

INSTITUT FÜR INFORMATIK  
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Masterarbeit

# Härtungs- und Sicherheitskonzepte für Web- und Applikationsserver

Christian Simon





Masterarbeit

# Härtungs- und Sicherheitskonzepte für Web- und Applikationsserver

Christian Simon

Aufgabensteller: PD Dr. Helmut Reiser

Betreuer: Tobias Adolph  
Felix von Eye  
Rosa Freund  
Dr. Wolfgang Hommel

Abgabetermin: 12. März 2014



Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 12. März 2014

.....  
*(Christian Simon)*



## Abstract

Während der letzten Jahre hat das Internet einen enormen Bedeutungsgewinn erfahren. Dies führte jedoch auch dazu, dass es für kriminelle Aktivitäten immer interessanter wurde. Daher werden im Rahmen dieser Arbeit Maßnahmen untersucht, welche Internetdienstanbieter treffen können, um die Sicherheit von Web- und Applikationsservern zu verbessern. Betrachtet werden dabei die gängigen Produkte „Virtuelle Maschinen“, „Betreutes Hosting“ und „Betreute Webanwendungen“.

Nach Betrachtung von Angriffsszenarien und der derzeitigen Situation der entsprechenden Dienstleistungen am Leibniz-Rechenzentrum (LRZ), werden durch die dabei entdeckten Sicherheitsrisiken allgemeine Anforderungen an mögliche Lösungsansätze aufgestellt. Anschließend werden konkrete Maßnahmen betrachtet und anhand der Anforderungen bewertet. Durch Auswahl von geeigneten Lösungsansätzen wird ein individueller Maßnahmenkatalog für das LRZ erstellt.

Die praktische Umsetzbarkeit des Kataloges wird durch eine prototypische Implementierung von einzelnen Maßnahmen nachgewiesen: Für das Produkt „Virtuelle Maschinen“ wird die sichere Einrichtung eines Applikationsservers durch das Konfigurationsmanagement-Tool *Puppet* gezeigt. Mit *webapp\_discover* wird im Laufe der Arbeit ein Skript entwickelt, welches die installierten Webanwendungen beim Produkt „Betreutes Hosting“ ermitteln kann. Schließlich wird vorgeführt, wie mit *Docker* eine automatisierte Einspielung von Aktualisierungen beim Produkt „Betreute Webanwendungen“ durchgeführt werden kann.





# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Zielsetzung und -gruppe . . . . .	2
1.2	Vorgehen . . . . .	3
<b>2</b>	<b>Technische Grundlagen</b>	<b>5</b>
2.1	Webarchitektur . . . . .	5
2.2	Serverseitiger Aufbau . . . . .	6
2.2.1	Hardware / virtualisierte Hardware . . . . .	6
2.2.2	Betriebssystem . . . . .	7
2.2.3	Laufzeitumgebung . . . . .	8
2.2.4	Webanwendung . . . . .	8
2.2.5	Netzanbindung . . . . .	8
2.3	Abgrenzung Web- und Applikationsserver . . . . .	9
2.3.1	Webserver . . . . .	9
2.3.2	Applikationsserver . . . . .	11
2.4	Produkte . . . . .	11
2.4.1	Virtuelle Maschinen . . . . .	12
2.4.2	Betreutes Hosting . . . . .	12
2.4.3	Betreute Webanwendungen . . . . .	13
2.5	IT-Sicherheit . . . . .	13
2.5.1	ISO/OSI Sicherheitsarchitektur . . . . .	13
2.5.2	ISO/IEC 27001:2005 . . . . .	15
2.5.3	IT-Grundschutz . . . . .	16
2.6	Angriffsvektoren . . . . .	17
2.6.1	DoS & DDoS . . . . .	17
2.6.2	Unsichere bzw. bekannte Passwörter . . . . .	18
2.6.3	Bekannte Sicherheitslücken veralteter Software . . . . .	19
2.6.4	Verwundbare Konfiguration . . . . .	20
2.6.5	Verwundbare Webapplikationen . . . . .	20
<b>3</b>	<b>Anforderungsanalyse</b>	<b>23</b>
3.1	Situation am LRZ . . . . .	23
3.1.1	Virtuelle Maschinen . . . . .	23
3.1.2	Betreutes Hosting . . . . .	24
3.1.3	Analyse der Sicherheitsvorfälle im Jahr 2013 . . . . .	26
3.2	Anforderungen . . . . .	28
3.2.1	Prävention . . . . .	29
3.2.1.1	Sichere Passwortrichtlinien . . . . .	29
3.2.1.2	Regelmäßige Wartung von Webanwendungen . . . . .	29
3.2.1.3	Automatisierte Installation und Aktualisierung . . . . .	29
3.2.1.4	Dokumentation & Schulung . . . . .	29

