FH JOANNEUM Gesellschaft mbH

Sicherheitsanalyse moderner Webtechnologien

HTML, JavaScript, Ajax, PHP

Bachelorarbeit 1

eingereicht am 30.01.2015
Fachhochschul-Studiengang Software Design
Jahrgang: 2012

Betreuer: Dipl.-Ing. Johannes Feiner

eingereicht von: Daniel Hösele
Personenkennzahl: 1210418031

Jänner 2015

Abstract

Modern technologies of the web provide a large number of possibilities for the development of rich web applications. Because of diversity, a large number of known vulnerabilities exist for each of these technologies. The thesis at hand therefore aims at giving an overview of the functionality of modern web technologies and analyses the Top-10 security vulnerabilities, which were discovered and published by the OWASP Foundation in 2013. The implementation of a working prototype will help to further investigate the vulnerabilities examined in the theoretical part of the work. In order to support programmers with the development of safer web applications, the results of the thesis are finally visualized and recommendations of avoiding such security risks are provided.

Zusammenfassung

Moderne Technologien im Web bieten umfangreiche Möglichkeiten zur Programmierung von Webapplikationen. Doch wie sicher sind diese Technologien und welche Maßnahmen sollten gesetzt werden um Sicherheitslücken vorzubeugen? Diese Arbeit gibt einen Überblick über die modernen Technologien des Webs und analysiert deren Sicherheitslücken. Durch die Erstellung eines Prototyps werden diese Sicherheitslücken exemplarisch umgesetzt und Präventionsmaßnahmen vorgeschlagen. Am Ende werden die Ergebnisse in einer Tabelle zusammengefasst, die als Checkliste für die Entwicklung sicherer Webapplikationen dienen soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ACL Access Control List

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BOM Browser Object Model

CSRF Cross-Site Request Forgery

CVE Common Vulnerabilities and Exposures

DHTML Dynamic HTML

DTD Document Type Definition

DOM Document Object Model

GPL General Purpose Language

HTML Hyper Text Markup Language

HTTP Hyper Text Transfer Protocol

JSON JavaScript Object Notation

PHP Hypertext Preprocessor

SAX Simple API for XML

SGML Standard Generalized Markup Language

SSE Server-Side Events

SVG Scalable Vector Graphics

URL Uniform Resource Locator

W3C World Wide Web Consortium

Web World Wide Web

XML Extensible Markup Language

XSS Cross Site Scripting

1 Einleitung

HTML5, JavaScript und Ajax sind die am meisten verbreiteten clientseitigen Technologien für die Entwicklung von Webapplikationen. Diese Technologien werden prinzipiell am Rechner des Clients ausgeführt, für gewöhnlich in dessen Browser, und sind deshalb sehr verwundbar für eine große Anzahl verschiedener Cyber-Attacken. Da die meisten Webentwickler keine Sicherheits-Spezialisten sind, haben viele über das World Wide Web (Web) erreichbare Webseiten Schwachstellen, da bei der Entwicklung der Fokus auf Design und Funktionalität, anstatt auf Sicherheit gelegt wurde.

Serverseitig gibt es unzählige Technologien, wie zum Beispiel PHP, Python, ASP.NET, Ruby uvm. In dieser Arbeit wird das Hauptaugenmerk jedoch auf clientseitige Technologien gelegt. Deshalb wird serverseitig PHP nur dann verwendet, wenn dies zur Umsetzung des Prototypen bzw. zur bewussten Implementierung von Schwachstellen notwendig ist. Diese Technologien werden direkt am Server ausgeführt und schicken nur die, beispielsweise von einem PHP-Skript erzeugten, HTML-Daten an den Client. Dieser Ansatz ist um einiges sicherer, da am Client keine Manipulation des PHP-Skripts vorgenommen werden kann.

Abbildung 1.1 zeigt den Prozentsatz der Programmierer, die bereits aktiv mit HTML5 entwickeln.

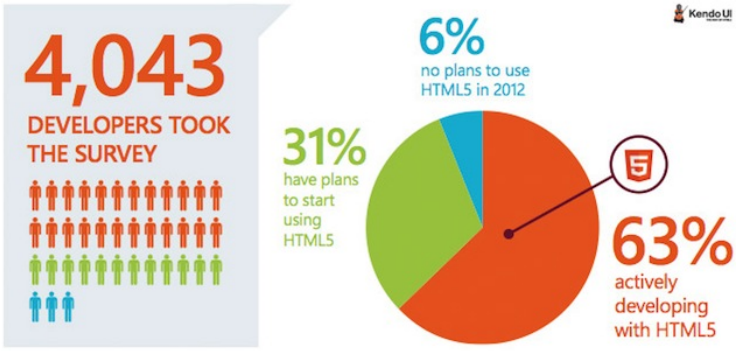

Abbildung 1.1: HTML5 Benutzung - Quelle: [11]

In den letzten Jahren haben sich diese Technologien ständig weiterentwickelt. Noch nie war es so leicht, Webapplikationen zu entwickeln und online zu stellen. Wie sicher diese Technologien sind und wie Entwickler bekannte Sicherheitslücken umgehen bzw. schließen können, wird in dieser Arbeit erforscht. Laut Chuan Yue und Haining Wang, die 6805 bekannte Webseiten auf Sicherheitslücken untersucht haben, verlinkten 66,4% dieser Seiten direkt im Header auf externe JavaScript-Dateien. Jeder, der sich Zugriff zu einer dieser Dateien beschafft, könnte darin Schadcode platzieren, welcher beim nächsten Aufruf der Website ausgeführt werden würde. Weiters fanden sie heraus, dass auf 44% der untersuchten Seiten die Funktion eval(), die document.write()-Methode und innerHTML-Property verwendet wurden, um JavaScript Code zu generieren.[24]

Abbildung 1.2 zeigt die Adaption von HTML5, und damit verbunden die Nutzung weiterer Technologien wie JavaScript, im Jahr 2013.

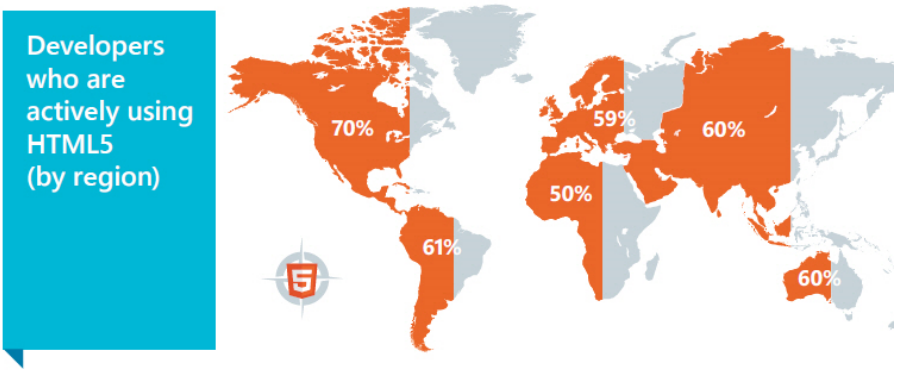

Abbildung 1.2: HTML5 Entwickler/Welt - Quelle: [18]

In dieser Arbeit wird zuerst ein Überblick über den Funktionsumfang moderner Webtechnologien gegeben. Anschließend wird eine Analyse der 2013 von der OWASP Foundation veröffentlichten Top-10 Sicherheitslücken von Webapplikationen durchgeführt. Um die Ergebnisse exemplarisch und grafisch aufzubereiten, wurde ein Prototyp erstellt. Abschließend fasst eine Checkliste die gefundenen Ergebnisse zusammen und ermöglicht so einen umfassenden Überblick über die Sicherheit der untersuchten Technologien. Grundlegende Kenntnisse des Lesers in der Programmierung von Webapplikationen werden vorausgesetzt.

2 Technologische Grundlagen

In den folgenden Unterkapiteln werden die Grundlagen der in dieser Arbeit verwendeten Technologien erläutert. Kapitel 2.7 gibt zuletzt einen Überblick, welche Technologien im Web noch zusätzlich Anwendung finden.

2.1 HTML5

HTML5 ist eine der Kerntechnologien, um Inhalte im Web zu strukturieren und präsentieren. Die fertige Spezifikation wurde im Oktober 2014 vom World Wide Web Consortium (W3C) vorgelegt, die letzte Version davor (HTML 4) stammte aus dem Jahr 1997.[2, vgl.] Sie ersetzt die Standards HTML 4.01, XHTML 1.0 und DOM HTML Level 2 und bietet unzählige neue Funktionen, die vor allem für mobile Endgeräte optimiert wurden. So wird nun auch das direkte Einbinden von Audio und Video-Elemente unterstützt, was bisher nur mit Hilfe von Browser Plug-Ins möglich war.

> HTML5 is a rebel. It was dreamt up by a loose group of freethinkers who weren't in charge of the official HTML standard. It allows page-writing practices that were banned a decade ago. It spends thousands of words painstakingly telling browser makers how to deal with markup mistakes, rather than rejecting them outright. It finally makes video playback possible without a browser plug-in like Flash. And it introduces an avalanche of JavaScript-fueled features that can give web pages some of the rich, interactive capabilities of traditional desktop software[9, S.11].

Wie MacDonald in seinem Buch "HTML5: The Missing Manual" schreibt, sind die neuen Funktionen von HTML5 vor allem auf Interaktivität und Kompatibilität ausgerichtet. Webseiten sollen nicht mehr rein aus statischen HTML bestehen, das bei jeder Interaktion komplett neu geladen werden muss, sondern mit Hilfe von JavaScript werden Daten dynamisch nachgeladen und mit DOM-Manipulationen im Browser angezeigt.

