Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Zabihullah Safai

Sicherheit im E-Business

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Zabihullah Safai

Sicherheit im E-Business

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Martin Hübner
Zweitgutachter : Prof. Dr. Kai von Luck

Abgegeben am 27.08.2012

Zabihullah Safai

Thema der Bachelorarbeit

Sicherheit im E-Business

Stichworte

Apache OfBiz, Penetrationtests, E-Business, E-Commerce, Source Code Analyse, Statische Tests, Nessus Framework

Kurzzusammenfassung

In dieser Arbeit wird die Sicherheit von E-Business-Systemen untersucht. Gegenstand der Untersuchung ist das E-Business-Framework Apache OfBiz, als repräsentatives E-Commerce-System. Die Untersuchung findet in Form von Penetrationstests statt. Dabei werden statische Analysen in Form von Sourcecode-Analysen durchgeführt. Außerdem werden Exploit-Versuche mittels Nessus Framework durchgeführt. Als Ergebnis wird versucht, eine Antwort auf die Frage zu geben, ob es sichere E-Business-Systeme gibt, die man „out ofthe box“ produktiv einsetzen kann.

Zabihullah Safai

Title of the paper

Security in E-Business

Keywords

Apache OfBiz, Penetrationtests, E-Business, E-Commerce, Source Code Analysis, Static Tests, Nessus Framework

Abstract

This thesis investigates the security issue of e-business systems. The test subject will be the apache OFBiz, which is representative for e-commerce systems. For this purpose there will be performed penetration tests and source code analysis. Also exploit attempts using Nessus Framework are performed. As a result, attempts to provide an answer to the question of whether there are secure e-business systems, which can be used "out ofthe box" productive

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Problemstellung

Die weltweite Nutzung des Internets ermöglicht, dass Konsumenten für ihre Einkäufe immer häufiger auf den elektronischen Handel (E-Commerce[1]) zurückgreifen. Deshalb wird in den E-Commerce-Systemen eine Vielzahl von Kundendaten, z.B. Namen, Rechnungs- und Email-Adressen, Benutzernamen und Passwörter, Kreditkartendaten etc. gespeichert. Aus der Menge dieser Daten kann man durch Anwendung geeigneter Analyseinstrumente Rückschlüsse auf die Interessen und Neigungen und das Kaufverhalten der Kunden ziehen.

Aus den genannten Gründen sind die Daten für Cyber-Kriminelle sehr wertvoll. Angreifer nutzen solche Daten, um finanzielle Vorteile zu erlangen, SPAM[2] zu verteilen oder mit Social Engineering[3] vorbereitete gezielte Attacken zu verursachen (vgl. [BSI03] S. 5).
Datenraub wird auch zu anderen illegalen Zwecken genutzt, wie beispielsweise Industriespionage, Wettbewerbsvorteile bei Angeboten und Erpressungen.

Ein häufiges Phänomen im Online-Handel bzw. E-Commerce ist die missbräuchliche Nutzung von Kreditkartendaten. Im Internet kursieren z. B. Downloads, die für alle gängigen Kreditkartentypen Nummern generieren, die zwar in der Regel keinem Kartenkonto zugeordnet werden können, die aber von E-Commerce-Anbietern alle offline vorgenommenen Plausibilitätsüberprüfungen ohne Beanstandung passieren. Bei einer anderen Variante werden rechtmäßige Karteninhaber, z.B. mittels Ausspähung durch Trojaner, ihrer Identität beraubt und später mit betrügerischen Umsätzen konfrontiert.

---

[1] engl. *Electronic Commerce*
[2] engl. *Unerwünschte Werbebotschaften*
[3] engl. *Zwischenmenschliche Beeinflussung*

Immer mehr sind auch Angriffstechniken auf die Infrastruktur von E-Business[4]-Applikationen zu beobachten. Dabei werden bestimmte Dienste durch gezielte DoS[5] Attacken außer Betrieb gesetzt. Außerdem können in Folge von Datendiebstahl, Datenverlust und bösartigen Programmen interne Unternehmensinformationen zugänglich gemacht werden bzw. verloren gehen, wodurch den Unternehmen erheblicher finanzieller Schaden entstehen kann.

[4] engl. *Electronic Business*
[5] engl. *Denial of Service*, Nichtverfügbarkeit eines Dienstes

1.2 Zielsetzung

Dem E-Commerce-Anbieter, der diese Daten erhebt, verarbeitet und speichert, kommt eine hohe Verantwortung zu, den ausreichenden Schutz dieser Daten zu gewährleisten. Um den im vorherigen Absatz genannten Schwachstellen entgegenzuwirken, müssen E-Business-Systeme als Untermenge von Webanwendungen in regelmäßigen Abständen Sicherheitstests unterzogen werden. Das heißt, man muss sich der Methoden der proaktiven Sicherheit bedienen.

Proaktive Sicherheit bedeutet eine Verbesserung des Schutzes einer IT-Infrastruktur durch Anwendung vorbeugender Techniken. Penetrationstests, als ein wesentlicher Teil der proaktiven Sicherheit, können ein wichtiges Feedback zum Sicherheitsstand der E-Business-Systeme liefern, Problembereiche offen legen und Verbesserungspotenziale identifizieren.

In dieser Arbeit wird ein Sicherheitskonzept zum Durchführen von Penetrationstests für ein E-Business-System ausgearbeitet. Das Konzept wird an den Vorgaben von BSI[6] in [BSI03] vorgestellten Maßnahmen angelehnt. Die Durchführung wird in drei Hauptbereiche unterteilt: Informationssammlung, statische Analyse von Source Code und praktische Exploit-Versuche. Die Tests werden als Withebox-Test[7] durchgeführt. Die Gründe dafür werden in Kapitel 3.3 ausführlich erklärt.

Für die Durchführung der Penetrationstests wird das Nessus[8] Frameworks[9] auf einer repräsentativen E-Business-Plattform angewendet. Als repräsentatives Testsystem wird das Apache OfBiz[10] Framework ausgewählt. Als Ergebnis der Arbeit werden die Penetrationstests analysiert und ausgewertet. Ferner wird die Frage behandelt, ob es fertigkonfigurierte E-Business-Systeme gibt, die „out of the box[11]“ sicher sind.

1.3 Abgrenzung

[6] *Bundesamt für Sicherheit in der Informationstechnik*
[7] *Die innere Funktionsweise des Testgegenstandes ist bekannt*
[8] *Nessus 5.0* ist ein Produkt von Tenable Network Security Inc. für Durchführung von Penetrationstests
[9] engl. *Programmiergerüst* für Entwicklungsansätze
[10] Open for Business
[11] engl. *im initialen Zustand*

Die vorliegende Studie behandelt die Sicherheit im E-Business-System auf Anwendungsebene. Die Betriebssicherheit und Netzwerksicherheit wird im Rahmen dieser Arbeit nicht behandelt, sondern vorausgesetzt. E-Commerce-Systeme sind ein wichtiger Teil eines E-Business-Systems. Die Penetrationstests werden auf Anwendungsebene durchgeführt, daher wird der Untersuchungsgegenstand ein E-Commerce-System sein. Deshalb wird im weiteren Verlauf dieser Arbeit von E-Commerce-Anwendungen die Rede sein. Wann immer die Unterscheidung zwischen E-Commerce und E-Business notwendig ist, wird auf das konkrete System hingewiesen.

1.4 Aufteilung und Organisation der Arbeit

Im zweiten Kapitel wird der grundsätzliche Aufbau von E-Business-Anwendungen erklärt. Außerdem wird der Unterschied zwischen E-Business und E-Commerce näher erläutert. Es wird kurz auf die repräsentative Architektur von E-Commerce-Anwendungen eingegangen. In Kapitel drei liegt das Hauptaugenmerk auf Sicherheit von E-Business-Systemen. In Kapitel 3.1 werden die repräsentativen Bedrohungen behandelt, mit der ein E-Commerce-System konfrontiert sein kann. Es folgt eine Auflistung der Bedrohungen. Einige wichtige Sicherheitslücken werden in diesem Kapitel ausführlich beschrieben. In Kapitel 3.3 wird ein Sicherheitskonzept herausgearbeitet, das auf die Besonderheiten des Testgegenstands Apache OfBiz zugeschnitten ist. Das Konzept stellt den Leitfaden für die folgenden drei Kapitel dar.

Das vierte Kapitel beschäftigt sich mit Methoden der Informationsrecherche über den Testgegenstand. Dazu werden in Kapitel 4.2 der Umfang der Untersuchung und deren Ziele definiert. Kapitel 4.3 beschreibt die Informationsbeschaffung. Außerdem wird auf den Umgang mit Benutzerdaten eingegangen. Im darauffolgenden Kapitel geht es um eine statische Analyse des Source-Codes.

In Kapitel fünf werden die konkreten Maßnahmen für die Source-Code-Analyse beschrieben. Kapitel 5.3 beschäftigt sich mit dem Aufdecken offener Schwachstellen des Testgegenstands. Es folgen Bewertung und Priorisierung der gefunden Schwachstellen. Darüber hinaus wird in Kapitel 6 der praktische Penetrationstest durchgeführt und dokumentiert. Im letzten Kapitel werden die bekannten Ergebnisse analysiert und eine Aussage darüber formuliert, ob Open Source[12] Frameworks existieren, die von Grund auf

[12] engl. *Quelloffene Softwareprojekte*

sicher sind, und ob man diese Frameworks für den kommerziellen Einsatz heranziehen kann.

2 Architektur von E-Business-Systemen

Um zu verstehen, welche Sicherheitslücken in E-Business-Anwendungen zu erwarten sind, muss man sich zunächst mit der allgemeinen Architektur von E-Business-Systemen beschäftigen. Zu Anfang des Kapitels werden Begrifflichkeiten beleuchtet. In Kapitel 2.2 folgt dann eine Ausführung über den architektonischen Aufbau von E-Business-Systemen.

2.1 Abgrenzung: E-Business und E-Commerce

Der Begriff E-Commerce, oder „Electronic Commerce“ ist ein vom Marketing geprägter Sammelbegriff, welcher die elektronische Integration von Geschäftsprozessen zwischen Geschäftspartnern bezeichnet. Er bezeichnet also nicht nur den Handel, wie der Name suggeriert, sondern mehr als das.

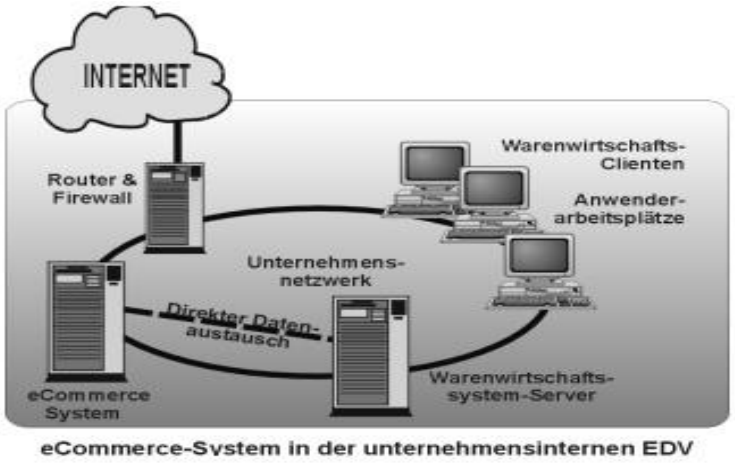

Abbildung 1: E-Commerce-Systeme, Quelle BSI

Zieschhang definiert den Begriff in ([BSI01], S. 212) so:

> *„Unter E-Commerce Applikationen verstehen wir IT-Anwendungen (Software und Hardware) zur Kommunikation und Durchführung von Transaktionen zwischen mehreren Parteien, die potenzielle Geschäftspartner darstellen. Beispiele hierfür sind Online-Shops, Buchungs- / Abrechnungs- / und Informationssysteme für den elektronischen Zahlungsverkehr, Systeme für die Auswahl und Bestellung von Waren und Informationen im Internet, für die Teilnahme an Online-Auktionen,…“*

Unter dem Begriff E-Business oder „Electronic Business“, der nach Bartelt und Lamersdorf als umfassender Oberbegriff für alle elektronisch abgewickelten Geschäftstätigkeiten gilt, (vgl. [BL2000] S. 2), versteht man die Neuausrichtung der Kernprozesse eines Unternehmens, um es an die Anforderungen der elektronischen Medien anzupassen.

Während E-Commerce sich auf handelsbasierte Geschäftsprozesse konzentriert, bezieht E-Business sämtliche, auch verteilte Geschäftsprozesse innerhalb und außerhalb von Organisationen ein, deren Ablauf auf dem Einsatzelektronischer Medien basiert. Das E-Business wird nach (vgl. [BL2000] S. 2)wiederum in die Teilbereiche:

- Electronic Information (EI)

- Electronic Cooperation (ECoop)
- Electronic Commerce

unterteilt, wobei der Electronic Commerce weiterhin die stärkste treibende Kraft im Electronic Business ist (vgl. [BL2000] S. 2).

In der vorliegenden Arbeit wird das Thema Sicherheit im E-Business behandelt. Die meisten Sicherheitsthemen gelten abstrakt auf allen Ebenen des E-Business. Für die Durchführung der Tests eignet sich jedoch Electronic Commerce (E-Commerce) am besten. Die Gründe hierfür liegen in der hohen Verbreitung der Technologie, der relativ hohen Verfügbarkeit von Frameworks und Software für E-Commerce-Anwendungen und dem hohen Bekanntheitsgrad von E-Commerce.

Aus den genannten Gründen wird als Untersuchungsgegenstand ein E-Commerce-Framework ausgewählt und behandelt.

2.2 Architektur von E-Business-Anwendungen

Der Begriff Architektur[13] in der Informationstechnik wird sehr weit gefasst und kann je nach Kontext unterschiedliche Dinge beschreiben. Kahlbrand definiert den Begriff in ([Kah98], S. 70) so:

> *„Unter der Architektur eines Systems versteht man seine logische und physische Struktur, wie sie durch die strategischen und taktischen Entscheidungen in Analyse, Design und Implementierung geformt wird.“*

Die Architektur eines E-Business-Systems ist je nach Anforderung sehr unterschiedlich, daher beschränkt sich diese Arbeit auf die Beschreibung des architektonischen Aufbaus von E-Commerce-Systemen.