3.2.1.5	Verhinderung von unberechtigten Datenverkehr . . . . .	30
3.2.1.6	Minimale Berechtigungen für Webanwendungen . . . . .	30
3.2.2	Detektion von Verwundbarkeiten . . . . .	30
3.2.2.1	Entdeckung von anfälliger/veralteter Software . . . . .	30
3.2.2.2	Ermittlung von schwachen Passwörtern . . . . .	30
3.2.3	Detektion von Angriffen . . . . .	30
3.2.3.1	Meldung durch Dritte . . . . .	31
3.2.3.2	Auffälligkeiten im Datenverkehr . . . . .	31
3.2.3.3	Beobachtung des Dateisystems . . . . .	31
3.2.4	Reaktion auf Angriffe . . . . .	31
3.2.4.1	Isolation von Mandanten . . . . .	31
3.2.4.2	Limitierung von Ressourcen . . . . .	32
3.2.4.3	Automatisierte Deaktivierung . . . . .	32
3.2.5	Überblick . . . . .	32
<b>4</b>	<b>Lösungsmöglichkeiten</b>	<b>35</b>
4.1	Organisatorische Maßnahmen . . . . .	35
4.1.1	Festlegung einer Passwortrichtlinie . . . . .	35
4.1.2	Zeitliche Befristung von Produkten . . . . .	36
4.1.3	Dokumentation für Kunden . . . . .	37
4.1.4	Berücksichtigung von externen Abuse-Datenbanken . . . . .	38
4.2	Betriebssystem . . . . .	39
4.2.1	Einschränkung von Dateiberechtigungen . . . . .	39
4.2.2	Ausführung von Diensten in einem Chroot . . . . .	41
4.2.3	Linux Containers (LXC) . . . . .	43
4.2.4	Docker . . . . .	44
4.2.5	Mandatory Access Control mit <i>AppArmor</i> . . . . .	47
4.3	Webserver . . . . .	49
4.3.1	Einsatz einer Web Application Firewall (WAF) . . . . .	49
4.3.2	Einsatz von HTTPS . . . . .	50
4.3.3	Sicherheitsrisiken bei Webanwendungen . . . . .	52
4.3.3.1	Lösungen auf Dateisystemebene . . . . .	53
4.3.3.2	Lösungen mit Zugriff über HTTP . . . . .	55
4.4	Applikationsserver . . . . .	55
4.4.1	Absicherung der Konfiguration von <i>Apache Tomcat</i> . . . . .	55
4.4.2	Absicherung von <i>Apache Tomcat</i> mit <i>AppArmor</i> . . . . .	58
4.5	Infrastruktur . . . . .	62
4.5.1	Konfigurationsmanagement mit <i>Puppet</i> . . . . .	62
4.5.2	Firewall . . . . .	66
4.5.3	Intrusion-Detection-System (IDS) . . . . .	67
4.5.4	Zentrale Logauswertung . . . . .	68
4.6	Gegenüberstellung mit Anforderungen . . . . .	70
4.7	Auswahl von Lösungsansätzen für das LRZ . . . . .	73
4.7.1	Virtuelle Maschinen . . . . .	73
4.7.2	Betreutes Hosting . . . . .	74

<b>5</b>	<b>Prototypische Implementierung am LRZ</b>	<b>77</b>
5.1	Sichere Konfiguration von <i>Apache Tomcat</i> durch <i>Puppet</i> . . . . .	77
5.1.1	Installation der nötigen Pakete . . . . .	79
5.1.2	Erstellung eines Zugangs zur Administration . . . . .	80
5.1.3	Aktivierung des HTTPS-Zugriffes . . . . .	82
5.1.4	<i>AppArmor</i> -Profil für den <i>Tomcat</i> -Server aktivieren . . . . .	83
5.1.5	Test des <i>AppArmor</i> -Profils . . . . .	86
5.2	Installierte Webanwendungen detektieren mit <i>webapp_discover</i> . . . . .	87
5.2.1	Implementierung von <i>webapp_discover</i> . . . . .	88
5.2.2	Durchführung eines Scans beim LRZ . . . . .	91
5.2.3	Statistiken zum Webhosting am LRZ . . . . .	92
5.2.4	Erzeugen der Definition einer Webanwendung . . . . .	96
5.3	„Betreute Webanwendungen“ mit <i>Docker</i> . . . . .	98
5.3.1	<i>Docker</i> installieren . . . . .	98
5.3.2	Erstellung eines Base-Images für <i>Debian Wheezy</i> . . . . .	99
5.3.3	Erstellung eines Images für den LAMP-Stack . . . . .	100
5.3.4	Erstellung des <i>Wordpress</i> Images . . . . .	102
5.3.5	Veröffentlichung von <i>Docker</i> -Images in der Registry . . . . .	103
5.3.6	Erzeugung von <i>Wordpress</i> -Container . . . . .	105
5.3.7	Upgrade einer <i>Wordpress</i> -Instanz . . . . .	105
<b>6</b>	<b>Fazit und Ausblick</b>	<b>109</b>
6.1	Zusammenfassung der Ergebnisse . . . . .	109
6.2	Schritte zur Umsetzung am LRZ . . . . .	109
6.3	Ausblick auf Folgearbeiten . . . . .	110
	<b>Abbildungsverzeichnis</b>	<b>113</b>
	<b>Literatur</b>	<b>115</b>
	<b>Inhalt der beigelegten CD</b>	<b>123</b>



# 1 Einleitung

Durch den enormen Bedeutungsgewinn des Internets in den letzten Jahren wird es auch zunehmend Ziel krimineller Aktivitäten. Ein herausragendes Beispiel für solche kriminellen Aktivitäten ist ein Sicherheitsvorfall bei der Firma Sony im Jahr 2011: Die Online-Dienste *Sony PlayStation Network* und *Qriocity* des Herstellers von Spielekonsolen wurden durch Unbekannte angegriffen und diese erbeuteten Kundendaten von 77 Millionen Benutzern. [Jur11]

Als erste Reaktion auf den Angriff, der zwischen dem 17. und 19. April 2011 erfolgte, schaltete Sony das *Sony PlayStation Network* am 20. April 2011 ab. Dieser Dienst ermöglicht Besitzern von PlayStation-Spielekonsolen sowohl das Spielen mit anderen, als auch den Bezug neuer digitaler Inhalte über das Internet. Dies war durch die Abschaltung nun nicht mehr möglich. In den ersten Mitteilungen an die Kunden sprach Sony lediglich von einem ungeplanten Ausfall. Erst am 26. April wurden die Kunden über den Einbruch sowie das Abhandenkommen von persönlichen Daten informiert. Neben dem *Sony PlayStation Network* war auch der Musikstreaming-Dienst *Qriocity* betroffen. [Sey11]

Die unbekanntes Angreifer erhielten Zugriff auf persönliche Kundendaten wie Name, Adresse, Geburtsdatum, Benutzername und Passwort. Außerdem konnte nicht ausgeschlossen werden, dass weitere Daten aus dem Kundenprofil - dazu gehören Kaufhistorie, Rechnungsanschriften und Sicherheitsfragen - ebenfalls entwendet wurden. Ebenso konnten die Angreifer auf Kreditkartendaten zugreifen, sofern diese im Kundenprofil hinterlegt waren, was bei 12,3 Millionen Nutzern der Fall war.

