

Hochschule der Medien
Studiengang Computer Science and Media
Nobelstraße 10
70569 Stuttgart

Java Server Faces und Java EE Security

Seminararbeit

vorgelegt von

Daniel Kuhn

dk047@hdm-stuttgart.de

Matrikelnummer: 20486

Christof Strauch

cs134@hdm-stuttgart.de

Matrikelnummer 20492

Dozenten	Jochen Bauer, Michael Gerlinger, Moritz Seltmann
Vorlesung	Praktikum Anwendungssicherheit
Modul	Sicherheit für Medien und E-Commerce Systeme (SMC)
Studiengang	Computer Science and Media (CSM)
Semester	Wintersemester 2008/2009

Inhaltsverzeichnis

Abbildungsverzeichnis.....	iii
Tabellenverzeichnis.....	iii
Abkürzungsverzeichnis.....	iv
Preface.....	v
1 Einleitung.....	1
2 Authentifizierung in Java EE-Anwendungen.....	2
2.1 Sicherheitsanforderungen für Authentifizierung.....	2
2.2 Ziele einer Sicherheitsarchitektur im Java EE Umfeld.....	3
2.3 Sicherheitsmechanismen der Java EE-Spezifikation.....	3
2.4 Zusammenfassung.....	14
3 Sicherheit für JSF- und Java EE-Webapplikationen.....	15
3.1 Auswahl der betrachteten Sicherheitsaspekte.....	15
3.2 Sicherheitsaspekte und Angriffstypen.....	16
3.3 Cross-Site Scripting (XSS).....	17
3.4 Injection Flaws.....	23
3.5 Malicious File Execution.....	27
3.6 Insecure Direct Object References.....	30
4 Zusammenfassung und Fazit.....	32
Anhang.....	v
1 Blacklists potentiell gefährlicher Zeichen für Interpreter.....	v
Literatur- und Quellenverzeichnis.....	vii

Abbildungsverzeichnis

Abbildung 1: Mapping von Rollen zu Benutzern und Gruppen.....	5
Abbildung 2: HTTP Basic Authentifizierung.....	11
Abbildung 3: Form-basierende Authentifizierung.....	12
Abbildung 4: Zertifikat-basierende gegenseitige Authentifizierung.....	13
Abbildung 5: Benutzername- und Passwort-basierende gegenseitige Authentifizierung.....	14
Abbildung 6: Verteilung der Top 10 Sicherheitsrisiken für Webanwendungen (Quelle: [OWA07b, S. 6]).....	15
Abbildung 7: JavaScript-Code in einer Datenbanktabelle.....	21
Abbildung 8: Entity-encodierte Ausgabe.....	21

Tabellenverzeichnis

Tabelle 1: Schwachstellen in Webapplikationen und mögliche Angriffstypen (nach [OWA07b, S. 6]).....	16
Tabelle 2: Standardvalidatoren der JSF Core Tag Library.....	19
Tabelle 3: Blacklist potentiell gefährlicher Zeichen für SQL-Interpreter (nach [BSI06, S. 21]).....	v
Tabelle 4: Blacklist potentiell gefährlicher Zeichen in Systemaufrufen (nach [BSI06, S. 21f]).....	vi
Tabelle 5: Blacklist potentiell gefährlicher Zeichen für Verzeichnisdienst-Interpreter (nach [BSI06, S. 22])...vi	
Tabelle 6: Blacklist potentiell gefährlicher Zeichen in XML-Dokumenten [OWAGuide, S. 183].....	vi
Tabelle 7: Blacklist weiterer potentiell gefährlicher Zeichen für diverse Interpreter (nach [BSI06, S. 22]).....	vi

Abkürzungsverzeichnis

API	Application Programming Interface	JNDI	Java Naming and Directory Interface
AST	Abstract Syntax Tree	JPA	Java Persistence API
BSI	Bundesamt für Sicherheit in der Informationstechnik	JPAQL	JPA Query Language
CA	Certificate Authority	JSF	Java Server Faces
CORBA	Common Object Request Broker Architecture	JSP	Java Server Pages
CSRF	Cross Site Request Forgery	JSTL	JavaServerPages Standard Tag Library
(L)DAP	(Lightweight) Directory Access Protocol	LAN	Local Area Network
DBMS	Database Management System	MIME	Multipurpose Internet Mail Extensions
DOM	Document Object Model	ORB	Object Request Broker
DTD	Document Type Definition	ORM	Object-relational Mapping
EJB	Enterprise Java Bean	OWASP	Open Web Application Security Project
ERP	Enterprise Resource Planning	RMI	Remote Method Invocation
(X)HTML	(Extensible) Hypertext Markup Language	SHA	Secure Hash Algorithm
HTTP	Hypertext Transfer Protocol	SJSAS	Sun Java System Application Server
HTTPS	Hypertext Transfer Protocol Secure	SQL	Structured Query Language
HQL	Hibernate Query Language	SSL	Secure Socket Layer
IDL	Interface Definition Language	TLS	Transport Layer Security
IIOB	Internet Inter-ORB Protocol	URI	Uniform Resource Identifier
IPSec	Internet Protocol Security	URL	Uniform Resource Locator
JAAS	Java Authentication and Authorization Service	UI	User Interface
JAF	Java Activation Framework	VLAN	Virtual LAN
Java EE	Java Enterprise Edition	VPN	Virtual Private Network
Java SE	Java Standard Edition	XML	Extensible Markup Language
JCE	Java Cryptography Extension	XPath	XML Path Language
JDBC	Java Database Connectivity	XSL	Extensible Stylesheet Language
JMS	Java Messaging Service	XSLT	XSL Transformation
		XSS	Cross Site Scripting

Preface

Sicherheit ist ein sich schnell veränderndes Umfeld – vergleichbar mit einem Katz- und Mausspiel – zwischen Programmierer und Angreifer. Täglich werden neue Sicherheitslücken in den verschiedensten Anwendungen, Plattformen und Betriebssystemen entdeckt. Diese Schwachstellen werden auf kurz oder lang vom Hersteller geschlossen. Um eine größtmögliche Sicherheit zu gewährleisten, sind alle Systeme und Anwendungen von den betroffenen Firmen und Benutzern mit diesen Sicherheitspatches zu versorgen. Andernfalls setzen sie sich diversen Gefahren wie der Einsicht schützenswerter Daten, der Daten- und Infrastrukturmanipulation bis hin zu kompletten Systemkompromittierungen aus.

Sicherheit erstreckt sich über alle Schichten des ISO/OSI-Modells. Von der physikalischen bis hin zur Anwendungsschicht sind vielfältige Aspekte der Sicherheit zu gewährleisten: Wie wird Datenübertragung gesichert und wie können Kommunikationspartner authentifiziert werden? Wie werden Man-in-the-middle-Attacken verhindert? Wie werden also Vertraulichkeit, Integrität, Authentizität und Nicht-Abstreitbarkeit erzielt?

Zudem spielt die gesamte IT-Infrastruktur des Servers oder des Clients eine wichtige Rolle im Sicherheitsprozess, denn nur wenn diese gegenüber Angriffen geschützt sind, bleiben sensitive Daten auch geheim. So sind neben sicherer Datenübertragung auch die Endpunkte der Kommunikation abzusichern. Für Angriffe sind häufig nicht einmal Sicherheitslücken in Betriebssystemen, Plattformen und Anwendungen selbst notwendig: durch fehlerhafte Konfiguration, z.B. durch Vergabe von Root-Rechten für Serverprozesse, können Angreifer möglicherweise mittels legitimer Applikationsfunktionen etwa auf Systemressourcen und eventuell andere Server oder Clients im Netzwerk zugreifen, Daten extrahieren und manipulieren.

Sicherheit ist also keine Anforderung, die sich auf einzelne Komponenten und Schichten beschränkt, sondern die system- und netzwerkweit für alle interagierenden Komponenten zu erzielen ist. Je besser jede einzelne Komponente abgesichert wird, desto sicherer ist demnach auch ein verteiltes Anwendungssystem als Ganzes.

In dieser Arbeit liegt der Schwerpunkt auf der Anwendungssicherheit im Umfeld der JSF und Java EE Technologien. Es werden Konzepte und Technologie vorgestellt, mittels derer Sicherheit für Webapplikationen gegenüber diversen typischen Schwachstellen und Verwundbarkeiten erzielt werden kann.

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit Sicherheitsaspekten Java Server Faces (JSF) sowie der Java EE-Technologien. Dazu werden typische Sicherheitsprobleme in Web-Anwendungen knapp vorgestellt und Handlungsempfehlungen gegeben, wie diesen durch Konzepte und Mittel der JSF- und Java EE-Technologie begegnet werden kann.

Im zweiten Kapitel wird zunächst vorgestellt, welche Konzepte die Java EE-Spezifikation bezüglich der Authentifizierung, bestehend aus Authentisierung und Autorisierung, bietet, um Sicherheitsrollen zu definieren, innerhalb der (Web-)Anwendungslogik zu prüfen und mit den in der Betriebsumgebung der Web-Applikation definierten Benutzern und Gruppen zu integrieren.

Im dritten Kapitel der Arbeit werden die häufigsten Schwachstellen und Verwundbarkeiten von Webanwendungen sowie Maßnahmen zu deren Behandlung diskutiert. Die Auswahl der Sicherheitsaspekte erfolgt nach den durch das Open Web Application Security Project (OWASP) zusammengestellten Top 10 Sicherheitsproblemen in Web-Anwendungen (vgl. [OWA07a] und mit Fokus Java EE [OWA07b]). Für die häufigsten Schwachstellen und Verwundbarkeiten von Webanwendungen werden anhand der OWASP Top 10, des OWASP Guides ([OWAGuide]) sowie des vom BSI zusammengestellten Maßnahmenkataloges „Sicherheit von Webanwendungen“ ([BSI06]) und weiteren Quellen Maßnahmen zur Absicherung gegen die jeweilige Schwachstelle vorgestellt.

2 Authentifizierung in Java EE-Anwendungen

Autor: Daniel Kuhn

Authentifizierung und Session-Management spielen eine wichtige Rolle bei der Absicherung von Web-Anwendungen. Sicherheitslücken bedeuten in diesen Punkten meist, dass Benutzerdaten oder Sessions während ihres Lebenszyklus nicht ausreichend geschützt werden und Angreifer sich dadurch Zugang zu weitergehenden Berechtigungen bis hin zu administrativen Accounts verschaffen können. Vielfach treten diese Sicherheitslücken nicht allein beim Anmeldeprozess auf. Häufig sind auch kleine Features wie „Remember-Me“, Sicherheitsfragen bei verlorenen Passwörtern und das Ändern des Benutzeraccounts sowie Logout-Funktionen und Passwort-Management kritische Punkte einer Webanwendung.

Ziel von Authentifizierung im Allgemeinen ist also ein sicherer Anmeldeprozess an der Anwendung durch eine sichere Kommunikation, durch Geheimhaltung von der angemeldeten Identität und durch eine sichere Speicherung von Benutzeraccounts in der Anwendung selbst.

2.1 Sicherheitsanforderungen für Authentifizierung

Grundlegende Überlegungen, die bei der Planung von Java EE Web-Anwendungen und deren Sicherung bezüglich des Authentifizierungsprozesses zu berücksichtigen sind, werden in einer Empfehlung des OWASP zur Schwachstelle „Broken Authentication and Session Management“ genannt (vgl. [OWA07b, S. 27f]). Einige wichtige Maßnahmen zur Verhinderung von Sicherheitslücken in der Authentifizierung sind:

- Prüfung und Logging der Authentifizierung und Autorisierung: die Anwendung sollte zu jeder Zeit wissen, welche Benutzer eingeloggt sind, wann und von wo Benutzer eingeloggt wurden, welche Transaktionen der Benutzer getätigt und welche Daten er angesehen hat.
- Benutzung des in JEE 5 eingebauten Sessionmanagement-Mechanismus: niemals sollte ein eigenes Sessionmanagement benutzt werden, da dies potentiell unsicher ist.
- Abweisen von neuen, voreingestellten und ungültigen Sessions, die über eine URL oder durch einen Request aufgerufen werden, um „Session Fixation“-Angriffen entgegen zu wirken.
- Verwendung von nur einem (sicherem) Authentifizierungsmechanismus
- Definition von Passwortrichtlinien, so dass Benutzer keine unsichere Passwörter (z.B. kurze Passwörter, Passwörter nur aus Zahlen bestehend ...) benutzen können.
- Definition von Authentifizierungsrichtlinien, sodass User-Accounts nach mehrmalig falscher Passworteingabe gesperrt werden.
- Absicherung des Login-Prozesses durch eine verschlüsselte Verbindung: keine Authentifizierung über eine potentiell unsichere Verbindung erlauben.

Weitere Maßnahmen, die sich über alle Bereiche der Authentifizierung (Konfiguration, Validierung, Übertragung, Management) bezieht, gibt das BSI in einem Maßnahmenkatalog zur „Sicherheit von Webanwendung – Maßnahmenkatalog und Best Practices“ (vgl. [BSI06, S. 40ff, S. 68ff u.a.]).

2.2 Ziele einer Sicherheitsarchitektur im Java EE Umfeld

Im Java Umfeld, aber auch in anderen Anwendungen, gibt es verschiedene Ziele, die bei der Planung einer Sicherheitsarchitektur laut der Java EE-Spezifikation zu beachten sind (vgl. [Sun06b, S. 29f]).