2.2.1 Aufbau von E-Commerce-Systemen

[13] *Beschreibt den grundlegenden Aufbau eines Sofwaresystems (Komponentensicht)*

Die Basistopologie von E-Commerce-Systemen ist nach wie vor die Client-Server-Architektur. Die E-Commerce-Anwendung läuft auf dem Server und überträgt die Daten per HTTP an den Client-Rechner. Dabei wird die Server-Schicht im Wesentlichen aufgebrochen, um eine bessere Trennung von Datenhaltung, Anwendungslogik, Geschäftslogik und Präsentation zu erreichen. Der Client-Rechner mit seinem Webbrowserzeigt die empfangenen Inhalte.
Die E-Commerce-Anwendung besteht in der Regel aus drei Schichten:

- der Präsentationschicht,
- der Businesslogik-Schicht und
- der Datenschicht.

Diese Schichten sind nicht physisch getrennt, sondern sie sind als logische Ebene zu verstehen. Das bedeutet, es können mehrere der oben genannten Schichten auf einem physikalischen Server laufen. Umgekehrt kommt es ebenfalls vor, dass eine logische Schicht auf mehrere physikalische Server verteilt wird. Mit zusätzlichen Servern
werden dabei umfangreiche Funktionen umgesetzt, die durch die Komplexität der Anwendung bestimmt sind. Als Beispiel ist zu nennen die Verteilung von Applikationsservern auf mehrere Rechner, um eine bessere Lastverteilung zu erzielen.

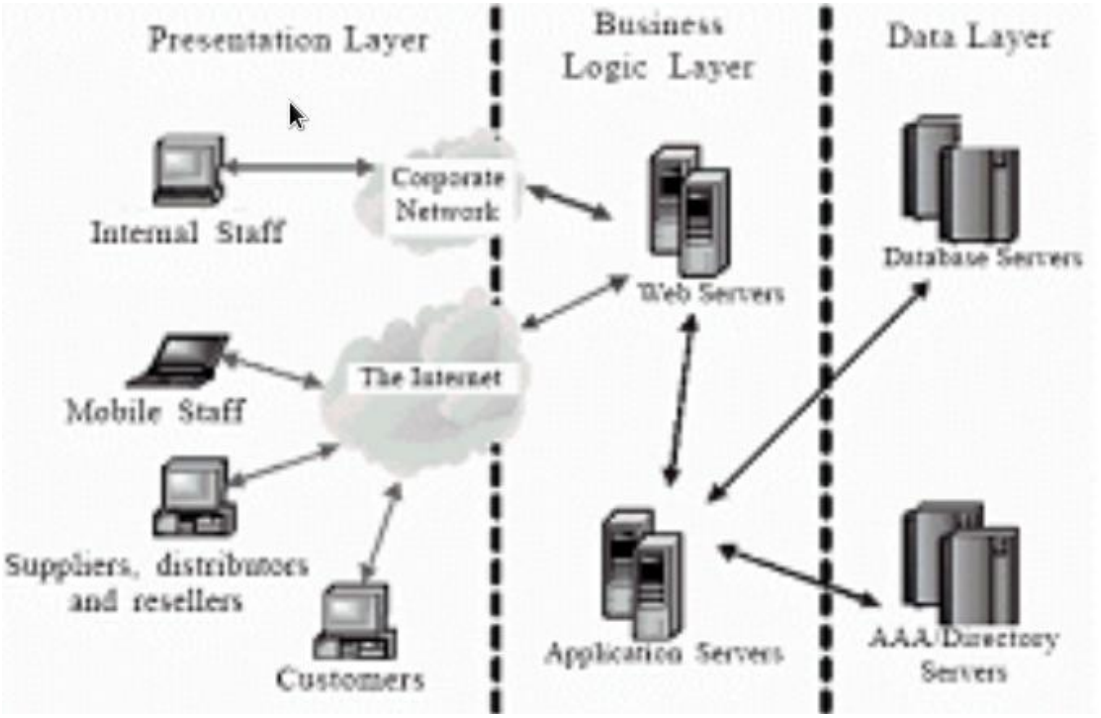

Abbildung 2: Architektur von E-Commerce-Systemen

Die Präsentationsschicht wird meistens im Browser des Endnutzers vollzogen. Die Businesslogik-Schicht wird durch Applikationsserver dargestellt, die die eigentlichen Prozesse ausführen. Außerdem dient diese Schicht als „Service Point“ und konzentriert sich auf die kundengerechte Bedienung (schnelle und benutzerfreundliche Dialogfelder, individuelle Beantwortung von Client-Anfragen, Werbung, Sicherheit usw.)“, so Strobel in ([STR-03], S. 21). Die Datenschicht wird in den aktuellen Versionen von E-Commerce-Applikationen durch eine sogenannte Entity-Schicht präsentiert. Die Entity-Schicht abstrahiert das Datenmodell und macht den Datenbankserver transparent.

3 Sicherheit von E-Business-Systemen

Mit Sicherheit sind dabei in erster Linie der sichere Transport und die sichere Speicherung von Daten gemeint. Aber auch die angeschlossenen Rechner können bei Sicherheitsproblemen in Mitleidenschaft gezogen werden. Viele potenzielle Kunden sind wegen ihrer Sicherheitsbedenkenzurückhaltend beim Thema E-Business. Der Grund dafür mag auch darin liegen, dass viele E-Business-Anbieter nebensinnvollen auch eine Vielzahl überflüssiger Angaben von ihren Kunden erfragen. In diesem Kapitel wird die Bedrohungslage von E-Business-Systemen erläutert. Anschließend (Kap. 3.2) wird auf die Sicherheitslücken von E-Commerce-Systemen eingegangen, sie werden kurz skizziert. Für die Durchführung von Penetrationstest wird ein Sicherheitskozept benötigt, das wird in Kapitel 3.3 beschrieben.

3.1 Bedrohung für E-Commerce-Systeme

E-Commerce-Systeme sind Webanwendungen, die kontinuierlich an die Gegebenheiten im World Wide Web angepasst werden und somit einem ständigen Wandel unterzogen sind. Damit über das WWW auf Webanwendungen zugegriffen werden kann, müssen diese flexibel, aber gleichzeitig auch zuverlässig sein. Bei der Entwicklung von Webanwendungen ergeben sich erhebliche Anforderungen, die im Detail nur sehr schwer und inkrementell evaluiert werden können. Analog zur Anforderungsanalyse für Softwareentwicklung sind

bei der Entwicklung von Webanwendungen neben den funktionalen Anforderungen ebenso nicht-funktionale Anforderungen zu berücksichtigen. Das Thema Sicherheit ist eine Qualitätsanforderung der nicht-funktionalen Gruppe, die zunehmend an Bedeutung gewinnt.

Die Sicherheit in der Webkommunikation und insbesondere auch in Webanwendungen beschreibt keinen finalen Zustand, sondern ein iterativer Prozess, der immer wieder an die aktuellen Gegebenheiten angepasst werden muss. Stuttard und Pinto beschreiben es in ([WebHack], S. 38) so:

> *„Web applications bring with them new and significant security threats. Each application is different and may contain unique vulnerabilities.“*

Folgerichtig sollte Sicherheitsarchitektur in E-Business-Systemen als ein ganzheitlicher entwicklungsbegleitender Prozess verstanden werden. Das größte Sicherheitsproblem, mit dem die E-Commerce-Systeme konfrontiert sind, ist die Tatsache, dass die Anwendungen eine unbekannte Varietät möglicher Benutzereingaben verarbeiten müssen. Das bedeutet, Nutzer können völlig willkürlich Eingabedaten an den Server schicken. Der Server muss alle Eingabedaten zunächst als potenziell bösartig einstufen, um einen nicht-authentifizierten Zugriff abzuwehren und die Datenintegrität zu erhalten. Stuttard und Pinto beschreiben das Problem in ([WebHack], S. 43) so:

> *„The majority of attacks against web applications involve sending input to the server which is crafted to cause some event that was not expected or desired by the application’s designer.“*

Das heißt, E-Commerce-Systeme werden in der Designphase auf alle bekannten Sicherheitslücken vorbereitet. Diese Abschirmung ist aber keine generische Lösung für alle Bedrohungen.
Die Kernursachen für Sicherheitsprobleme in Web-Applikationen gehen zurück auf (vgl. [WebHack], S. 44):

- unreifes Sicherheitsbewusstsein der Entwickler
- Eigenentwicklung mit Fokus auf Features
- trügerische Einfachheit der eingesetzten Frameworks zu Herstellung von Webanwendungen und E-Commerce-

Anwendungen
- sich rasch entwickelnde und anpassende Bedrohungsprofile
- Ressourcen und Zeitdruck in IT-Projekten und
- überforderte Technologien

Gefahren, die aus den Sicherheitsrisiken bei E-Commerce-Anwendungen hervorgehen, betreffen deren Merkmale Integrität, Vertraulichkeit, Verbindlichkeit, Verfügbarkeit sowie Authentizität.

Eckert formuliert es in ([Eck06], S. 85) so:

> *„Sicherheitsbedrohungen ergeben sich durch das unautorisierte Lesen elektronischer Nachrichten, die sensible, personenbezogene Informationen oder Geschäftsdaten beinhalten können, durch das unautorisierte Verändern von gesendeten Daten, so dass gefälschte Informationen beim Empfänger ankommen, oder auch durch das Maskieren und Vortäuschen einer falschen Absenderidentität, so dass der Kommunikationspartner im Vertrauen auf diese Identität vertrauliche Informationen preisgibt.“*

3.1.1 Was ist zu schützen?

Gefahren für die Sicherheit von E-Commerce-Anwendungen können gewollt oder ungewollt (beispielsweise aus Unwissenheit) in den verschiedenen Entwicklungsphasen einer Anwendung von Entwicklern, Benutzern und externen Personen ausgehen. Soweit der Wille zur Schädigung bzw. Ausforschung vorhanden ist, kann ein Angriff formuliert werden, für den die Beschreibung eines Angreifer-Modells erfolgt. Die Systeme sind sowohl vor internen als auch vor externen Gefahren zu schützen.

Folgerichtig ist ein E-Commerce-System vor unbefugtem Informationsgewinn (Verlust der Vertraulichkeit), vor unbefugter Veränderung von Daten (Verlust der Integrität), und vor unbefugter Beeinträchtigung der Funktionalität (Verlust der Verfügbarkeit) zu schützen.

3.2 Sicherheitslücken in E-Commerce-Anwendungen

Es gibt unterschiedliche Formen von Sicherheitsproblemen im E-Business. Man kann sie unterteilen in:

- die gezielte Überlastung einzelner Dienste (Denial-of-Service-Attacken)
- das Manipulieren von Webangeboten (die Internetdarstellung eines Unternehmenswird verändert)
- die unberechtigte Inanspruchnahme von Dienstleistungen
- Unbefugtes Lesen und Missbrauchen der Kundendaten

Das OWASP[14] ist eine nicht profitorientierte Organisation, welche im Jahr 2001 gegründet wurde. OWASP veröffentlicht in regelmäßigen Abständen eine Top-Ten-Liste der Sicherheitsschwachstellen in Webapplikationen. Die List aus dem Jahr 2010
stellt die Reihenfolge der Bedrohung auf die Internet-Applikationen wie folgt vor:

1. Injection
2. Cross Site Scripting
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross Site Request Forgery
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Invalidated Redirects and Forwards

E-Commerce-Systeme als eine Unterklasse von webbasierten Applikationen können auch direkt von den oben genannten Schwachstellen betroffen sein. Aus diesem Grund befasst sich diese Arbeit mit der Top-Ten-Liste.

[14] engl. *Open Web Application Security Project*

3.2.1 Injection

Unter Injection im Allgemeinen und SQL-Injection im Besondern wird die Konkatenation von unsicheren Zeichenketten (Input aus Benutzereingaben) zu sicheren Zeichenketten und deren Ausführung als Datenbank-Abfrage oder Kommando verstanden. Das Ziel derartiger Angriffe ist, Daten zu stehlen bzw. zu verändern, oder Kontrolle über den Datenbankserver zu erhalten (vgl. [EckO6], S. 157) Ein Beispiel dafür ist folgender Code:

```
String sql = "select * from user where username='" +
username + "' and password='" + password + "'";

Statement stmt = conn.createStatement();

rs = stmt.executeQuery(sql);
if (rs.next()) {
      boolean loggedIn = true;
      System.out.println("Successfully logged in");
      } else {
      System.out.println("Username and/or password not
recognized");
}
```

In diesem Beispiel werden der Benutzername und das Passwort ungefiltert aus den Nutzereingaben generiert, wenn ein Attacker den Parameter Benutzername so manipuliert: username="admin`OR `1`=`1". Diese Eingabe versetzt den Angreifer in die Lage, sich ohne Eingabe eines Passwortes auf der Seite anzumelden, weil die Oder-Klausel die Abfrage als wahr auswerten lässt.

In Java-Programmen kann man SQL-Injection vermeiden, in dem man für die Generierung von SQL-Abfragen die Klasse PreparedStatement einsetzt. Dabei wird das eigentliche SQL-Statement vorkompiliert und die Benutzereingaben werden als Parameter dem Statement übergeben. Die Daten werden nicht interpretiert und somit wird ein SQL-Statement verhindert.

3.2.2 CrossSiteScripting

Webseiten können durch Manipulation aus dynamisch erzeugten Inhalten Schadcode in Form von HTML-Tags, oder Javascript-Code enthalten. Grundsätzlich kann man sagen, dass Inhalte, die durch Benutzerinteraktion an die Webserver gesendet werden, von Webseiten-Programmierern als nicht vertrauenswürdig einzustufen und deshalb zu überprüfen sind.

Man spricht von einer Cross Site Scripting[15] Schwachstelle wenn Benutzerdaten, die an den Webserver gesendet werden, vom Server ungeprüft an den Nutzer zurückgesendet werden und im Nutzers Browsers zur Ausführung kommen (vgl. [Eck06], S. 149). Angreifer setzen diese Techniken gezielt ein, um Benutzerdaten zu stehlen, oder Schadcode zu verbreiten.

Weil bei diesem Angriff Seiten durch Benutzerinteraktion Schadcode an andere Seiten verschicken, wird XSS in der Regel als webseitenübergreifender Angriff bezeichnet.

Abbildung 3 zeigt ein Szenario für Cross Site Scripting.