Welche Daten die Einbrecher letztendlich abgegriffen hatten, konnten auch hinzugezogene externe Experten nach einer umfangreichen forensischen Untersuchung nicht feststellen. Es wurde lediglich herausgefunden, dass mehrere Anfragen an die Datenbank des Dienstes Antworten von erheblicher Größe erzeugt hatten und diese anschließend nach außen übertragen wurden. Da bei Kreditkartenfirmen kein signifikanter Anstieg der Missbrauchsfälle seit diesem Einbruch aufgetreten ist, wird jedoch davon ausgegangen, dass die gespeicherten Kreditkartendaten nicht in fremde Hände gelangt sind.

Über den genauen Ablauf des Angriffs wurde nichts bekannt. Es wurde lediglich festgestellt, dass der Angreifer auf zehn Servern durch sog. Privilege Escalations volle Rechte auf den jeweiligen Systemen erlangte. Mit deren Hilfe wurde u.a. durch Manipulationen an Logdateien versucht, die Kompromittierung der Systeme möglichst lange vor den Systemadministratoren zu verbergen. [Hir11]

Es existieren keine Angaben über den durch den Angriff entstandenen Schaden, jedoch ging Sony einen Monat nach den Ereignissen von einem geschätzten Schaden von 171 Mio. \$ aus. [Hac11]

Anhand des Vorfalls bei Sony lassen sich die Gefahren, denen Webapplikationen ausgesetzt sind, ganz gut veranschaulichen. Doch auch wesentlich kleinere Webdienste sind vor diesen nicht geschützt. Während bei den großen Diensten meist Spezialisten und deren Sachverstand und Erfahrung nötig sind, wird bei kleinen Webapplikationen häufig automatisiert angegriffen. Oftmals kommt dabei den Angreifern zu Gute, dass solche Projekte nach der Erstellung der entsprechenden Webseiten nicht ausreichend gewartet werden. Ebenso werden dort die grundlegendsten Sicherheitsstandards missachtet, wie z.B. durch die Verwendung von zu

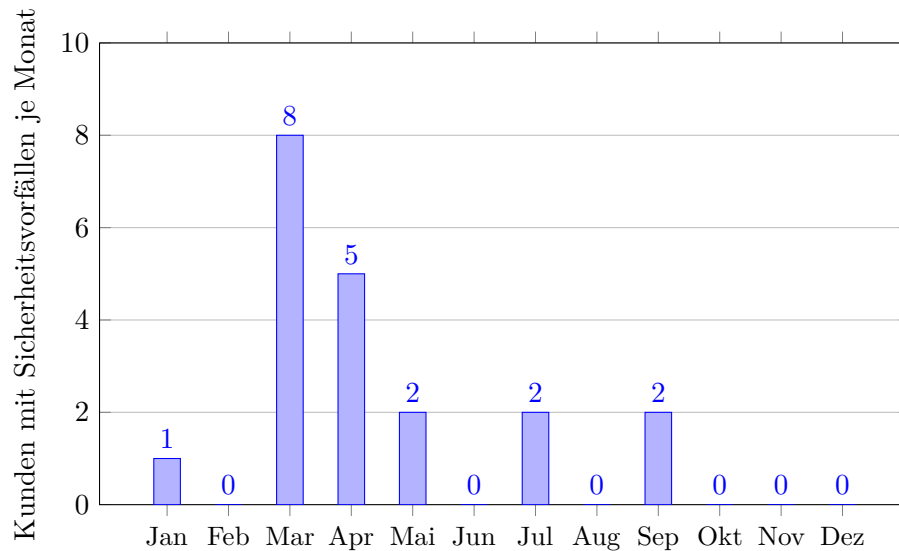


Abbildung 1.1: Sicherheitsvorfälle Webhosting am LRZ im Jahr 2013 (Datenquelle: [LRZ13d])

einfachen Passwörtern. Auf diese Ursachen lassen sich auch die meisten Sicherheitsvorfälle beim Webhosting am Leibniz-Rechenzentrum (LRZ) zurückführen, die im Jahr 2013 entdeckt wurden (vgl. Abschnitt 3.1.3). In Abbildung 1.1 sind diese Sicherheitsvorfälle dargestellt. Erfahrungsgemäß werden nicht alle Vorkommnisse durch das Team des LRZs erkannt bzw. diesem gemeldet. Daher ist davon auszugehen, dass die Dunkelziffer höher liegt.

### 1.1 Zielsetzung und -gruppe

Auf Grund dieser hohen Gefährdung von Webdiensten soll im Rahmen dieser Arbeit ein Härtings- und Sicherheitskonzept für Webanwendungen erstellt werden. Dieses Konzept soll Internetdiensteanbieter bei der Absicherung ihrer Produkte unterstützen. Es wird dabei auf diese Produktvarianten eingegangen:

- **Betreute Webanwendungen**

Der Anbieter stellt seinen Kunden eine betriebsfertige Installation einer Webanwendung zur Verfügung. Die Wartung der Anwendung fällt in den Zuständigkeitsbereich des Anbieters.

- **Betreutes Hosting**

Der Anbieter stellt seinen Kunden eine Laufzeitumgebung zur Verfügung, um eigene Webanwendungen, die mit der jeweiligen Laufzeitumgebung kompatibel sind, zu betreiben. Dabei kümmert sich ausschließlich der Kunde um die Wartung der Anwendungen.