- **Portabilität - Write once, run everywhere:** Eine Anwendung soll von einem Server auf einen anderen mit minimalem Aufwand migriert werden können.
- **Transparenz:** Die Bereitsteller von Applikationskomponenten müssen nichts über Sicherheitsaspekte wissen. Diese sollen automatisch angewendet werden, wenn sie benötigt werden.
- **Isolation:** Authentifizierung und Zugriffskontrolle sollen nach Anweisungen des Deployers durchgeführt werden und vom Systemadministrator verwaltet werden können, nicht durch manuelle Codeänderungen des Entwicklers.
- **Erweiterbarkeit:** Die Benutzung von Plattformdiensten in Anwendungen, darf die Portabilität nicht verletzen. Die Spezifikation stellt eine API zum Zugriff auf Sicherheitsinformationen bereit. Wird diese verwendet, dann ist eine Portabilität der Komponente gewährleistet.
- **Flexibilität:** Die in der Spezifikation vorgestellten Sicherheitsmechanismen und -deklarationen sollen keine spezielle Richtlinien auferlegen, sondern die Implementierung von Sicherheitsrichtlinien für die spezifische Java EE-Installation oder -Applikation erleichtern und fördern.
- **Abstraktion:** Sicherheitsanforderungen können im Deployment Descriptor spezifiziert werden. Diese bilden logische Sicherheitsrollen und Zugriffsanforderungen auf umgebungsspezifische Rollen, Benutzer und Richtlinien ab. Damit können Sicherheitseigenschaften in einer konsistenten Art und Weise zur Entwicklungs- und Betriebsumgebung angegeben werden. Der Deployment Descriptor legt fest, welche Sicherheitseigenschaften modifizierbar sind und welche nicht.
- **Unabhängigkeit:** Der Deployment Deskriptor kann als eine Schnittstelle zwischen Web-Anwendungen und Web-Container angesehen werden. Er definiert unter anderem ein Sicherheitsverhalten, das mit einer Vielzahl populärer Sicherheitstechnologien implementierbar sein muss.
- **Kompatibilitätstests:** Die Java EE-Sicherheitsarchitektur muss so gestaltet sein, dass für eine Implementierung eindeutig prüfbar ist, ob sie kompatibel dazu ist oder nicht.
- **Sichere Interoperabilität:** Anwendungskomponenten in einem Java EE-Produkt müssen auf Dienste eines anderen Java EE-Produkts zugreifen können, unabhängig davon, ob die zugegriffene Komponente gleichen oder anderen Sicherheitsrichtlinien genügt und ob es sich dabei um eine Web- oder EJB-Komponente handelt.

2.3 Sicherheitsmechanismen der Java EE-Spezifikation

Um Java EE Anwendungen sicher zu gestalten, gibt es nach [Sun07 S. 841 ff] verschiedene Möglichkeiten. Ein häufig vorzufindender Ansatz ist wohl eine programmatische Herangehensweise: viele Entwickler planen und entwickeln bei neuen Web-Anwendungen ihr eigenes Sicherheitssystem, zugeschnitten für ihr Umfeld und stellen damit die Authentifizierung und Autorisierung der Benutzer und deren Rechte in der

Anwendung sicher. Neben einem programmatischen Ansatz kann jedoch auch auf die durch die Java EE Spezifikation definierten Verfahrensweisen zurückgegriffen werden. Java EE unterstützt den Entwickler durch eigene Sicherheitskonzepte bei der Erreichung der Ziele von Autorisierung und Authentisierung und erlaubt die Definition von Benutzerrollen, Zugriffskontrollen und Authentifizierungen innerhalb der Web-Applikation. Diese Rollen können dann vom Entwickler an den entscheidenden Stellen benutzt werden, um Zugriffskontrollen zu realisieren. Dieser Ansatz entspricht den Handlungsempfehlungen der OWASP aus Abschnitt 2.1 . Neben dem Rollen- und Rechtekonzept lassen sich auch Sicherheitsanforderungen an die Kommunikation, die Authentifizierungsmethode und die Bindung an URLs definieren, welche automatisch von der Web-Anwendung und dem Web-Container durchgesetzt werden.

Die Definition dieser Sicherheitskonzepte kann neben dem programmatischen Ansatz grundsätzlich auf verschiedene Arten erfolgen: Durch „Metadata“-Annotationen (Java Annotations) im Java-Quellcode, durch Beschreibung der Richtlinien im Deployment-Deskriptor der Web-Anwendung oder der Kombination beider. Anzumerken ist jedoch, dass die im Deployment Deskriptor definierten Sicherheitsanforderungen eine höhere Priorität haben und im Konfliktfall vom Container gegenüber den Annotations bevorzugt werden.

2.3.1 Terminologie

Bevor der Sicherheitsmechanismus genauer diskutiert werden kann, muss zuerst die Terminologie geklärt werden. Sicherheitsrollen gliedern sich nach ([Sun07, S. 813ff]) in diesem Konzept in Umgebungen (realms), Benutzer (users), Gruppen (groups) und Rollen (roles) auf.

Realm: Ein Bereich ist eine vollständige Datenbank von gültigen Benutzern und Gruppen auf dem Applikationsserver oder in einer Webapplikation, die von einer gemeinsamen Authentifizierungsinstanz validiert wurden.

User: Ein Benutzer ist ein Individuum (oder eine Programm), das im Applikationsserver definiert wurde. In der Regel werden für einen Benutzer Authentifizierungsdaten bestehend aus Benutzername und Passwort verwendet. Ein Benutzer kann einer bestimmten Gruppe (des Servers) angehören und kann in verschiedenen Webapplikation in verschiedenen Rollen angemeldet sein.

Group: Eine Gruppe ist eine Menge von Benutzern, die meist eine Gemeinsamkeit haben. Gruppen sind wie einzelne Benutzer in der Betriebsumgebung von Java EE-Anwendungen definiert (im Applikationsserver selbst oder von diesem aus einem externen System bezogen).

Role: Eine Rolle ist ein abstrakter Name für eine für ein Recht, eine bestimmte Ressource (einer Website) zuzugreifen. Rollen werden für jede einzelne Webapplikation definiert.

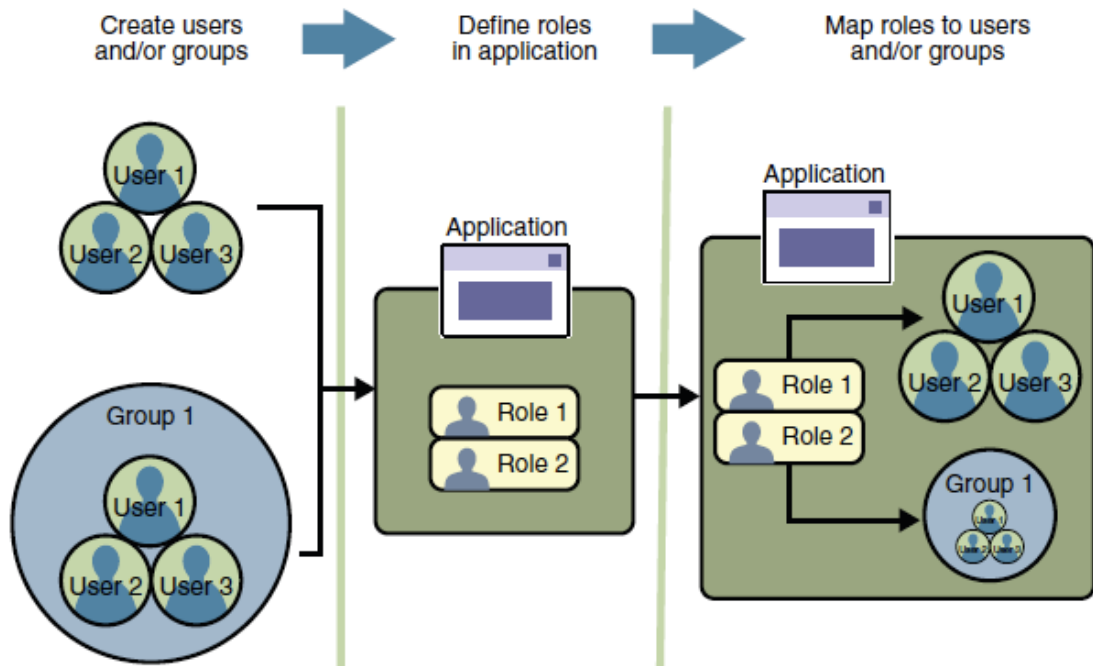


Abbildung 1: Mapping von Rollen zu Benutzern und Gruppen

2.3.2 Definieren von Sicherheitsrollen

Sicherheitsrollen können, wie bereits beschrieben, auf zwei verschiedene Arten in JEE 5 definiert werden, mittels Annotationen im Code und durch die Konfiguration im Web bzw. EJB Deployment Descriptor (`web.xml` bzw. `ejb-jar.xml`).

2.3.2.1 Sicherheitsrollen via Annotations

Für eine Servlet Klasse, eine Java-Bean oder eine Methode können mittels Annotations-Rollen zugewiesen werden, indem die Annotation `@DeclareRoles(„Rollenname“)` vor deren Deklaration gestellt wird. Durch diese Deklaration wird an die Servlet Klasse, Bean oder Methode die angegebene Rolle „geheftet“ und ist damit für diese Ressource definiert.

```
@DeclareRoles („employee“)
public class showNewsServlet {
    // ...
}
```

2.3.2.2 Sicherheitsrollen via Application Deployment Deskriptor

Diese Deklaration über Annotations ist identisch zur folgenden Deklaration im Deployment Deskriptors. In `<security-role>` lassen sich verschiedene Rollennamen deklarieren, die in der gesamten Anwendung verfügbar sein sollen. Diese Rollen werden über den Security Constraint an eine Ressource (z.B. eine Bean, ein Servlet / eine JSP) gebunden. Im Gegensatz zu den Annotations lassen sich an dieser Stelle auch die

HTTP-Zugriffsmethoden (PUT, GET, ...) spezifizieren, über welche diese spezielle Ressource zugänglich sein soll.

```
<security-role>
  <role-name>employee</role-name>
</security-role>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/intranet/showNews.jsp</url-pattern>
    <http-method>PUT</http-method>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>role1</role-name>
    <role-name>employee</role-name>
  </auth-constrain>
</security-constraint>
```

2.3.2.3 Manuellen Rollenprüfung

Werden Rollen im Deployment Descriptor oder via Anntoations für eine Klasse oder Methode definiert, kann in dieser Ressource auf die deklarierte Rolle geprüft werden. Diese Prüfung kann automatisch (siehe 2.3.2.4) oder manuell erfolgen.

Eine manuelle Prüfung kann mittels der Methoden: `isUserInRole(<rollenname>)`, `getRemoteUser()` und `getUserPrincipal()` erfolgen. Die `isUserInRole(<rollenname>)` Methode prüft, ob der Benutzer, der die Ressource aufruft, in der angegebenen Rollen angemeldet ist und gibt `true` oder `false` zurück. Damit kann programmatisch auf die Rollenzugehörigkeit geprüft werden.

```
@DeclareRoles("employee")
@Stateless public class EmployeeBean implements Employee {
  @Resource SessionContext ctx;
  public void updateEmployeeInfo(EmplInfo info) {
    if (!ctx.isUserInRole("employee")) {
      throw new SecurityException(...);
    }else{
      ...
    }
  }
  ...
}
```

Die Methoden `getRemoteUser()` und `getUserPrincipal()` geben beide ein Benutzerobjekt zurück, wenn der Aufrufer angemeldet ist. Ist der Aufrufer nicht an der Anmeldung authentifiziert, wird

null zurückgeliefert. Anhand des Rückgabewertes kann somit geprüft werden, ob der Benutzer angemeldet ist, und anhand der Benutzerdaten weitere Entscheidungen getroffen oder Ausgaben durchgeführt werden.

```
@DeclareRoles("employee")
@Stateless public class EmployeeBean implements Employee {
    @Resource SessionContext ctx;
    public void updateEmployeeInfo(EmplInfo info) {
        if (!tx.getUserPrincipal() != null) {
            // display principal
        }else{
            throw new SecurityException(...);
        }
    }
    ...
}
```

2.3.2.4 Automatische Rollenprüfung

Neben der manuellen Rollenprüfung bietet Java EE die Möglichkeit diese Prüfung automatisiert handzuhaben. Wiederum können die Deklarationen über Annotations oder den Deployment Descriptor getätigt werden. In beiden Fällen können für einzelne Methoden oder für ganze Klassen die Zugriffsrechte spezifiziert werden.

Über die Annotations `@RolesAllowed(<list of roles>)`, `@PermitAll` und `@DenyAll` können für Ressourcen explizit alle Rollen erlaubt oder alle Rollen ausgeschlossen werden.

```
@RolesAllowed("admin")
public class SomeClass {
    public void aMethod () {...}
    public void bMethod () {...}
    ...
}
```

```
@Stateless public class MyBean implements A extends SomeClass {

    @RolesAllowed("HR")
    public void aMethod () {...}

    public void cMethod () {...}
    ...
}
```

Die Deklaration der Zugriffsrechte im Deployment Descriptor erfolgt analog zu den Annotations. Es gibt drei Arten der Auszeichnung: das explizite Binden von Rollen an eine Ressource oder Methode, das Binden einer Ressource/Methode an alle Rollen und das Verbieten von Rollen für eine Ressource/Methode. Dazu

führt der Deployment Descriptor eine „Exclude-List“. Die Deklaration dieser ist jedoch weder in der Spezifikation ([Sun06a]) noch im JavaEE Tutorial ([Sun07]) vermerkt.

```

<method-permission>
  <role-name>employee</role-name>
  <method>
    <ejb-name>EJB_NAME1</ejb-name>
    <method-name>METHOD1/*</method-name>
    <method-params>
      <method-param>PARAM1</method-param>
      <method-param>PARAM2</method-param>
    </method-params>
  </method>
  <method>
    <ejb-name>EJB_NAME2</ejb-name>
    <method-name>METHOD2/*</method-name>
    <method-params>
      <method-param>PARAM1</method-param>
      <method-param>PARAM2</method-param>
    </method-params>
  </method>
</method-permission>

```

2.3.2.5 Annotations vs. Deployment-Deskriptor

Durch die Deklaration mittels Annotations lassen sich Klassen/oder Methoden schnell und einfach an Sicherheitsrollen binden. Allerdings sind die Konfigurationsmöglichkeiten im Vergleich zu einer Deklaration im Deployment Deskriptor beschränkt. Über den Deployment Deskriptor lassen sich zusätzlich zu den Rollen und der Referenz zu einer Ressource die HTTP-Methoden einschränken und Sicherheitsanforderungen für die Datenübertragung definieren, welche im Kapitel 2.3.3 näher erläutert werden. Insbesondere die Beschreibung der Sicherheitsanforderungen und die Einschränkung der HTTP-Methoden machen eine Konfiguration über den Deployment Deskriptor zum Mittel der Wahl. Bei der Rollendeklaration können beide Vorgehensweisen genutzt werden. Eine Vorgehensweise mit Annotations ist jedoch, je nach Anwendungszweck, einfacher und leichter für den Entwickler.