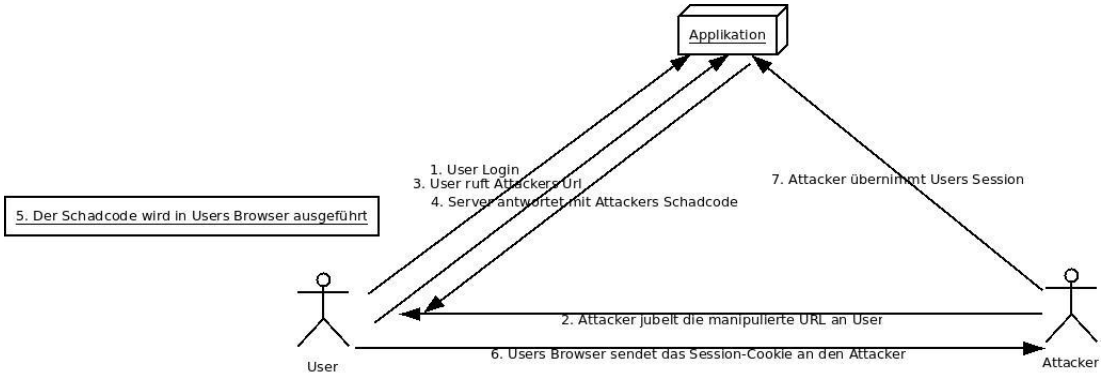

Abbildung 3: XSS Session- Hijacking

In dem Schaubild wird das Cookie-Stealing[16] praktisch dargestellt. Bei dieser Technik versucht ein Attacker in eine authentifizierte Session einzudringen, um dieselben Privilegien in der Session zu erlangen, wie der rechtmäßig authentifizierte Session-Benutzer.

[15] auch als XSS abgekürzt
[16] engl. *Stehlen von Session-Cookie*, oder auch Session-Hijacking

Der User in diesem Fall ist Opfer einer XSS-Attacke. Der Attacker (rechts im Bild) sendet z.B. Massenmails an verschiedene Nutzer mit einem ähnlichen Inhaltwie diesem:

> Sehr geehrter Kunde, wir führen eine Statistik durch und bitten Sie, uns dabei zu helfen. WasSie brauchen, ist 5 Minuten Zeit, am Ende bedanken wir uns bei Ihnen miteinem Gutschein von 10 Euro.
>
> Bitte melden Sie sich in Ihrem Onlineshop an und klicken bitte auf den folgenden Link:
>
> https://OnlineShopFuerTee.com/%65%72%72%6f%72%2e%70%68%70?message%3d%3c%73%63%72ipt>var+i=ne%77+Im%61ge%3b+i.s%72c=”ht%74%70%3a%2f%2f%77my-att%61%63%6ber.co%6d%2f”%2bdocum%65%6e%74%2e%63ookie;</%73%63ript%3e

Die Url sieht decoded so aus:

> https://OnlineShopFuerTee.com/error.php?message=<script>var+i=new+Image;+i.src=”http://my-attacker.com/“%2bdocument.cookie;</script>

Dann vollzieht sich folgendes Szenario:

1. Der Nutzer loggt sich ein und bekommt ein Session-Cookie vonOnlineShopFuerTee.com zugewiesen: sessionid=12234344kjhh55656

2. Der Nutzer hat den manipulierten Link vom Angreifer erhalten.

3. Der Nutzer klickt auf den Link und bekommt eine Fehlerseite vom Server, weil die Seite error.php auf dem Server nicht existiert.

4. Der Server antwortet und schickt den eingeschleusten

Javascript-Code mit an den User.

5. Der Code wir im Browser des Benutzers ausgeführt.

6. Der manipulierte Code bringt den Browser dazu den folgenden Codeaufzurufen:

```
var i = new Image; i.src=”http://my-attacker.com/“+document.cookie;
```

Die Anfrage beinhaltet das Session-Cookie vom Benutzer.
Das obige Szenario beschreibt nur eine der vielen Möglichkeiten von Cross Site Scripting.

3.2.3 Broken Authentication and Session Management

Man spricht von „Brocken Authentication and Session Management“, wenn Zugangsdaten von Benutzern, oder Session-Tokens, nicht hinreichend geschützt sind. Angreifer können mittels Session-Tokens im Namen und mit den Rechten des Opfers unautorisiert in die gesperrten Bereiche vordringen und Dienste in Anspruch nehmen (z.B. kostenpflichtige Downloads) (vgl. [OWASP-Top10])

3.2.4 Insecure Direct Object References

Bei der Entwicklung von Web-Anwendungen werden oft Objektreferenzen verwendet, um auf ein bestimmtes internes Implementierungsobjekt zu verweisen, z.B. Verweise auf eine Datei, ein Verzeichnis oder Datenbankeinträge. Die Schwachstelle dabei ist, dass die Zugriffskontrollen auf diese Objekte entweder unzureichend sind oder gänzlich fehlen.

Ein Angreifer kann durch geschicktes Manipulieren unautorisiert auf Dateien und Inhalte zugreifen. Vor allem, wenn Applikationen mit sensiblen Daten arbeiten, sind diese Attacken gefährlich. Dabei verwenden Angreifer meist manipulierte IDs oder Pfadangaben, um etwa fremde Datensätze aus der Datenbank auszulesen oder unautorisiert auf Dateien des Web-Servers zuzugreifen. Ein möglicher Fall wäre, wenn bei Fehlermeldungen Informationen ausgegeben würden über Art und Weise der Fehler und wo sie aufgetreten sind (in welcher

Datei). Diese Informationen können vom Angreifer in Form von manipulierten Links dazu benutzt werden, sensible Informationen aus dem System zu holen (vgl. [OWASP-Top10]).

3.2.5 Cross Site Request Forgery

Bei Cross Site Request Forgery geht es um eine Attacke aus einer bestehend gültigen Session auf eine verwundbare Funktion. Dabei bleibt der Angriff dem autorisierten Nutzer verborgen. Der Angreifer bringt den Browser dazu, eine geschützt Funktion, z.B. die Überweisungsfunktion einer Bank-Applikation aufzurufen. Die Funktion liefert dann das gewünschte Ergebnis, weil der Nutzer sich noch in einer gültig authentifizierten Session befindet.

Bei dieser Angriffsvariante wird der rechtmäßig angemeldete Benutzer zum "Bauernopfer", indem - ohne sein Wissen - von seiner authentisierten Session eine Anfrage durchgeführt wird. Möglich ist das beispielsweise, wenn sich der bei einer Web-Anwendung angemeldete User beim Verlassen derselben nicht abmeldet und beim weiteren Surfen auf eine vom Angreifer präparierte Seite gelangt. CSFR ist die Variante von Angriffen auf Web-Anwendungen, deren Anzahl derzeit am schnellsten zunimmt.

3.2.6 Security Misconfiguration

Bei Security Misconfiguration geht es um falsche oder unzureichende Sicherheitskonfiguration von Servern, auf denen Webapplikationen laufen. Diese Sicherheitslücke stellt ein echtes Risiko dar und aufgrund ihrer hohen Verbreitung wird sie als ein Punkt in den Top-Ten aufgeführt (vgl. [OWASP-Top10]).

3.2.7 Insecure Cryptographic Storage

Diese Schwachstelle entsteht durch den fehlerhaften Einsatz von Verschlüsselungstechniken zum Schutz von sensitiven Daten. Dabei ist die Verschlüsselung schützenswerter Daten wie Zugangs- und Kreditkarteninformationen oder die Nicht-Vorhersagbarkeit etwa von Session-Tokens ein wichtiger Mechanismus zur Absicherung einer Web-Applikation. In vielen Web-Applikationen fehlen jedoch

kryptografische Funktionen oder sind schlecht implementiert. Grund hierfür ist meistens die Verwendung eigenimplementierter Verschlüsselungsalgorithmen, die entweder schwach (etwa SHA[17]-1 oder MD5[18]) oder fehlerhaft sind.

Diese Funktionen können gebrochen werden und bieten somit keinen Schutz gegen Angriffe. Zudem werden die kryptografischen Schlüssel häufig an unsicheren Stellen (etwa in der Anwendung herunterladbar) aufbewahrt, wodurch die Sicherheit des gesamten Verschlüsselungsmechanismus aufgehoben wird.

3.2.8 Failure to Restrict URL Access

Kritische Informationen in einer Web-Anwendung werden häufig lediglich dadurch geschützt, dass die entsprechende URL einem unautorisierten Benutzer nicht angezeigt wird oder nicht bekannt ist. Für eine Attacke lässt sich das ausnutzen, indem die URL direkt angesprochen wird. Die bekannteste Angriffsmethode, die diese Lücke missbraucht, nennt sich "Forceful Browsing": Der Angreifer versucht, durch systematisches "Ausprobieren" ungeschützte Seiteninhalte oder Anwendungsfunktionen zu identifizieren und darauf zuzugreifen.
Das Ziel ist häufig, versteckte Dateien oder URLs ausfindig zu machen, die bei der Implementierung der Berechtigungen übersehen wurden. Für den Betreiber einer Web-Anwendung sind solche Schwachstellen besonders brisant, da ein Angreifer Informationen über deren Aufbau und Struktur gewinnt oder sogar Zugriff auf administrative Funktionen der Seite erhält.

3.2.9 Insufficient Transport Layer Protection

Bei dieser Art von Schwachstelle werden sensible Daten über einen unsicheren Kommunikationskanal im Klartext oder nur teilweise, bzw. unsicher verschlüsselt übertragen. In diesen Fällen kann ein Angreifer durch passives Mitlesen oder eine Man-in-the-Middle-Attacke[19] auf die transferierten Daten zugreifen. Vor allem bei der Übermittlung

[17] *eine kryptographische Hashfunktion*
[18] engl. *Message-Digest Algorith*, Hashfunktion zur Prüfsummenbildung
[19] engl. *Mittelsmannangriff*, Angreifer steht zwischen Kommunikationspartnern und lauscht den Datenverkehr

vertraulicher Informationen wie Zugangs-, Zahlungs- oder Kundendaten ist es notwendig, die Datenkommunikation zu verschlüsseln. Jedoch gibt es Web-Anwendungen, die nur den Austausch der Login-Informationen über einen sicheren Übertragungskanal abwickeln und die anschließende Kommunikation unverschlüsselt lassen. Dabei wird häufig vergessen, dass auch bei den folgenden Anfragen sicherheitsrelevante Authentifizierungsinformationen wie Session-Tokens übertragen werden. Das ermöglicht dem Angreifer, unverschlüsselte Tokens mitzulesen und sich mit der fremden Identität bei der Web-Anwendung zu authentisieren.

3.2.10 Invalidated Redirects and Forwards

Webapplikationen nutzen in Ihrem Prozessablauf oft Weiterleitungen. Dabei werden die Ziele der Weiterleitungen aus unsicheren Daten bestimmt. Diese Schwachstelle können vom Angreifer durch manipulierte Weiterleitungsziele so ausgenutzt werden, dass die Nutzer am Ende auf Phishing-Seiten weitergeleitet werden.

Nachfolgende Tabelle zeigt eine im Rahmen der Arbeit entstandene Liste der Sicherheitslücken für Webanwendungen. Die Spalte Relevanz bezieht sich auf die Rolle der Sicherheitslücken in Webanwendungen. Die Spalte Priorisierung beschreibt

3.3 Sicherheitskonzept

Um die Sicherheitsschwachstellen von E-Commerce-Systemen an dem Testgegenstand aufzuspüren, werden Penetrationstests durchgeführt. Penetrationstests werden in der Regel für große Systemlandschaften oder Unternehmensnetzwerke entworfen. In der Fachliteratur werden Konzepte zur Durchführung von Penetrationstests mit dem o. g. speziellen Blickpunkt beschrieben.

In der vorliegenden Arbeit geht es aber um das Aufspüren von Sicherheitsschwachstellen eines bestimmten E-Business-Frameworks. Dabei werden die Netzwerkkomponente und die Schnittstellen zu Nachbarsystemen außer Acht gelassen.

Der zweite Unterschied liegt in der Art und Weise, wie die Tests durchgeführt werden. Die herkömmlichen Penetrationstests werden zumeist von externen Experten durchgeführt. Der innere Zusammenbau des zu untersuchenden Systems ist den Experten aber nicht zwingend bekannt. Aus diesem Grund wird einem herkömmlichen Penetrationstest der Ansatz von Blackbox-Testing zugrunde gelegt.

OfBiz ist aber eine quelloffene Software. Der innere Aufbau ist gut dokumentiert und für alle zugänglich. Die Programmierschnittstellen sind bekannt. Daher werden in dieser Arbeit Whitebox-Tests angewendet.

Um die Schwachstellen der Webapplikationen zu untersuchen, wird auch eine Überprüfung der Source Code notwendig. Diese wiederum stellt eine weitere Abweichung von den herkömmlichen Pentetrationstests.

3.3.1 Phasen des Penetrationstests angepasst auf die vorliegende Studie

In dieser Arbeit werden Penetrationstests in folgenden drei Teilbereichen unterteilt:

1. Penetrationstests – Vorbereitung und Datensammlung
2. Penetrationstests – Statisch Tests
3. Penetrationstests – Aktive Exploitversuche

Im ersten Teil geht es um die Vorbereitungsphase und die Informationssammlung. Der zweite Teil beschäftigt sich mit der statischen Analyse von Source Code. Im dritten Teilbereich werden aktive Exploitversuch behandelt.

Unter einem Penetrationstest versteht man den kontrollierten Versuch, von "außen" in ein bestimmtes Computersystem bzw. -netzwerk einzudringen, um Schwachstellen zu identifizieren, bevor diese von unautorisierten Dritten ausgenutzt werden können (vgl. [LDSH]). Demzufolge liefern solche Tests eine Einschätzung darüber, wie erfolgreich vorsätzliche Angriffe unter gegebenen Bedingungen die Sicherheitslücken des Systems ausnutzen können ([Eck06], S. 194).