- **Virtuelle Maschinen**

Der Anbieter stellt dem Kunden lediglich (virtuelle) Ressourcen zur Verfügung. Der Kunde muss sich um Betriebssystem, Laufzeitumgebung und Webanwendung kümmern.

Dieses Konzept soll für jedes der Produkte eine möglichst optimale Absicherung gegenüber den Gefahren, welche beim Betrieb von Webapplikationen darauf bestehen, ermitteln. Das Konzept soll generell allgemein gültig sein, jedoch in gewissen Teilen auf die speziellen Anforderungen am Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften eingehen.

Die Arbeit behandelt dabei nicht die Konzeption/Programmierung von sicheren Webapplikationen. Sie beschränkt sich darauf, bestehende Webapplikationen, auch wenn diese gewisse Sicherheitsdefizite aufweisen, als Provider möglichst sicher zu betreiben.

### 1.2 Vorgehen

Die Gliederung der Arbeit stellt sich folgendermaßen dar: In Kapitel 2 werden die technischen Hintergründe besprochen, auf die dann später zurückgegriffen werden kann. Dies umfasst sowohl eine Klassifizierung von denkbaren Angriffsvektoren, als auch eine genauere Einteilung der Produkte von Internetdiensteanbietern.

Anschließend wird im Kapitel 3 die derzeitige Situation am LRZ dargestellt. Insbesondere werden die dortigen Sicherheitsvorfälle analysiert. Aus diesen Betrachtungen sowie den technischen Hintergründen im vorigen Kapitel werden die Anforderungen entwickelt, welche mögliche Lösungen zur Verbesserung der Sicherheit von Web- und Applikationsservern erfüllen müssen.

Danach werden in Kapitel 4 Lösungsansätze ausführlich erläutert und hinsichtlich der Anforderungen bewertet. Zudem erfolgt eine Zuordnung zu den geeigneten Produkten von Providern. Daraus werden schließlich die Maßnahmen ausgewählt, welche am LRZ umgesetzt werden sollen. Dabei werden die besonderen Begebenheiten am LRZ berücksichtigt.

Um zu zeigen, dass eine praktische Umsetzung der Lösungsvorschläge möglich ist, werden im Kapitel 5 drei Lösungsansätze prototypisch implementiert.

Das Kapitel 6 fasst die Erkenntnisse der Arbeit zusammen und wagt einen Ausblick, inwiefern sich die Produkte und Aufgaben eines Providers zukünftig entwickeln könnten.





## 2 Technische Grundlagen

In diesem Kapitel werden die nötigen technischen Grundlagen eingeführt, die im Fortgang der Arbeit immer wieder Grundvoraussetzungen für die jeweilig diskutierten Themen darstellen.

Zuerst wird im Abschnitt 2.1 die Webarchitektur, welche die im Rahmen der Arbeit zu untersuchende serverseitige Komponente beinhaltet, erläutert. Im anschließenden Abschnitt 2.2 wird ein detaillierter Blick auf den Aufbau dieser Komponente geworfen. Zwischen den Server-Komponenten Web- und Applikationsserver wird in Abschnitt 2.3 differenziert. In Abschnitt 2.4 werden dann typische Produkte zusammengestellt, wie sie von Internetdiensteanbieter – u.a. auch vom LRZ – angeboten werden. Der Abschnitt 2.5 behandelt IT-Sicherheit im Allgemeinen. Nachfolgend werden in Abschnitt 2.6 mögliche Schwachstellen und Angriffsvektoren der serverseitigen Systeme aufgezählt und bewertet.

### 2.1 Webarchitektur

Die Kommunikation zwischen Webapplikationen und den Endnutzern findet im Allgemeinen über das Hypertext Transfer Protocol (HTTP) statt. Dieses Protokoll ist nach dem TCP/IP-Schichtenmodell der Anwendungsschicht zuzuordnen. Es nutzt in der Regel das TCP-Protokoll zum Transport der Daten. Es kommt das Client-Server-Modell zum Einsatz, indem sich der Endnutzer als Client zum Server verbindet und dort mittels eines oder mehrerer Requests bestimmte Daten anfragt, welche der Server mit einem Response an den Client ausliefert. Als Client wird in der Regel ein Browser verwendet, welcher aus der vom Nutzer eingegebenen URL über das Domain Name System (DNS) ermittelt, welcher Server zuständig ist.

Serverseitig wartet ein sog. Webserver auf die Anfragen der Clients und liefert diesem die angeforderten Daten aus. Es kann sich bei diesen Daten um statische Dateien oder auch um je nach Aufruf dynamisch generierte Inhalte handeln. Dynamische Inhalte werden meist von Webanwendungen auf Anfrage des Webserver generiert und dann durch den Webserver ausgeliefert. Ein Sequenzdiagramm für eine beispielhafte Anfrage an einen Webserver ist in Abbildung 2.1 skizziert. Dabei ruft der Client mit einem Request die Seite `/index.html` ab. Nachdem der Server durch einen Response den Inhalt der Seite geliefert hat, erkennt der Client, dass zum vollständigen Anzeigen der Seite noch das Bild `/images/image.png` benötigt wird. Also wird auch dieses mit einem Request beim Server angefragt. An diesem Beispiel kann man auch leicht erkennen, dass HTTP keine persistente Sitzung besitzt. Jeder Request wird somit unabhängig vom vorherigen betrachtet. Deshalb wird HTTP auch als ein zustandsloses Protokoll bezeichnet. Wird eine Sitzungsverwaltung benötigt, muss diese dann ggf. durch die Anwendungsschicht implementiert werden. [RFC2616]

Das gilt analog für einen Applikationsserver, welcher als Schnittstelle zur Außenwelt einen Webserver enthalten kann. Die Unterscheidung der beiden Servertypen wird im Abschnitt 2.3 behandelt.