Beide Methoden schließen sich daher einander nicht zwingend aus. Zwar haben die Deklarationen im Deployment Descriptor eine höhere Priorität, jedoch lassen sich weiterhin Sicherheitsrollen über Annotations definieren. Damit Administratoren jedoch die Annotations überschreiben können, gibt es im Deployment Descriptor die Option „metadata-complete“. Dieses Element des Deployment Descriptor der Web-Anwendung gibt an, ob neben dem Descriptor auch Annotations zugelassen werden, um so zusätzliche Deployment Informationen beizusteuern.

Die Konfiguration des Sicherheitssystems durch den Web-Anwendung Deployment Descriptors erfüllt damit die in Kapitel 2.2 beschriebenen Sicherheitsanforderungen: Transparenz, Isolation, Erweiterbarkeit, Abstraktion, Flexibilität und Unabhängigkeit. Eine Portabilität durch beide Vorgehensweisen – Annotations und Deployment Descriptor – gewährleistet.

2.3.3 Anforderungen an den Transport

Die OWASP nennt in ihren „Top ten most critical application security vulnerabilities for java enterprise applications“ ([OWA07a]) im Kapitel „Broken Authentication and Session-Management“ unter anderem eine Handlungsempfehlung zur Absicherung des Login-Prozesses durch eine verschlüsselte Verbindung. Demnach soll keine Authentifizierung über eine potentiell unsichere Verbindung erlaubt werden. Im Kapitel „A9 Insecure Communications“ wird dieser Punkt nochmals als eine Verwundbarkeit von Anwendungen detailliert behandelt. Viele Anwendungen übertragen demnach den Datenverkehr, auch bei sensiblen Daten, nicht auf eine sichere Art und Weise. Insbesondere schützenswerte Daten, wie medizinische Informationen, Kreditkartenzahlungen und Zahlungsinformationen im Allgemeinen, sollten jedoch immer verschlüsselt übertragen werden, um das Abhören der Daten zu verhindern. Angriffe auf den Datenverkehr werden als „Sniffing“ bezeichnet und können mit relativ einfachen Methoden und Programmen (wie z.B. Wireshark [WirWir]) durchgeführt werden. Zur Verschlüsselung des Datenverkehrs im Internet wird in der Regel SSL/TLS verwendet, um den Datenverkehr bei Benutzerauthentifizierungen, Einkäufen oder Transaktionen mit hohen Sicherheitsanforderungen generell zu schützen.

In Java EE 5 existieren automatisierte Methoden, um Sicherheitsanforderungen „on-demand“ für sicherheitskritische Bereiche durchzusetzen. Wie die Rollen, das URL-Muster und die HTTP-Methoden können auch die Sicherheitsanforderungen im Deployment Descriptor der Web-Anwendung definiert werden. Dazu wird im `security-constraint` neben der `web-resource-collection` und der `auth-collection` ein `user-data-constraint` angelegt. Über diese Konfiguration kann für ein bestimmtes Verzeichnis oder eine bestimmte Datei, die in der `web-resource-collection` definiert wurde, eine Sicherheitsanforderungen deklariert werden. Wird `CONFIDENTIAL` oder `INTEGRAL` im `user-data-constraint` als `transport-guarantee` angegeben, so erfolgt die Datenübertragung verschlüsselt bzw. integritätsgeschützt über HTTPS. Ist kein `user-data-constraint` definiert, oder wird `NONE` angegeben, so erfolgt die Übertragung ungesichert über HTTP.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/intranet/showNews.jsp</url-pattern>
    <http-method>PUT</http-method>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>role1</role-name>
    <role-name>employee</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

2.3.4 Authentifizierungsmechanismen

Im Abschnitt 2.3.1 wurde die Terminologie von Realms, Gruppen und Benutzer und deren Herkunft kurz erläutert. Benutzer und Gruppen werden in der Regel in einer Datenbank oder einem LDAP-Verzeichnis gespeichert. Rollen werden über Annotations oder durch den Web Deployment Descriptor definiert.

Beim Authentifizierungsprozess werden die Benutzer gegen die Benutzerdatenbank validiert und verifiziert. Die Zuordnung von Benutzern oder deren Gruppen zu den Rollen der Annotations oder des Web-Anwendung Deployment Descriptor wird im Runtime Deployment Descriptor spezifiziert. Der Runtime Deployment Descriptor (sun-web.xml, sun-ejb-jar.xml) ist der Deployment Deskriptor des Applikationservers. Dieser ist – im Gegensatz zum Web Deployment Descriptor – für alle Web- und EJB-Anwendungskomponenten.

Der Authentifizierungsprozess kann – wie die bereits beschriebenen Sicherheitsanforderungen – in JEE 5 automatisch sichergestellt werden, indem ein weitere Konfiguration zum Deployment-Deskriptor der Web-Anwendung hinzugefügt wird. Außerhalb des `security-constraint`-Elements kann eine Authentifizierungsmethode (`auth-method`) innerhalb einer Login-Konfiguration (`login-config`) hinzugefügt werden. Das Java EE 5 Tutorial ([Sun07, S.860ff]) sowie die Spezifikation ([Sun06a, S.35ff]) definieren als Authentifizierungsmethoden BASIC, DIGEST, FORM, CLIENT-CERT oder herstellerspezifische Methoden. Allgemeine Sicherheitshinweise und Sicherheitsanforderung zu Authentifizierung sowie Methoden zu deren Umsetzung – nicht nur für Java EE 5 – werden in der OWASP Guide ([OWAGuide, S. 107-137]) vorgestellt.

Alle Authentifizierungsmethoden zielen meist nur auf die Authentizität, Vertraulichkeit und die Integrität der Anmeldedaten (Benutzername/Passwort oder Zertifikat) ab. Der Schutz der Anmeldedaten selbst wird nicht berücksichtigt. Alle diese Authentifizierungsmechanismen setzen voraus, dass starke Passwörter und Zertifikate mit der entsprechenden Sicherheit verwendet werden.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/intranet/showNews.jsp</url-pattern>
    <http-method>PUT</http-method>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>role1</role-name>
    <role-name>employee</role-name>
  </auth-constrain>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

2.3.4.1 HTTP Basic Authentifizierung

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

Die HTTP Basic Authentifizierung benutzt eine Standardauthentifizierung mit Benutzername und Passwort. Möchte der Benutzer auf eine geschützte Ressource zugreifen, wird nach seinen Anmeldedaten verlangt. Diese werden vom Server gegen eine Benutzerdatenbank validiert. Nur wenn die Daten korrekt sind, darf der Benutzer die angeforderte Ressource zugreifen.

Die Datenübertragung selbst läuft unverschlüsselt und ohne Authentifizierung des Servers ab. Passwort und Benutzername sind Base64-encodiert und können von Angreifern im Klartext eingesehen werden. Diese Authentifizierungsmethode bietet somit allein keinen Schutz gegen Man-In-The-Middle- und Replay-Angriffe sowie Spoofing. Zudem bietet die Methode aufgrund der Base64-encodierten Passwörter keinen Schutz, wenn ein Angreifer an die Benutzerdatenbank gelangt. Vertraulichkeit und Integrität können in dieser Authentifizierungsmethode nur durch die Verwendung einer sicheren Kommunikation sichergestellt werden. Nur auf diese Weise können bei dieser Authentifizierungsmethode die Man-In-The-Middle-Angriffe und Spoofing verhindert werden. Ohne die Verschlüsselung des Datenverkehrs ist diese Art von Authentifizierung potentiell unsicher.

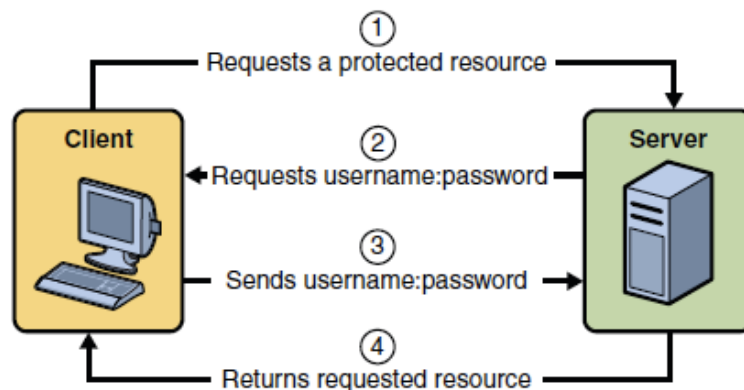


Abbildung 2: HTTP Basic Authentifizierung

2.3.4.2 Form-basierende Authentifizierung

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>file</realm-name>
  <form-login-config>
    <form-login-page>/pfad/login.jsp</form-login-page>
    <form-error-page>/pfad/loginError.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Eine Form-basierte Authentifizierung erlaubt es dem Entwickler das „Look and Feel“ der Login- und Fehler-Seiten nach seinen Wünschen zu gestalten. Möchte der Benutzer eine geschützte Ressource zugreifen, wird er auf die vom Entwickler bereitgestellte Login Seite geleitet. Diese leitet die Anmeldedaten an ein vom Web-Container bereitgestelltes Authentifizierungs-Servlet weiter, dass die Daten verifiziert. Sind diese korrekt, wird der Benutzer auf die Ressource umgeleitet, sind sie nicht korrekt, wird die Fehler-Seite des Entwicklers angefordert. Eine einfache Login-Seite kann wie folgt aussehen:

```
<form method="post" action="j_security_check">
  <input type="text" name="j_username"/>
  <input type="password" name="j_password"/>
</form>
```

Bei dieser Authentifizierungsmethode werden die Anmeldedaten im Klartext übertragen. J_security_check stellt an dieser Stelle keine Möglichkeit einer verschlüsselten Passwortübertragung bereit. Zudem erfolgt wiederum keine Authentifizierung des Servers bei der Datenübertragung. Ohne eine Kommunikation über ein sicheres Protokoll (SSL, VPN, IPSec) ist diese Methode unsicher und ermöglicht Angriffe analog zur HTTP-Basic Authentifizierung.

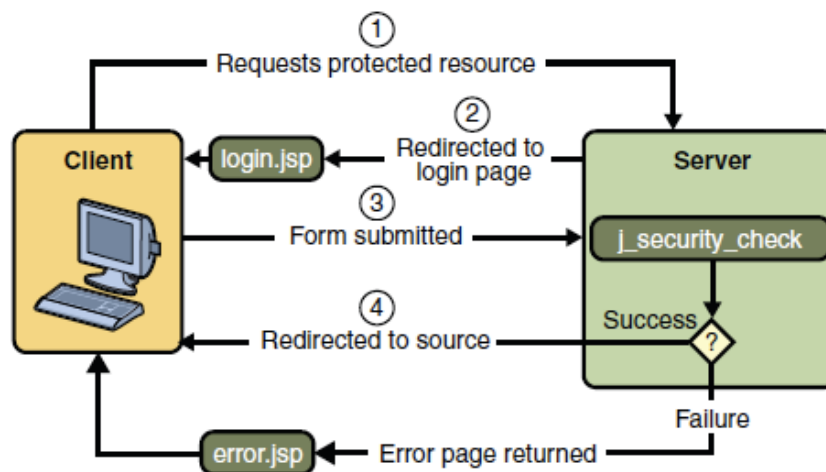


Abbildung 3: Form-basierende Authentifizierung

2.3.4.3 HTTPS Client Authentifizierungen

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

Bei der HTTPS Client Authentifizierung benötigen die Benutzer ein Zertifikat mit dem sie sich beim Server authentifizieren können. Dieser prüft das Zertifikat das für den jeweiligen Benutzer eindeutig ist. Dieses Zertifikat wird in der Regel von einer vertrauenswürdigen Stelle (CA) signiert oder ausgestellt. Die Datenübertragung erfolgt über eine sichere Verbindung, meist SSL.

Diese Authentifizierungsmethode ist sicher gegen Spoofing und Man-in-the-middle Angriffen. Anmelde-
daten liegen bei dieser Methode bereits mit dem Client-Zertifikat vor, so dass Benutzernamen und Pass-
wörter überflüssig werden. Im Allgemeinen ist die Authentifizierung sicher, allerdings authentifiziert sich
lediglich der Benutzer beim Server.

2.3.4.4 Gegenseitige Authentifizierung

Die Authentifizierungsmethode stellt sicher, dass sich Benutzer und Server gegenseitig authentifizieren.
Dies kann auf zwei Arten erfolgen: durch ein Zertifikat oder durch Benutzername und Passwort. Durch
diese Form der Authentifizierung können beide Kommunikationspartner sicher sein, dass ihr Gegenüber
zuverlässig authentifiziert wird. Erfolgt die Übertragung über einen sicheren Kanal und vorausgesetzt, dass
der trustStore nicht kompromittiert wurde, ist die Methode sicher gegen Replay-Attacks, Spoofing-Attacks
und Man-in-the-middle-Attacks.

Zertifikat-basierend

Bei der Zertifikat-basierenden Variante fordert der Benutzer wiederum eine geschützte Ressource an. Der
Server übermittelt dem Client in Schritt 2 sein Zertifikat. Der Client verifiziert dieses Zertifikat durch eine
vertrauenswürdige Stelle (trustStore) und übermittelt daraufhin dem Server sein Zertifikat, das dieser
gegen den trustStore validiert. Haben sich beide Parteien gegenseitig authentifiziert, kann der Benutzer die
Ressource zugreifen.