Allgemein wird die Durchführung von Penetrationstests in folgende Phasen unterteilt:

1) Vorbereitung des Penetrationstests
2) Informationsbeschaffung
3) Bewertung der Informationen/Risikoanalyse
4) aktive Eindringversuche
5) Abschlussanalyse

Die Abbildung 4 illustriert die Durchführung der allgemeingültigen Penetrationstestphasen nach [BSI02]

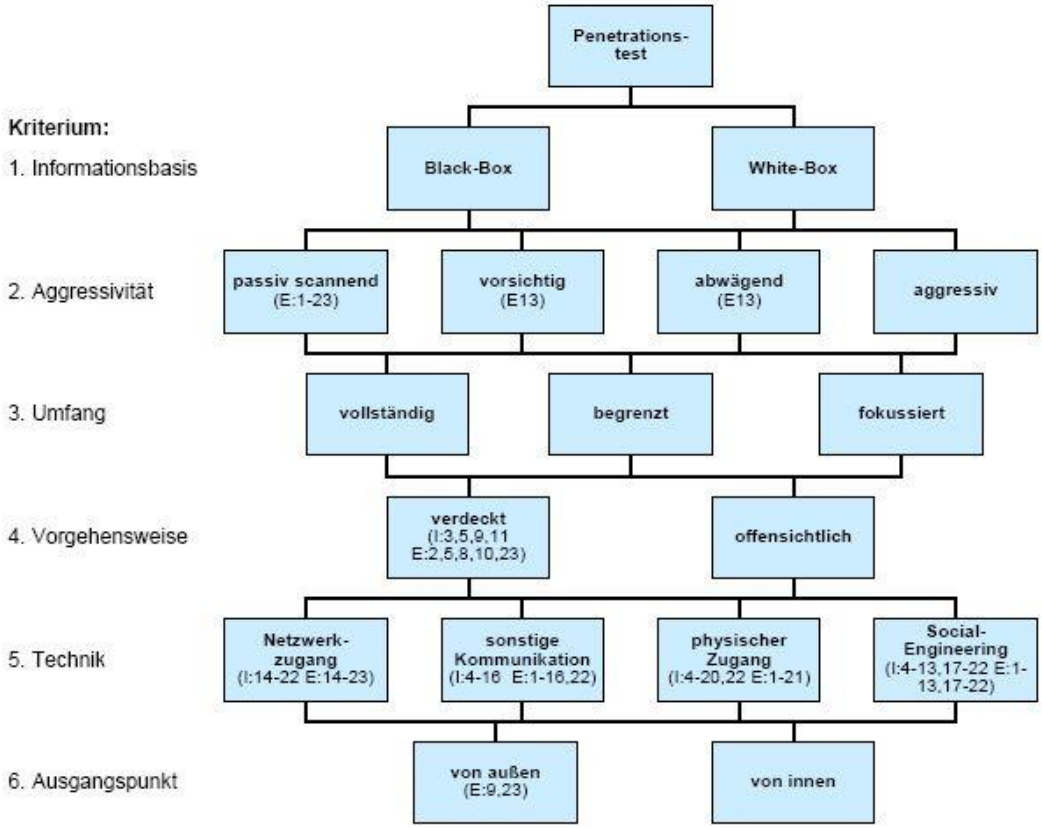

Abbildung 4 Phasen eines Penetrationstests, Quelle www.datenschutzzentrum.de

Analog der im Schaubild dargestellten Gliederung wird die vorliegende Untersuchung in unterschiedlichen Phasen folgendermaßen organisiert.

Es geht um White-Box-Tests, weil die Anforderung klardefiniert ist und die Quellen und Dokumentationen offen vorliegen. Diese Tests werden mit der Intensität aggressiv durchgeführt – es gilt nicht ein Online-System zu untersuchen, dessen Ausfall größere Schäden verursachen würde. Möglichst alle Schwachstellen auf Anwendungsebene sollen offenbart werden.
Der Umfang der Arbeit ist fokussiert, das bedeutet, die Tests beschränken sich auf die Anwendungsebene, speziell werden die Schwachstellen einer Webanwendung mit dem Schwerpunkt eines E-Business-Systems untersucht.
Die Tests werden offensichtlich durchgeführt, die Maßnahme ist nicht verdeckt. Das Kommunikationsmedium ist das Netzwerk und die Untersuchungen gehen von innen aus.

Es gibt bestimmte Variationen von Angriffsmethoden, die für E-Commerce-Anwendungen, als Unterklasse von Webanwendungen, speziell sind. Diese Besonderheit wird hier berücksichtigt, und die Penetrationstests an die besonderen Anforderungen angepasst.

Zu den Anforderungen zählen eine Liste der „Mindestanforderungen zur Informationssicherheit bei eCommerce-Anbietern“ von BSI [BSI03] bezogen auf Anwendungsebene, und Beseitigung von offenen Top-10 Schwachstellen von OWASP [OWASP-Top10].

4 Penetrationstests – Vorbereitung und Datensammlung

Diese Phase leitet die Penetrationstests ein. Der Verlauf der Penetrationstests hängt stark von der Vorbereitung und der Planung ab. Zu Anfang dieser Phase muss geklärt werden, welchen Umfang ein solcher Test haben sollte, welche Notfallmaßnahmen getroffen werden müssen und welche Vorgehensweise erwünscht ist. Damit die rechtliche Absicherung gegeben ist, müssen vor allem die Entscheider (das Management) in dieser Phase miteinbezogen werden.

In Kapitel 4.1 wird der Umfang dieser Untersuchung festgelegt, Ziele der Penetrationstests werden abgestimmt und definiert. Nachdem der Umfang bekannt ist werden in Kapitel 4.2 Informationen über das Framework gesammelt. Ziel ist es, eine möglichst komplette und detaillierte Übersicht über die potenziellen Angriffspunkte bzw. die bekannten Sicherheitsmängel zu erlangen.

4.1 Vorbereitung und Zieldefinition

Die Vorbereitungsphase beginnt mit der Definition der Ziele, die durch Penetrationstests erreicht werden sollen (vgl. [BSI2]). Mögliche Ziele sind die Erhöhung der Sicherheit der

technischen Systeme und Sensibilisierung der Mitarbeiter für sicherheitsrelevante Themen im Unternehmen.

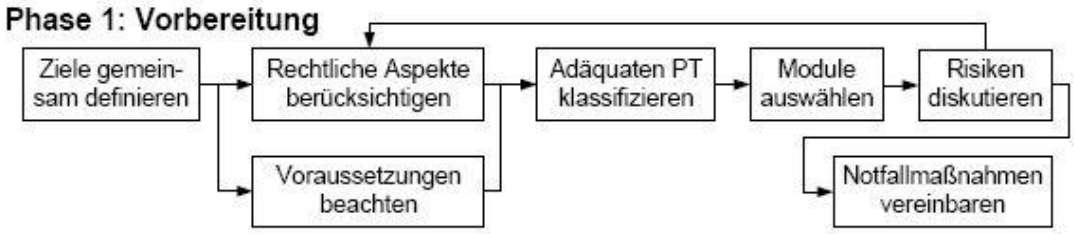

Abbildung 5: Vorbereitungsphase, Quelle BSI

Das Ziel dieser Arbeit ist herauszufinden, ob es repräsentative E-Business-Anwendungen gibt, die von Grund auf sicher sind. Sicherheit in diesem Kontext bedeutet, dass es keine gravierenden Schwachstellen gibt, die nach erfolgter Installation von Dritten unautorisiert ausgenutzt werden können. Des Weiteren soll untersucht werden, wie sich die Sicherheitsaspekte nach einer Anpassung des Systems ändern. Deshalb wird in dieser Arbeit zwischen einer Initialfassung und einer angepassten Version von OfBiz unterschieden.

Können Entwickler durch nachträgliche Anpassung (angepasste Version) von Logik und Prozessabläufen die Sicherheit des Systems gravierend beeinträchtigen? Diese Fragestellung soll durch den Einsatz von Penetrationstests ebenfalls beleuchtet werden, außerdem soll die Auswertung mögliche Antworten liefern.

4.1.1 Umfang

Die Sicherheitsüberprüfung des Systems bezieht sich nur auf die Anwendungsebene. Das Netzwerk und die angeschlossene IT-Infrastruktur (Schnittstellen zu anderen Systemen) werden nicht untersucht. Auf Anwendungsebene werden bekannte Schwachstellen von Webanwendungen untersucht. Außerdem wird eine Source-Code-Analyse durch sogenannte Code Scanner durchgeführt. Ferner wird geprüft, ob das Framework eine robuste Schale mit sich bringt, die nachträglich programmierte Schwachstellen kompensieren kann oder ob und wie das Framework durch nachlässige programmatische Anpassung geschwächt werden kann.

4.1.2 Vorgehen

Als Untersuchungsobjekt wird das OFBiz als ein großes Open Source Framework, welches die ganze Bandbreite an Applikationen wie ERP[20], CRM[21] und E-Commerce vereint, gewählt.

Das Framework ist nach dem MVC[22]-Pattern durch den Einsatz von Standard-Technologien aufgebaut. Aus diesem Grund ist es repräsentativ für andere (kommerzielle und nicht-kommerzielle) E-Business-Systeme auf Basis von Java. Aus den genannten Gründen ist diese OfBiz für die Untersuchung in dieser Arbeit sehr gut geeignet.

Der Fokus dieser Arbeit ist auf die sicherheitstechnische Überprüfung der E-Commerce-Plattform des OfBiz Frameworks gelegt. Die anderen Bereiche des Frameworks werden im Rahmen der vorliegenden Arbeit nicht behandelt.

Das BSI hat im Jahr 2011 in (vgl. [BSI03], S. 8-12) Mindestanforderungen für E-Commerce-Anbieter definiert. Darin werden die Anforderungen in sechs Klassen unterteilt:

1) Physische Sicherheit
2) Netzsicherheit
3) Sicherheit von IT-Systemen
4) Anwendungssicherheit
5) Verschlüsselung und Schlüsselmanagement
6) Patch- und Änderungsmanagement

Im Rahmen dieser Studie wird die Anwendungssicherheit betrachtet. Aspekte der Anwendungssicherheit, die mit Hilfe von Penetrationstests beleuchtet werden sollen, sind:

1) Datensparsamkeit
2) Sichere Software-Entwicklung
3) Sichere Datenspeicherung
4) Offene Schwachstellen

[20] engl. *Enterprise Resource Planning*
[21] engl. *Customer Relationship Management*
[22] engl. *Model View Controler*

Daraus ergeben sich folgende Ziele für die Penetrationstests:

- Wie viele Benutzerdaten werden im Rahmen der Benutzer-Registrierung und Online-Kaufabwicklung von OfBiz verarbeitet und gespeichert?
- Wie sicher sind die Einzelmodule in OfBiz programmiert? Sind die Prinzipien der sicheren Sofware-Entwicklung eingehalten?
- Wie werden die Benutzerdaten z.B. persönliche Daten, Kreditkartendaten erfasst und gespeichert?
- Wird für die Kommunikation (Datenaustausch) auf sichere Kanäle zurückgegriffen?
- Wie geht OfBiz mit den offenen Schwachstellen, wie XSS, SQL-Enjection usw. um? Wird durch regelmäßige Updates auf die aktuellen Bedrohungen reagiert?

4.2 Informationsbeschaffung

Nachdem Ziele, Umfang und Vorgehen definiert worden sind, kann mit der Sammlung von Informationen über das Zielsystem begonnen werden. Die Informationsbeschaffungsphase wird auch als passiver Penetrationstest bezeichnet. Ziel ist es, eine möglichst komplette und detaillierte Übersicht über die installierten Systeme inklusive der potenziellen Angriffspunkte bzw. der bekannten Sicherheitsmängel zu erlangen (vgl. [BSI02], S. 45).

Anlehnend an die von [BSI03] und [BSI02] vorgestellten Anforderungen und das Phasenkonzept wird dieses Kapitel in folgende Unterkapitel unterteilt:

- Umgang mit Benutzerdaten
- Sichere Software-Entwicklung
- Aufdecken von offenen Schwachstellen

4.2.1 Umgang mit Benutzerdaten

Die Frage nach Datensparsamkeit kann man durch die Betrachtung der Bereiche beantworten, wo Benutzerdaten gesammelt werden. Generell gilt: Nur solche Daten sollten erhoben, verarbeitet und gespeichert werden, die für die Abwicklung eines Online-Kaufprozesses notwendig sind. Altdatenbestände, für die keine Speicherungsnotwendigkeiten mehr bestehen, sind sicher zu löschen

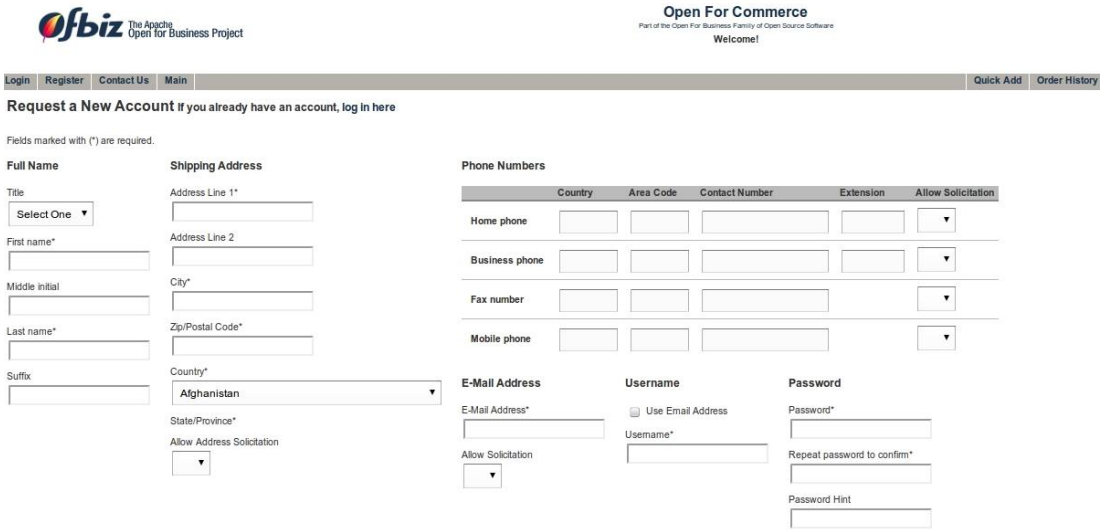

Abbildung 6: OfBiz Account-Formular

Die Abbildung 6 zeigt das Neukunden-Registrierungsformular in der Initialfassung (ohne Anpassung des Systems). Die Daten, die hier erfasst werden, werden benötigt, um eine eventuelle Bestellung des Kunden zu bearbeiten. Bankdaten, wie zum Beispiel Kreditkartennummer und Kontonummer werden im Lauf des Benutzer-Registrierungsprozesses nicht erfasst. Dieser Zustand kann nach einer Anpassung des Systems auf Anforderung des Betreibers geändert werden. Somit können mehr Daten erfasst werden als unbedingt nötig.