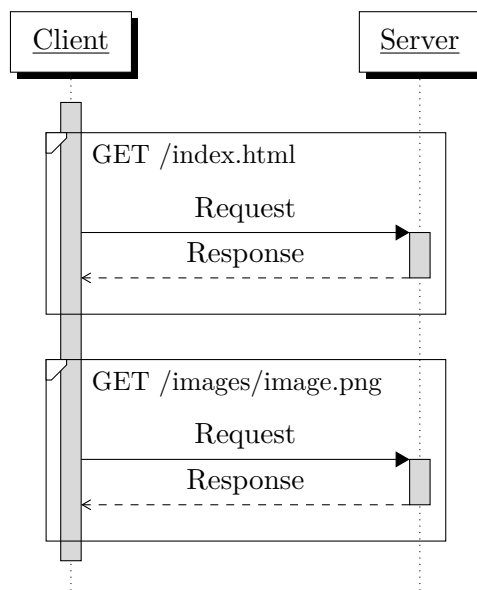


Abbildung 2.1: Sequenzdiagramm HTTP-Aufruf

## 2.2 Serverseitiger Aufbau

Es wird nun die typische Architektur beim Betrieb von Web- und Applikationsservern betrachtet. Dies dient zur Klärung der Zuständigkeiten im Provider-Kunden-Verhältnis, welches anschließend im Abschnitt 2.4 besprochen wird.

Der serverseitige Aufbau ist in Abbildung 2.2 skizziert. Ganz außen steht dabei die entsprechende Hardware, die ggf. durch einen Hypervisor virtualisiert sein kann. Ein Hypervisor kann die physischen Ressourcen so verwalten, dass mehrere Betriebssysteme auf einem Rechner ausgeführt werden können.

Auf der (virtualisierten) Hardware wird das Betriebssystem ausgeführt, welches sich dann bekanntlich um die Verwaltung der zur Verfügung stehenden Ressourcen kümmert. Auf dem Betriebssystem läuft dann die jeweilige Laufzeitumgebung ab. Diese besteht in der Regel aus einem Web- bzw. Applikationsserver, der die Anwendung ausführt. Durch eine Netzanbindung können sich zum einen die Clients zur Anwendung verbinden, zum anderen ermöglicht diese einen Zugriff auf externe Datenquellen.

In den nun folgenden Unterabschnitten werden diese einzelnen Komponenten detaillierter besprochen. Dabei wird auch ein besonderes Augenmerk auf Verpflichtungen und Aufgaben bei dem Betrieb des jeweiligen Bereiches geworfen.

### 2.2.1 Hardware / virtualisierte Hardware

Auf der untersten Ebene ist die Hardware angesiedelt. Sie stellt die für die Erbringung von Internet-Diensten nötigen grundlegenden Ressourcen zur Verfügung. Dabei kann auf der physikalischen Hardware mit Hilfe eines Hypervisors eine virtualisierte Hardware bereitgestellt werden. Diese verfügt über vielfältige Vorteile im Vergleich zur Verwendung von rein physikalischer Hardware. Gegen Ende dieses Abschnittes wird auf die Besonderheiten von virtualisierter Hardware detailliert eingegangen.

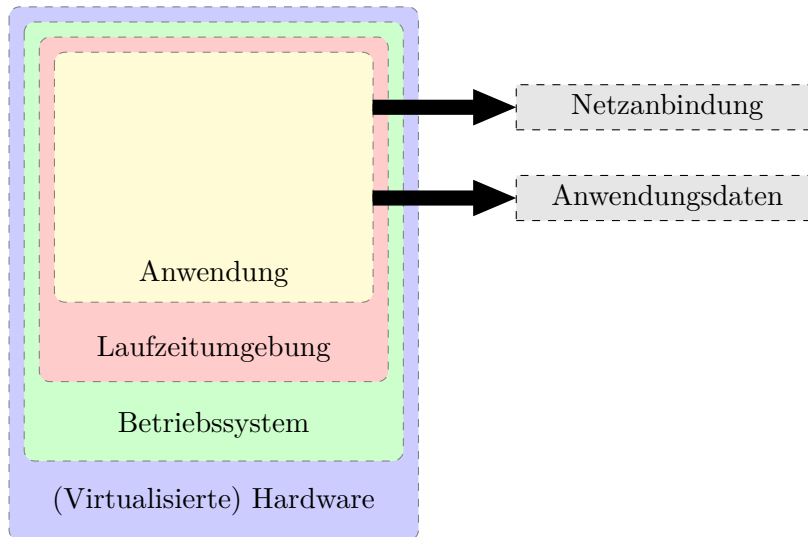


Abbildung 2.2: Überblick serverseitiger Aufbau

Der Internetdiensteanbieter hat bei dieser Hardware u.a. dafür zu sorgen, dass diese die nötige Verfügbarkeit bietet. Dazu ist es nötig, ausreichend Ersatzteile für ausgefallene Hardwarekomponenten, sowie zusätzliche Hardware zur aktuell Benötigten vorzuhalten, sofern die Kundendienste weitere Ressourcen benötigen. Durch ein umfassendes Monitoring muss sichergestellt sein, dass Fehlfunktionen zeitnah erkannt werden, so dass die Hardware ohne große Einflüsse für die Kunden getauscht werden kann. Eine weitere Herausforderung ist die Absicherung des physikalischen Zugriffs auf die Hardwarekomponenten. Zudem ist es nötig, dass ausschließlich entsprechend autorisierte Personen Zugriff erhalten.