Listing 2.1 gibt einen Überblick, wie eine HTML5-Seite aufgebaut ist:

HTML-Code:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>HTML5 Markup</title>
  </head>
  <body>
    <h1>HTML5 rocks!</h1>
  </body>
</html>

```

Listing 2.1: HTML Beispiel

Am Anfang eines HTML-Dokuments muss der DOCTYPE deklariert werden. Da Hyper Text Markup Language (HTML) 4.01 auf SGML basierte, musste die DOCTYPE-Deklaration auf eine Document Type Definition (DTD) verweisen. Die DTD spezifiziert die Regeln einer Auszeichnungssprache, damit der Browser den Inhalt richtig anzeigen kann. HTML5 basiert nicht auf SGML, deshalb ist keine Referenz mehr auf eine DTD erfordlich.[21]

Das <head>-Element dient als Container für alle Kopfelemente. Es beinhaltet meistens den Titel der Seite, JavaScripts, Stylesheets und Meta-Tags. In HTML5 ist die Verwendung des <head>-Elements nicht mehr zwingend erforderlich.

Das <body>-Element definiert den Bereich für den Inhalt einer Seite. Üblicherweise sind das Texte, Hyperlinks, Grafiken, Tabellen, Listen, Formen uvm.

Damit die jeweilige Website in allen Browsern gleich angezeigt wird, ist es empfehlenswert, den Source-Code auf http://validator.w3.org/ validieren zu lassen.

Neu in HTML5

Wie bereits erwähnt, gibt es einige neue Elemente und Attribute in HTML5. Einige davon sind semantischer Natur und ersetzen generische Block und Inline-Elemente.

Semantische Elemente (z.B. <header>, <footer>, <article>, <section>, <nav>, <figure>,...) helfen dem Entwickler den Inhalt besser zu strukturieren.

Durch die Einführung der grafischen Elemente <svg> und <canvas> können nun direkt Vektor bzw. Rastergrafiken in den HTML5-Code eingebunden werden. Der Vorteil von Scalable Vector Graphics (SVG) liegt darin, dass jede Form als Objekt im DOM vorhanden ist und anschließend als Bitmap gerendert wird. Sobald sich ein Attribut des SVG-Objektes ändert, kann der Browser automatisch die Szene neu rendern.

Weiters wurden die multimedialen Elemente <audio> und <video> eingeführt, mit denen Audio und Video-Dateien, ohne der Benutzung von Plug-Ins, eingebettet werden können.

Listing 2.2 zeigt die Verwendung eines solchen Video-Elements:

HTML-Code:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

Listing 2.2: HTML Video-Element

Weiters spezifiziert HTML5 API's, die mit JavaScript oder über Mark-Up Komponenten genutzt werden können[19]:

- Media API
  Abspielen von Audio und Video-Dateien über die neuen <audio> und <video>-Elemente.
- Geolocation API
  Zugriff auf Ortungsdienstinformationen für die weitere Verwendung in Skripten.
- Drag and Drop API
  Elemente können mit dem "draggable"-Attribute versehen werden.
- Local/Web Storage API
  Daten können im Cache des Browsers gespeichert werden, um später verfügbar zu sein.

- Web Workers API
  Erlaubt die Ausführung von Skripten im Hintergrund (Multi-Tasking).
- WebSockets API
  Hält eine Verbindung zwischen Client und Server offen, so das Daten in Echtzeit hin und her übertragen werden können (Server-Side Events).

Nachteile von HTML5

Auch wenn die Vorteile durch die Neuerungen von HTML5 überwiegen, gibt es dennoch Nachteile, die hier anzumerken sind:

Im Gegensatz zu XHTML müssen bei HTML5 Tags nicht XML-konform geöffnet und geschlossen werden und Attribute dürfen minimalisiert werden. Ein Beispiel ist das "defer"-Attribut, welches die Ausführung eines JavaScripts bis zur vollständigen Parsung der Seite verzögert:

> In XHTML, attribute minimization is forbidden, and the defer attribute must be defined as <script defer="defer">[22].

Weiters erlaubt HTML5 die Ausführung von nicht well-formed Code, der Entwickler kann aber jederzeit XML-konformen Code schreiben, wenn er folgende Regeln befolgt:

- Element korrekt geöffnet und auch wieder geschlossen: Bold
- Leeres Element:
, <hr /> etc.
- Korrekt verschachtelt: ItalicBold
- Attribute klein schreiben:
- Attribute nicht minimalisieren: <script defer="defer">

Ein großer Nachteil war auch, dass HTML5 lange nicht offiziell standardisiert war. Im Oktober 2014 wurde jedoch von der W3C eine "Recommendation" und damit De-facto-Norm von HTML5 veröffentlicht[2, vgl.].

2.2 Document Object Model

Das Document Object Model (DOM) ist eigentlich kein Modell, sondern eine Schnittstelle, um den Inhalt und die Struktur eines Dokuments dynamisch durch ein Computerprogramm ändern zu können. Dabei werden Dokumente hierarchisch in einem Baum dargestellt, Knoten stehen über Beziehungen miteinander in Verbindung.

> The DOM is an API for HTML and XML documents. The DOM represents a document as a hierarchical tree of nodes, allowing developers to add, remove, and modify individual parts of the page. Evolving out of early DHTML innovations from Netscape and Microsoft, the DOM is now a truly cross-platform, language-independent way of representing and manipulating pages for markup[25, S.309].

Abbildung 2.1 zeigt, wie das HTML5-Codebeispiel als Objektmodell dargestellt wird:

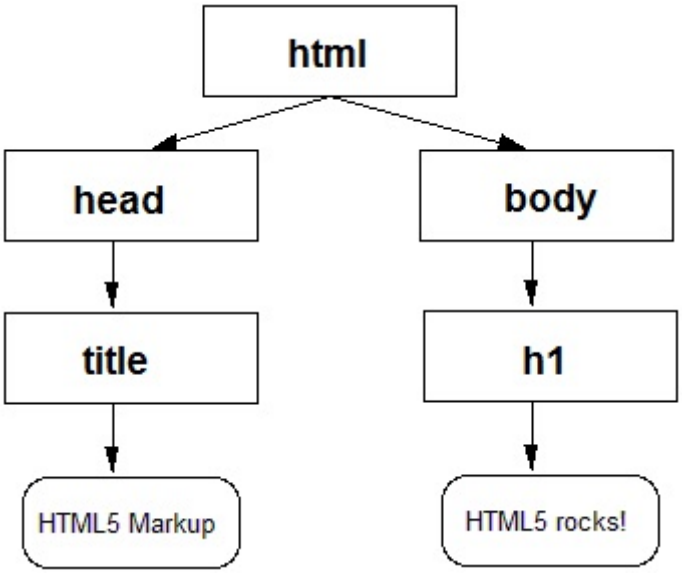

Abbildung 2.1: DOM - Hierarchische Baumstruktur eines HTML-Dokuments

Der Wurzelknoten ist in diesem Fall das html-Element, “head” und “body” sind jeweils die Kinder von “html”, und umgekehrt ist “html” das Elternteil der beiden. Das gleiche gilt für “title” und “h1”. Durch diese Parent-Child-Beziehungen kann von jedem beliebigen Knoten im Dokument, auf darüber oder darunterliegende Knoten zugegriffen und Manipulationen vorgenommen werden.

Simple API for XML (SAX)

Alternativ zu DOM gibt es die SAX, ein Application Programming Interface (API) zum Parsen von XML-Daten.

> Document Object Model (DOM) and Simple API for XML (SAX) are the two major programming models for XML document processing. Each, however, has its own efficiency limitation. DOM assumes an in-core representation of XML documents which can be problematic for large documents. SAX needs to scan over the document in a linear manner in order to locate the interesting fragments.[6].

Der Unterschied zu DOM besteht darin, dass SAX XML-Daten als sequentiellen Datenstrom liest und für im Standard definierte Ereignisse vorgegebene Rückruffunktionen (Callbacks) aufruft. SAX definiert eine Menge von Ereignissen, die beim Lesen des Dokuments vorkommen können und startet beim Erkennen einer syntaktischen Struktur eine Behandlungsroutine.

Dadurch kann bereits nach wenigen eingelesenen Zeichen bereits mit der Auswertung des Dokuments begonnen werden, was die subjektive Zugriffszeit verkürzen kann.

2.3 Browser Object Model

Das Browser Object Model (BOM) erlaubt den Zugriff und Manipulationen des Browserfensters. Durch die Einführung von HTML5 wurde das BOM erstmals formal spezifiziert. Prinzipiell ist das BOM für die Interaktion mit dem Browserfenster und dessen Frames zuständig[25, S.9].

Folgende Möglichkeiten bietet das BOM:

- Erstellen und Pop-up neuer Fenster
- Verschieben, Änderung der Größe und Schließen des Fensters
- Navigator-Objekt: Details über den verwendeten Browser
  Wichtig für SPA (Single-page applications)

- Location-Objekt: Details über die geladene Webseite
- Screen-Objekt: Details über die Bildschirmauflösung
- Unterstützung von Cookies

Nicholas C. Zakas, ein ehemaliger Mitarbeiter von Yahoo!, beschreibt in seinem Buch "Professional JavaScript for Web Developers" das bisherige Problem mit BOM sehr treffend:

> Because no standards existed for the BOM for a long time, each browser has its own implementation. There are some de facto standards, such as having a window object and a navigator object, but each browser defi nes its own properties and methods for these and other objects. With HTML5 now available, the implementation details of the BOM are expected to grow in a much more compatible way[25, S.9-10].

2.4 JavaScript

JavaScript ist eine dynamische Programmiersprache, und ist im heutigen Web sehr weit verbreitet. Es wird benutzt, um mit Hilfe von clientseitigen Skripten mit dem User zu interagieren, Daten asynchron nachzuladen und den Inhalt der geladenen Seite zur Laufzeit zu verändern.

Eine komplette JavaScript-Implementierung besteht meist aus den folgenden Teilen[25, S.3]:

- Core (ECMAScript)
- Document Object Model
- Browser Object Model

Skripte werden entweder direkt in die HTML-Seite eingebunden, oder in externe JS-Dateien ausgelagert. Listing 2.3 zeigt eine simple Java-Script Anwendung:

JavaScript:

```
1  <title>BA1: Sicherheitsanalyse moderner Webtechnologien</title>
2  <script>
3  //fixes layout issue on small screens at onload
4  var w = window.innerWidth;
5  if (w < 512) {
6    document.getElementById("title").textContent = "BA1: Sicherheitsanalyse";
7  }
8  //fixes layout issue on small screens dynamically at runtime
9  $(window).resize(function() {
10   var viewportWidth = $(window).width();
11   if (viewportWidth < 512) {
12     document.getElementById("title").textContent="BA1: Sicherheitsanalyse";
13   } else {
14     document.getElementById("title").textContent=
15               "BA1: Sicherheitsanalyse moderner Webtechnologien";
16   }
17 });
18 </script>
```

Listing 2.3: JavaScript Beispiel

Beim Aufruf einer Website, in der dieses Skript inkludiert ist, wird abgefragt, ob die Breite des Fensters den Wert “512” unterschreitet und wenn dies der Fall ist, der Titel der Website geändert, damit er zum Beispiel auf mobilen Endgeräten nicht über den Bildschirmrand hinausragt.

Sollte sich die Größe des Fensters zur Laufzeit ändern (z.B.: Hochformat/Querformat durch Kippen des Smartphones) wird, durch den Code ab Zeile 9, der Titel dynamisch geändert.

Sicherheit

JavaScript und DOM bieten Angreifern die Möglichkeit, Skripte zu schreiben, die Schadcode enthalten, der im Browser des Clients ausgeführt werden kann. Deshalb wurden zwei Sicherheitskonzepte in JavaScript integriert. Zum einen werden Skripte prinzipiell in einer Sandbox ausgeführt, d.h. es können nur Operationen im Browser durchge-

führt werden, und nicht Systemspezifische wie das Erstellen einer Datei (über File-API möglich). Zweitens müssen sich Skripte an die “Same Origin Policy” halten. Die Skripte einer Webseite haben keinen Zugriff auf Usernamen, Passwörter und Cookies, die an eine andere Webseite geschickt werden. Die meisten Sicherheitslücken sind jedoch Verletzungen dieser beiden Konzepte.

2.5 Ajax

Ajax steht für Asynchronous JavaScript and XML und ist ein Ansatz, um Daten asynchron senden und empfangen zu können, ohne dabei die Webseite neu laden zu müssen. Daten können mittels des XMLHttpRequest-Objekts geholt werden. Um Daten zwischen Client und Server zu schicken, werden oft Extensible Markup Language (XML) oder JavaScript Object Notation (JSON) verwendet.

In Listing 2.4 wird ein Ajax-Request erläutert:

JavaScript:

```
var ajaxRequest;
function sendAjaxRequest() {
  ajaxRequest = new XMLHttpRequest();
  ajaxRequest.open("GET", "http://api.website.org/data/api?var=xxx");
  ajaxRequest.onreadystatechange = ajaxCallback;
  ajaxRequest.send(null);
}
function ajaxCallback() {
  if (ajaxRequest.readyState == 1) {
    console.log ( 'server connection established' );
  } else if (ajaxRequest.readyState == 2 ) {
    console.log ( 'request received' );
  } else if (ajaxRequest.readyState == 3 ) {
    console.log ( 'processing request' );
  } else if (ajaxRequest.readyState == 4 ) {
    console.log ( 'request finished and response is ready' );
    postMessage(JSON.parse(ajaxRequest.responseText));
  }
}
```

```
20 sendAjaxRequest();
```

Listing 2.4: JavaScript Ajax-Call

Zuerst wird die Variable “ajaxRequest” deklariert und die beiden nötigen Funktionen implementiert. Durch das Aufrufen von “sendAjaxRequest();” wird ein XMLHttpRequest erzeugt und mit der Methode “GET” eine Verbindung zur API der Website geöffnet. Die Zeilen 9-15 zeigen, welchen Wert die readyState-Eigenschaft im Laufe einer Anfrage an den Server hält. Wenn die HTTP-Antwort vollständig geladen wurde, wird der Wert der readyState-Eigenschaft auf 4 gesetzt und der Ajax-Request ist abgeschlossen. Die dieses Snippet aus einem WebWorker stammt, wird der Inhalt der Antwort geparsed und mit “postMessage()” zurück an den Aufrufer geschickt.

2.6 PHP

Hypertext Preprocessor (PHP) ist eine serverseitige Skriptsprache, die eigentlich für die Web-Entwicklung entwickelt wurde, jedoch auch für GPL-Programmierung verwendet wird. Die neueste stabile Version 5.6.4 stammt vom 18. Dezember 2014. PHP-Code kann leicht mit HTML-Code vermischt werden und wird von einem PHP-Interpreter, der normalerweise im Web-Server integriert ist oder nachinstalliert werden kann, ausgeführt. Der Server generiert aus dem PHP-Skript üblicherweise HTML-Quelltext, der an den Client geschickt und entsprechend angezeigt wird.

> PHP is the engine behind millions of dynamic web applications. Its broad feature set, approachable syntax, and support for different operating systems and web servers have made it an ideal language for both rapid web development and the methodical construction of complex systems[20, S.15].

PHP ist aufgrund seiner Heterogenität weit verbreitet und ist mit den meisten Datenbanken kompatibel. Das Parsen von Form-Daten ist ebenso leicht umsetzbar wie das Ausführen von HTTP-Requests.

Das Code-Beispiel in Listing 2.5 zeigt ein triviales PHP-Skript:

PHP-Skript:

```
1 <?php
```

```
2  $file = fopen("./message", "w") or die("Unable to open file!");
3  if ($_GET["msg"]=="rocks") {
4    $msg = "PHP rocks!";
5  } else {
6    $msg = "No Message";
7  }
8  fwrite($file, $msg);
9  fclose($file);
10 ?>
```

Listing 2.5: PHP Beispiel

Hierbei wird mit “fopen” die Datei “message” geöffnet oder erstellt, falls sie nicht existiert. Sollte dabei ein Fehler auftreten, zum Beispiel aufgrund fehlender Rechte, bricht das Programm ab. Ansonsten wird der GET-Parameter “msg” ausgelesen und der Wert der gleichnamigen Variable zugewiesen. Falls der GET-Parameter nicht gesetzt ist, wird der Variable der String “No Message” zugewiesen. Zum Schluss wird der Wert der Variable in die Datei geschrieben und dieselbe geschlossen.

2.7 Weitere Technologien

Um die Vollständigkeit zu gewährleisten, ist an dieser Stelle anzumerken, dass im heutigen Web natürlich weit mehr Technologien involviert sind. Clientseitig sind unter anderem CSS, MS J-Script, MS VB-Script und Flash in Verwendung und serverseitig meist ASP.NET, Perl, Python, JSP, Ruby uvm.

Die Kapitel 2.1 - 2.6 haben einen grundlegenden Überblick über die modernen Technologien des Webs gegeben. Zusammenfassend ist zu sagen, dass Sicherheitslücken oft durch das Zusammenspiel von mehr als einer dieser Technologien auftreten. Im folgenden Kapitel werden nun die 10 häufigsten Sicherheitslücken von Webapplikationen untersucht, die in vielen Fällen ebensolche Technologien betreffen.

3 Analyse bekannter Sicherheitslücken

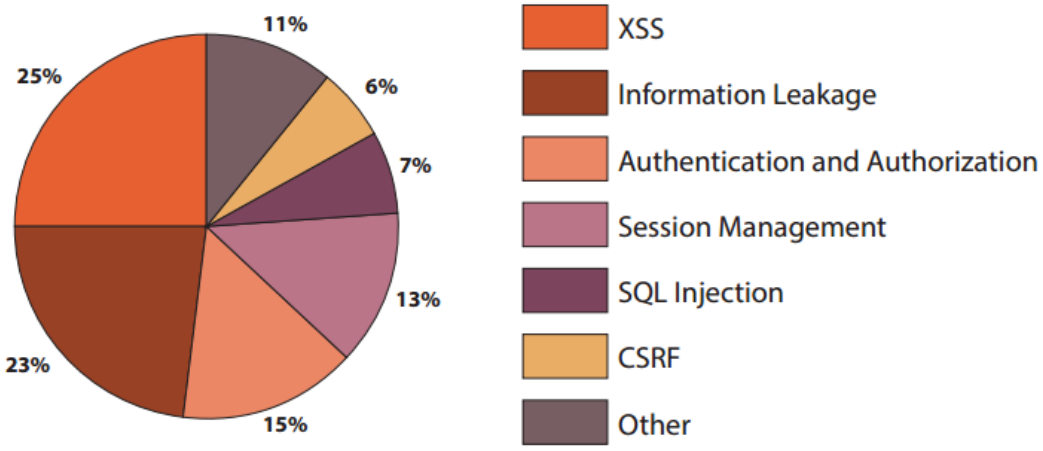

Abbildung 3.1: Sicherheitslücken von Webapplikationen - Quelle: [4]

Abbildung 3.1 zeigt die Häufigkeit von Sicherheitslücken in Webapplikationen in Prozent. Für die Analyse wurden die Top 10-Sicherheitsrisiken von Webapplikationen, die 2013 von der OWASP Foundation veröffentlicht wurden, herangezogen[13]. Der Unterschied zur Studie der OWASP Foundation liegt darin, dass mit der Erstellung des Prototyps, jeweils ein interaktives Beispiel zu jeder Sicherheitslücke erstellt wird, und so die Funktionsweise näher betrachtet werden kann.

3.1 Cross-Site Scripting (XSS)

Wie bereits in Kapitel 2.4 beschrieben, basiert Security im Web auf einigen Mechanismen, wie zum Beispiel der “Same Origin Policy”. Diese besagt, dass der Browser den Zugriff einer Website auf Ressourcen des Systems nur für die jeweilige Domäne zulässt, jeder anderen Webseite müssen separate Berechtigungen erteilt werden. Dies

verhindert, dass ein Skript einer Website auf Daten (Passwörter, Cookies, ...) einer anderen Website zugreifen und dadurch bewusste Schädigungen anrichten kann.

Cross Site Scripting (XSS) tritt auf, wenn[12, CWE-79]:

1. Nicht vertrauenswürdige Daten in eine Webapplikation eingeschleust werden
2. Die Webapplikation dynamisch eine Seite generiert, die solche Daten enthält
3. Während der Erstellung der dynamischen Seite keine Mechanismen vorhanden sind, die das Erstellen von ausführbaren Daten (JavaScripts, HTML-Tags, ActiveX etc.) unterbinden.
4. Ein Besucher die Webseite aufruft, die nun eine Script mit Schadcode enthält.
5. Da das Script nun von der aufgerufenen Webseite kommt, führt der Browser dieses mit den Berechtigungen der jeweiligen Domäne aus.
6. Dadurch wird die “Same Origin Policy” verletzt.

Generell wird zwischen 3 Arten von XSS unterschieden[12, CWE-79]:

1. Reflected XSS (Nicht Persistent)
   Dieser XSS-Typ ist der am weitesten verbreitete. Der Server liest die Daten direkt aus einem HTTP-Request und sendet die Antwort in einer HTTP-Response. Der Angreifer bringt das Opfer dazu, gefährlichen Code an eine ungeschützte Webapplikation zu schicken, die dann vom Server in der HTTP-Response zurückgeschickt und vom Browser ausgeführt werden. Die häufigste Methode ist, den Schadcode in einem Parameter einer URL zu inkludieren, und diesen öffentlich zu machen oder dem Opfer per E-Mail zu schicken.
2. Stored XSS (Persistent)
   Bei dieser Variante speichert die Applikation den Schadcode in einer Datenbank, einem Forum, Log oder einem anderen vertrauenswürdigen Datenspeicher. Später wird der Schadcode von der Applikation wieder geladen und in einen dynamischen Inhalt inkludiert. Dadurch kann der Angreifer zum Beispiel bestimmte Benutzer mit Administrator-Rechten attackieren.
3. DOM-Based XSS
   Dieser Typ von XSS tritt auf, wenn entweder der Client den Schadcode selbst in die Seite injiziert, oder der Server dies verursacht. Wenn der Server zum Beispiel erlaubt, dass die bereitgestellten Daten von Benutzern ausgeführt werden

können, ist DOM-Based XSS möglich.

Prävention

Die OWASP-Foundation hat sieben Regeln erstellt, um dieser Art von Attacke vorzubeugen [14]. Prinzipiell sagen diese aus, dass eine Applikation nicht vertrauenswürdigen Date nicht annehmen/einfügen soll, außer an sicheren Stellen. An diesen sicheren Stellen, muss ein "Escape-Mechanismus" implementiert werden, um sicherzustellen, dass kein ausführbarer Schadcode injiziert werden kann. Zusätzlich wird die Implementierung einer Cookie-Security empfohlen, wie zum Beispiel das Binden eines Session-Cookies an die IP des Benutzers, der sich ursprünglich eingeloggt hat.

3.2 Injection Flaws

Injection Flaws erlauben es Angreifern, Schadcode durch eine Webapplikation an andere Systeme weiterzuleiten. Bei solchen Attacken kommen Zugriffe auf das Betriebssystem über "System calls", die Benutzung von externen Programmen über Shells-Befehle, sowie Zugriffe auf Datenbanksysteme (SQL-Injection), zum Einsatz[15].

> Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws[13].

Injection Flaws betreffen also jede Webapplikation, welche einen Interpreter benutzt. Wenn keine nötigen Maßnahmen getroffen wurden, und ein Angreifer spezielle Kommandos eingibt, werden diese blindlings an das externe System zur Ausführung weitergegeben.

Prinzipiell gibt es 2 Arten von Injection Flaws:

1. Command Injection
   Hierbei wird durch spezielle Elemente die dynamische Generierung eines Kom-

mandos manipuliert und der Angreifer erhält somit Privilegien, die er sonst nicht hätte[12, CWE-77].

2. SQL-Injection
   Ohne auszureichendes Entfernen oder Auskommentieren von SQL-Syntax in Eingabefeldern, können die Daten in so generierten SQL-Abfragen, als SQL anstatt Benutzerdaten interpretiert werden. Dadurch können Sicherheitschecks umgangen oder das Ausführen von System-Befehlen ausgelöst werden[12, CWE-89].

Prävention

Der einfachste Weg das System gegen Injection Flaws zu schützen ist, wann immer möglich, externe Interpreter zu vermeiden. Viele Shell-Befehle und einige "System calls" sind über programmiersprachenspezifische Bibliotheken umsetzbar. Weiters ist anzuraten, jede Benutzereingabe als Schadcode anzusehen, und praktisch über eine White-List ungewünschte Eingaben wegzufiltern. Ebenso sollte jede involvierte Software nur mit den jeweils benötigten Rechten ausgeführt werden.

3.3 Broken Authentication and Session Management

Authentication und Session Management inkludiert alle Aspekte der Authentifizierung und dem Management aktiver Sitzungen eines Benutzers. Durch Funktionen wie das Zurücksetzen oder Ändern des Passworts können sogar sichere Authentifizierungsmechanismen untergraben werden. In den meisten Fällen wird nur ein Benutzername und ein Passwort benötigt, um sich bei einer Webapplikation anmelden zu können. Sicherere Methoden, wie zum Beispiel RSA-Tokens oder biometrische Hardware, sind zwar erhältlich, aber für eine simple Webapplikation keine adäquate Lösung[16].

Prävention

Folgende Punkte sollten zur Absicherung beachtet werden[16]:

1. Passwortstärke/länge

2. Begrenzte Anzahl an Loginversuchen
3. Eingabe von altem Passwort bei Passwortänderung
4. Speicherung von Passwörtern in gehashtem oder enkodiertem Format
5. Verwendung von SSL bei Login-Vorgang
6. Verwendung von SSL bei Session-Verwendung
7. Caching: Authentifizierung oder Session-Daten sollten niemals über GET übertragen werden, sondern über POST
8. Verwendung des “no cache” Tags in Authentifizierungsseiten
9. Verwendung von “AUTOCOMPLETE=OFF”

3.4 Insecure Direct Object References

Insecure Direct Object References treten auf, wenn durch eine Benutzereingabe direkter Zugriff auf ein Objekt der Webapplikation erlangt werden kann. In seiner simpelsten Form kann zum Beispiel einfach in einer URL die User-ID auf eine Andere geändert, und somit der Zugriff auf Datenbankeinträge oder Dateien erlangt werden. Dies ist möglich, wenn die Applikation Benutzereingaben ohne Berechtigungsabfrage annimmt, und das jeweilige Objekt direkt holt.

> A 2010 security breach that leaked the email addresses of over 114,000 Apple iPad users — including politicians and CEOs — was the result of hackers exploiting an insecure direct object reference in AT&T’s systems architecture. Similar attacks have been carried out against the Australian Tax Office and other major organizations[3].

Prävention

Um Insecure Direct Object References zu vermeiden, muss jedes Objekt, auf das ein Benutzer Zugriff hat, einen Schutzmechanismus implementieren[13, S.10]

Möglichkeiten:

1. Indirekte Objekt-Referenzen per User oder Session
2. Zugriffskontrolle für jedes direkt referenzierte Objekt

3.5 Security Misconfiguration

Eine solche Sicherheitslücke kann auftreten, wenn eine Komponente der Webapplikation falsch bzw. unsicher konfiguriert worden ist. Diese Sicherheitslücken treten meistens durch eine unsichere oder schlecht dokumentierte Standard-Konfiguration oder durch Nebeneffekte von optionalen Konfigurationen auf.

> Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc [13].

Prävention

Grundsätzlich sollte die involvierte Software immer Up-to-date gehalten werden. Weiters sollte für die Applikation, die Frameworks, den Applikations/Web/Datenbank-Server und für die Plattform eine sichere Konfiguration definiert und eingesetzt werden. Standard-Konfigurationen sind meistens nur für den Entwicklungsprozess gedacht, und sollten nicht für den produktiven Einsatz im Web verwendet werden.

3.6 Sensitive Data Exposure

Wenn eine Webapplikation sensible Daten nicht ausreichend schützt, können Angreifer Zugriff auf diese Daten erlangen. Meistens betrifft ein solcher Angriff die Passwörter von Benutzern, es kann sich aber auch um Kreditkartendaten, Session Token oder andere Zugangsdaten handeln. Wenn Daten unverschlüsselt, also in Klartext, gespeichert oder übertragen werden, können Angreifer leicht an diese Daten kommen.

Ein Angreifer kann zum Beispiel, mit Hilfe eines “Netzwerksniffers” übertragene Pakete abfangen. Wenn die darin enthaltenen Daten nicht verschlüsselt sind, kann er sie einfach ablesen.

Prävention

Folgende Maßnahmen sollten als Minimum betrachtet werden[13, S.12]:

1. Verschlüsselung von sensiblen Daten bei Übertragung und Speicherung
2. Keine dauerhafte Speicherung von unnötigen sensiblen Daten
3. Starke Verschlüsselungsalgorithmen und Keys
4. Passwörter mit speziellen Algorithmen verschlüsseln (bcrypt, PBKDF2, scrypt)
5. Deaktivieren von Autocomplete und Caching auf Seiten, die sensible Daten enthalten

3.7 Missing Function Level Access Control

Wenn der unautorisierte Zugriff auf private Funktionen möglich ist, können Angreifer zum Beispiel durch die Abänderung eines URLs, sensible Daten eines anderen Benutzers einsehen. Weiters können so Rechte erlangt werden, die der Angreifer normalerweise nicht besitzen würde. Das Verstecken dieser Funktionen (private vs. public) alleine, bietet keinen ausreichenden Schutz gegen unautorisierten Zugriff.

> When developers create web interfaces, they have to restrict which users can see various links, buttons, forms, and pages. Developers usually get this right because it is very visible. Unfortunately, making it pretty doesn’t make it secure. Developers often forget that they also have to put access controls in the business logic that actually performs business functions[23].

Prävention

Ein Ansatz, um diesem Problem vorzubeugen ist, standardmäßig den Zugriff auf alle Funktionen zu verbieten. Über eine White-List kann dann benutzerspezifisch Zugriff

auf eine bestimmte Funktion erteilt werden. Weiters helfen Access Control Lists und rollenbasierte Authentisierungsmechanismen gegen diese Sicherheitslücke[8].

3.8 Cross-Site Request Forgery (CSRF)

Bei einer Cross-Site Request Forgery-Attacke nutzt der Angreifer die Art und Weise, wie die Authentifizierung einer Webapplikation funktioniert. Der Angreifer platziert auf seiner Website einen Link oder ein Script, welches sich mit der Ziel-Webapplikation verbindet. Ist das Opfer dort bereits angemeldet oder ein Cookie noch nicht abgelaufen, werden die Aktionen des Angreifers ausgeführt und für die Applikation sieht es so aus, als wäre das vom Benutzer gewünscht gewesen. Potentielle Ziele sind unter anderem Social Web Applikationen, Online-Banking-Portale und Online-Email-Clients[7].

Prävention

Die gängigste Methode um Cross-Site Request Forgery-Attacken vorzubeugen, besteht darin einen unvorhersehbaren Token an jeden Request anzuhängen. Dieser Token steht in Zusammenhang mit der Session des Benutzers und sollte im besten Fall pro Request generiert werden[7]. Weiters besteht die Möglichkeit, die Referer und Origin-Header auszulesen und mit der IP-Adresse des eingeloggten Benutzers abzugleichen[17].

3.9 Using Components with Known Vulnerabilities

Sicherheitslücken in Bibliotheken und Software von Drittanbietern sind weit verbreitet und kann die Sicherheit der Applikation gefährden. In den letzten Jahren wurden ca. 4500 solcher CVE's pro Jahr veröffentlicht.

> Modern applications frequently leverage hundreds of libraries. All of this code runs with the full privilege of the application, so vulnerabilities can be devastating. A recent study by Aspect Security of over 113 million library downloads by developers in 60,000 organizations, showed that 26 percent of those downloads contain known vulnerabilities[23].

Als Beispiel ist die “Remote-code execution with Expression Language Injection” - Sicherheitslücke zu nennen, die kürzlich als Teil des Spring-Frameworks in Java veröffentlicht wurde. Die Lücke wurde schnell geschlossen und eine neue Version des Frameworks veröffentlich. Trotzdem enthalten ca. 29,8 Millionen Downloads diese Sicherheitslücke[1].

Prävention

Folgende Punkte wurden von der OWASP-Corporation definiert[13, S.15]:

1. Identifizierung aller Komponenten und Versionen, die verwendet werden inkl. Abhängigkeiten
2. Kontrolle dieser Komponenten in Exploit-Datenbanken
3. Erstellen von Sicherheitsrichtlinien für die Verwendung von Komponenten
4. Verwendung von “Security-Wrappern”, um nicht benötigte Funktionalität zu deaktivieren

3.10 Unvalidated Redirects and Forwards

Dieses Problem tritt auf, wenn eine Webapplikation einen Parameter annimmt, ohne diesen zu validieren und den Benutzer auf den Wert dieses Parameters weiterleitet. Angreifer können zum Beispiel Phishing-Mails mit einem Redirect-URL versenden. Der Wert des Parameters zeigt dann auf eine Webseite die Schadcode oder Malware enthält. Einem Laien fällt dies nicht auf, da der Name der eigentlichen Website in der URL aufscheint.

> The user may be subjected to phishing attacks by being redirected to an untrusted page. The phishing attack may point to an attacker controlled web page that appears to be a trusted web site. The phishers may then steal the user’s credentials and then use these credentials to access the legitimate web site[12, CWE-601].

Prävention

Die einfachste Möglichkeit besteht darin, keine Redirects und Forwards zu verwen-

den. Wenn doch, sollte kein Parameter zur Berechnung der Zieladresse verwendet werden. Eine weitere Möglichkeit besteht darin, statt dem eigentlichen URL einen Mapping-Wert zu verwenden, der serverseitig in den richtigen URL übersetzt wird.

Zusammenfassung

Die Analyse der Sicherheitslücken hat gezeigt, dass durch kleine Unachtsamkeiten oder Unwissenheit von Seiten des Entwicklers schwerwiegende Sicherheitsrisiken in Webapplikationen auftreten können. Um die Sicherheit auf Seiten des Benutzers zu erhöhen, kann es von Nutzen sein, "clientseitige Security-Policies" zu verwenden:

> There are a number of benefits to the use of client-side security policies, with auditability being one of them. Client-side security policies are typically sent via HTTP response headers, thus it is straightforward to determine a website's security expectations by passive analysis. By instructing browsers to enforce protection through server provided policies, websites can address a number of security issues in a very straightforward manner. Today, several client-side security policies exist, all of which address various security issues such as cross-site scripting (XSS) and clickjacking attacks. Although these policies are not security panacea, their usage on a website can indicate the "security consciousness" of that website and its security objectives[5].

Um CSP (Content Security Policy) zu aktivieren, muss "X-Content-Security-Policy" im Header einer HTTP-Response enthalten sein. Der Inhalt des Headers enthält entweder die Policy, die erzwungen werden soll oder zeigt auf eine Datei, in der die jeweiligen Regeln enthalten sind. Diese Datei muss vom selben Ursprung (Domäne), wie das zu schützende Dokument kommen. Der Zweck liegt darin festzulegen, welche Ressourcen von wo geladen werden dürfen.[10, vgl.].

Um die Ergebnisse dieses Kapitels genauer untersuchen zu können, wurde die Umsetzung eines Prototypen geplant, die im nächsten Kapitel näher beschrieben wird.

4 Prototyp

In diesem Kapitel werden die Planung und die Umsetzung des Prototyps beschrieben.

Planung

Der Prototyp soll dazu dienen, die in Kapitel 3 gewonnenen Erkenntnisse anhand von Beispielen exemplarisch umzusetzen und Entwicklern so eine Plattform mit den grundlegenden Funktionsweisen und Präventationsmaßnahmen der untersuchten Sicherheitslücken zur Verfügung zu stellen. Der Prototyp soll mit Hilfe der in Kapitel 2 beschriebenen modernen Webtechnologien umgesetzt werden. Da das Design in Bezug auf Sicherheitslücken nicht wirklich von Relevanz ist, wird der Prototyp mit Hilfe des Frameworks “Bootstrap” gestaltet.

Ansonsten soll die Applikation clientseitig rein aus HTML5, JavaScript und Ajax-Aufrufen und serverseitig aus PHP bestehen. Datenbankspezifische Operationen werden mit SQLLite3 umgesetzt. Als Webserver dient das Programm “Server.app”, welches ebenfalls aus dem Hause Apple stammt. Der Source Code wird mit dem Programm Google Chrome Version 39 ausgeführt. Um Fehler zu finden bzw. Sicherheitslücken und deren Verhalten näher untersuchen zu können, werden die in Google Chrome integrierten “Web Developer Tools” benutzt.

Umsetzung

Wie bereits erwähnt, wurden für die Umsetzung des Prototyps, die von der OWASP Foundation veröffentlichten und in Kapitel 3 analysierten Sicherheitslücken herangezogen. Für jede Sicherheitslücke wurde eine Webseite erstellt, diese darin beschrieben, mit einem interaktivem Beispiel versehen und Präventionsmaßnahmen vorgeschlagen. Weiters wurden am Ende jeder Seite weiterführende Links zum Thema angeführt.

Da der Prototyp über das Internet verfügbar sein wird, wurden eine Authentifizie-

rungseite und ein Session-Management implementiert. Weiters wurden alle Inhalte nach dem Prinzip “Mobile First” gestaltet, und durch die Verwendung des Frameworks “Bootstrap” für das Design, kann die Applikation auf Handys, Tablets und herkömmlichen Desktop-PCs verwendet werden.

Folglich soll veranschaulicht werden, wie eine Sicherheitslücke in der Applikation aufbereitet wurde:

1. Beschreibung:
   Zu allererst wird die jeweilige Sicherheitslücke beschrieben und eventuell ein Überblick über die Funktionsweise und deren Hintergründe gegeben.

2. Beispiel:
   Dieser Punkt kann als Hauptziel der Entwicklung des Prototyps gesehen werden. Hier soll exemplarisch gezeigt werden, wie eine Sicherheitslücke durch falsche oder unsachgemäße Implementierungen entstehen kann und welche Auswirkungen das auf den Benutzer bzw. den Browser des Benutzers haben kann.

3. Prävention:
   Anschließend werden Tipps und Hilfestellungen zur Vermeidung dieser spezifischen Sicherheitslücke vorgeschlagen.

4. Weiterführente Links:
   Am Schluss werden weiterführende Links angegeben, um dem Benutzer einen tieferen Einblick in die Materie gewährleisten zu können.

Beispiel: XSS

Die Website liest über ein PHP-Skript einen GET-Parameter “msg” aus und zeigt den Wert auf der Seite an, wenn dieser über GET übergeben wurde. Da in Listing 4.1 kein Escape-Mechanismus implementiert wurde, kann nicht nur Text, sondern auch Schadcode z.B. in Form eines JavaScripts übergeben und in die Seite injiziert werden.

PHP-Skript:

```
1  if ($_GET["msg"]) {
2    $message = $_GET['msg'];
3  echo '<p>'. $message . '</p>';
4  }
```

Listing 4.1: PHP-XSS Beispiel

Durch das Formular in Listing 4.2 kann zum Beispiel, ein solches Skript mit Schadcode an die Seite gesendet werden:

HTML-Code:

```
<form action="index.php" method="get">
  <input type="hidden" name="pid" value="xss">
  Message: <input type="text" name="msg" value="
  <script>alert('Attacked through XSS!');</script>">
  <input type="submit" value="Ausprobieren">
</form>
```

Listing 4.2: HTML-Formular

Nachdem auf den Button mit dem Wert “Ausprobieren“ geklickt wurde, wird die Seite mit dem übergebenen Schadcode neu geladen, und ein Pop-Up mit der Meldung “Attacked through XSS!” erscheint. Natürlich kann jeder x-beliebige Schadcode übergeben werden, der nicht nur ein harmloses Pop-Up erzeugt.

Um Entwicklern bei der Planung und Implementierung von Webapplikationen die Erkenntnisse der letzten beiden Kapitel zur Verfügung zu stellen, wurde im nächsten Kapitel eine Checkliste mit Präventionsmaßnahmen erstellt und die Implementierung der Sicherheitslücken näher erläutert.

5 Ergebnisse

In diesem Kapitel werden die Ergebnisse dieser Arbeit beschrieben. Zum einen wird eine Checkliste mit einem Überblick über die untersuchten Sicherheitslücken erstellt und zum anderen der fertige Prototyp betrachtet.

5.1 Checkliste

In folgender Tabelle wurden die in Kapitel 3 gefundenen Ergebnisse zusammengefasst, um für Entwickler einen Überblick über die Sicherheit moderner Webtechnologien zu schaffen. Entwickler können so Schritt für Schritt vorhandene Implementierungen überprüfen oder zukünftige Entwicklungen besser planen.

Cross-Site Scripting	
Beschreibung/Funktionsweise	Prävention
XSS tritt auf, wenn in eine Webapplikation nicht vertrauenswürdige Daten eingeschleust werden und diese über keine Validierungs oder Escapemechanismen verfügt, um die Daten auf ausführbaren (Schad)-Code zu überprüfen.	Implementierung von Validierungs u/o Escapemechanismen, Implementierung einer Cookie-Security durch Binden des Cookies an die Session des verifizierten Benutzers.

Injection Flaws	
Beschreibung/Funktionsweise	Prävention
Durch Injection Flaws können Angreifer Schadcode über die Webapplikation an andere Systeme weiterleiten. Dabei kommt unter anderem der Zugriff auf System Calls, die Benutzung von externen Programmen sowie der Zugriff auf Datenbanksysteme zum Einsatz.	Vermeidung von externen Interpretern, Umsetzung von Shell-Befehlen und System Calls über programmiersprachenspezifische Bibliotheken, White-List für Benutzereingaben, sinnvolles Rechte-Management.
Broken Authentication and Session Management	
Beschreibung/Funktionsweise	Prävention
Betrifft alle Aspekte der Authentifizierung und des Managements aktiver Sessions. Durch Funktionen wie das Zurücksetzen oder Ändern des Passworts können sogar sichere Authentifizierungsmechanismen untergraben werden.	Verwendung von starken und langen Passwörtern, Begrenze Anzahl an Loginversuchen, Eingabe von altem Passwort bei Änderung, Speicherung von Passwörtern in gehashtem oder kodierten Format, Verwendung von SSL bei Login-Vorgang und Session-Verwendung Verwendung von POST anstatt GET bei Authentifizierung und Session, Verwendung des “no cache” Tags und “AUTOCOMPLETE=OFF”
Insecure Direct Object References	
Beschreibung/Funktionsweise	Prävention
Tritt auf, wenn durch Benutzereingabe direkter Zugriff auf ein Objekt der Webapplikation ohne Berechtigungsabfrage erlangt werden kann.	Um Insecure Direct Object References zu vermeiden, muss jedes Objekt, auf das ein Benutzer Zugriff hat, einen Schutzmechanismus implementieren. Möglichkeiten: indirekte Objekt-Referenzen per User oder Session, Zugriffskontrolle für jedes direkt referenzierte Objekt.

Security Misconfiguration	
Beschreibung/Funktionsweise	Prävention
Kann auftreten, wenn eine Komponente der Webapplikation falsch oder unsicher konfiguriert wurde. Meist durch unsichere oder schlecht dokumentierte Standard-Konfiguration oder Nebeneffekte von optionalen Konfigurationen.	Up-to-date halten der involvierten Software, für die Applikation, die Frameworks, den Applikations/Web/Datenbank-Server und für die Plattform eine sichere Konfiguration definieren und einsetzen, keine Standard-Konfigurationen für produktiven Einsatz der Webapplikation verwenden.
Sensitive Data Exposure	
Beschreibung/Funktionsweise	Prävention
Wenn sensible Daten nicht ausreichend geschützt werden, zum Beispiel durch die Speicherung eines Passworts in Klartext, können Angreifer Zugriff auf diese Daten erlangen. Meistens sind Passwörter betroffen, aber auch Kreditkartendaten, Session Token und andere Zugangsdaten.	Verschlüsselung von sensiblen Daten bei Übertragung und Speicherung, keine dauerhafte Speicherung von unnötigen sensiblen Daten, starke Verschlüsselungsalgorithmen und Keys, Passwörter mit speziellen Algorithmen verschlüsseln, Deaktivieren von Autocomplete und Caching auf Seiten, die sensible Daten enthalten.
Missing Function Level Access Control	
Beschreibung/Funktionsweise	Prävention
Wenn der unauthorisierte Zugriff auf private Funktionen möglich ist, können Angreifer zum Beispiel durch die Abänderung eines URLs, sensible Daten eines anderen Benutzers einsehen. Weiters können so Rechte erlangt werden, die der Angreifer normalerweise nicht besitzen würde.	Ein Ansatz, um diesem Problem vorzubeugen ist standardmäßig den Zugriff auf alle Funktionen zu verbieten. Über eine White-List kann dann benutzerspezifisch Zugriff auf eine bestimmte Funktion erteilt werden. Weiters helfen Access Control Listen und rollenbasierte Authentisierungsmechanismen gegen diese Sicherheitslücke

Cross-Site Request Forgery	
Beschreibung/Funktionsweise	Prävention
Bei einer CSRF-Attacke nutzt der Angreifer die Art und Weise der Authentifizierung einer Webapplikation. Dazu wird ein Link oder Skript auf einer Website platziert, das sich mit der Ziel-Webapplikation verbindet. Ist das Opfer dort bereits angemeldet oder ein Cookie noch nicht abgelaufen, werden die Aktionen des Angreifers ausgeführt.	Einzigartigen Token an jeden Request anhängen, Token aus der Session-ID des Benutzers generieren und im besten Fall pro Request neu berechnen, IP-Adresse des Benutzers mit der IP aus dem Referer oder Origin-Header abgleichen.
Using Components with Known Vulnerabilities	
Beschreibung/Funktionsweise	Prävention
Sicherheitslücken in Bibliotheken und Software von Drittanbietern sind weit verbreitet und können die Sicherheit der Applikation gefährden. In den letzten Jahren wurden ca. 4500 solcher CVE's pro Jahr veröffentlicht.	Identifizierung aller Komponenten und Versionen, die verwendet werden, inkl. Abhängigkeiten, Kontrolle dieser Komponenten in Exploit-Datenbanken, Erstellen von Sicherheitsrichtlinien für die Verwendung von Komponenten, Verwendung von 'Security-Wrappern' um nicht benötigte Funktionalität zu deaktivieren.
Unvalidated Redirects and Forwards	
Beschreibung/Funktionsweise	Prävention
Wenn eine Webapplikation einen Parameter annimmt, ohne diesen zu validieren, und den Benutzer auf den Wert dieses Parameters weiterleitet. Angreifer können zum Beispiel Phishing-Mails mit einem Redirect-URL versenden. Der Wert des Parameters zeigt dann auf eine Webseite, die Schadcode oder Malware enthält. Einem Laien fällt dies nicht auf, da der Name der eigentlichen Website am Anfang der URL aufscheint.	Die einfachste Möglichkeit besteht darin, keine Redirects und Forwards zu verwenden. Wenn dies doch der Fall ist, sollte kein Parameter zur Berechnung der Zieladresse verwendet werden. Eine weitere Möglichkeit besteht darin, statt dem eigentlichen URL einen Mapping-Wert zu verwenden, der serverseitig in den richtigen URL übersetzt wird.

Tabelle 5.1: Checkliste für Entwickler

5.2 Prototyp

Da der Prototyp auf einem voll funktionsfähigen Webserver läuft und somit über das Internet erreichbar ist, wurde eine Authentifizierungsseite implementiert:

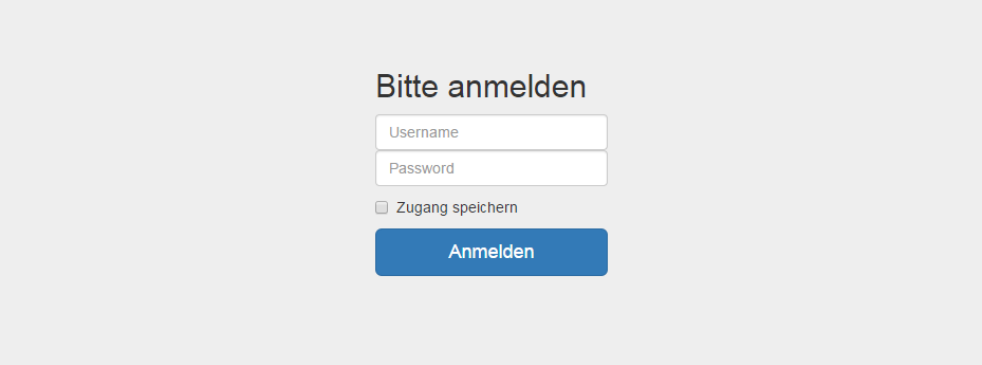

Abbildung 5.1: Login Seite des Prototypen - Quelle: Screenshot

Nach der Anmeldung mit den Zugangsdaten gelangt man auf die erste Seite der Applikation. Wird dabei die Checkbox “Zugang speichern” angehakt, werden die Zugangsdaten in einem Cookie gespeichert, welches 14 Tage lang gültig ist. Ansonsten wird, wie in in Listing 5.1 ersichtlich ist, ein Cookie angelegt, das beim Schließen des Browsers (Session) wieder gelöscht wird.

PHP-Skript:

```
function persistLogin($username,$password,$time){
  if($time!=0){
  setcookie($username,$password,time()+$time);
  }else{
  setcookie("sessionCookie",$password);
}}
if(isset($_SESSION['username'],$_SESSION['password'])){
 if($_SESSION['username']==$user_name&&$_SESSION['password']==$user_pw){
  if(isset($_POST['checkbox'])){
   persistLogin($_SESSION['username'],$_SESSION['password'],1209600);
  }else{
```

```
12          persistLogin($_SESSION['username'],$_SESSION['password'],0);
13        }
14      }
15    }
```

Listing 5.1: Cookies in PHP erstellen

Um das Passwort aus diesem Cookie später im Beispiel der Sicherheitslücke “Sensitive Date Exposure” auslesen zu können, wird es in Klartext gespeichert. Nach dem Login-Vorgang kommt man auf die erste Seite der Applikation, in der die Sicherheitslücke “Cross-Site Scripting” beschrieben wird:

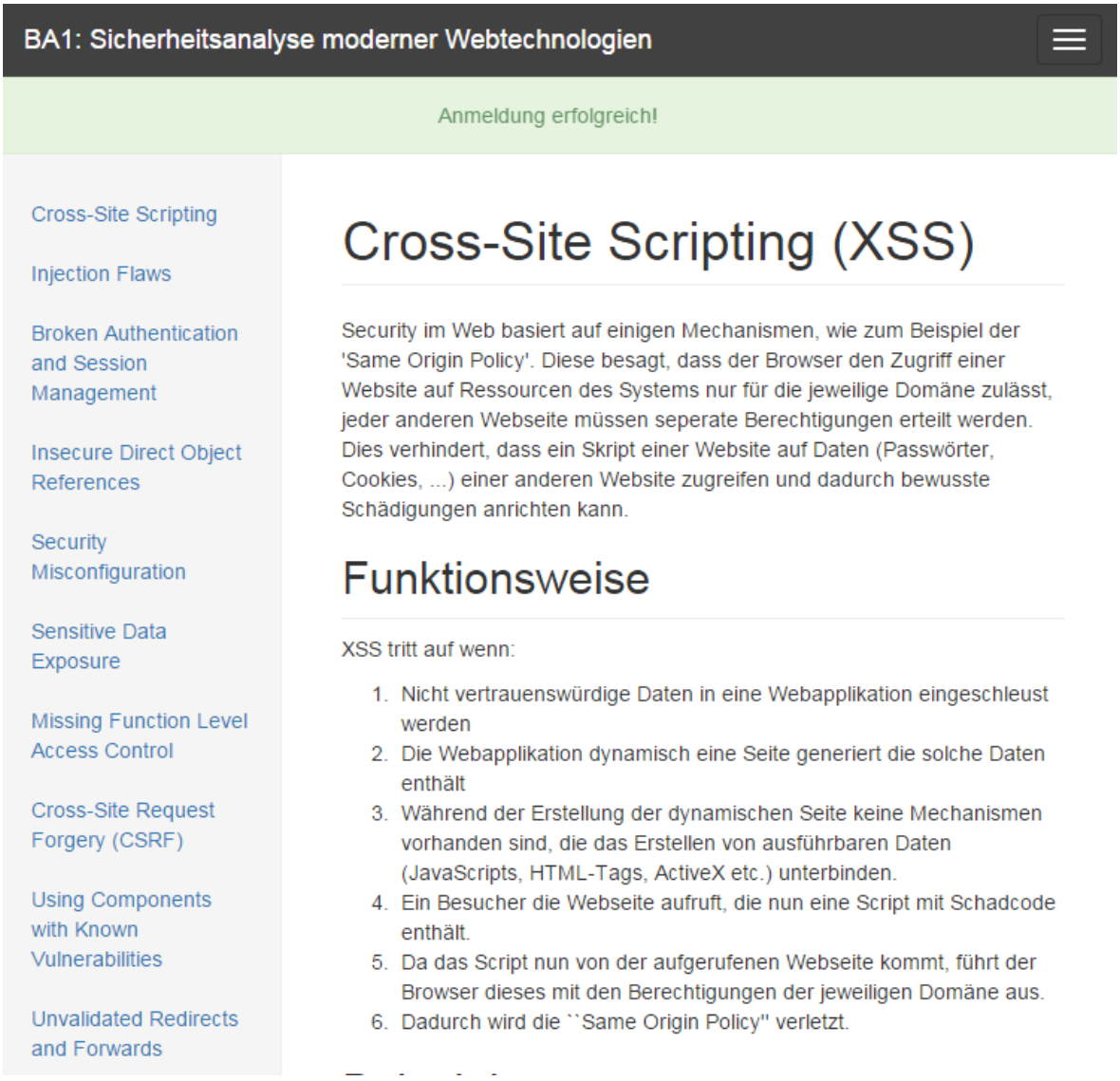

Abbildung 5.2: Startseite des Prototypen - Quelle: Screenshot

Da die Beschreibungen, Funktionsweisen und Präventionsmaßnahmen größtenteils aus Kapitel 3 übernommen wurden, liegt der Fokus auf den nächsten Seiten auf den Beispielen, die ausprogrammiert wurden.

Cross-Site Scripting

Im Falle von XSS wurde eine Sicherheitslücke in Form eines PHP-Skripts (Listing 5.2) implementiert, welches aus einem GET-Parameter übertragene Werte einliest und als Absatz in die Seite einfügt.

PHP-Skript:

```
1  // XSS VULNERABLE:
2  // if get-parameter "msg" is set, get the value and display it
3  if ($_GET["msg"]) {
4    $message = $_GET['msg'];
5    echo '<p>'. $message . '</p>';
6  }
```

Listing 5.2: PHP-Skript mit XSS

Wenn dieser Parameter zum Beispiel den Wert "<script>alert('Attacked through XSS!');</script>" trägt, wird ein Pop-Up mit der entsprechenden Meldung angezeigt. Natürlich kann so jeder beliebige Schadcode in die Applikation gelangen.

Injection Flaws

In diesem Beispiel (Listing 5.3) wird ein SQL-Statement direkt aus den GET-Parametern ausgelesen, ohne den Inhalt zu validieren und an die Datenbank geschickt.

injection.php

```
1  // If get-parameters are set, invoke operation on database
2  if($_GET["user"]&&$_GET["pw"]){
3    $user=$_GET["user"];
4    $pw=$_GET["pw"];
5    $sql="SELECT NAME FROM USER WHERE NAME = '". $user ."'
6      AND PASSWORD = '". $pw ."'";
7    echo $sql;
8    $ret = $db->query($sql);
9    while($row = $ret->fetchArray(SQLITE3_ASSOC) ){
10     echo "User = ". $row['NAME'] ."<br>";
11   }
```