Im Bestellgenerierungsprozess werden je nach ausgewählter Zahlungsmethode Bankdaten oder Kreditkartendaten erfasst. Die Daten sind an dieser Stelle notwendig, um den Kaufprozess abzuwickeln. Das Problem, welches sich hinter der Speicherung dieser Daten

verbirgt, ist die Tatsache, dass sie in der Initialfassung von OfBiz in der lokalen Datenbank gespeichert werden. Das ist ein offenes Sicherheitsrisiko, weil Cyber-Kriminelle sich durch gezielte Angriffe Zugang zu der Datenbank oder den Kommunikationskanälen, welche die Daten durchlaufen, verschaffen und dadurch in den Besitz wertvoller Daten gelangen können. Abbildung 7 zeigt die Zusammenfassung der gesammelten Daten im Bestellprozess.

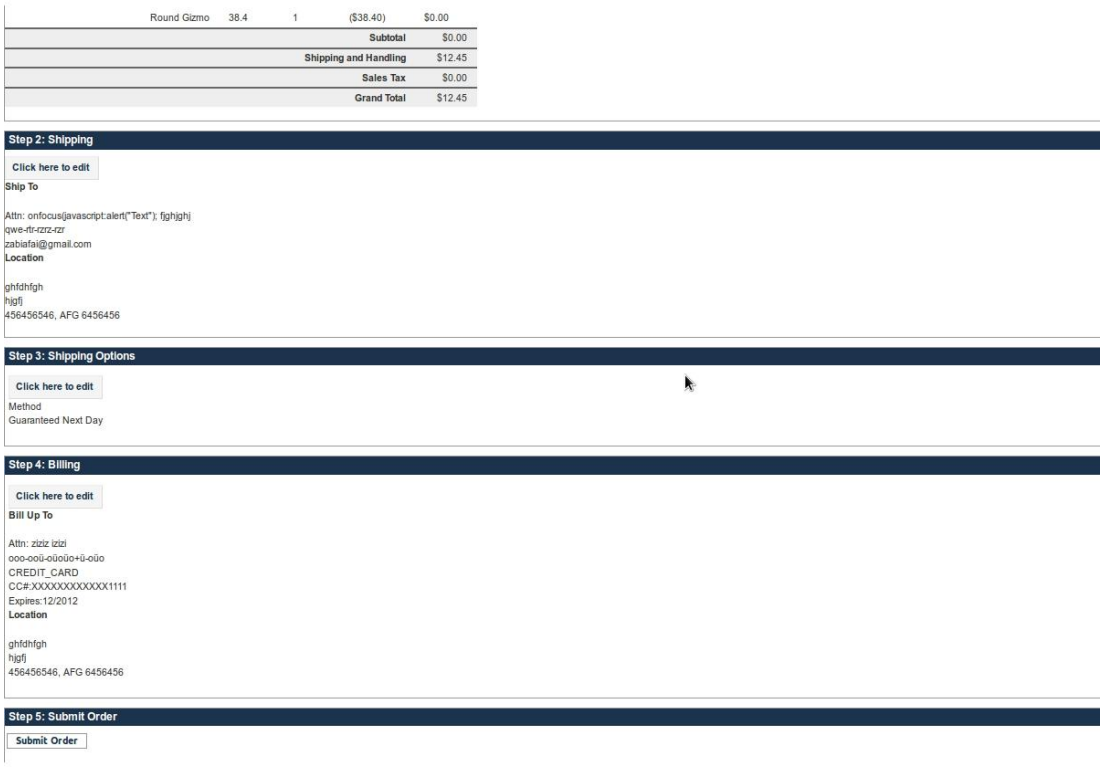

Abbildung 7: OfBiz, Zusammenfassung der gesammelten Daten

Man kann anhand der Fülle der Daten, die im System verarbeitet und gespeichert werden, schon eine Aussage darüber machen, wie sparsam das System mit Kundendaten umgeht.

Im Fall von OfBiz in der Initialfassung ist die Datensparsamkeit gegeben. Für den Datenaustausch werden SSL[23]-Verbindungen zur sicheren Übertragung von Daten genutzt.

Was die Datenspeicherung angeht, ist es in der Initialfassung sicherheitstechnisch sehr bedenklich. Das System wird, wie die meisten java-basierten Systeme mit HSQL-Datenbank[24] geliefert. HSQL ist eine Embedded-Datenbank[25] und eigentlich zum Testen von Software gedacht und für den produktiven Einsatz nicht geeignet.
Außerdem wird die Datenbank mit initialen Benutzernamen und Passwörtern ausgestattet, was ein großes Sicherheitsrisiko darstellt, weil die Daten im Internet schon bekannt sind. Diese Daten müssen vor einem produktiven Einsatz von OfBiz unbedingt geändert werden.

[23] engl. *Secure Sockets Layer*
[24] Hypersonic SQL Database, http://hsqldb.org/
[25] in einer Anwendung *eingebettetes Datenbanksystem,* das nach außen nicht sichtbar in Erscheinung tritt

5 Penetrationstest – Statische Tests

In diesem Kapitel wird die Qualität des Quelltextes (Source Code) untersucht. Der Quellcode wird dabei anhand von formalen Methoden auf Fehlersituationen geprüft. Sinn und Zweck dieser Untersuchungen ist, Fehler zu finden, bevor sie in die Produktion gehen.

In diesem Kapitel wird der Quelltext des Testgegenstands untersucht und analysiert. Kapitel 5.1 beschreibt die Methoden der sicheren Softwareentwicklung. Im weiteren Verlauf (Kap 5.2) werden diese Maßnahmen auf den Testgegenstand (OfBiz) übertragen und die Ergebnisse dargestellt. In Kapitel 5.3 geht es darum, offene Schwachstellen von OfBiz in den Foren und Sicherheitsportalen aufzuspüren. Es folgt die Bewertung der gewonnenen Informationen in Kapitel 5.4. Im Anschluss werden die Sicherheitslücken priorisiert und für den praktischen Test definiert.

5.1 Sichere Softwareentwicklung

Die Quelltext-Qualität kann die Sicherheitsmerkmale des Systems direkt beeinflussen. Die Analyse des Quelltextes ist ein weiterer Bestandteil des passiven Penetrationstests und kann eine Aussage über den Sicherheitsstand des Gesamtsystems liefern.
Allgemein werden Source Code Scanner zur Analyse des Quelltextes (Source Code Scanning) von Anwendungen, speziell Webanwendungen, eingesetzt. Dadurch werden insbesondere die Software-Entwickler bei der Implementierung von Webanwendungen unterstützt.

Eine solche Analyse kann zwar nie sorgfältiges Design und Implementierung, noch ausgiebiges Testen und Debuggen ersetzen, ist jedoch als weiteres Werkzeug bei der Suche nach Fehlern und bekannten Sicherheitslücken sehr hilfreich.

Die Source Code Analyse wird als statischer Test durchgeführt, weil dabei nicht das System parametrisiert getestet wird, sondern der Quelltext formalen Analyse-Methoden unterzogen wird (vgl. [SPLI], S. 93). Source Codes sind formal strukturiert, deshalb ist eine toolbasierte Analyse möglich. Statische Analyse arbeitet nach fest definierten Regeln, d.h. es müssen Regeln existieren, die nach Codefragmenten suchen. Die statische Analyse beschränkt sich nicht auf die Suche eines bestimmen Fehlerfalls, sondern auf eine Menge von Fällen, sodass der zu suchende Fehler nicht bekannt sein muss. Das Ergebnis einer Analyse ist in der Regel eine Reportdatei mit Verweisen auf das potenziell fehlerhafte Codefragment, den Fehlertyp und die möglichen Folgen.

Die Vorteile sind: eine sehr einfache Durchführung, eine Untersuchung bereits in einem frühen Entwicklungsstadium des Quellcodesund eine Abdeckung des gesamten Codebaums.

Das Hauptziel der Quelltextanalyse bei einem bestehenden System ist, die vorhandenen Sicherheitslücken zu finden. Dazu wird der Quelltext systematisch nach vorhandenen Mustern, z.B. ungeeigneten Funktionsaufrufen, Ausgabe systeminterner bzw. schützenswerter Informationen und ungeprüften Zugriffen auf Subsysteme durchsucht.

5.2 Durchführung einer Source Code Analyse

Für die statische Source Code Analyse gibt es einige freie (Open Source) und kommerzielle Tools, die man einsetzen kann. Auf den Internetseiten von OWASP werden einige diese Tools vorgestellt. Es gibt auch zwei OWASP-eigene Tools, die jedoch nicht mehr weiterentwickelt werden.

In dieser Arbeit kommt das aktuelle Analyse-Tool von Google, das CodePro AnalytiX[26], zum Einsatz. Der Grund für die Auswahl ist neben der Aktualität des Werkzeugs sowie der guten Anbindung an Eclipse die einfache Benutzbarkeit und die umfangreichen Aspekte der Analyse, unter denen man die statische Auswertung durchführen kann. Aufgrund des

[26] *Google`s CodePro AnalytiX* ist ein Tool für statische Codeanalyse

Projektumfangs und der umfassenden Analyse benötigt das Tool mehr Arbeitsspeicher, deshalb muss Eclipse mit bis zu 1024 MB zugesichertem Arbeitsspeicher gestartet werden.

```
./eclipse –vmargs –Xms40m –Xmx1024m
```

Wenn man Eclipse mit den Standardwerten für die Analyse startet, bricht das Analyse-Tool bei ungefähr 51% die Arbeit mit einem OutOfMemoryException[27] ab.

Wie oben erwähnt, kann das CodePro AnalytiX in verschiedenen Modi die Qualität des Source Codes analysieren. Es stehen die folgenden Analysemodi zur Verfügung:

- CodePro Core
- CodeProDefault
- Effective Java
- Internal API
- Potential Errors and Refactorings
- Security
- Spelling
- The Element of Java Style

Da der Focus dieser Arbeit auf Sicherheit liegt, ist die Analyse im Modus Security durchgeführt wurden. Nach ca. 35 Minuten hat das CodePro AnalytiX die statische Auswertung komplett durchgeführt und das Ergebnis steht fest.

Es wurden 454 Sicherheitsschwachstellen (Security Violations) mit dem Prädikat „high“ und 7304 mit der Gefährdungsstufe „low“ festgestellt.

Die Abbildung 8 zeigt die Auflistung der Security Violations sortiert nach Gefährdungsstufen, welche durch die statische Source Code Analyse herausgefunden wurde.

[27] *Erschöpfung von allocierten Programmspeicher*

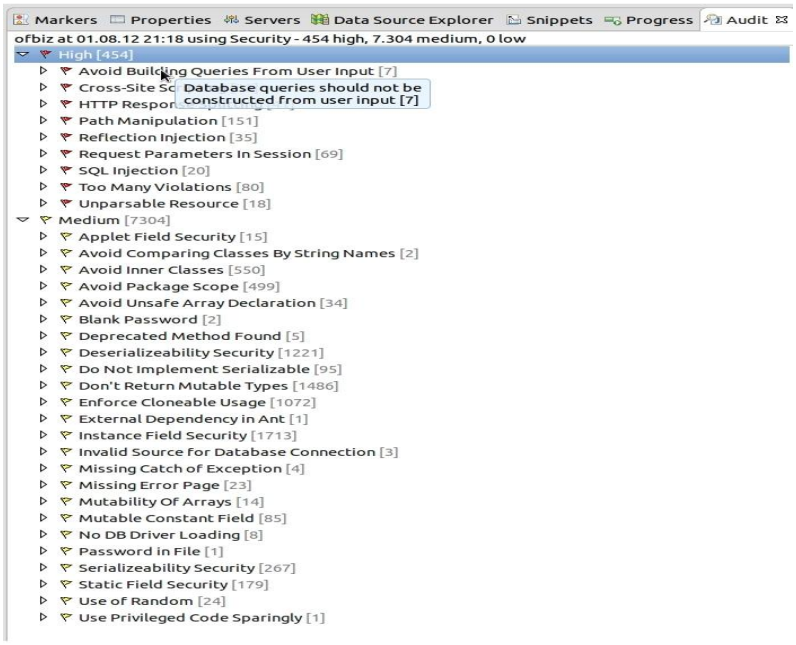

Abbildung 8: Ergebnisse der Statischen Source Code Analyse

Nachfolgend werden die Ergebnisse aus der "high"-level gekennzeichneten Gruppe durchgesprochen. Die jeweilige Schwachstellengruppe wird zu Anfang erläutert. Es wird je nach Möglichkeit ein Praxisbeispiel gebracht. Anschließend wird eine der gefundenen Schwachstellen mit Source Code vorgestellt.

5.2.1 SQL-Injection:

Bei der ersten Gruppe handelt es sich um Injection-Schwachstellen im Allgemeinen. Insbesondere wird aber dort die Technik SQL-Injection betrachtet.
Unter SQL-Injection wird allgemein die Konkatenation von unsicheren Zeichenketten (Input aus Benutzereingaben) zu sicheren Zeichenketten verstanden. (vgl. Kapitel 3.2.1)

Die Auswertung des Codes zeigt, dass in Source Code von OfBiz in sieben Fällen aus den Benutzereingaben direkt SQL-Statements zusammengebaut werden.

```
protectedboolean next() throws SQLException {
```

```
if (rs.next()) returntrue;
        System.err.println("executing page fetch(2)");
rs = stmt.executeQuery(query);
returnrs.next();
}
```

Diese Methode, in der der Code-Ausschnitt vorkommt, wird nur aus den internen Operationen aufgerufen. Die Parameter haben keinerlei Verbindung zu den Endnutzern und werden auch nicht aus den Eingaben von Endnutzern zusammengebaut. Deshalb kann man mit hoher Wahrscheinlichkeit davon ausgehen, dass von dieser Stelle keine Injection zu befürchten ist. Das gleiche gilt auch für die weiteren fünf Fälle.

Ein ernst zunehmender Hinweis kommt aber in der Klasse DatabaseUtil.java in der Zeile 1822 vor. In dieser Zeile wird ein SQL-Statement ausgeführt, welches über Umwege aus Benutzereingaben stammen könnte.

```
try {
            stmt = connection.createStatement();
            stmt.executeUpdate(sqlBuf.toString());
        } catch (SQLException e) {
return"SQL Exception while executing the following:\n" +
sqlBuf.toString() + "\nError was: " + e.toString();
}
```

Die Annahme des Tools beruht eventuell darauf, dass man das Gegenteil (der Input der Methode **nicht** aus Benutzerdaten generiert wird) nicht ganz ausschließen kann. Der Weg, über den die Methode aufgerufen wird, und mit Input versorgt wird, ist zu lang, um das Risiko ganz auszuschließen.