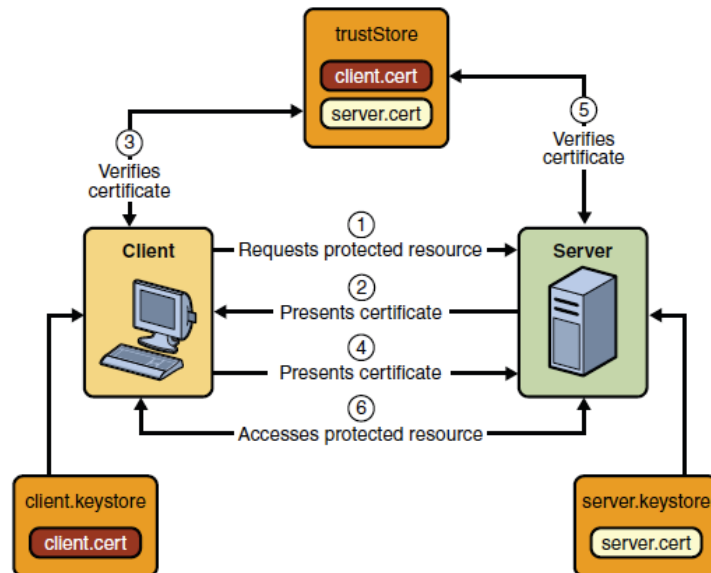


Abbildung 4: Zertifikat-basierende gegenseitige Authentifizierung

Benutzername-Passwort-basierend

Bei dieser Variante authentifiziert sich der Server durch ein Zertifikat, das er dem Client nach der Anfrage sendet. Der Client verifiziert dieses durch den trustStore und authentifiziert sich gegenüber dem Server anschließend mit seinen Zugangsdaten. Stimmen beide Seiten zu, kann die angeforderte Ressource zugegriffen werden.

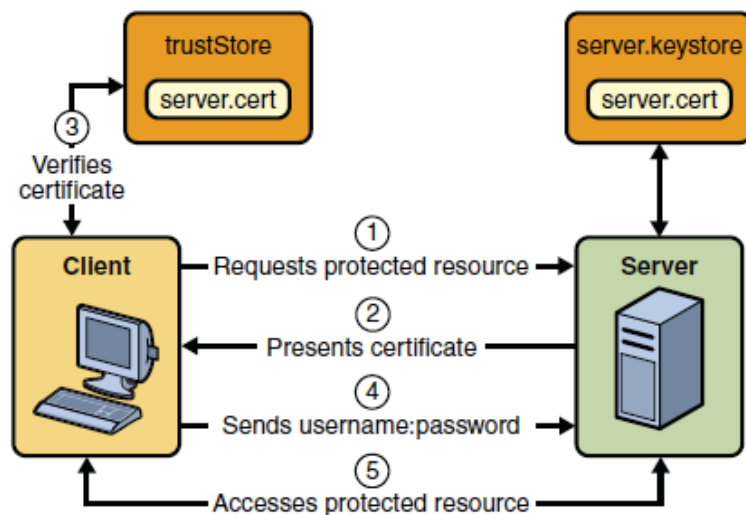


Abbildung 5: Benutzername- und Passwort-basierende gegenseitige Authentifizierung

2.4 Zusammenfassung

In diesem Kapitel wurden Sicherheitsmechanismen für die Authentifizierung und Autorisierung vorgestellt, die von JEE 5 bereit gestellt werden. Mithilfe dieser Mechanismen können Sicherheitsanforderungen von (Web-)Applikationskomponenten, welche zum einen über Annotations oder den Deployment Descriptor definiert werden, sichergestellt werden. Die vorgestellten Konzepte und Technologien sind somit dazu geeignet, im Rahmen der vom OWASP und vom BSI empfohlenen Maßnahmen gegenüber typischen Schwachstellen hinsichtlich Authentifizierung, Autorisierung sowie dem Datentransport eingesetzt zu werden.

3 Sicherheit für JSF- und Java EE-Webapplikationen

Autor: Christof Strauch

In diesem Kapitel soll es nun um die Diskussion häufiger Sicherheitsprobleme sowie um Maßnahmen zu deren Behandlung gehen. Es wird jeweils untersucht, inwiefern die Java Server Faces sowie die Java EE Technologie Mittel und Konzepte bereitstellen, die Entwickler zur Absicherung gegen die jeweiligen Schwachstellen und Verwundbarkeiten anwenden können.

3.1 Auswahl der betrachteten Sicherheitsaspekte

Für die hier betrachteten Sicherheitsaspekte wurde eine Auswahl aus den durch das Open Web Application Security Project (OWASP) ermittelten zehn kritischsten Sicherheitsproblemen in Webapplikationen getroffen, die als OWASP Top 10 zuletzt 2007 veröffentlicht wurden (vgl. [OWA07a]). Die Erhebung dieser Top 10-Liste erfolgte aus den „Vulnerability Trends for 2006“ der gemeinnützigen MITRE Corporation (vgl. [Mit07]), aus denen die zehn am häufigsten berichteten Verwundbarkeiten in Webanwendungen extrahiert wurden, während Aspekte der Betriebssystem- und generellen Anwendungssicherheit (selbst Buffer Overflows) unberücksichtigt blieben. Der OWASP Top 10 2007 Report gewichtet die ausgewählten Sicherheitsaspekte anhand der durch MITRE erhobenen Rohdaten wie in Abbildung 6 dargestellt. Darin wurden die in den MITRE-Daten dokumentierte Häufigkeit der zehn ausgewählten Sicherheitsprobleme summiert und der Anteil der jeweiligen Schwachstelle an dieser Summe errechnet.

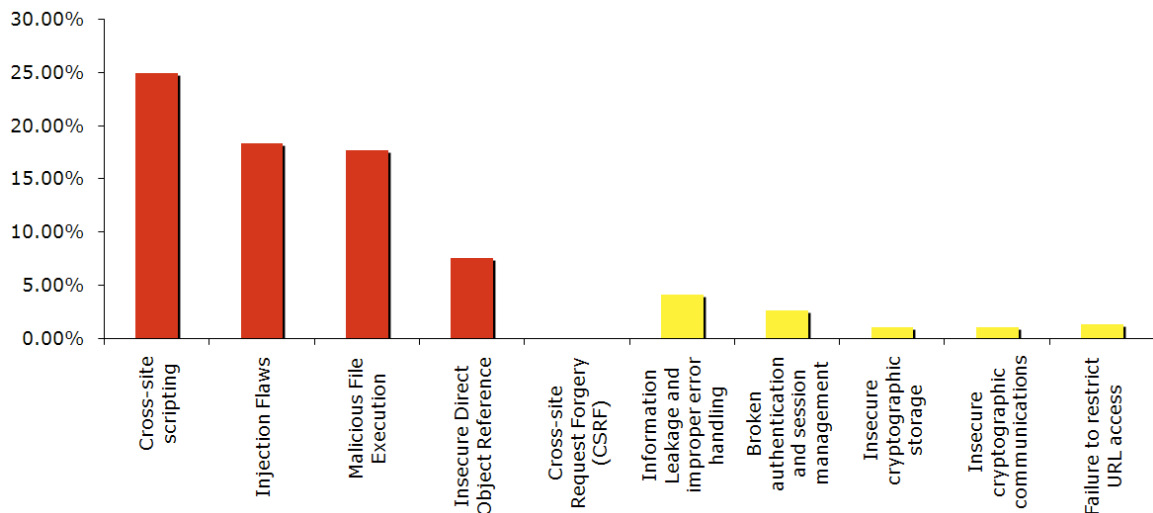


Abbildung 6: Verteilung der Top 10 Sicherheitsrisiken für Webanwendungen (Quelle: [OWA07b, S. 6])

Die vorliegende Arbeit diskutiert im Folgenden die vier am häufigsten berichteten Schwachstellen und Verwundbarkeiten und Schwachstellen. Neben deren Auswirkungen werden konstruktive Maßnahmen erörtert, mit denen der jeweiligen Verwundbarkeit begegnet werden kann. Im Rahmen dieser Erörterung wird insbesondere berücksichtigt, ob und inwiefern Konzepte und Technologien der JSF- und Java EE-Tech-

nologien eingesetzt werden können, um Webanwendungen bezüglich des betrachteten Aspekts abzusichern.

3.2 Sicherheitsaspekte und Angriffstypen

Durch die in der OWASP Top 10 Liste aufgeführten Schwachstellen und Verwundbarkeiten werden diverse Angriffstypen ermöglicht, die in folgender Tabelle aufgeführt und den sie bedingenden Schwachstellen zugeordnet werden.

Schwachstelle aus OWASP Top 10	Mögliche Angriffstypen
A1 Cross-Site-Scripting	<ul style="list-style-type: none"> • Phishing Attacken
A2 Injection Flaws	<ul style="list-style-type: none"> • Vertraulichkeitsverletzungen bezüglich schützenswerter (z.B. privater) Daten • Kompromittierung von Systemen, Datenmanipulation und -löschung
A3 Malicious File Execution	<ul style="list-style-type: none"> • Identitätsdiebstahl • Kompromittierung von Systemen, Datenmanipulation und -löschung
A4 Insecure Direct Object Reference	<ul style="list-style-type: none"> • Phishing Attacken • Vertraulichkeitsverletzungen bezüglich schützenswerter (z.B. privater) Daten • Identitätsdiebstahl • Finanzieller Verlust durch unautorisierte Transaktionen und CSRF-Attacken
A5 Cross Site Request Forgery	<ul style="list-style-type: none"> • Finanzieller Verlust durch unautorisierte Transaktionen und CSRF-Attacken
A6 Information Leakage and Improper error handling	<ul style="list-style-type: none"> • Vertraulichkeitsverletzungen bezüglich schützenswerter (z.B. privater) Daten
A7 Broken Authentication and Session Managemet	<ul style="list-style-type: none"> • Phishing Attacken • Vertraulichkeitsverletzungen bezüglich schützenswerter (z.B. privater) Daten • Identitätsdiebstahl • Finanzieller Verlust durch unautorisierte Transaktionen und CSRF-Attacken
A8 Insecure Cryptographic Storage	<ul style="list-style-type: none"> • Identitätsdiebstahl
A9 Insecure Communications	<ul style="list-style-type: none"> • Identitätsdiebstahl
A10 Failure to Restrict URL Access	<ul style="list-style-type: none"> • Phishing Attacken • Vertraulichkeitsverletzungen bezüglich schützenswerter (z.B. privater) Daten • Identitätsdiebstahl • Finanzieller Verlust durch unautorisierte Transaktionen und CSRF-Attacken

Tabelle 1: Schwachstellen in Webapplikationen und mögliche Angriffstypen (nach [OWA07b, S. 6])

Neben den genannten Angriffstypen weist die OWASP zusätzlich auf einen Reputationsverlust hin, der durch die Ausnutzung aller genannten Schwachstellen für Unternehmen und Organisationen entstehen kann (vgl. [OWA07b, S. 7]).

3.3 Cross-Site Scripting (XSS)

3.3.1 Beschreibung der Schwachstelle

Zu den seit Jahren für Webapplikationen am häufigsten gemeldeten Schwachstellen (vgl. MITRE Vulnerability Trends der Jahre 2005 und 2006 in [Mit07]) zählen Cross Site Scripting Schwachstellen. Bei dieser Art der Schwachstelle ist es möglich, HTML-Markup (häufig mit darin enthaltenem JavaScript-Code) in die Webanwendung einzuschleusen und dieses durch die Webapplikation an einen Browser ausliefern zu lassen. Zur Ausnutzung der Schwachstelle sind zwei Voraussetzungen erforderlich (nach [Sey06]):

1. Es muss die Möglichkeit gegeben sein, Markup und / oder Script-Code in die betreffende Webapplikation einzuschleusen (z.B. durch Formularfelder, URL-Parameter).
2. Das eingeschleuste Markup oder der eingeschleuste Script-Code muss erfolgreich zur Interpretation / Ausführung gebracht werden, also z.B. durch die Webanwendung auf einer Ausgabeseite ausgeliefert und von einem Web-Browser interpretiert / gerendert werden.

Die OWASP beschreibt Cross Site Scripting als eine Teilmenge der HTML-Injection (vgl. [OWA07b, S. 8]) bzw. als „User Agent Injection“ (vgl. [OWAGuide, S. 173]). Es handelt sich also um eine Injektionsschwachstelle (vgl. nächstes Unterkapitel), mit der Besonderheit, dass injizierte Inhalte nicht von einem Server, Betriebs- oder einem sonstigem Backendsystem (Datenbank, LDAP, ERP o.ä.) zur Ausführung gebracht wird, sondern vom Webbrowser eines Benutzers.

3.3.2 Typen von Cross Site Scripting-Schwachstellen

In der Literatur wird bezüglich der Cross Site Scripting Schwachstellen zumindest zwischen den Typen „transientes XSS“ (auch „reflected XSS“) und „persistentes XSS“ (auch „stored XSS“) unterschieden (vgl. [OWA07b, S. 8], [OWAGuide, S. 174-175], [WASCXSS], [Sey06]). Während beim ersten Typ einem Opfer eine URL untergeschoben werden muss, welche die injizierten Inhalte enthält (z.B. auch URL-encodiert), ist es im zweiten Fall möglich, die eingeschleusten Inhalte durch eine Webapplikation zu persistieren (z.B. in Datenbanktabellen für Gästebücher, Foren, Blog-Kommentare u.ä.). Das Open Web Application Security Project führt als weiteren Typus von XSS-Schwachstellen die von Klein ([Kle05]) beschriebene DOM-basierte XSS-Injektion auf ([OWA07b, S. 8f.], [OWAGuide, S. 175]), bei der clientseitig interpretierter JavaScript-Code manipuliert bzw. mit Parametern versorgt wird, die das durch den Browser beim Interpretieren eines (X)HTML-Dokuments aufgebaute Document Object Model (DOM) verändern und zum Einschleusen schädlichen Script-Codes verwenden.

3.3.3 Angriffspotenziale

Cross Site Scripting Attacken können die genannten Typen in Reinkultur oder auch gemischt/hybrid nutzen und betreffen neben Webbrowsern auch andere Applikationen, welche (X)HTML- und JavaScript-Code interpretieren, wie die von OWASP erwähnte Desktop-Suchmaschine Google Desktop ([OWA07b, S. 9]). Für Angriffe wird typischerweise Code in der Sprache JavaScript eingeschleust, der jegliche Aspekte des DOM-Baums manipulieren sowie asynchrone Requests mittels des aus dem AJAX-Ansatz (vgl. [Gar05]) bekannten

XMLHttpRequest-Objektes absetzen kann. Ferner werden für Browser häufig JavaScript-Schwachstellen gemeldet, die Angreifer in betroffenen Browsern ausnutzen können, um weitere Angriffsoptionen zu erhalten.