Durch die bereits angesprochene Virtualisierung von Hardware wird eine weitere Abstraktionsebene zwischen den Ressourcen und den darauf laufenden Betriebssystemen eingeführt. Es ist dadurch möglich, dass auf einer Hardwaremaschine mehrere Betriebssysteme weitestgehend isoliert voneinander ausgeführt werden können. Durch den Betrieb von solchen virtualisierten Ressourcen muss der Provider weitere Aufgaben rund um den Hypervisor erfüllen. Zum einen muss die Kompatibilität der realen Hardware mit der Virtualisierungslösung sichergestellt werden, zum anderen ist auch ein umfangreiches Monitoring dieser nötig.

### 2.2.2 Betriebssystem

Die nächste Schicht des serverseitigen Aufbaus stellt das Betriebssystem dar. Es kümmert sich um die Verwaltung der Ressourcen, die durch physische bzw. virtualisierte Hardware bereitgestellt werden. Es handelt sich dabei beispielsweise um Arbeitsspeicher, Prozessor sowie Ein- und Ausgabegeräte. Diese Verwaltung ist erforderlich, da im Allgemeinen ein Betriebssystem mehrere Prozesse parallel ausführt und die Ressourcen immer nur einen Prozess zu einem gewissen Zeitpunkt zugewiesen werden können. [Tan92]

Ein Internetdiensteanbieter muss für die darauf laufenden Anwendungen ein passendes Betriebssystem auswählen. Dieses Betriebssystem muss kompatibel mit der eingesetzten (virtualisierten) Hardware sein. Da die Betriebssystem-Versionen nur für eine gewisse Zeit

von den Herstellern bzw. der Community gepflegt werden, sind umfangreiche Updates, in denen das komplette Betriebssystem aktualisiert wird, von Zeit zu Zeit nötig. Bei der Linux Distribution *Debian*, welche häufig auf Webservern benutzt wird, werden mindestens ein Jahr nach der Veröffentlichung einer neuen Version Sicherheitsupdates für den Vorgänger erstellt. Bei kommerziellen Betriebssystemen werden solche Updates im Allgemeinen deutlich länger gepflegt: *Red Hat Enterprise Linux 6* wird für 13 Jahre und *Microsoft Windows Server 2008* für knappe 12 Jahre nach Release gepflegt. [DST13] [Red13] [Mic13]

### 2.2.3 Laufzeitumgebung

Bei der serverseitigen Architektur folgt auf das Betriebssystem die Laufzeitumgebung, welche für die letztendliche Webanwendung erforderliche Werkzeuge und Bibliotheken zur Verfügung stellt. Ähnlich wie bei Betriebssystemen haben Programme, die zur Laufzeitumgebung zählen, bestimmte Releasezyklen und erfordern bisweilen Sicherheitsupdates. Diese Updates müssen regelmäßig und möglichst zeitnah nach deren Erscheinen eingespielt werden. Dabei kann es beim Upgrade auf eine neue Version auch vorkommen, dass Inkompatibilitäten mit den direkt benachbarten Komponenten Betriebssystem und Webanwendung auftreten.

Eine sehr gebräuchliche Laufzeitumgebung stellt der sogenannte AMP-Stack dar. Das Akronym steht dabei für den Webserver *Apache*, das Datenbankmanagementsystem *MySQL* und die Skriptsprache *PHP*. Die gängigen Desktop-Betriebssysteme *Windows*, *Linux* und *Mac OS X* werden mit entsprechend vorkonfigurierten Entwicklungsumgebungen wie z.B. *XAMPP* unterstützt. [AFr14]

Dieser Stack lässt sich somit für Entwickler einfach einrichten und daher erfreut sich *PHP* einer sehr weiten Verbreitung. Der sechste Platz für *PHP* im TIOBE-Community-Index vom November 2013 spricht ebenso dafür. Dieser Index versucht über die Anzahl an Suchmaschinentreffern für bestimmte Programmiersprachen die Verbreitung der einzelnen Sprachen festzustellen. [TIO13]

Eine weitere häufig eingesetzte Sprache ist auf Rang 2 des TIOBE-Community-Index *Java*. Da Programme für *Java* in einen plattformunabhängigen Bytecode übersetzt werden, wird zu dessen Ausführung ein *Java Runtime Environment* benötigt. Häufig wird bei *Java* ein Applikationsserver (vgl. Abschnitt 2.3.2) eingesetzt, welcher weitere nützliche Middleware zum Betrieb von Webanwendungen enthält.

### 2.2.4 Webanwendung

Auf der obersten Ebene der serverseitigen Architektur steht schließlich die Webanwendung. Sie benötigt zur Ausführungszeit eine passende Laufzeitumgebung. Auch bei der Webanwendung müssen sowohl Sicherheitsupdates als auch Releasezyklen berücksichtigt werden, um einen sicheren Betrieb zu gewährleisten. Die einwandfreie Funktionalität einer Webanwendung sollte über umfangreiche, automatisierte, funktionale Tests festgestellt werden, denn nur so lassen sich etwaige Probleme beim Betrieb der Webanwendung zeitnah erkennen. Solche Tests lassen sich z.B. mit der quelloffenen Testumgebung *Selenium* erstellen. [Sel14]

### 2.2.5 Netzanbindung

Eine weitere notwendige Komponente bei der serverseitigen Architektur stellt die Netzanbindung dar. Darüber werden alle externen Systeme erreicht, so dass ohne zuverlässige

Netzanbindung weder externe Datenquelle eingebunden werden, noch Clients auf die Webanwendung zugreifen können. Dazu ist es nötig, dass die Netzanbindung ausreichend Reserven vorweist, um auf einen plötzlichen Anstieg der Zugriffszeiten auf die entsprechenden Webanwendungen reagieren zu können.

Sollte durch externe Angriffe versucht werden, die Seite durch eine Vielzahl von Zugriffen zu überlasten (DoS & DDoS vgl. Abschnitt 2.6.1), muss das rechtzeitig durch den Internetdienstanbieter erkannt werden, damit er ggf. Gegenmaßnahmen einleiten kann.