5.2.2 CrossSiteScripting

Bei der zweiten Gruppe handelt es sich um die Cross Site Scripting Schwachstelle, die insbesondere in Webapplikationen so populär ist.
Webseiten können durch Manipulation aus dynamisch erzeugten Inhalten Schadcode in Form von HTML-Tags, oder Javascript-Code enthalten (vgl. Kap 3.2.2)

In der Auswertung von CodePro werden 33 Fälle von XSS gefunden. Der Hintergrund für die Einschätzung des Tools ist wahrscheinlich, dass die Webseiten im Fehlerfall viele Informationen über Art und Weise der Fehler und die fehlende Ressource ausgeben. Diese Informationen können vom Angreifer dazu genutzt werden, Ihre Angriffe gezielt daran anzupassen. Ein Beispiel dafür wird durch den folgenden Code-Schnipsel verdeutlicht:

```
out.println("Thread: " + t.getName() + " [" + t.getId()
+ "] @ " + (g != null ? g.getName() : "[none]") + " : "
+ t.getPriority() + " [" + t.getState().name() + "]");

out.println("--- Alive: " + t.isAlive() + " Daemon: " +
t.isDaemon());
```

Die dritte Gruppe aus den Auswertungsergebnissen ist ebenfalls ein Cross Site Scripting Fall, weil die Eingaben vom Benutzer direkt in der Ausgabeseite ausgegeben bzw. ausgeführt (in Form von URL für Redirect) werden:

```
// set the order in the session for cancelled orders
request.getSession().setAttribute("PAYPAL_ORDER",
orderId);

// redirect to paypal
try {
        response.sendRedirect(redirectString);
    } catch (IOException e) {
        Debug.logError(e, "Problems redirecting to
PayPal", module);
       request.setAttribute("_ERROR_MESSAGE_",
       UtilProperties.getMessage(resourceErr,
       "payPalEvents.problemsConnectingWithPayPal",
       locale));
return"error";
    }
```

5.2.3 Path Manipulation

Bei der vierten Gruppe handelt es sich um Path Manipulation Schwachstellen. Es geht hierbei wieder darum, dass Benutzereingaben ungefiltert in den Programmkontext übernommen werden. Es gibt u. a. zwei Möglichkeiten, diese Schwachstelle auszunutzen:

1) Löschen von Dateien: Wenn die zu löschende Datei aus den Nutzereingaben generiert wurde, dann hat ein Angreifer die Möglichkeit, sämtliche Dateien des Verzeichnisses zu löschen, indem die zu löschende Datei erwartet wird:

```
        String fileName = request.geParameter("report");
        File toBeDelete = new File("/usr/local/reports/"
+ fileName);
        boolean tmpBool;
        if (toBeDelete.exists()) {
                toBeDelete.delete();
                tmpBool = true
        }
```

2) Überschreiben von Konfigurationsdateien: Der Fall tritt ein, wenn ein Programm aus einer Konfigurationsdatei liest, welche Datei es zu öffnen hat und es diese Info dem Angreifer mitteilt. Wenn der Angreifer nun durch die Nutzung anderer Schwachstellen die Möglichkeit hat, die Konfigurationsdatei zu überschreiben, dann kann er im Kontext des laufenden Programms (mit den Ausführungsrechten) jede beliebige andere Datei mit einem bestimmten Muster (z.B. mit der „.txt“ Endung) auf dem Dateisystem öffnen und ausführen.

Im Ergebnis der statischen Analyse werden 144 Fälle von Path Manipulation gefunden. In allen diesen Fällen wird angenommen, dass die Dateinamen (Ressourcen) aus den Eingaben des Nutzers gebildet werden. Diese Annahme geht davon aus, dass das Tracing von Variablen, aus denen die Dateinamen generiert werden, zu lang ist und außerdem nicht mit Sicherheit ausgeschlossen werden kann, dass die Inhalte aus den Nutzereingaben kommen. Deswegen muss davon ausgegangen werden.
Der untere Code-Auszug zeigt einen solchen Fall:

```
privatestatic File makeNewDirectory(File parent) {
File latestDir = null;
boolean newDir = false;
```

```
while (!newDir) {
            latestDir = new File(parent, "" +
System.currentTimeMillis());
if (!latestDir.exists()) {
                latestDir.mkdir();
newDir = true;
            }
        }
return latestDir;
}
```

Der Methodenparameter „parant“ kommt über mehrere Ebenen und der Pfad ist einfach zu lang, um sicher zu sein.

5.3 Aufdecken von offenen Schwachstellen

In diesem Kapitel geht es darum, Informationen über die möglichen Schwachstellen der zu untersuchenden Anwendung herauszufinden. Hierzu werden unterschiedliche Sicherheitsportale und Foren systematisch nach Einträgen über Apache OfBiz durchsucht. Diese Ergebnisse werden ausgewertet und sie fließen in die nächste Phase (Bewertung der Informationen) mit ein.

Das Ergebnis der Recherche nach potenziellen Schwachstellen von Apache OfBiz wird nachfolgend tabellarisch dargestellt. Dabei werden die Ergebnisse chronologisch sortiert:

Fall 1	Cross Site Scripting
Beschreibung	Multiple vulnerabilities have been reported in Apache OfBiz, which can be exploited by malicious people to conduct cross-site scripting attacks and compromise a vulnerable system. 1) Certain unspecified input is not properly

	sanitised within the "getServerError()" function in checkoutProcess.js before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site. 2) Certain input passed as parameter arrays is not properly sanitised before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site. 3) Certain input related to content IDs and map-keys is not properly sanitised before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site. 4) Certain unspecified input passed to the Webslinger component is not properly sanitised before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site. 5) An error in the FlexibleStringExpander class when handling strings with nested scripts can be exploited to execute arbitrary code.
OfBiz Version	10.4.1
Quelle	[OBVul1]
Status	behoben

Fall 2	Cross Site Scripting
Beschreibung	Cross-Site Scripting attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites.

	Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it. This vulnerability can be exploited to force a logged in Administrator to run arbitrary SQL commands or create a new user with Full Privileges . You can find customized XSS PoC payloads here.
OfBiz Version	9.04
Quelle	[OB-Vul-2]
Status	behoben

Die passive Informationsbeschaffung wird anhand der offenen Quellen und der Dokumentationen durchgeführt. In den beiden o. g. Fällen des offenen Schwachstellen-Reports über Apache OfBiz ist die Rede von Cross Site Scripting und Code Execution[28]. Zu dieser Kategorie gehören Schwachstellen, die es Angreifern erlauben, Zugriff auf das System zu erlangen und mit den Rechten des lokalen Benutzers (angemeldeter Benutzer) beliebige Codes auszuführen. Die entdeckten Cross Site Scripting Schwachstellen erlauben dem Angreifer mittels Skripts in Systeme einzudringen und interne Anfragen zu fälschen.

Alle bisher gemeldeten Sicherheitsschwachstellen sollen nach internen Bugreports von OfBiz-Projekt in der aktuellen Version10.4.x behoben sein. Bei den bisher bekannten Schwachstellen handelt es sich um Cross Site Scripting Schwachstellen.

Es sind bislang keine Informationen über SQL-Injection -Schwachstellen oder Ähnliches bekannt geworden. Das liegt u.a. am architektonischen Aufbau von OfBiz. Die unterschiedlichen Schichten der Applikation (Oberfläche, Applikationslogik und Datenbanklogik) werden getrennt. Die Schicht, die mit der Datenbankengine kommuniziert,

[28] nngl. *Code-Ausführung*, durch einen Angreifer auf dem Zielsystem

wird als „EntityEngine“ bezeichnet. Diese Schicht prüft ankommende Daten und versucht, sie in Werte zu konvertieren. Sollte dies zum Beispiel aufgrund eines Manipulationsversuchs nicht gelingen, beendet die „EntityEngine“ die Verarbeitung mit einem Fehler und gibt die Daten erst gar nicht an die Datenbank weiter. Die „EntityEngine“ gilt als Garant gegen Injection-Schwachstellen.

Nach Angaben von OfBiz-Community soll das Framework auch nach Anpassung als sicher einzustufen sein, sofern bei der Entwicklung von neuen Features bzw. bei der Anpassung bestehender Funktionen die Best-Practices eingehalten werden.

5.4 Bewertung der Informationen / Risikoanalyse

In diesem Kapitel soll eine Bewertung der Bedrohung stattfinden. Dazu werden die bisher gewonnen Informationen über das Testobjekt analysiert und bewertet. In diese Bewertung müssen die vereinbarten Ziele, die potenzielle Gefährdung der Systeme und der geschätzte Aufwand für die Evaluierung der Sicherheitsmängel einbezogen werden. Im Anschluss sollen die Gefahren miteinander verglichen werden. Daraus folgt eine Priorisierung der Ergebnisse. Die priorisierte Liste gilt als Grundlage für die aktiven Eindringversuche.

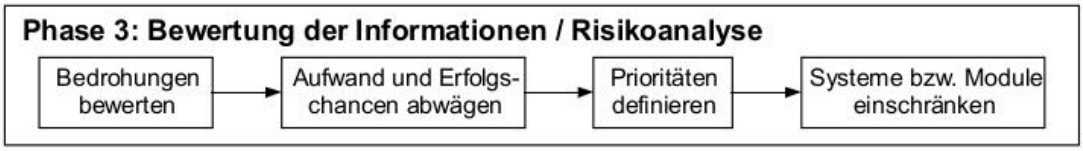

Abbildung 9: Penetrationsphase Bewertung der Information, Quelle BSI

Die Penetrationstests sollen speziell auf der Anwendungseben durchgeführt werden. Das bedeutet, der Fokus liegt bei den Schwachstellen der Anwendung.

E-Commerce Applikationen als eine Unterklasse von webbasierten Applikationen werden meist in Programmier- oder Skript-Sprachen wie Java, PHP, Perl oder Python programmiert. Sie werden auf dem Server ausgeführt und verarbeiten Daten, die in SQL-Datenbanken abgelegt sind (vgl. [Eck06], S. 137).

Ein sehr großer Teil der heutigen Sicherheitsprobleme in E-Commerce Applikationen resultiert aus verwundbaren Webanwendungen. Dabei liegt das Hauptaugenmerk auf den dynamischen Seitengenerierungen.

Die gewonnen Ergebnisse aus den Kapitel 5.2 und 5.3 verdeutlichen, dass die meisten Gefährdungssituationen, die bisher über OfBiz bekannt sind, sich auch in der Top-Ten-Liste von OWASP befinden.

Angreifer können potentiell viele verschiede Angriffswege innerhalb der Applikation verwenden um einen Schaden im System zu verursachen. Wie man aus der Abbildung sehen kann, kann jeder dieser Wege einen potenziellen Angriff darstellen. Eine Kombination aus mehreren Angriffen ist möglich.

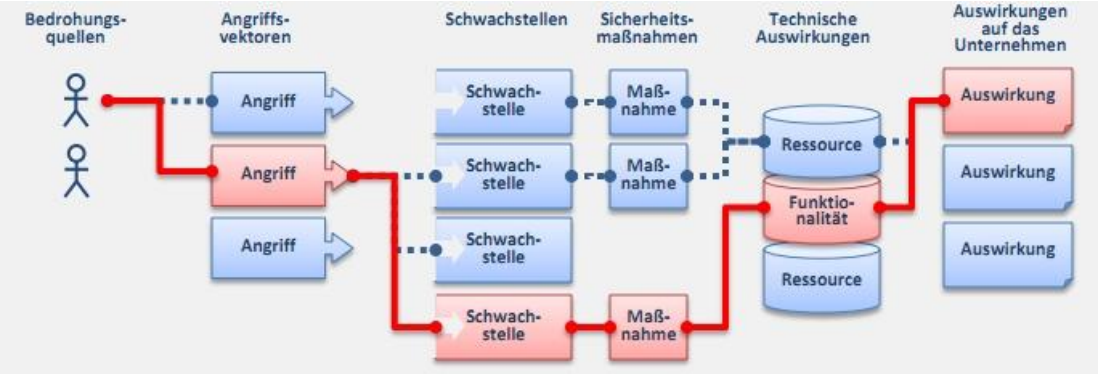

Abbildung 10: unterschiedlichen Bedrohungen in Webanwendungen, Quelle OWASP

Aus einer Umfrage von KPMG [KPMG] aus dem Jahr 2001 unter E-Business-Betreibern über mögliche Sicherheitsgefährdungen sehen die Mehrzahl (45%) der Teilnehmer die größte Gefahr für ihre E-Commerce-Systeme in „Hacker“-Angriffen. Die Zahl der veröffentlichten Sicherheitslücken war im Jahr 2010 weiterhin hoch. Ob der Trend auch 2011 zu erwarten ist, bleibt offen (vgl. [BSI4]). Aus den genannten Gründen wird die Top-Ten-Liste als Anforderung an das Testsystem angenommen und für die Penetrationstests modular aufbereitet. Penetrationstests für alle relevanten Risiken würden den Rahmen dieser Arbeit sprengen, daher wird der praktische Test nur auf einige wenige Testmodule beschränkt.

5.5 Fehlerarten bezogen auf Sicherheitslücken

Im Zusammenhang mit Sicherheitsproblemen bei Webapplikationen werden die folgenden drei Fehlerarten unterschieden.

5.5.1 Designfehler

Designfehler sind Fehler im Grundkonzept, die entweder bei der Definition der Anforderung an die Software, oder bei der Entwicklung des Softwaredesgins, auf dessen Grundlage das Programm entwickelt wird, auftreten. Solche Fehler entstehen entweder aufgrund mangelnder Kenntnis des Fachgebiets oder mangelnder Erfahrung der Softwareentwickler bzw. der Softwarearchitekten. Das Design von Software ist in vielen Fällen historisch gewachsen und durch die Weiterentwicklung fehleranfällig. Vielfach werden Neuentwicklungen ohne Design-Refactoring[29] durchgeführt, die steigende Komplexität der Software führt dann zu Designfehlern.