Das OWASP weist ferner darauf hin, dass grundsätzlich alle Java EE Applikationsframeworks diese Sicherheitslücke enthalten können und in der Vergangenheit auch XSS-Lücken in Fehlerseiten des Web Container-Frameworks Apache Struts ([ApaStr]) sowie in Fehler-, Administrations-, und Beispielseiten von Java EE Applikationsservern auftraten.

3.3.4 Maßnahmen zur Absicherung gegen Cross Site Scripting Schwachstellen

Aus den oben aufgeführten Bedingungen für die Ausnutzbarkeit von Cross Site Scripting Schwachstellen resultieren die folgenden Maßnahmen zur Absicherung eigener Webapplikationen: Validierung aller eingehenden Parameter sowie eine angemessene, starke Einkodierung aller ausgegebenen Daten (vgl. [OWA07b, S. 9-10] und [BSI06, S. 20]).

3.3.4.1 Validierung aller eingehenden Parameter mittels einer Whitelist

Alle eingehenden Daten sind auf Länge, Typ, Syntax und ggf. weitere definierte Regeln (z.B. Syntax einer Kreditkartennummer, Zahlenbereich von Postleitzahlen etc.) zu prüfen, bevor sie von der Anwendung akzeptiert werden. Für die Validierung ist ein Whitelist-Ansatz zu wählen, es sind also jeweils die erlaubten Wertebereiche und Muster zu definieren und nicht die unerlaubten (vgl. [BSI06, S. 24] und [OWA07b, S. 10]). Diese Validierung sollte stets serverseitig erfolgen ([BSI06, S. 32]) und nicht ausschließlich clientseitig (z.B. Formularprüfung mittels JavaScript). Das OWASP sowie das BSI weisen zusätzlich darauf hin, invalide Eingangsdaten lieber zurückzuweisen, als eine Bereinigung oder „Verschönerung“ dieser Daten zu implementieren (vgl. [OWA07b, S. 10] und [BSI06, S. 26 und 95]).

Für die Validierung eingehender Daten in Java Web-Applikationen stehen verschiedenen Möglichkeiten bereit. Die Java Server Faces Spezifikation fordert für Implementierungen die Realisierung eines Validierungsmodells, innerhalb dessen Validatoren registriert werden können, welche das Interfaces `javax.faces.validator.Validator` implementieren, das die Methode `validate` spezifiziert (vgl. [Sun06b, S. 3-34]). Ein Validator zur Prüfung übermittelter E-Mail-Adressen kann beispielsweise wie folgt implementiert werden:

```
public class EmailValidator implements Validator {
    @Override
    public void validate(FacesContext ctx, UIComponent component,
        Object obj) throws ValidatorException {
        String errorMsg = "E-Mail-Adresse inkorrekt";
        FacesMessage msg = new FacesMessage(errorMsg);
        if (obj == null) {throw new ValidatorException(msg);}
        // Pattern for valid eMail-Addresses
        String pattern =
```

```

    "^[\\w-]+(?:\\. [\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-Z]{2,7}$";
    Pattern p = Pattern.compile(pattern);
    Matcher m = p.matcher(obj.toString());
    if (!m.matches()) {throw new ValidatorException(msg);}
  }
}

```

Auf diese Weise implementierte Validatoren werden in der Konfigurationsdatei `faces-config.xml` wie folgt bekannt gemacht:

```

<validator>
  <validator-id>emailValidator</validator-id>
  <validator-class>
    de.hdmstuttgart.csm.validator.EmailValidator
  </validator-class>
</validator>

```

Anschließend kann der implementierte Validator innerhalb von Java Server Pages (JSP) Dokumenten mittels des `validator` Tags der JSF Core Tag Library (URI: <http://java.sun.com/jsf/core>) verwendet werden:

```

<h:inputText id="email" required="true"
  value="#{controller.person.email}" >
  <f:validator validatorId="emailValidator" />
</h:inputText>

```

Außer dem im vorangegangenen Code-Snippet verwendeten `validator`-Tag, das eine Referenzierung der in der JSF Konfigurationsdatei deklarierten Validatoren erlaubt, liefert die JSF Core Taglib gemäß JSF Spezifikation die folgenden Tags für Standardvalidatoren mit (vgl. [Sun06b, S. 9-42], [Sun07, S. 361f]):

Tag aus JSF Core	Beschreibung
<code>validateDoubleRange</code>	Prüfung reeller Werte (vom Typ <code>Double</code>) auf Ober- und/oder Untergrenze.
<code>validateLength</code>	Prüfung auf maximale und/oder minimale Länge einer Eingabe in eine UI-Komponente.
<code>validateLongRange</code>	Prüfung ganzzahliger Werte (vom Typ <code>Long</code>) auf Ober- und/oder Untergrenze.

Tabelle 2: Standardvalidatoren der JSF Core Tag Library

Neben der Java Server Faces Technologie finden sich auch in anderen Web Container Frameworks vergleichbare Validierungsmöglichkeiten. Das Framework Apache Struts etwa erlaubt die Validierung

eingehender Parameter, indem innerhalb von XML-Dateien für implementierte Actions Validierungsregeln in deklarativer Weise spezifiziert werden können (vgl. [ApaStrVal]); als Validator-Implementierungen verwendet Struts diejenigen des XWork-Frameworks von OpenSymphony ([OSyXWo]).

Falls keine Frameworks für Model 2-Web Anwendungen (Begriff nach [Sun02, Kapitel 4.4]) verwendet werden, so sind eingehende Parameter innerhalb der Requests-verarbeitenden JSP-Dokumente und Servlets zu validieren. Um eine zentrale Filterung der eingehenden Parameter zu ermöglichen und eine Verstreuung von Validatoren in einer Webapplikation zu verhindern, können ferner Filter eingesetzt werden. Dabei handelt es sich um Instanzen des Interfaces `javax.servlet.Filter` (vgl. [SunJEEFlt]), das neben den Lebenszyklus-Methoden `init` und `destroy` die Methode `doFilter` spezifiziert. In der Implementierung dieser `doFilter`-Methode kann auf Daten des Requests (z.B. Request-Parameter) und des Responses zugegriffen werden. Filter können als „vorverarbeitende Instanzen“ für Requests sowie als „nachverarbeitende Instanzen“ für Responses einer Web-Anwendung definiert und auch in Filterketten angeordnet werden (z.B. eine Kette aus Kompressionsfilter mit nachgeschaltetem Verschlüsselungsfilter für Responses der Webapplikation).

3.3.4.2 Angemessene und starke Enkodierung aller ausgegebenen Daten

Daten, die innerhalb von (X)HTML- und XML-Dokumenten ausgeliefert werden und die vom Anwender oder aus Backend-Systemen (Datenbank, ERP, Web Service o.ä.) stammen, sind für die Auslieferung in (X)HTML- und XML-Dokumenten in geeigneter Form zu enkodieren. Dies betrifft vor allem die Filterung und Enkodierung von HTML/XML-Entities sowie das Setzen eines Character Encodings für alle Ausgabeseiten (vgl. [OWA07b, S. 10] und [BSI06, 20ff]). Ziel dieser Maßnahmen ist es, keine Zeichen und Zeichenketten aus Eingabedaten des Benutzers oder aus Drittsystemen an den Browser des Anwenders auszuliefern, die zur Ausführung durch ein Plug-in oder einen JavaScript-Interpreter kommen oder durch den Webbrowser selbst gerendert werden.

Ausnahmen von dieser Regel können erlaubte HTML-Tags in Systemen wie Foren oder Wikis sein, die Anwender beispielsweise zur Formatierung ihrer Beiträge verwenden können. Diesbezüglich empfiehlt das BSI jedoch, eine eigene Syntax für solche strukturierenden und / oder formatierenden Tags zu verwenden (z.B. `{strong}...{/strong}` statt `...`), die Einhaltung dieser Syntax zu prüfen und diese Tags erst bei der Auslieferung an den Browser des Anwenders durch ihre (X)HTML/XML-Äquivalente zu ersetzen (vgl. [BSI06, S. 28]).

Die Enkodierung von (X)HTML-Entitäten muss durch Webanwendungen nicht selbst implementiert, sondern es kann innerhalb der typischerweise eingesetzten Technologien JSF oder Struts auf entsprechende Mittel zurückgegriffen werden. So sorgt die innerhalb der Applikationsserver JBoss ([RedJBo]) und Glassfish/Sun GlassFish EnterpriseServer¹ (SJSAS; [JavGla], [SunGla]) verwendete Implementierung Mojarra ([JavJSF]) bei Verwendung des Tags `outputText` aus der JSF HTML Tag Library (URI: <http://java.sun.com/jsf/html>) standardmäßig für eine Enkodierung aller HTML-Entities. Dieses Verhalten wurde in einer einfachen Webanwendung verifiziert, indem in eine Datenbanktabelle in für ein Attribut eines Datensatzes ein `script`-Element eingetragen wurde, das in Entity-enkodierter Form in einer Ausgabeseite der Anwendung ausgeliefert wurde (vgl. Abbildung 7 und Abbildung 8).

¹ `forms` Sun Java System Application Server (SJSAS)

	id	nachname	vorname	email	geburtsdatum	geschlecht
1	8	test	<script>alert('Hallo')</script>	mail@mail.de	1982-03-20 00:00:00.0	m
2	5	Kuhn	Daniel	dk047@hdm-stuttgart.de	1983-10-14 00:00:00.0	m
3	7	Brose	Alfred	ab103@hdm-stuttgart.de	1983-06-29 00:00:00.0	m

Abbildung 7: JavaScript-Code in einer Datenbanktabelle

Vorname	Nachname
<script>alert('Hallo')</script>	test
Daniel	Kuhn

```

<tr>
<td>&lt;script&gt;alert('Hallo')&lt;/script&gt;</td>

```

Abbildung 8: Entity-encodierte Ausgabe

Dieses Standardverhalten konnte auch für Apache MyFaces ([ApaMyF]), Icefaces ([ICEICE]) sowie Woodstock ([JavWoo]) nachgewiesen werden.

Das Model-2 Webanwendungs-Framework Apache Struts liefert in seinen Tag Bibliotheken ebenfalls Tags mit, die Inhalte standardmäßig enkodiert in eine JSP-Seite ausgeben (in Struts 1 u.a. das Tag `write` aus der Bean Taglib [ApaStr1Wri], in Struts 2 z.B. das Tag `property` [ApaStr2Pro]).

Sofern Java Server Faces oder Struts nicht verwendet werden, lässt sich innerhalb von Webanwendungen zur Ausgabe von Inhalten aus Drittsystemen oder vom Anwender selbst die JavaServerPages Standard Tag Library (JSTL; [SunJSTL]) nutzen, um Inhalte Entity-encodiert in JSP-Seiten auszugeben. Diese stellt unter anderem das Tag `out` aus der Core-Taglib bereit, dessen boole'sches Attribut `escapeXml` standardmäßig den Wert `true` annimmt und eine solche Encodierung aktiviert ([SunJSTLout]).

Neben der Encodierung von Entities und weiteren nicht ASCII-Zeichen empfiehlt OWASP ferner die Definition des Character Encodings der ausgelieferten Seiten (vgl. [OWA07b, S. 10]). Diese Maßnahme rührt von der Annahme her, dass Webbrowser bei nicht-gesetztem Character Encoding selbst zu ermitteln versuchen, welches Character Encoding für eine darzustellende Seite gelten könnte. Dies ermöglicht es Angreifern, durch Einschleusen von Zeichen ein gewisses Character Encoding für das Rendering im Browser zu forcieren und auf diese Weise die von der Webapplikation durchgeführten Encodierungen von Sonderzeichen zu unterlaufen, um dennoch Markup oder JavaScript-Code im Browser zur Interpretation zu bringen.

Zum Setzen des Encodings innerhalb von JSP-Seiten können die im folgenden Markup-Snippet fett markierten Direktiven und Tags verwendet werden:

```
<%@ page language="java"
    contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
...
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
..
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
```

Für XML-Dokumente kann das Character-Encoding im Prolog definiert werden:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Innerhalb von Servlets kann mittels der `setHeader` oder `setContentType`-Methode des `HttpServletResponse`-Interfaces (vgl. API [SunSrvRsp]) sowohl der MIME²-Type des zurückgelieferten Dokuments als auch im Falle von (X)HTML dessen Character-Encoding gesetzt werden:

```
response.setHeader("Content-Type", "text/html; charset=ISO-
8859-1");
// alternativ
response.setContentType("text/html; charset=ISO-8859-1");
```

3.3.4.3 OWASP-Tools

Das OWASP stellt innerhalb des OWASP Enterprise Security API ([OWAEnt]) die Interfaces `Validator` und `Encoder` mit Standard-Implementierungen bereit, die innerhalb von Java-Webapplikationen zur Validierung von Eingabedaten sowie zur Enkodierung ausgelieferter Inhalte verwendet werden können.

Ein Anwendungsbeispiel eines Validators findet sich im folgenden Code-Snippet (vgl. [OWA07b, S. 10]):

```
if (!Validator.getInstance().isValidHttpRequest(request)) {
    response.getWriter().write("<P>Invalid HTTP Request");
}
```

3.4 Injection Flaws

3.4.1 Beschreibung der Schwachstelle

Als Injection Flaws werden Schwachstellen bezeichnet, die es Angreifern ermöglichen, Daten durch einen Interpreter zur Ausführung kommen zu lassen (vgl. [OWA07b, S. 12], [OWAGuide, S. 171ff]), z.B.

- SQL-Interpreter einer Datenbank
- Kommando-Interpreter eines Betriebssystems
- (L)DAP-Interpreter eines Verzeichnisdienstes
- XPath-Interpreter
- XSLT-Interpreter/Prozessor
- XML-verarbeitende Interpreter
- (X)HTML-verarbeitende Interpreter (z.B. Browser, siehe Unterkapitel 3.3)

3.4.2 Angriffspotenziale

Zur Ausnutzung der Schwachstelle ist es notwendig, dass Eingabedaten (z.B. Parameter), die an eine Webanwendung übermittelt werden, nicht oder nur unzureichend gefiltert, validiert und enkodiert an einen solchen Interpreter übermittelt werden. Angreifer können Eingabedaten in diesem Fall entsprechend aufbereiten, um durch die interpretierten Kommandos Daten anzulegen, zu lesen, zu manipulieren oder auch zu löschen; die konkreten Angriffsmöglichkeiten hängen von der Art des Interpreters, den Rechten des ausführenden Benutzers/Prozesses sowie den durch die Webanwendung definierten Kommandos, in die die Eingabedaten eingefügt werden, ab.