```
12      echo "<br>Operation done successfully";
13      $db->close();
14  }
```

Listing 5.3: SQL-Injection

Von der Applikation wird über Ajax eine Anfrage mit den beiden GET-Parametern an das Skript geschickt und das Ergebnis dynamisch in die Seite geladen.

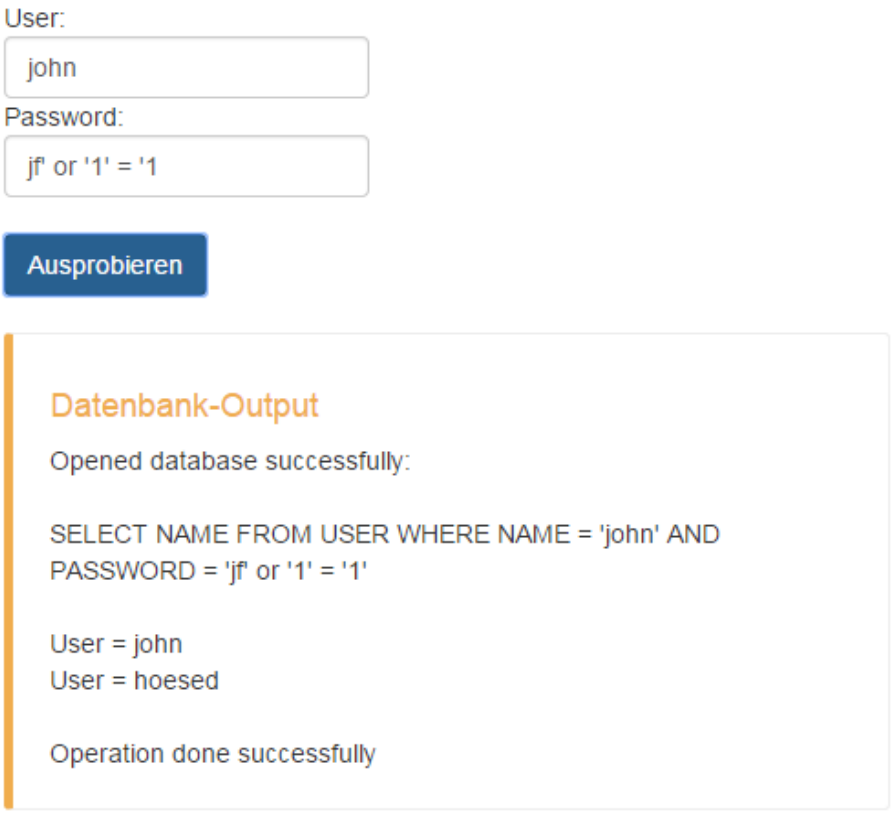

Abbildung 5.3: SQL Injection - Quelle: Screenshot

Abbildung 5.3 zeigt, dass durch die SQL-Syntax im Passwort-Feld nicht nur der Benutzer mit dem Passwort “jf”, sondern auch alle anderen Benutzer aus der Datenbank zurückgeliefert werden.

Broken Authentication and Session Management

Um zu zeigen, dass Session-IDs niemals mit URLs übertragen werden sollten, wurde in diesem Beispiel genau diese Methodik durchgeführt.

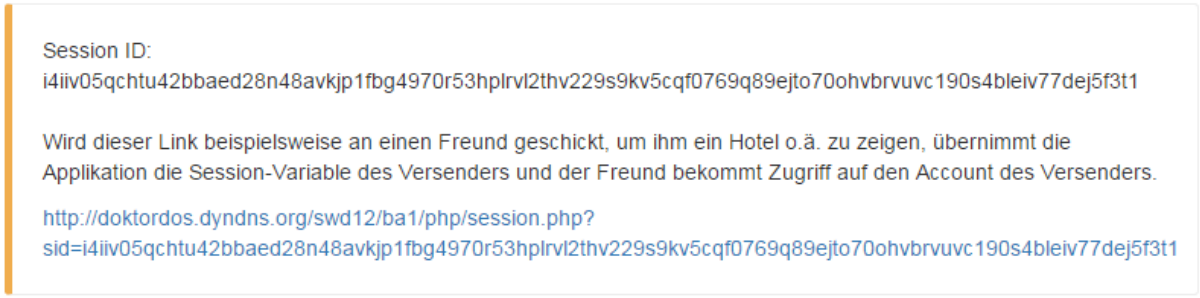

Abbildung 5.4: Session ID in URL - Quelle: Screenshot

In Abbildung 5.4 wird beschrieben, was durch diese äußerst unsichere Methode ausgelöst werden könnte.

Insecure Direct Object References

Dieses Beispiel ist eine Mischung aus XSS und unsicherer direkter Objektreferenzen. Die ID (1) des angemeldeten Benutzers ist in einem versteckten Input-Feld gespeichert. Gibt ein Angreifer folgenden Schadcode in das Message-Feld ein und schickt dieses ab, liefert die Datenbank den Kontostand eines anderen Benutzers

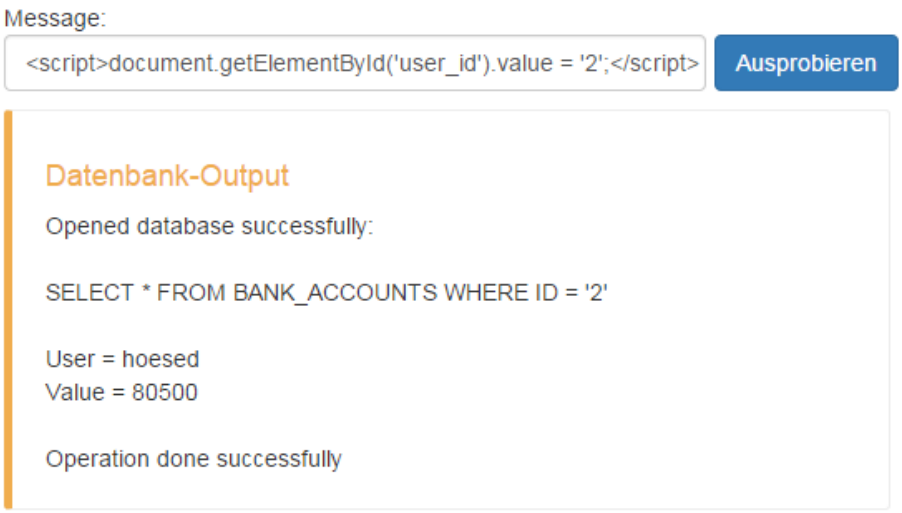

Abbildung 5.5: Datenbank-Output - Quelle: Screenshot

Abbildung 5.5 zeigt, dass der Kontostand eines anderen Users zurückgeliefert wurde.

Security Misconfiguration

Da hier ein interaktives Beispiel nicht möglich ist, wurde ein Beispiel aus der PHP-Konfigurationsumgebung gewählt.

In der php.ini-Datei gibt es eine Variable, die standardmäßig aktiviert ist:

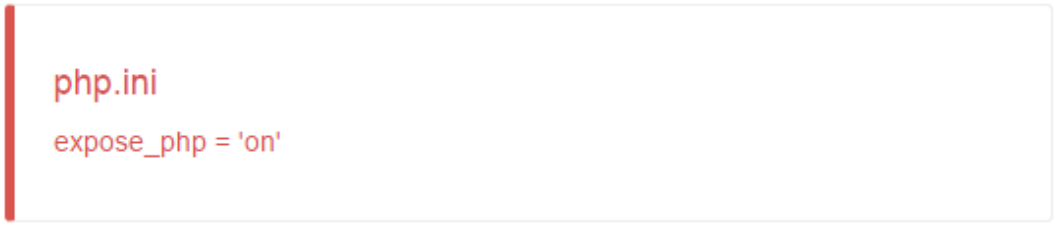

Abbildung 5.6: PHP Standard-Konfiguration - Quelle: Screenshot

Durch diese Einstellung wird im Header die spezifische PHP-Version mitgeschickt. Ein Angreifer könnte anhand dieser Versions-Nummer in einer Exploit-Datenbank nach versionsspezifischen Sicherheitslücken suchen und somit den Server attackieren.

Sensitive Data Exposure

Da die Zugangsdaten beim Login unkodiert in einem Cookie gespeichert wurden, kann das Passwort mit Hilfe von JavaScript ausgelesen werden, wie Listing 5.4 zeigt:

JavaScript

```
function getCookie(cname){
  var name=cname+"=";
  var ca=document.cookie.split(';');
  for(var i=0;i<ca.length;i++){
  var c=ca[i];
  while(c.charAt(0)==' ') c=c.substring(1);
  if(c.indexOf(name)==0) return c.substring(name.length,c.length);}
  return null;
}
function showCookieValue(){
 var cookieValue=getCookie("sessionCookie");
  if(cookieValue==null) {cookieValue="No Cookie!";}
  document.getElementById("output").textContent="Password: "+cookieValue
```

Listing 5.4: Cookie per JS auslesen

In Abbildung 5.7 sieht man, dass das Passwort in Klartext gespeichert war.

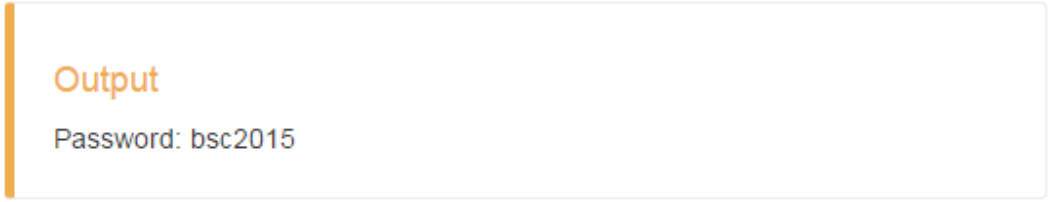

Abbildung 5.7: Cookie ausgelesen - Quelle: Screenshot

Missing Function Level Access Control

In Listing 5.5 wurde die Datenbank mit Hilfe von “prepared statements” gegen SQL-Injection geschützt. Die Abfrage eines anderen Accounts ist jedoch möglich, da keine zusätzliche Zugangskontrolle implementiert wurde.

missing.php

```
1  if($_GET["id"]){
2    $id=$_GET["id"];
3
4    $stmt=$db->prepare('SELECT * FROM BANK_ACCOUNTS WHERE ID =:id');
5    $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
6
7    $ret=$stmt->execute();
8    while($row=$ret->fetchArray(SQLITE3_ASSOC) ){
9    echo "User=". $row['NAME'] ."<br>";
10   echo "Value=". $row['VALUE'] ."<br>";
11   }
12 echo "<br>Operation done successfully";
13 $db->close();
14 }
```

Listing 5.5: Prepared Statements

Abbildung 5.8 zeigt, dass die Datenbank den Zugriff auf jedes Konto zulässt.

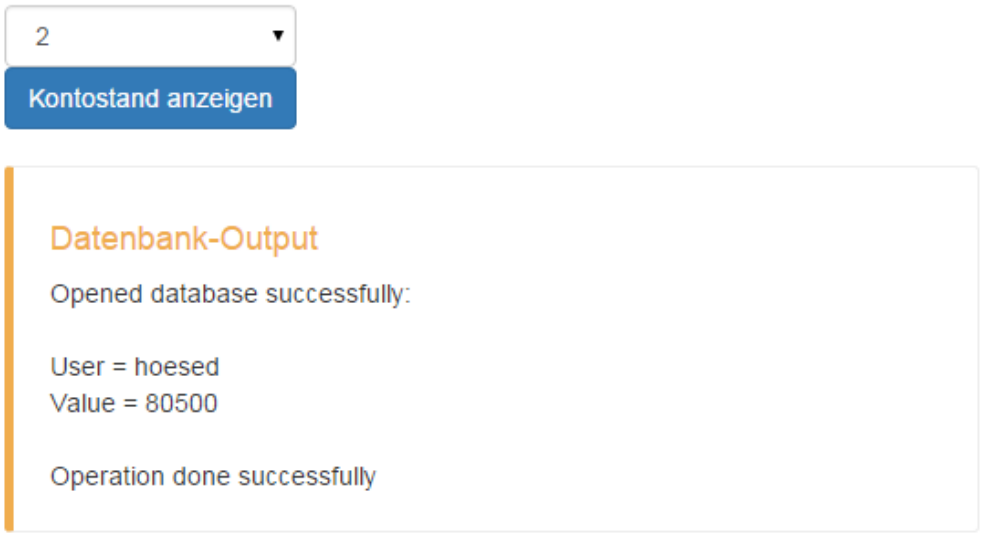

Abbildung 5.8: Datenbank-Output - Quelle: Screenshot

Cross-Site Request Forgery

In Listing 5.6 wurde in einem Bild Schadcode platziert. Der Browser versucht beim Aufruf der Seite das vermeintliche Bild unter dieser URL zu laden und da die Applikation keinen Token im Url benutzt, um den Aufruf an die Session eines verifizierten Benutzers zu binden, wird der Wert in Abbildung 5.9 bei jedem Aufruf der Seite erhöht.

Schadcode