5.5.2 Implementationsfehler

Implementierungsfehler oder Programmierfehler entstehen, wenn der Entwickler beim Umsetzen von Anforderungen bestimmte Zustände nicht berücksichtigt hat. Oder wenn Situationen in der Systemspezifikation nicht beachtet worden sind. Letzteres entsteht auch durch eine ungenaue Anforderungsanalyse, oder durch mehrdeutige Spezifikation des Systems.

5.5.3 Konfigurationsfehler

Man spricht von Konfigurationsfehlern, wenn Systeme durch falsche oder fehlerhafte Konfiguration Sicherheitslücken aufweisen bzw. fehleranfällig werden.

5.6 Liste der Bedrohungen bzw. Sicherheitslücken

[29] engl. *Restrukturierung* oder Strukturverbesserung von Programmen in der Software-Entwicklung

Nachfolgende Tabelle zeigt eine im Rahmen der Arbeit entstandene Liste der Sicherheitslücken für Webanwendungen. Die Spalte Relevanz bezieht sich auf die Rolle der Sicherheitslücken in Webanwendungen.

Nr	Name der Sicherheitslücke	Fehlerart	Relevanz
1	SQL-Injection	Designfehler und Implementierungs-fehler	Hoch
2	Cross Site Scripting	Implementierungs-fehler	Hoch
3	Broken Authentication and Session Management	Implementierungs-fehler	Hoch
4	Insecure Direct Object References	Implementierungs-fehler	Mittel
5	Cross Site Request Forgery	Implementierungs-fehler	Mittel
6	Security Misconfiguration	Konfigurations-fehler	Hoch
7	Insecure Cryptographic Storage	Implementierungs-fehler und Konfigurations-fehler	Hoch
8	Failure to Restrict URL Access	Designfehler und Implementierungs-fehler	Hoch
9	Insufficient Transport Layer Protection	Designfehler und Konfigurations-fehler	Mittel
10	Invalidated Redirects and Forwards	Implementierungs-fehler	Mittel

Die Liste gilt als Input für die praktischen Exploitversuche. Die praktischen Tests werden nach Möglichkeit so gestaltet, dass sie die o.g. Schachstellen bei der Durchführung berücksichtigen.

6 Penetrationstests – Aktive Exploitversuche

Unter dem Begriff Exploit versteht man in der Informationstechnologie den systematischen Versuch, mit Hilfe von Befehlsfolgen Sicherheitslücken und Fehlfunktionen von Programmen oder ganzen Systemen auszunutzen, um sich Zugang zu Ressourcen zu verschaffen oder Systeme zu beeinträchtigen. Diese Möglichkeit wurde beim Entwurf des Programms nicht berücksichtig.
In den folgenden Tests wird das E-Business-System Apache OfBiz angegriffen. Dabei kann der Angriff auf unterschiedliche Art und Weise durchgeführt werden. Das Ergebnis wird dokumentiert und zeigt den technischen Sicherheitsstand von Apache OfBiz.

6.1 Durchführung von praktischen Penetrationstests

Für die Durchführung der Tests ist die Auswahl auf das Tool Nessus der Firma Tenable Network Securitygefallen. Die Auswahl ist hauptsächlich aufgrund der Aktualität der Software und der umfangreichen Plug-In[30]-Bibliothek getroffen worden. Außerdem ist das Tool als Web-Applikation designt, dadurch ist es sehr einfach in der Bedienung und portabel in der Ausführung.

Nessus ist für nicht-kommerzielle Benutzung in Standardversion kostenfrei zu beziehen. Selbst in der Standardausführung enthält Nessus eine große Fülle von Sicherheits-Plug-Ins.

[30] engl. *Erweiterungsmodule* für Programme

Für die Penetrationstestsder webbasierten Anwendungen bringt Nessus das spezielle Plug-In „Web App Tests“ mit. Plug-Ins bei Nessus bedeuten eine Sammlung gleichartiger Tests, die zusammen oder einzeln ausgeführt werden können.

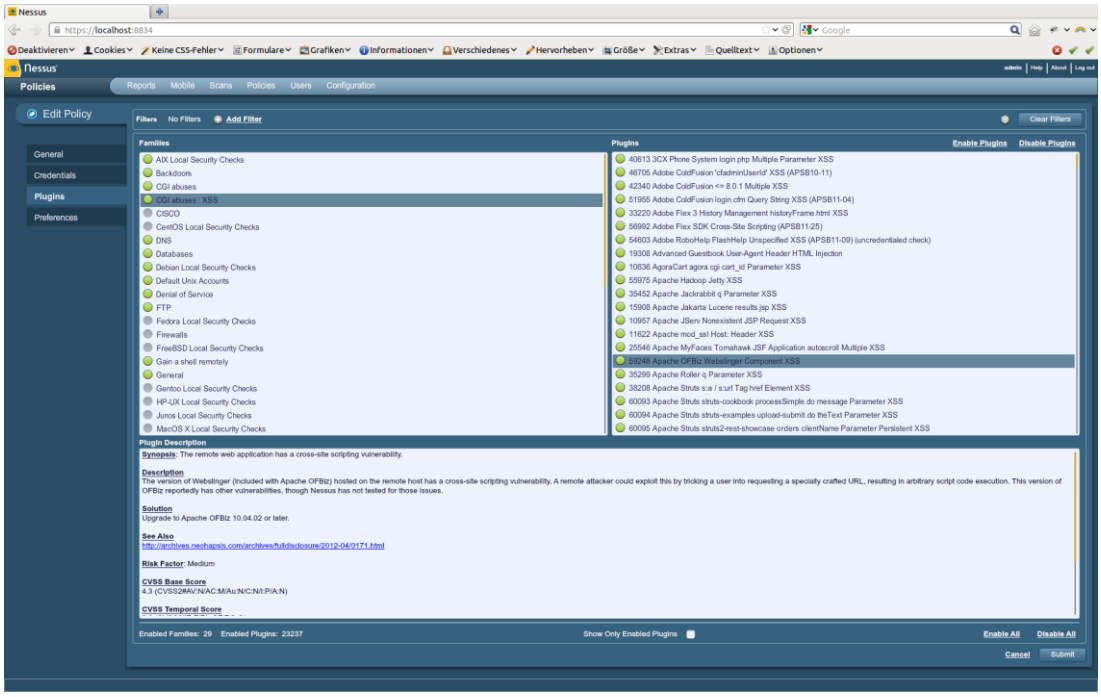

Abbildung 11: Active Exploitversuche mit Nessus

Wie die Abbildung 11 zeigt, besteht das Plug-In „Web App Tests“ aus einer Reihe von einzelnen Tests, die zu einem Testmodul zusammengeführt sind. Einzelne Tests können separat aktiviert oder deaktiviert werden, um den Scanvorgang exakt für den eigenen Bedarf maßzuschneidern.

Es sind neben Cross Site Scripting Tests, Injection-Tests, Remote Shell Tests, Denial of Service Tests, Database Tests etc. auch systemspezifische Tests enthalten, die z.B. Lücken in Betriebssystemen oder in speziellen Softwaremodulen finden können.

Die aktuellen Tests für diese Arbeit laufen auf dem Betriebssystem „Ubuntu“. Deshalb sind außer „Ubuntu“-spezifischen Tests alle anderen spezifischen Tests deaktiviert. Die Abbidung 11 zeigt den Inhalt von „CGI abuses: XSS“[31] Test an. Dabei sind hier alle bekannten Sicherheitslücken auf Anwendungsebene aufgeführt, die in diesem Modul

[31] *ein Testmodule für Cross Site Scripting Schwachstellen*

Testcases verfügbar sind. In dem Feld Plug-In-Description erscheint auch ein Hinweis zur Behebung der Sicherheitslücke; außerdem wird hier eine Risikobewertung angegeben.

Das Ergebnis der praktischen Tests – wie in der Abbildung gezeigt – sieht folgendermaßen aus:

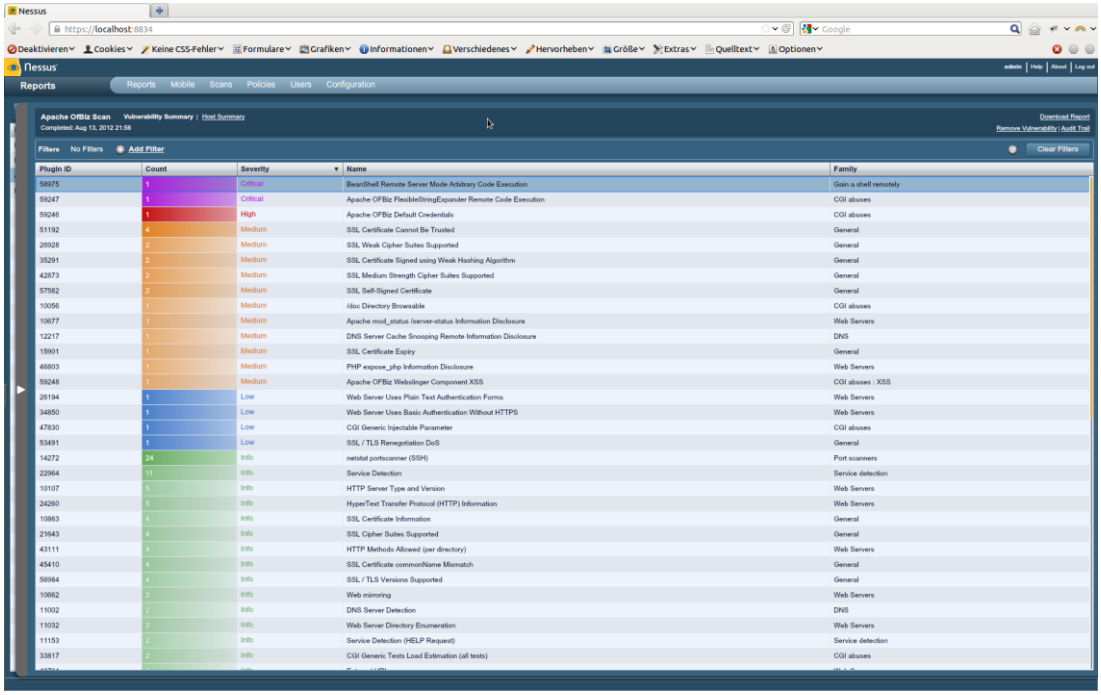

Abbildung 12: Ergebnisse der aktiven Exploitversuche mit Nessus

Die Ergebnisse der gefundenen Schwachstellen des „Web-App“ Tests von Nessus werden im Folgenden tabellarische dargestellt:

	Priorität	Gesamtanzahl der Sicherheitslücken	Anzahl der Sicherheitslücken, die direkt OfBiz betreffen
A	Critical	2	1
B	High	1	1
C	Medium	10	1
D	Low	4	0
E	Info	Viele	1

Die ersten drei Einträge der Tabelle (von Critical bis Medium) werden im Kapitel 6.2 ausführliche beschrieben. Auf die Einträge der Liste mit den Prioritäten Low und Info wird nicht mehr eingegangen.

6.2 Identifizierte OfBiz-Sicherheitslücken

6.2.1 Apache OfBiz FlexibleStringExpander (Critical)

Bei dieser Sicherheitslücke geht es um eine Art Code-Ausführung in der getInstance Methode von OfBiz. Die Klasse, um die es geht, ist: *FlexibleStringExpander.java*
In dieser Klasse kann man Code zur Ausführung bringen.

In dem gezeigten Beispiel wird z.B., die eingesetzte Java-Version in einem http-Post-Request angefragt. Der auszuführende Text wird als Inhalt eines Postparameters getarnt. Der Server beantwortet die Anfrage, in dem er den Parameterinhalt ausführt und die Ausgabe zurückliefert. Damit ist ein Angreifer in der Lage, einen beliebigen Code auszuführen.

```
------------------------------------------------------------
-----------
POST /ecommerce/control/addpromocode/showcart HTTP/1.1
Host: localhost:8080 Accept-Charset: iso-8859-1,utf-
8;q=0.9,*;q=0.1
Accept-Language: en Content-Type: application/x-www-
form-urlencoded Connection: Close
Cookie: OFBiz.Visitor=10083;
JSESSIONID=48E5F606DA7F8F557F86B8B7BF6D2BD8.jvm1
Content-Length: 60
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows
NT 5.1; Trident/4.0)
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, image/png, */*

productPromoCodeId=${bsh:System.getProperty("java.versio
n")}

Zurück gegebener Wert ist:
```

```
1.6.0_26

------------------------------------------------------------
-----------
```

Für diese Schwachstelle existiert eine verbesserte Version derselben Klasse als Patch, den man aber separat installieren muss.

6.2.2 Default Credentials (High)

Die zweite Sicherheitslücke, die durch Penetrationstests gefunden wurde, ist, dass in der installierten Version von OfBiz die Standardkombinationen aus Benutzernamen und Passwort noch gültig sind. Diese Kombination ist allgemein bekannt und sollte vor der Produktivstellung von OfBiz geändert werden. Ansonsten hat ein Angreifer einfaches Spiel, auf dem System Administrator zu werden und alles zu machen, – auch das System zu zerstören – was er will.

```
Logversuch mit den Standard Benutzer:

Aufruf von
https://localhost:8443/ordermgr/control/checkLogin
Benutzer: admin
Passwort: ofbiz
```

6.2.3 Apache OfBiz Webslinger Component XSS (Medium)

Hierbei geht es um eine Cross Site Scripting Schwachstelle in der Webselinger Komponente von OfBiz, fünf SSL-Schwachstellen, eine Schwachstelle bezieht sich auf die offen Erreichbarkeit des doc Verzeichnisses innerhalb von OfBiz-Installtion usw.

An dieser Stelle wird aber nur auf die Cross Site Scripting Schwachstelle eingegangen und die anderen Punkte nicht weiter vertieft.

Es taucht eine XSS-Sicherheitslücke auf:
Benutzereingaben werden ohne Prüfung und ohne sie URL-konform zu decodieren, ausgegeben. Dadurch entstehen Cross Site Scripting Lücken.

Der Aufruf folgender URL:

```
http://localhost:8443/webslinger/><script>alert(document
.cookie)</script>
```

liefert eine Alert-Box, in dem Cookie-Inhalte der aktuellen Seite angezeigt werden.