### 2.3 Abgrenzung Web- und Applikationsserver

In diesem Abschnitt wird erklärt, inwiefern sich Web- und Applikationsserver unterscheiden. Während ein Webserver nur Dienste über das HTTP(S)-Protokoll anbietet, kann ein Applikationsserver Anfragen von Clients über mehrere Protokolle beantworten. Dadurch eignet sich ein Applikationsserver im Allgemeinen besser für Interaktion mit anderen Anwendungen. Insbesondere kann ein Applikationsserver auch als Webserver fungieren und Anfragen von Browsern bearbeiten. [Sin02]

#### 2.3.1 Webserver

Ein Webserver liefert lediglich Daten über HTTP bzw., falls Verschlüsselung mittels TLS eingesetzt wird, über HTTPS aus. Es kann sich dabei sowohl um statische Daten, welche direkt aus dem Dateisystem gelesen werden, als auch um dynamische Daten handeln. Bei dynamischen Daten gibt es mehrere Typen von Generatoren. Diese Typen lassen sich folgendermaßen kategorisieren. Es wird dabei jeweils ein Beispiel für die weit verbreitete Skriptsprache PHP vorgestellt. Zur besseren Differenzierung wird der Vorgang für die verschiedenen Varianten durch die Abbildung 2.3 dargestellt:

- **Generierung durch den Webserver**

Der Webserver generiert hier die Antwort selbst. Dazu können Webserver zumeist durch Module erweitert werden, welche dann neue Arten der dynamischen Seitenerzeugung hinzufügen. Für die Skriptsprache PHP gibt es das Modul *Mod-PHP*, welches dynamische Seiten aus PHP Skripten generiert.

Vorteil der direkten Einbindung in den Webserver ist der geringe Ressourcenverbrauch, sowie die relativ schnelle Generierung der dynamischen Inhalte. Allerdings besitzen so ausgeführte Skripte dieselben Rechte wie der Webserver, was in einer Multibenutzerumgebung problematisch ist. [Apa13b]

- **Generierung durch jeweils einen Prozessaufruf**

Hier ruft der Webserver bei jeder eingehenden Anfrage nach dynamischen Daten einen eigenen Prozess auf, welcher die Inhalte generiert, die dann vom Webserver ausgeliefert werden. Damit der aufgerufene Prozess Zugriff auf die Metadaten (z.B. Quelladresse, HTTP-Header, ...) der HTTP-Anfrage bekommt, liefert der Webserver dem Prozess diese nach dem Common Gateway Interface (CGI) Standard. [RFC3875]

Im Vergleich zur Generierung durch den Webserver kann dort nun ein Benutzer mit eingeschränkten Rechten verwendet werden, was eine bessere Absicherung darstellt. Allerdings erzeugt der zusätzliche Prozessaufruf auch einen gewissen Overhead, welcher