```
1 <img src="http://doktordos.dyndns.org/swd12/ba1/vote.php?vote=true"/>
```

Listing 5.6: Schadcode

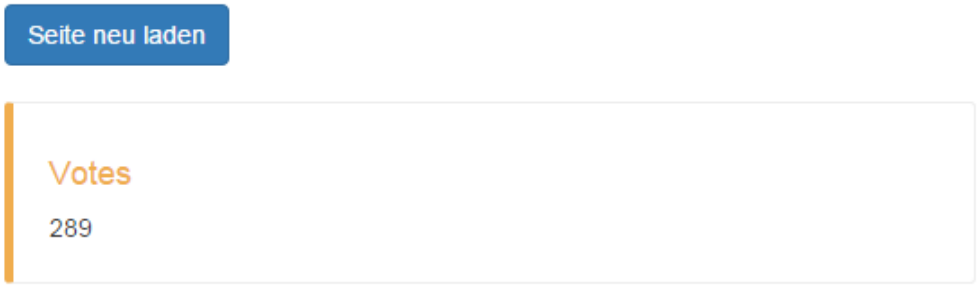

Abbildung 5.9: Anzahl der Votes - Quelle: Screenshot

Using Components with Known Vulnerabilities

Da es sich hier wieder um ein Problem handelt, welches programmatisch nicht darstellbar ist, wurde das Beispiel aus der Analyse übernommen:

“Als Beispiel ist die “Remote-code execution with Expression Language Injection” - Sicherheitslücke zu nennen, die kürzlich als Teil des Spring-Frameworks in Java veröffentlicht wurde. Die Lücke wurde schnell geschlossen und eine neue Version des Frameworks veröffentlicht. Trotzdem enthalten ca. 29.8 Millionen Downloads diese Sicherheitslücke.”

Unvalidated Redirects and Forwards

In Listing 5.7 bezieht das PHP-Skript den URL aus dem GET-Parameter und leitet den User direkt an die übergebene Adresse weiter:

redirect.php

```php
if (isset($_GET["url"])) {
  $url = $_GET['url'];
  header("Location: " . $url);
}
```

Listing 5.7: Direkte Weiterleitung

Ein Angreifer kann nun einen URL zusammenstellen, der auf eine Seite mit Schadcode weiterleitet:

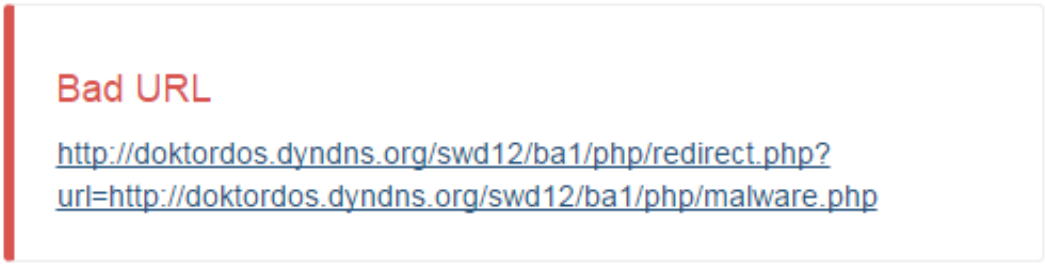

Abbildung 5.10: Bad URL - Quelle: Screenshot

Einem Laien fällt dies eventuell nicht auf, da der Name der eigentlichen Website am Anfang der URL in Abbildung 5.10 aufscheint.

6 Conclusio

In dieser Arbeit wurden die wesentlichen Sicherheitslücken moderner Webtechnologien herausgearbeitet. Zuerst wurde ein Überblick über den aktuellen Funktionsumfang dieser Technologien erstellt, um dann im zweiten Kapitel eine gründliche Analyse der, 2013 von der OWASP Foundation veröffentlichten, Top 10 Sicherheitslücken von Webapplikationen durchführen zu können.

Die Entwicklung des Prototyps hat gezeigt, wie einfach Risikofaktoren übersehen werden können und das ein gründliches Sicherheitskonzept quer durch alle Aspekte einer Webapplikation hindurchgezogen werden sollte. Dies fängt beim Betriebssystem an, zieht sich über die Wahl eines geeigneten Webservers und dessen Konfiguration, bis hin zu externen Programmen wie zum Beispiel durch Datenbanken.

Am Schluss wurde eine Checkliste erstellt, die einen Überblick über die bekanntesten Sicherheitslücken geben und Entwicklern als Grundlage für die Entwicklung von sichereren Webapplikationen dienen soll. Hundertprozentige Sicherheit gibt es bekanntlich nicht, ein Mindestmaß sollte aber auf jeden Fall umgesetzt werden, da Angriffe auf sensible Daten in Zukunft sicher nicht weniger werden.

Literaturverzeichnis

[1] Bernal Carlos: *Top 10 Web Security Risks: Using Components with Known Vulnerabilities*. https://blog.credera.com/technology-insights/java/top-10-web-security-risks-using-components-known-vulnerabilities-9/, 21.01.2014, eingesehen am 06.01.2015.

[2] Bright Peter: *HTML5 specification finalized, squabbling over specs continues*. http://arstechnica.com/information-technology/2014/10/html5-specification-finalized-squabbling-over-who-writes-the-specs-continues, 29.10.2014, eingesehen am 03.01.2015.

[3] Black Stratus: *Comprehensive Web Application Protection*. http://www.blackstratus.com/enterprise/log-management/insecure-direct-object-reference, 2013, eingesehen am 05.01.2015.

[4] Cenzic, Inc.: *Cenzic Vulnerability Report 2014*. http://www.cenzic.com/downloads/Cenzic_Vulnerability_Report_2014.pdf, 2014, eingesehen am 23.01.2015.

[5] Chen Ping, Nikiforakis Nick, DesmetLieven, Huygens Christophe: *Security Analysis of the Chinese Web: How well is it protected?*. SafeConfig '14 Proceedings of the 2014 Workshop on Cyber Security Analytics, DOI: 10.1145/2665936.2665938, 2014.

[6] Chuang Tyng-Ruey, Huang Chia-Hsin, Lee Hahn-Ming: *Prefiltering techniques for efficient XML document processing*. ACM Proceedings of the 2005 ACM symposium on Document engineering, DOI: 10.1145/1096601.1096641, 2005.

[7] DuPaul Neil: *Cross-Site Request Forgery Guide: Learn All About CSRF Attacks and CSRF Protection*. http://www.veracode.com/security/csrf, o.J., eingesehen am 06.01.2015.

[8] Hamit Josh: *Top Ten Web Security Risks: Missing Function Level Access Control*. http://blog.credera.com/technology-insights/open-source-technology-insights/top-ten-web-security-risks-missing-function-level-access-control-7/, 20.03.2014, eingesehen am 06.01.2015.

[9] MacDonald Matthew: *HTML5: The Missing Manual, 2nd Edition*. O'Reilly Media Inc., Sebastopol, 2013.

[10] Markham Gervase, Stamm Sid, Sterne Brandon: *Reining in the Web with Content Security Policy*. WWW '10 Proceedings of the 19th international conference on WWW, DOI: 10.1145/1772690.1772784, 2010.

[11] Mathur Ankit: *HTML5 still relevant for app developers: Kendo UI*. http://www.devworx.in/news/open-web/html5-still-relevant-for-app-developers-kendo-ui-136481.html, 08.11.2012, eingesehen am 23.01.2015.

[12] Mitre Corporation: *Common Weakness Enumeration List*. http://cwe.mitre.org, 2014, eingesehen am 05.01.2015.

[13] OWASP Foundation: *OWASP Top 10 - The Ten Most Critical Web Application Security Risks*. OWASP Foundation, o.O., 2013.

[14] OWASP Foundation: *XSS Prevention Cheat Sheet*. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet, 12.04.2014, eingesehen am 05.01.2015.

[15] OWASP Foundation: *Injection Flaws*. https://www.owasp.org/index.php/Injection_Flaws, 29.06.2010, eingesehen am 05.01.2015.

[16] OWASP Foundation: *Broken Authentication and Session Management*. https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management, 22.04.2010, eingesehen am 05.01.2015.

[17] OWASP Foundation: *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet, 14.08.2014, eingesehen am 06.01.2015.

[18] Rizzo Tony: *HTML5 Ascendent*. http://www.html5report.com/topics/html5/articles/328645-html5-ascendent.htm, 28.02.2013, eingesehen am 23.01.2015.

[19] Robbins Niederst Jennifer: *HTML5 Pocket Reference*. O'Reilly Media, Inc., Sebastopol, 2013.

[20] Sklar David, Trachtenberg Adam: *PHP Cookbook, Third Edition*. O'Reilly Media, Inc., Sebastopol, 2014.

[21] w3schools.com: *The world's largest web development site*. http://www.w3schools.com, eingesehen am 03.01.2015.

[22] w3schools.com: *HTML <script> defer Attribute*. http://www.w3schools.com/tags/att_script_defer.asp, eingesehen am 23.01.2015.

[23] Williams Jeff: *Application Security: We Still Have A Long Way To Go*. http://www.darkreading.com/application-security/application-security-we-still-have-a-long-way-to-go/d/d-id/1005798, 21.11.2013, eingesehen am 06.01.2015.

[24] Yue Chuan, Wang Haining: *A Measurement Study of Insecure JavaScript Practices on the Web*. ACM Transactions on the Web, Vol.7, No.2, Article 7, DOI: 10.1145/2460383.2460386, 2013.

[25] Zakas C. Nicholas: *Professional JavaScript for Web Developers, Third Edition*. John Wiley and Sons, Inc., Indianapolis, 2012.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese hier vorgelegte Arbeit mit dem Titel „Sicherheitsanalyse moderner Webtechnologien“ selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen oder anderen Quellen, insbesondere dem Internet, entnommen sind, sind als solche eindeutig und wieder auffindbar kenntlich gemacht. Alle diese Quellen sind in einem Literaturverzeichnis angegeben. Die vorliegende Arbeit ist in gleicher oder ähnlicher Form noch nicht veröffentlicht.