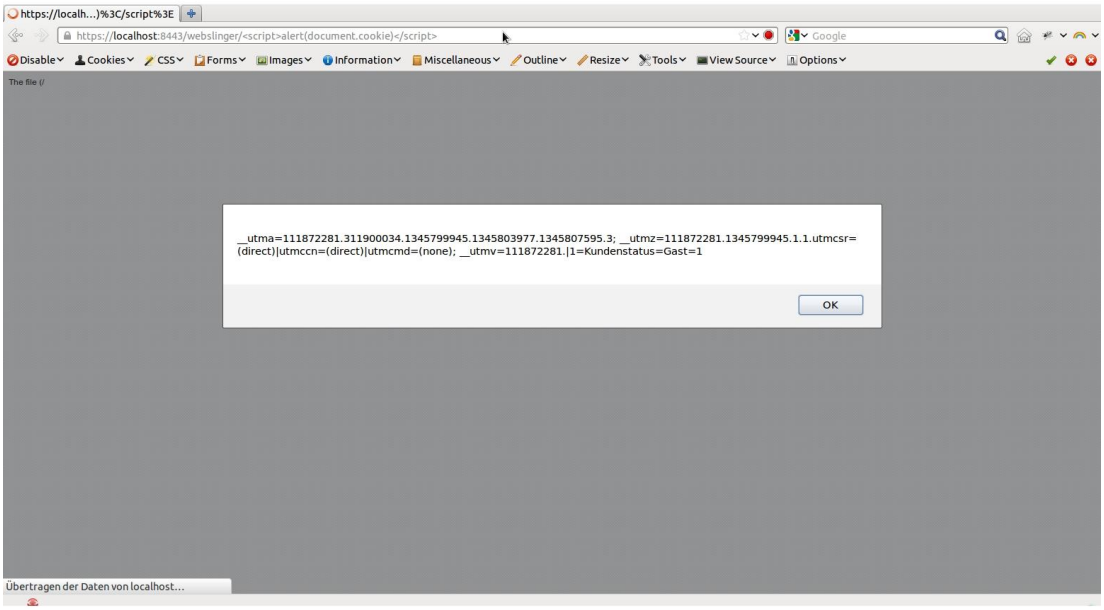

Abbildung 13: OfBiz, XSS-Schwachstelle

Auf diese Weise ist der Angreifer in der Lage, unterschiedliche Inhalte von der Seite zu stehlen. Das Auslesen von sogenannten Secure-Cookies ist durch diese Sicherheitslücke ebenfalls möglich. Secure-Cookies werden benutzt, um Session-Tokens zu speichern. In Session-Tokens werden die aktuellen Anmelde-Informationen über den Nutzer als Cookie gespeichert. Damit wird dem Nutzer eine komfortable Session ermöglicht, ohne dabei jedes Mal das Passwort einzugeben.

Ferner wird durch diese Schwachstelle ermöglicht, dass Benutzereingaben, ohne maskiert zu werden, in Fehlermeldungen auftauchen. Diese Eingaben können wiederum von Angreifern zur Path Manipulation genutzt werden.

Diese Schwachstelle ist in der Version 10.4.01 von OfBiz behoben.

Im Gegensatz zu der Webslinger-Komponente von OfBiz, stellt sich die E-Commerce-Komponente als sehr widerstandsfähig und robust dar und reagiert selbst auf ausgefallene XSS-Fallen als adäquat.

Eine Suche nach:

```
“ onmouseover=“alert(document.cookie)
```

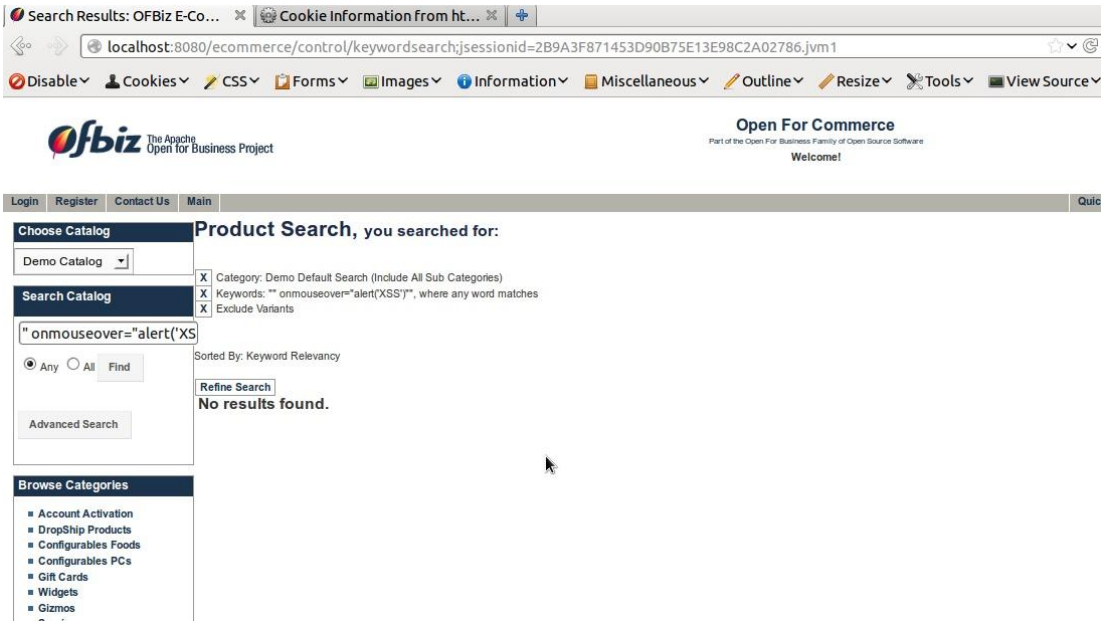

Abbildung 14: OfBiz, XSS-Schwachstelle

7 Fazit

Gerade im Internet ist das Thema der Datensicherheit zu Recht ein besonders sensibles. Wann immer persönliche Nutzerdaten gespeichert werden, stellt sich automatisch die Frage nach dem Schutz vor Missbrauch. Und besonders wenn dabei Geschäfte getätigt werden, muss Datensicherheit höchste Priorität haben. Nicht nur ist die Datensicherheit für den Kunden in höchstem Maße relevant, um die Seriosität einer Seite, z.B. eines Online-Shops, einschätzen zu können, auch für die Betreiber desselben sollten entsprechende Sicherheitsmaßnahmen von absolutem Interesse sein: Geraten die Daten tatsächlich in falsche Hände, geht das im besten Falle einher mit einem entsprechenden Imageschaden. Im schlimmsten Fall können sich Betreiber mit empfindlichen Schadensersatzansprüchen konfrontiert sehen.

Als Referenzsystem ist das Framework Apache OfBiz auf Sicherheit hin geprüft worden.

Zunächst ist der Source Code von OfBiz durch das Tool CodePro AnalytiX analysiert worden. Es ist u. a. 7 Fällen von Injection, 33 Fällen von Cross Site Scripting aufgedeckt worden.

Die Ergebnisse der statischen Source Code Analyse erwecken zunächst das Gefühl, dass Tatsächlich gravierende Fehler vorliegen. Beim nähren Ansehen stellt man aber fest, dass die Entscheidungsgrundlagen des Tools für diese Einschätzung nicht ganz nachvollziehbar sind.

Beispielsweise in den ausgewerteten Fällen von Injection-Schwachstellen geht es nicht um eine direkte Interaktion mit Benutzern (User der Website), sondern um interne Operationen des Systems. Daher sind alle diese Fälle im Bezug auf Injection unbedenklich

Oder im Fall von gefundenen 33 Fällen im Bezug von Cross Site Scripting.
Die vom Tool angemerkten Code-Stellen weisen keine eindeutigen Mustern auf, aus denen man schließen kann, dass z.B. Nutzereingaben uncodiert ausgegeben werden, oder sonstige Hinweise auf Ausführung von Scripts in den Webseiten vorliegen.

Die Recherche nach offenen Schwachstellen im Internet hat zwei Schwachstellen entdeckt. Diese Schwachstellen sind aber alle in der vorliegenden Version behoben.

Die Ergebnisse der aktiven Penetrationstests waren: Code Injection, Cross Site Scripting und Default Credentials. Für Code Injection gibt es einen Hotfix, die Cross Site Scripting Schwachstelle in der Webselinger komponente ist in der Version 10.4.2 behoben. Die Default Credentials sind ein Problem der Konfiguration. Das muss vor einer Produktivsetzung unbedingt geändert werden. Gleiches gilt für die Embedded Datenbank und weitere Standard-Einstellungen. Es gibt ein Guide-Line für Produktivsetzung von OfBiz des gleichnamigen Projekts (vgl. [OBSETUP]). Dieses Dokument sollte unbedingt befolgt werden, bevor man ein OfBiz-System online setzt.

Das Gesamtergebnis kann man als zufriedenstellend bezeichnen. Bis auf kleine Sicherheitsmängel, die auch alle entweder durch direkte Hotfixes[32] oder durch neues Release behoben worden sind, gab es keine gravierenden Sicherheitslücken.

Diese Beobachtung scheint durch den architektonischen Aufbau begründet. OfBiz trennt die Oberfläche, die Applikationslogik und die Datenbanklogik in Schichten. Die Schicht, die mit der Datenbank kommuniziert, wird als EntityEngine bezeichnet. Sie prüft die ankommenden Daten und versucht, sie in Werte zu konvertieren, die mit der jeweiligen Datenbank kompatibel sind. Sollte dies zum Beispiel aufgrund eines Manipulationsversuches (SQL-Injection) nicht gelingen, beendet die EntityEngine die Verarbeitung mit einem Fehler und gibt die Daten gar nicht erst an die Datenbank weiter.

Außerdem werden Felder, die als Passwort deklariert sind, automatisch lediglich als Hash-Wert in OfBiz hinterlegt, was ein Ausspähen quasi unmöglich macht. In der Datenbank selbst können besonders sensible Informationen auch verschlüsselt hinterlegt werden, z. B. Kreditkartendaten.

[32] engl. *Schnelle Update* zur Fehlerkorrektur

Zusätzlich kommen sogenannte “Map Processors” zum Einsatz, die direkt nach dem Absenden eines HTML-Formulars die jeweiligen Eingabefelder validieren. So wird beispielsweise geprüft, ob in Feldern, in denen numerische Werte (0-9) erwartet werden, diese auch tatsächlich stehen. Bei einer Datumsangabe kann darüber hinaus ausgelesen werden, ob die erwartete Syntax eingehalten wurde und die Angaben sich zu entsprechenden Date-Objekten konvertieren lassen. Diese Map Processors kommen im gesamten Backend stringent zum Einsatz und sorgen so für einen flächendeckenden Schutz.

Nachdem die Tests komplett durchgeführt wurden und die Ergebnisse vorliegen, kann man nun die Frage „Gibt es E-Business-Systeme oder E-Commerce-Systeme, die „out of the box“ sicher sind?“ natürlich insgesamt mit „ja“ beantworten. Ja, es gibt sie.

Natürlich darf man nicht vergessen, dass diese Systeme vor der Produktivsetzung entsprechend konfiguriert werden müssen. Andernfalls sind sie nicht sicher, wie es aus den Ergebnissen der Penetrationstests insbesondere unter dem Punkt „Default Credintials (High)**“** deutlich wurde.

Literaturverzeichnis

[BL2000] Andreas Bartelt, Winfried Lamersdorf: Geschäftsmodelle des Electronic Commerce: Modellbildung und Klassifikation, Universität Hamburg 2000

[BSI01] Bundesamt für Sicherheit in der Informationstechnik: Odyssee im Cyberspace? Sicherheit im Internet

[BSI02] Bundesamt für Sicherheit in der Informationstechnik: Durchführungskonzept für Penetrationstests, Bundesamt für Sicherheit in der Informationstechnik

[BSI03] Bundesamt für Sicherheit in der Informationstechnik: Mindestanforderungen zur Informationssicherheit bei eCommerce-Anbietern, Bundesamt für Sicherheit in der Informationstechnik, Bonn 2011

[BSI4] Bundesamt für Sicherheit in der Informationstechnik: Die Lage der IT-Sicherheit in Deutschland 2011, Bonn, im Mai 2011

[CERTXSS] http://www.cert.org/advisories/CA-2000-02.html - abgerufen am 13.06.2012

[CERTMAL] http://www.cert.org/tech_tips/malicious_code_mitigation.html - abgerufen am 13.06.2012

[Eck06] Claudia Eckert, IT-Sicherheit Konzepte-Verfahren-Protokolle, Oldenbourg Wissenschaftsverlag GmbH, 2006

[Kah98] Bernd Kahlbrandt: Software Engineering: Objektorientierte Software-Entwicklung mit der Unified Modeling Language, Springer Verlag 1998

[KPMG] Umfrage zur Wirtschaftskriminalität im eCommerce: KPMG 2001

[LDSH] https://www.datenschutzzentrum.de/systemdatenschutz/meldung/sm92.htm , abgerufen am 24.07.2012

[OWASP] The Open Web Application Security Project, https://www.owasp.org/index.php/Main_Page. abgerufen am 07.08.2012

[OWASPTop10] OWASP, Die häufigsten Sicherheitsrisiken für Webanwendungen, OWASPTop10_DE_Version_1_0.pdf, https://www.owasp.org/index.php/OWASP_Top_Ten_Project - abgerufen am 06.06.2012

[OBSEC] Apache OfBiz Community: http://www.ofbiz.biz/features/enterprise-ready/sicherheit/, abgerufen am 07.08.2012

[OBVul1] Apache OfBiz Cross-Site Scripting and Code Execution Vulnerabilities: http://secunia.com/advisories/48800, abgerufen am 05.08.2012

[OBVul2] Apache OfBiz Sicherheit-Schwachstellen: http://www.bonsai-sec.com/en/research/vulnerabilities/apacheofbiz-multiple-xss-0103.php, abgerufen am 07.08.2012

[OBSETUP] Apache OfBiz Technical Production Setup Guide, https://cwiki.apache.org/OFBTECH/apache-ofbiz-technical-production-setup-guide.html , abgerufen am 17.08.2012

[SPLI] Andreas Spillner, Tilo Linz, Basiswissen Softwaretest, dpunkt.verlag, 2003

[STR03] Claus Strobel, Web-Technologien: in E-Commerce-Systemen, Oldenbourg Wissenschaftsverlag GmbH, 2004

[WebHack] Dafydd Stuttard, Marcus Pinto, The Web Applikation Hackers Handbook Discovering and Exploiting Security Flaws, Wiley Publishing, Inc, 2008

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____