Potentiell sind alle Webapplikationen, die Interpreter (z.B. innerhalb von Libraries zur XML-Verarbeitung oder zum Zugriff auf Namens- und Verzeichnisdienste) verwenden oder andere Prozesse (z.B. Datenbank-Server-Prozesse, Kommando-Interpreter von Betriebssystemen) aufrufen, für Injection Flaws anfällig (vgl. [OWA07b, S. 12]).

3.4.3 Maßnahmen zur Absicherung gegen Injection Flaws

Ziel der Absicherung gegenüber Injection Flaws ist es, jegliche von Interpretern und Prozessen verarbeiteten Kommandos, die die Webanwendung an diese absetzt, vor Manipulation durch Benutzerdaten zu schützen (vgl. [OWA07b, S. 12]). Aufgrund der Vielzahl angreifbarer Interpreter und Angriffstypen (vgl. Tabelle 1) sind verschiedene Maßnahmen zur Absicherung gegenüber Injection Flaws in Betracht zu ziehen, die im Folgenden diskutiert werden.

3.4.3.1 Validierung von Eingabedaten

Analog zur oben beschriebenen Maßnahme gegenüber Cross Site Scripting sind auch zur Absicherung von Injection Flaw-Schwachstellen sämtliche Eingabedaten auf Länge, Typ, Syntax und ggf. weitere geltende Regeln zu prüfen (vgl. [BSI06, S. 20]). Wie oben beschrieben, ist ein Whitelist-Verfahren einem Blacklist-Verfahren vorzuziehen (vgl. [BSI06, S. 24] und [OWA07b, S. 13]). Für die Erstellung einer solchen Whitelist empfiehlt das BSI, zunächst von einer möglichst geringen Menge an Zeichen auszugehen und die Whitelist nur bei Bedarf zu erweitern. Ferner sollten einzelne Filter möglichst einfach gestaltet und komplexe Filter durch eine sequentielle Kette von Filtern nachgebildet werden (vgl. [BSI06, S. 21]). Sofern eine Input-Filterung dennoch nur mittels eines Blacklist-Verfahrens erfolgen kann, führt das BSI eine Liste potentiell gefährlicher Zeichen für diverse Interpreter auf, die in Anhang 1 (ergänzt um kritische Zeichen in XML-Dokumenten nach [OWAGuide, S. 183]) wiedergegeben werden. Das OWASP-Projekt gibt in seinem „Guide to Building Secure Web Applications and Web Services“ in einem eigenen Kapitel weitere Hinweise zur Datenvalidierung ([OWAGuide, S. 161ff]).

Im vorangegangenen Unterkapitel wurden ausführlich verschiedene Möglichkeiten der Validierung und Filterung von Eingabedaten mit Mitteln der JSF-, Struts- und Servlet/Filter-Technologien vorgestellt, die auch als Bestandteil der Maßnahmen gegenüber Injection Flaws eingesetzt werden können. Auch Validatoren des OWASP Enterprise Security API ([OWAEnt]) können zur Absicherung gegenüber Injection Flaws eingesetzt werden.

3.4.3.2 Verwendung stark typisierter, parametrisierter Abfrage-APIs

Das OWASP empfiehlt ferner die Verwendung stark typisierter und parametrisierter APIs zur Kommunikation mit Backend-Systemen ([OWA07b, S. 13]). Ziel dieser Maßnahme ist es, durch ein Framework oder ein API Eingabedaten für Abfragen zumindest auf ihren Typ prüfen zu lassen, für das Backend-System zu enkodieren (insbesondere Escaping von Sonderzeichen) und keine Manipulation zuzulassen, die die Struktur einer von der (Web-)Anwendung zusammengesetzten Abfrage verändern.

Für diverse Backendsysteme, mit denen eine (Web-)Applikation kommuniziert, sind verschiedene solcher Frameworks und APIs verwendbar. Am Beispiel der Kommunikation mit relationalen Datenbanken wird der Zugriff über typisierte und parametrisierte APIs und Frameworks im Folgenden erläutert.

APIs und Frameworks für Zugriffe auf relationale Datenbanken

Zur Verhinderung von SQL-Injections wird in der Literatur einhellig vor der Verwendung dynamisch – d.h. durch String-Verkettung – zusammengesetzter SQL-Statements gewarnt ([OWA07b, S. 13], [OWAGuide, S. 179f], [BSI06, 29ff]). Diese bieten Angreifern die Möglichkeit, durch Eingabedaten, welche in derartige SQL-Statements eingehen, Strukturveränderungen an Statements vorzunehmen, beispielsweise im WHERE-Teil eines SELECT-Statements tautologische (immer wahre) Anteile oder-verknüpft einzufügen oder mehrere SQL-Statements zu verschachteln oder zu verketteten (via Sub-Selects oder Kommandokonkatenation).

Um derartige Angriffe zu verhindern, wird die Verwendung stark typisierter und parametrisierter APIs empfohlen. Zu diesen zählen gemäß OWASP ([OWAGuide, S. 179]):

- parameterisierte Stored Procedures, insbesondere solche mit stark typisierten Parametern

- Prepared Statements
- Frameworks für objektrelationales Mapping

Bezüglich der Stored Procedures und Prepared Statements weist das BSI darauf hin, dass bei Übergabe von Daten des Benutzers zusätzlich eine Validierung notwendig ist, um problematische Eingaben abzufangen (vgl. [BSI06, S. 29] und Hinweise von Microsoft zu Stored Procedures unter dem Microsoft SQL Server-DBMS [MicSSP]). Ferner können Stored Procedures bei Verwendung von `exec()` sowie durch Anhängen von Parametern gegenüber Injektion anfällig sein ([OWA07b, S. 13]). Sowohl Stored Procedures als auch Prepared Statements erlauben einem SQL-Interpreter grundsätzlich jedoch das Parsen und die Erstellung eines abstrakten Syntaxbaums (AST) vor dem Absetzen eigentlicher Datenbank-Abfragen. Da eine solche Vorverarbeitung seitens des DBMS geschieht und für konkrete Queries lediglich noch Parameter eingesetzt werden, wird verhindert, dass diese Parameter sich strukturverändernd auf das Statement auswirkend. Auch weist das BSI auf die umfangreichen Formulierungsmöglichkeiten für Bedingungen in SQL-Statements hin, so dass in aller Regel keine Notwendigkeit für dynamische, d. h. durch die Applikationslogik mittels String-Verkettung zusammengesetzte, SQL-Statements besteht (vgl. [BSI06, S. 29]).

Zum Aufruf DBMS-seitig definierter Stored Procedures stellt das Java Database Connectivity (JDBC)-API das Interface `CallableStatement` ([SunSqlCal]) bereit, das wie folgt verwendet werden kann:

```
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

Zur Verwendung von Prepared Statements steht das Interface `PreparedStatement` ([SunSqlPre]) bereit, das – wie im folgenden Snippet dargelegt – eingesetzt werden kann:

```
PreparedStatement ps = con.prepareStatement (
    "select * from user where id=?");
ps.setInt(1, param);
ResultSet rs = preparedStatement.executeQuery();
```

Das Snippet zeigt außerdem, dass Parameter nicht als Strings übergeben werden müssen, sondern über Setter gesetzt werden können, die eine Typprüfung vornehmen. Das Interface `PreparedStatement` stellt solche Setter für gängige DBMS-Datentypen bereit, von denen bei Verwendung mittels JDBC Gebrauch gemacht werden sollte.

Um Injection Flaws zu schützen, können ferner Frameworks für objektrelationales Mapping (ORM) eingesetzt werden, die als Service-Provider-Implementierungen für das Java Persistence API (JPA; siehe [Sun06c], [Sun06a, S. 11 und 142f], [Sun07, S. 683ff]) oder aber nativ verwendet werden können. Als Beispiele solcher ORM-Frameworks sind unter anderem das bekannte quelloffene ORM-Framework `Hibernate` ([RedHib]) sowie die kommerzielle ORACLE-spezifische Implementierung `TopLink` zu nennen

([OraTop]). Diese Frameworks erlauben einen objektorientierten Zugriff auf relationale Datenbanken und definieren Abfragesprachen (JPAQL, HQL), in denen Instanzen definierter Klassen als Parameter übergeben werden können, für die das Framework eine Typprüfung durchführt. Es ist zu vermuten, dass diese Frameworks für die meisten Abfragen Prepared Statements verwenden, insbesondere wenn im Rahmen des JPA NamedQueries (vgl. z.B. [Sun07, S. 720f) definiert werden, die eine Verwendung von Prepared Statements nahelegen. Sowohl OWASP ([OWAGuide, S. 180f], [BSI06, S. 30]) als auch das BSI weisen dennoch darauf hin, dass dies im Einzelfall zu verifizieren ist und dass zusätzliche Sicherheitslücken entstehen können, da die Frameworks zum einen eine eigene – von ihnen selbst interpretierte – Abfragesprache definieren und zum anderen häufig Methoden für das Absetzen nativer SQL-Queries bereitstellen (z.B. für Performance-kritische oder auch komplexe Abfragen, die durch die eigene Query Language nicht unterstützt werden).

3.4.3.3 Zugriff mit minimalen Rechten

Zum Zugriff auf Ressourcen und Dienste innerhalb von Web- und Applikationsservern werden in Kapitel 2 die Konzepte von Sicherheitsrollen sowie dem Mapping dieser Rollen auf konkrete Benutzer und Benutzergruppen erläutert. Andere Backendsysteme wie Datenbanken, ERP-Systeme, Namens- und Verzeichnisdienste u.a. umfassen Rollen- und Rechtekonzepte, bei denen die Art des Zugriffs (z.B. lesen, Datensätze erstellen, editieren, löschen) sowie die für den Zugriff freigegebenen Datenbereiche definiert werden können (z.B. mittels Access Control Lists, Capability Lists, Discretionary Access Control, Mandatory Access Control, Role Based Access Control).

Zum Zugriff auf Backend-Systeme sind stets Identitäten mit minimalen Rechten zu verwenden (vgl. [OWA07b, S. 13], [OWAGuide, S. 138ff] und für Datenbankserver [BSI06, S. 87ff]). So sind zumindest unterschiedliche Benutzer für lesenden und schreibenden Zugriff zu verwenden und wenn möglich sollten Schreibberechtigungen weiter unterteilt werden in Erstellung, Manipulation und Löschung von Daten sowie administrative Befugnisse. Bezüglich der Datenbereiche, auf die mittels der genannten Berechtigungen zugegriffen werden darf, sollte ebenfalls eine Unterteilung erfolgen. Diese Hinweise gelten auch bei Verwendung von Funktionsbenutzern, die nicht Personen oder Gruppen, sondern Prozessen zugeordnet werden. So sollte etwa bei rein lesenden Operationen aus einer Webanwendung auf ein Datenbanksystem auch eine Benutzerkennung verwendet werden, die ausschließlich lesende Zugriffe besitzt, um Angreifern nicht mittels der Webapplikation eine Rechteauserweiterung zu ermöglichen.

Ferner sind auch die Prozesse des Web- und Applikationsservers unter Benutzerkennungen zu betreiben, die mit minimal notwendigen Rechten ausgestattet sind und nicht von mehreren Prozessen verwendet werden (z.B. die Unix-Gruppe nobody, vgl. [BSI06, S. 75f sowie 82ff]).

3.4.3.4 Vermeidung detaillierter Fehlermeldungen

Detaillierte Fehlermeldungen, die etwa Stacktraces oder Teile von SQL-Statements umfassen, können für Angreifern hilfreich hinsichtlich der Informationsgewinnung und Identifizierung verwendeter Technologien und Versionen sein (Spotting und Fingerprinting), die auf das Vorhandensein von Injection Flaws und anderer Schwachstellen hindeuten können. Für Fehlermeldungen ist grundsätzlich das Prinzip der minimalen Information anzuwenden. Fehlermeldungen sollten für legitime Benutzer der Webanwendung aussagekräftig sein und ihnen Handlungsempfehlungen zur Problemlösung und Fehlervermeidung geben

(vgl. etwa die Usability-Heuristiken von Nielsen [NieUsa] und Shneiderman [Shn87]), jedoch ist eine Offenlegung der verwendeten Technologien und Implementierungsdetails zu vermeiden ([OWAGuide, S. 13], [BSI06, S. 58f und 72]). So können etwa bei Verwendung der Java Server Faces oder Struts Technologie Exceptions in Action-Klassen abgefangen, serverseitig geloggt und Fehlermeldungen aus einem Resource Bundle (vgl. [Sun07, S. 448]) für die ausgelieferten JSP-Seiten gesetzt werden. Auf diese Weise bleiben technische Details für Endnutzer verborgen und Administratoren sowie Entwicklern wird dennoch die Möglichkeit gegeben, anhand der Log-Dateien Fehlersituationen nachzuvollziehen.

3.4.3.5 Vermeidung von Betriebssystemaufrufen

Da Java EE-Applikationen bestehend aus Web- und EJB-Komponenten grundsätzlich in Containern von Web- und Applikationsservern ausgeführt werden und für sie eine Ortstransparenz vorgesehen ist, um etwa Clustering zu ermöglichen, sind gemäß der Java EE Spezifikation grundsätzlich keine Betriebssystemaufrufe aus Web- und EJB- Anwendungen erlaubt. Java EE-Anwendungen, die sich in diesem Aspekt spezifikationsgemäß verhalten, sollten daher nicht der Gefahr von Command-Injections ausgesetzt sein. Die Java EE-Spezifikation schreibt ferner vor, dass Java EE-Komponenten niemals direkt mit anderen Java EE-Komponenten oder externen Systemen interagieren, sondern stets Dienste und Protokolle des Containers verwenden sollen (vgl. [Sun06a, S. 8]). Diese Dienste umfassen etwa Netzwerkprotokolle (HTTP, HTTPS, RMI/IIOP, CORBA IDL), Zugriffe auf Datenbanken (JDBC, JPA), asynchronen Nachrichtenaustausch (JMS), Namens- und Verzeichnisdienste (JNDI), Mail- und MIMI-Type-Verarbeitung (JAF), Sicherheitsdienste (z.B. JAAS) und Web Services (vgl. [Sun06a, S. 10ff.]). Sofern Ressourcen zugegriffen werden, die keinen dieser Dienste des Applikationsservers nutzen, besteht die Möglichkeit der Verwendung von Ressourcen-Managern und Konnektoren, die die Java EE-Spezifikation vorsieht (vgl. [Sun06b, S. 9 und 12f]). Es wird daher empfohlen, in Web- und EJB-Komponenten den Zugriff auf sämtliche externe Systeme über Dienste des Containers zu realisieren, anstatt direkt auf das jeweilige Backend-System zuzugreifen, da die Dienste zum einen hinsichtlich Berechtigungs- und Sicherheitsanforderungen von Administratoren konfiguriert werden können und zum anderen durch viele der genannten Dienste parameterisierte Zugriffe und starke Typisierung ermöglicht sowie eine für das Backend-System angemessene Encodierung durchgeführt wird. Diese Empfehlung entbindet Entwickler und Administratoren jedoch nicht, Anwendungen auf vorhandene Injection- und andere Schwachstellen zu prüfen und auch die Implementierungen der genannten Dienste hinsichtlich durchgeführter Validierungen, Filterungen und Encodierungen zu prüfen.

3.5 Malicious File Execution

3.5.1 Beschreibung der Schwachstelle

Malicious File Execution Schwachstellen treten auf, wenn Applikationen Eingabeparameter oder vom Anwender übertragene Dateien direkt oder verkettet mit Datei- oder Byte-/Zeichenstromfunktionen verwenden. Angreifen wird es dadurch ermöglicht, entweder selbst (schädlichen) Code zur Ausführung, Interpretation oder Ausgabe zu bringen (beispielsweise Backdoors) oder aber Zugriff auf Dateien des Filesystems eines Web-/Applikationsserver (z.B. Konfigurationsdateien des Servers, Dateien des Server-

Betriebssystemen) sowie externen Dateisystemen zu erhalten, falls diese durch den Web Server erreichbar sind und zugegriffen werden dürfen (vgl. [OWA07b, S. 15]).

3.5.2 Angriffspotenziale

Angreifbar sind grundsätzlich alle Webentwicklungstechnologien, die Dateien oder Dateinamen vom Anwender akzeptieren. Beispiele im Umfeld von Java-Webanwendungen können Servlets sein, die Dateinamen als URL-Parameter entgegennehmen, oder Programmcode, der Anwender lokale Dateien auswählen lässt. Des Weiteren können Datei-Uploads zu dieser Sicherheitslücke führen, wenn die Inhalte der von Anwendern übermittelten Dateien nicht oder nur unzureichend durch die Webapplikation geprüft werden, so dass Angreifer potentiell eigene Java-Klassen oder JSP-Dokumente zur Ausführung bringen können (vgl. [OWA07b, S. 15]). Auch zunächst wenig offensichtliche Angriffsmöglichkeiten wie der Upload von XML-Dokumenten, die entfernte DTDs nachladen und anschließend von XML-Prozessoren interpretiert werden, wurden bereits demonstriert ([Grz07], zitiert nach [OWA07b, S. 15]).

Die potentiellen Schäden durch Malicious File Execution-Angriffe hängen unmittelbar mit der Absicherung des Web-(und EJB-)Containers zusammen. Dieser kann durch die Verwendung und adäquate Konfiguration eines Security Managers dahingehend abgesichert werden, dass Webanwendungen in einer Sandbox, d.h. isoliert voneinander und von anderen Systemen und Ressourcen, ausgeführt werden. Häufig ist jedoch ein solcher Security Manager nicht aktiviert oder unzureichend konfiguriert (vgl. [OWA07b, S. 16]).

3.5.3 Maßnahmen zur Absicherung gegen Malicious File Execution

Zu den Absicherungsmaßnahmen ist zunächst zu sagen, dass bei sorgsamer Planung und Modellierung einer Webapplikation Benutzereingaben nicht benötigen sollten, die serverseitige Ressourcen über Pfade und Dateinamen (oder Teile daraus) referenzieren. Ferner können über Firewall-Regeln sowie durch eine entsprechende Konfiguration des Webservers (z.B. Prozessrechte und Dateizugriffsberechtigungen [BSI06, S. 77ff, 82ff und 94]) ausgehende Verbindungen sowie Dateisystemzugriffe unterbunden werden (vgl. [OWA07b, S. 16]).

Sollten Benutzereingaben, die serverseitige Ressourcen referenzieren, sowie Datei-Uploads von der Webanwendung benötigt werden, so sind die im Folgenden vorgestellten Maßnahmen zur Absicherung gegen Malicious File Execution geeignet ([OWA07b, S. 16f]).

3.5.3.1 Indirekte Objekt-Referenzierung

Zur Referenzierung serverseitiger Ressourcen sind keine Pfade und Dateinamen oder Teile daraus zu verwenden, sondern indirekte Referenzen. OWASP schlägt die Verwendung von Hashes für Ressourcen vor, die serverseitig auf konkrete Pfade und Dateinamen gemappt werden ([OWA07b, S. 16f]). Die Verwendung aktuell als sicher eingestufte Hash-Funktionen (z.B. der SHA-2-Generation) aus der Java Cryptography Extension (JCE) API gegenüber einfachen Index-Zählungen verhindert, dass Anwender durch Kenntnis einer Ressourcen-Referenz auf andere schließen und diese durch Interieren / Enumerieren zur Ausgabe oder Ausführung bringen können (vgl. auch [BSI06, S. 64]). Sollte dennoch eine indirekte Referenzierung über einfache Index-Zählungen erfolgen, so ist vor der Ausgabe oder Ausführung der Ressource stets zu prüfen,

ob der (un-)angemeldete Anwender gemäß eines Rollen- und Rechtekonzeptes Zugriff auf diese Ressource erhalten darf. Ferner können auch schwache oder selbst-entwickelte Hash-Algorithmen Benutzern den Zugriff auf oder die Ausführung von Ressourcen gewähren, den / die von der Webapplikation üblicherweise verwehrt geblieben wären (siehe z.B. die in den vergangenen Jahren bekannt gewordenen Sicherheitslücken des StudiVz hinsichtlich Profilen und Bildergalerien [HeiSec06a] [HeiSec06b] [HeiSec08]). Auch bei indirekter Objekt-Referenzierung ist die Webanwendung somit nicht von der Pflicht entbunden, Zugriffe auf Ressourcen hinsichtlich der Berechtigungen des anfordernden Benutzers zu prüfen (vgl. [OWA07b, S. 19]).

3.5.3.2 Benutzereingaben validieren und Datei-Uploads prüfen

Die für die Cross Site Scripting und Injection Flaw Schwachstellen oben beschriebenen Maßnahmen zur Validierung von Benutzereingaben sind ebenfalls zur Absicherung gegenüber Malicious File Execution Verwundbarkeiten geeignet ([OWA07b, S. 16]).

Für den Upload von Dateien ist in jedem Fall eine Whitelist erlaubter MIME-Typen und/oder Dateierweiterungen anzugeben, die von der Webanwendung akzeptiert werden, und übertragene Dateien sind auf Konformität mit diesen Regeln zu prüfen. Der Upload von Dateitypen, welche vom Server ausgeführt oder interpretiert werden, ist dabei zu unterbinden.

Vom Benutzer hochgeladene Dateien oder Referenzen auf Dateinamen sind ferner dahingehend zu prüfen, ob sie andere Daten der Benutzer-Sitzung zugreifen und manipulieren können (vgl. [OWA07b, S. 17]).

3.5.3.3 Firewall-Konfiguration

Um den Web-Server am Aufbau von Verbindungen zu externen Websites und internen Systemen zu hindern, sind Firewall-Regeln zu definieren. Geschäfts- und sicherheitskritische Systeme können ferner in einem eigenen virtuellen lokalen Netzwerk (VLAN) isoliert werden ([OWA07b, S. 16]).

3.5.3.4 Verwendung des Java EE Security Managers

Für den Betrieb von Webanwendungen wird die Verwendung des Java EE Security Managers empfohlen. Dieser kann u.a. den Zugriff auf Dateien des Server-Dateisystems verbieten und lediglich Dateien unterhalb des Wurzelverzeichnisses für Web-Ressourcen einer Webanwendung erlauben (vgl. [OWA07b, S. 16f]). Die Java EE Spezifikation definiert eine minimale Menge an Berechtigungen, die Webanwendungen zur Laufzeit erwarten können ([Sun06a, S. 117f]). Weitergehende Berechtigungen aus der für die Java Standard Edition (Java SE) definierten Menge von Berechtigungen ([SunJsePrm]) sind explizit gegenüber Server-Administratoren und Betreibern von Anwendungskomponenten zu kommunizieren und einzeln freizuschalten.

3.5.3.5 Verwendung von FileServlets vermeiden

Einige Webserver liefern in ihren Distributionen ein Servlet mit, das Inhalte des Dateisystems an den Aufrufer zurückliefert (etwa Sun und ORACLE). Die Verwendung solcher Servlets in eigenen Webanwendungen ist zu vermeiden (vgl. [OWA07b, S. 17]); sie sollten – falls durch den Webserver bereitgestellt – also keinesfalls im Web-Deployment-Deskriptor auf URL-Patterns gemappt werden.

3.6 Insecure Direct Object References

3.6.1 Beschreibung der Schwachstelle

Eine Insecure Direct Object Reference Schwachstelle liegt vor, sofern eine Webapplikation eine Referenz auf ein implementierungs-internes Objekt gegenüber Anwendern exponiert. Bei einem auf diese Weise referenzierten Objekt kann es sich etwa um eine Datei, ein Verzeichnis, einen Datensatz aus einer Datenbank (z.B. Primärschlüssel) oder auch um Session-Informationen (wie Session-IDs) handeln. Falls eine solche Objektreferenz mittels eines URL- oder Formular-Parameters durch den Anwender übermittelt wird und keine Zugriffskontrolle erfolgt, kann sie von Angreifern manipuliert werden, um Zugriff auf andere Daten zu erlangen (vgl. [OWA07b, S. 18]).

3.6.2 Angriffspotenziale

Im Jahre 2000 gelang es einem Angreifer durch eine Lücke in einer Anwendung der australischen Steuerbehörde steuerlich relevante Daten von 17.000 Firmen auszuforschen, da eine eindeutige Steuernummer als URL-Parameter an die Webanwendung wurde und keine Autorisierungsprüfung bezüglich des Zugriffs auf die angeforderten Steuerdaten erfolgte (nach [OWA07b, S. 18]).

Angreifbar hinsichtlich unsicheren direkten Objekt-Referenzen sind viele Webanwendungen, da es häufig nahe liegt und wenig Implementierungsaufwand verursacht, eindeutige interne Namen und Schlüssel für Daten und Ressourcen nach außen zu geben. Sofern des weiteren auf einen „security through obscurity“-Ansatz gesetzt wird und keine Autorisierungsprüfungen erfolgen, können Angreifer durch Parameter-Manipulation ihnen eigentlich verwehrt Informationen erlangen und Ressourcen zugreifen, um eine zwar ggf. vorhandene aber nur unzureichend durchgesetzte Zugriffskontrolle zu unterlaufen ([OWA07b, S. 18]).

3.6.3 Maßnahmen zur Absicherung gegenüber Insecure Direct Object Reference

Maßnahmen zur Absicherung gegen über Insecure Direct Object Reference Schwachstelle leiten sich zunächst aus den für diese Schwachstelle notwendigen Bedingungen ab: so ist einerseits die Exponierung interner Objekt Referenzen zu vermeiden, andererseits sind stets Autorisierungsprüfungen hinsichtlich des Zugriffs auf Daten und Ressourcen durchzuführen. Ferner kann die Webanwendung Parameter-Manipulationen durch Anwender erkennen und darauf reagieren. Zusätzlich sind – wie in den vorangegangenen Unterkapiteln bereits ausgeführt – Eingabedaten wie GET- und POST-Parameter von der Webapplikation stets zu validieren ([OWA07b, S. 19]).

3.6.3.1 Keine internen Objekt Referenzen exponieren

Interne Objektreferenzen auf Daten und Ressourcen sollten gegenüber Anwendern nicht publiziert und nach außen gegeben werden. Stattdessen können serverseitig Hashes oder Indizes für Datensätze (z.B. einer Datenbank) und Ressourcen (z.B. Dateien, Pfade) generiert, an Anwender übermittelt und serverseitig auf die angeforderten Ressourcen gemappt werden (vgl. [OWA07b, S. 19]). Es ist dabei auf die

Verwendung als sicher eingestufte Hash-Funktionen zu achten (z.B. SHA-2-Generation). Sofern Indizes verwendet werden, ist zusätzlich eine Autorisierungsprüfung sowie eine Prüfung auf manipulierte Parameter durchzuführen, da Anwender ansonsten Indizes iterieren, enumerieren oder erraten können (vgl. Hinweis des BSI in [BSI06, S. 64ff]).

Das OWASP Enterprise Security API ([OWAEnt]) stellt für das serverseitige Mapping von Hashes das Interface `AccessReferenceMap` bereit, das – wie im folgenden Code-Snippet dargestellt – verwendet werden kann ([OWA07b, S. 19f]):

```
HttpSession session = request.getSession();
AccessReferenceMap arm =
    (AccessReferenceMap) session.getAttribute("usermap" );
if ( arm == null ) {
    arm = new AccessReferenceMap();
    request.getSession().setAttribute( "usermap", arm );
}
String param = request.getParameter("user");
String accountName = (String)arm.getDirectReference(param);
```

Die Methode `getDirectReference` besitzt mit der Methode `getIndirectReference(Object directReference)` eine Inverse, die zu einer direkten Objektreferenz eine indirekte generiert, welche im Rahmen der Webanwendung an Clients ausgeliefert werden kann ([OWAEntAPI]).

3.6.3.2 Autorisierungsprüfungen

Unabhängig davon, ob Objektreferenzen direkt oder in Form von Hashes/Indizes nach außen gegeben werden und ob die Webanwendung Parameter-Manipulationen erkennt, sind für Zugriffe auf Daten und Ressourcen stets Autorisierungsprüfungen durchzuführen (vgl. [OWA07b, S. 19]).

3.6.3.3 Parameter-Manipulationen erkennen

Um die vom Anwender übergebenen Parameter auf Manipulationen zu prüfen, können diese mittels gängiger Verschlüsselungs- und Hash-Funktionen auf Integrität hin überprüft werden. Eine solche Identitätsprüfung kann mittels zusätzlich eingefügter GET- und POST-Parameter (z.B. versteckte Formularfelder) erzielt werden (vgl. [OWA07b, S. 19]). Diese können entweder unter Zuhilfenahme der Java Cryptography Extension (JCE) selbst implementiert oder es kann auf Projekte wie das quelloffene HTTP Data Integrity Validator Framework ([HDIHDI]) zurückgegriffen werden.

4 Zusammenfassung und Fazit

Die vorliegende Arbeit beschäftigt sich mit Sicherheitsaspekten der Java Server Faces und Java EE-Technologien. Es werden häufige Sicherheitsschwachstellen in Webanwendungen vorgestellt und Maßnahmen diskutiert, um Webapplikationen gegenüber diesen abzusichern. Die Auswahl der betrachteten Aspekte erfolgt nach den durch das Open Web Application Security Project zusammengestellten häufig gemeldeten Schwachstellen in Webapplikationen ([OWA07a] und [OWA07b]) sowie dem Maßnahmenkatalog für sichere Webapplikationen des Bundesamtes für Sicherheit in der Informationstechnik ([BSI06]).

Im zweiten Kapitel der Arbeit werden Konzepte der Authentifizierung und Autorisierung von Benutzern diskutiert. Diesen Aspekten der Applikationssicherheit kann mit den in der Java EE Spezifikation vorgesehenen Mechanismen und Technologien zur deklarativen und programmatischen Rollendefinition, zur Authentifizierung sowie zur sicheren Datenübertragung Rechnung getragen werden. Java EE 5 gibt Entwicklern die Möglichkeit, Sicherheitsrollen für Anwendungskomponenten zu deklarieren und Zugriffe auf Ressourcen und Anwendungsfunktionalität programmatisch sowie durch den Container zu prüfen. Ferner definiert die Java EE Spezifikation Authentifizierungsmechanismen (HTTP-Basic Authentication, HTTPS mit ein- oder beidseitiger Authentifizierung, Form-basierte Authentifizierung), die deklarativ für Web- und EJB-Anwendungskomponenten aktiviert werden können. Des Weiteren werden verschlüsselte und integritätsgeschützte Transportkanäle für Webanwendungen unterstützt. Die genannten Maßnahmen können zur Absicherung gegen typische Sicherheitsschwachstellen verwendet werden, insbesondere gegenüber den von OWASP aufgeführten Verwundbarkeiten „Broken Authentication and Session Management“ sowie „Insecure Communications“.

Im dritten Kapitel werden weitere typische Schwachstellen in Webanwendung vorgestellt und Maßnahmen zur Absicherung mittels JSF und Java EE-Technologien diskutiert. Es wird festgestellt, dass die Java Server Faces Technologie Webentwickler hinsichtlich der Validierung in Eingabedaten und Enkodierung von Ausgabedaten unterstützt, um Cross Site Scripting und Injection Schwachstellen zu begegnen. Ferner bietet die Java EE Plattform weitere Technologien und Konzepte an, um sich gegen typische Schwachstellen in Web- und EJB-Applikationen abzusichern, etwa stark typisierte APIs für den Zugriff auf Backendsysteme, objektrelationale Mapping Frameworks, das in Kapitel 2 vorgestellte Rollen- und Rechtekonzept sowie kryptographische Funktionen für Datenübertragung (z.B. SSL) und Datenhaltung (z.B. Hashfunktionen für indirekte Objektreferenzen).

Zusammenfassend kann bezüglich der Sicherheit im JSF- und Java EE-Umfeld gesagt werden, dass diese Technologien diverse Konzepte und Mittel bereitstellen, um Webanwendungen gegenüber typischen und häufig auftretenden Sicherheitslücken abzusichern.

Anhang

1 Blacklists potentiell gefährlicher Zeichen für Interpreter

SQL

Zeichen	Erläuterung
'	String
%	Wildcards
–	
[Escape-Zeichen in MS SQL
) (Verknüpfungen
@	Funktion oder Variable, min. in MS SQL
;	Kommandokonkatenation
+	Textkonkatenation
=	Vergleichsoperatoren
<	
>	
# -- /*	Kommentar-Einleitungen
\0 \r \n \t \h	weitere potentiell gefährliche Zeichen

Tabelle 3: Blacklist potentiell gefährlicher Zeichen für SQL-Interpreter (nach [BSI06, S. 21])

Systemaufrufe

Zeichen	Erläuterung
'	Parameterübergabe
"	
`	Kommandoaufruf
	Kommandoaufruf, Pipelining
>	Redirection, Ausgabe
<	Redirection, Eingabe
* ?	Wildcards
;	Kommandokonkatenation
\$	Variablennamen
.	Path Traversal
&	Kommandokonkatenation

Zeichen	Erläuterung
! () \0 \r \n	weitere potentiell gefährliche Zeichen

Tabelle 4: Blacklist potentiell gefährlicher Zeichen in Systemaufrufen (nach [BSI06, S. 21f])

LDAP und LDAP Metazeichen

Zeichen	Erläuterung
* = () & "	Problematische Zeichen in LDAP-Anfragen
; <= <= ~= :	Problematische LDAP-Metazeichen

Tabelle 5: Blacklist potentiell gefährlicher Zeichen für Verzeichnisdienst-Interpreter (nach [BSI06, S. 22])

XML-Dokumente

Zeichen	Erläuterung
<	Beginnzeichen eines Tags
>	Endezeichen eines Tags
/	Sonderzeichen in schließenden und Empty-Element-Tags
' "	Einfaches und doppeltes Hochkomma umschließt Attributwerte
=	Sonderzeichen zwischen Attributname und -wert

Tabelle 6: Blacklist potentiell gefährlicher Zeichen in XML-Dokumenten [OWAGuide, S. 183]

Weitere problematische Zeichen

Zeichen	Erläuterung
0x0 bis 0x19	nicht druckbare Zeichen
\0 bzw. %0	Null-Byte
\r bzw. %0a	Carriage Return
\n bzw. %0d	Line Feed
\t bzw. %09	Tabulator

Tabelle 7: Blacklist weiterer potentiell gefährlicher Zeichen für diverse Interpreter (nach [BSI06, S. 22])

Literatur- und Quellenverzeichnis

- [ApaMyF] The Apache Software Foundation: Apache MyFaces
Online im Internet
<http://myfaces.apache.org/>
- [ApaStr] The Apache Software Foundation: Struts
Online im Internet
<http://struts.apache.org/>
- [ApaStr1Wri] The Apache Software Foundation:
Online im Internet
<http://struts.apache.org/1.3.10/struts-taglib/tagreference.html#bean:write>
- [ApaStr2Pro] The Apache Software Foundation: Apache Struts 2 Documentation - property
Online im Internet
<http://struts.apache.org/2.1.6/docs/property.html>
- [ApaStrVal] The Apache Software Foundation: Apache Struts 2 Documentation - Validation
Online im Internet
<http://struts.apache.org/2.1.6/docs/validation.html>
- [BSI06] Bundesamt für Sicherheit in der Informationstechnik: Sicherheit von Webanwendungen
Online im Internet
<http://www.bsi.bund.de/literat/studien/websec/WebSec.pdf>
- [Gar05] Garret, J. J.: Ajax: A New Approach to Web Applications
Online im Internet
<http://www.adaptivepath.com/ideas/essays/archives/000385.php>
- [Grz07] Grzelak, D.: Log Injection Attack and Defence
Online im Internet
<http://www.sift.com.au/assets/downloads/SIFT-Log-Injection-Intelligence-Report-v1-00.pdf>
- [HDIHDI] hdiv.org: HTTP Data Integrity Validator
Online im Internet
<http://www.hdiv.org/>
- [HeiSec06a] Heise Zeitschriften Verlag: Datenleck beim StudiVZ? [Update]
Online im Internet
<http://www.heise.de/security/Datenleck-beim-StudiVZ-Update--/news/meldung/81373>
- [HeiSec06b] Heise Zeitschriften Verlag: StudiVZ nach Attacken und Umbau wieder online
Online im Internet
<http://www.heise.de/security/StudiVZ-nach-Attacken-und-Umbau-wieder-online--/news/meldung/82089>
- [HeiSec08] Heise Zeitschriften Verlag: Zugriff auf gesperrte StudiVZ-Fotoalben möglich [Update]
Online im Internet
<http://www.heise.de/security/Zugriff-auf-gesperrte-StudiVZ-Fotoalben-moeglich-Update--/news/meldung/114974>
- [ICEICE] ICESOFTECHNOLOGIES Inc.: ICEfaces
Online im Internet
<http://www.icefaces.org/main/home/index.jsp>

- [JavGla] java.net: glassfish
Online im Internet
<https://glassfish.dev.java.net/>
- [JavJSF] java.net: javaserverfaces
Online im Internet
<https://jvaserverfaces.dev.java.net/>
- [JavWoo] java.net: woodstock
Online im Internet
<https://woodstock.dev.java.net/>
- [Kle05] Klein, A.: DOM Based Cross Site Scripting or XSS of the Third Kind
Online im Internet
<http://www.webappsec.org/projects/articles/071105.shtml>
- [MicSSP] Microsoft Corporation: Transact-SQL Reference (SQL Server 2000) - xp_cmdshell
Online im Internet
<http://msdn.microsoft.com/en-us/library/aa260689.aspx>
- [Mit07] Christey, S., Martin, R. A.: Vulnerability Type Distributions in CVE
Online im Internet
<http://cve.mitre.org/docs/vuln-trends/vuln-trends.pdf>
- [NieUsa] Nielsen, J.: Ten Usability Heuristics
Online im Internet
http://www.useit.com/papers/heuristic/heuristic_list.html
- [OraTop] Oracle Corporation: Oracle TopLink
Online im Internet
<http://www.oracle.com/technology/products/ias/toplink/index.html>
- [OSyXWo] OpenSymphony: XWork - XWork Documentation
Online im Internet
<http://www.opensymphony.com/xwork/>
- [OWA07a] OWASP Foundation: OWASP Top 10 2007
Online im Internet
http://www.owasp.org/index.php/Top_10_2007
- [OWA07b] OWASP Foundation: OWASP Top 10 for Java EE
Online im Internet
https://www.owasp.org/images/8/89/OWASP_Top_10_2007_for_JEE.pdf
- [OWAEnt] OWASP Foundation: OWASP Enterprise Security API
Online im Internet
<http://www.owasp.org/index.php/ESAPI>
- [OWAEntAPI] OWASP Foundation: OWASP Enterprise Security API - JavaDoc
Online im Internet
http://owasp-esapi-java.googlecode.com/svn/trunk_doc/index.html
- [OWAGuide] OWASP Foundation: OWASP Guide Project
Online im Internet
http://www.owasp.org/index.php/Category:OWASP_Guide_Project

- [RedHib] Red Hat Middleware, LLC.: Hibernate
Online im Internet
<http://hibernate.org/>
- [RedJBo] Red Hat, Inc.: JBoss Application Server
Online im Internet
<http://www.jboss.org/jbossas/>
- [Sey06] Seyffer, D.: Web Applikationssicherheit
Vorlesung "Sicherheit", Berufsakademie Mosbach, Sommersemester 2006
- [Shn87] Shneiderman, B.: Designing the user interface
Addison Wesley Reading, Massachusetts 1987
- [Sun02] Sun Microsystems, Inc.: Designing Enterprise Applications with the J2EE Platform, Second Edition
Online im Internet
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html
- [Sun06a] Sun Microsystems, Inc.: Java Platform, Enterprise Edition (Java EE) Specification, v5
Online im Internet
<http://jcp.org/en/jsr/detail?id=244>
- [Sun06b] Sun Microsystems, Inc.: JavaServer Faces Specification, Version 1.2 Final Draft
Online im Internet
<http://www.jcp.org/en/jsr/detail?id=252>
- [Sun06c] Sun Microsystems, Inc.: Enterprise JavaBeans™ 3.0
Online im Internet
<http://jcp.org/en/jsr/detail?id=220>
- [Sun07] Sun Microsystems, Inc.: The Java EE 5 Tutorial
Online im Internet
<http://java.sun.com/javaee/5/docs/tutorial/doc/>
- [SunGla] Sun Microsystems, Inc.: Sun GlassFish Enterprise Server
Online im Internet
<http://www.sun.com/software/products/appsrvr/>
- [SunJEEFIt] Sun Microsystems, Inc.: Java EE 5 API - Interface Filter
Online im Internet
<http://java.sun.com/javaee/5/docs/api/javax/servlet/Filter.html>
- [SunJsePrm] Sun Microsystems, Inc.: Permissions in the Java 2 Standard Edition Development Kit (JDK)
Online im Internet
<http://java.sun.com/j2se/1.5.0/docs/guide/security/permissions.html>
- [SunJSTL] Sun Microsystems, Inc.: JavaServer Pages Standard Tag Library
Online im Internet
<http://java.sun.com/products/jsp/jstl/>
- [SunJSTLOut] Sun Microsystems, Inc.: JSTL core - Tag out
Online im Internet
<http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/c/out.html>

- [SunSqlCal] Sun Microsystems, Inc.: Interface CallableStatement
Online im Internet
<http://java.sun.com/javase/6/docs/api/java/sql/CallableStatement.html>
- [SunSqlPre] Sun Microsystems, Inc.: Interface PreparedStatement
Online im Internet
<http://java.sun.com/javase/6/docs/api/java/sql/PreparedStatement.html>
- [SunSrvRsp] Sun Microsystems, Inc.: Interface HttpServletResponse
Online im Internet
<http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpServletResponse.html>
- [WASCXSS] Web Application Security Consortium: Cross-site Scripting
Online im Internet
http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- [WirWir] Wireshark Foundation: Wireshark
Online im Internet
<http://www.wireshark.org/>