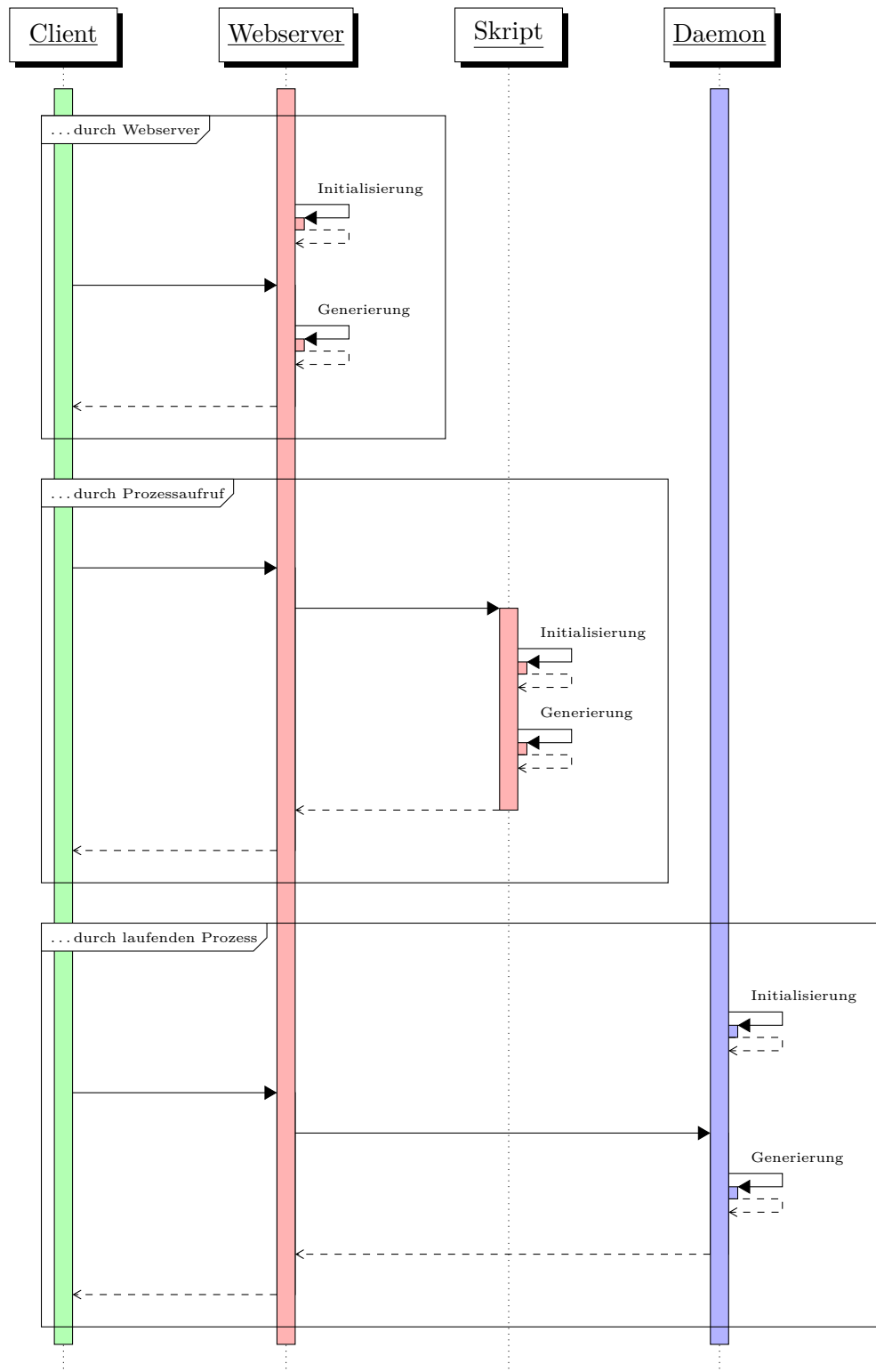


Abbildung 2.3: Generierung von dynamischen Inhalten bei Webservern ...

sich in der erhöhten Ausführungszeit zeigt. Im Falle von PHP wird das Skript nun direkt über das PHP-CGI Kommando aufgerufen.

- **Generierung durch einen bereits laufenden Prozess**

Eine weitere Alternative zur Einbindung von dynamischen Seiten stellt die Ausführung einer Skriptsprache oder auch einer ganzen Anwendung als eigener Prozess dar. Dieser Prozess kann mehrere Anfragen bearbeiten und wird unabhängig von einer einzelnen Anfrage gestartet und läuft im Hintergrund, wartend auf Anfragen, weiter.

Durch den eigenen Prozess können dessen Rechte wieder einfach eingeschränkt werden. Dabei bleiben die Ausführungszeiten ähnlich kurz wie bei der direkten Generierung durch den Webserver. Die Anfragen an den Prozess können durch die Protokolle FastCGI, HTTP oder auch Apache JServ Protocol (AJP) weitergereicht werden. Auch PHP kann als FastCGI Server im Hintergrund ausgeführt werden. Dafür bietet sich der FastCGI Process Manager (FPM) an, welcher bei erhöhter Auslastung weitere PHP Instanzen startet. [Bro96]

### 2.3.2 Applikationsserver

Ein Applikationsserver ist dagegen nicht nur auf die Verwendung des HTTP/HTTPS- Protokolls beschränkt. Er stellt allgemein Funktionen für entfernte Anwendungen über gewisse Schnittstellen zur Verfügung. Diese Definition schließt auch graphische Oberflächen, Webanwendungen und Programmierschnittstellen mit ein. Häufig werden Applikationsserver im Java Umfeld eingesetzt. Häufig eingesetzte Beispiele für Applikationsserver sind die Produkte *Apache Tomcat* und *WildFly* (ehemals *JBoss*). Ein Applikationsserver stellt im Gegensatz zu einem reinen Webserver auch weitergehende Verwaltungsfunktionen, wie z.B. Monitoring und Lifecycle Management zur Verfügung.

Bezüglich Sicherheits- und Performanceüberlegungen ist ein Applikationsserver mit der direkten Generierung von dynamischen Seiten beim Webserver vergleichbar, da der Applikationsserver innerhalb eines Prozesses ausgeführt wird. Daher wird eine kurze Antwortzeit auf Anfragen erreicht. Jedoch ist es schwierig, mehrere Instanzen voneinander zu isolieren. [Apa14]

## 2.4 Produkte

Dieser Abschnitt behandelt typische Produkte, welche durch Internetdiensteanbieter angeboten werden. Diese unterscheiden sich im Wesentlichen darin, wie die Zuständigkeiten zwischen Kunden und Provider verteilt sind. Zum besseren Überblick sind diese Zuständigkeiten aus der Sicht eines Providers in Tabelle 2.1 zusammengefasst. Bei allen Produkten ist der Provider für die Bereitstellung und Pflege von Netzanbindung und Hardware zuständig. In den anderen Teilbereichen hängt die Zuständigkeit vom Provider ab.

Die Einteilung in virtuelle Maschinen, Webhosting und Webanwendungen ist weitestgehend analog zu der Definition der Servicemodelle beim Cloud-Computing durch die US-Bundesbehörde *National Institute of Standards and Technology* (NIST). Dabei entsprechen virtuelle Maschinen Infrastructure-as-a-Service (IaaS), welcher lediglich den Zugriff auf virtualisierte Hardware erlaubt und die Ausführung eines Betriebssystems ermöglicht. Das Pendant zu Webhosting ist bei Cloud-Computing Platform-as-a-Service (PaaS). Bei beiden Produkten wird die Ausführung von Anwendungen für bestimmte Laufzeitumgebungen ermöglicht. Die

betreuten Webanwendungen entsprechen schließlich dem Modell Software-as-a-Service (SaaS). Diese Einteilung wird nun in den folgenden Unterabschnitten ausführlich vorgestellt. [MG11]

	Virtuelle Maschinen	Webhosting	Webanwendung
Hardware	ja	ja	ja
Betriebssystem	teilweise	ja	ja
Laufzeitumgebung	nein	ja	ja
Webanwendung	nein	nein	ja
Netzanbindung	ja	ja	ja
Zielgruppe	Administratoren	Entwickler	Endnutzer

Tabelle 2.1: Zuständigkeiten eines Providers bei den Produkten

### 2.4.1 Virtuelle Maschinen

Bei virtuellen Maschinen (VMs) erhält der Kunde vom Provider Zugriff zu virtuellen Ressourcen für Rechenoperationen, Hintergrund- und Arbeitsspeicher und Netzanbindung. Damit diese verwendet werden können, muss sich der Kunde auf diesen VMs ein Betriebssystem installieren.

Damit der Kunde eine VM verwalten kann, stellt ihm der Provider entsprechende Oberflächen zur Verfügung. Über diese kann der Kunde ihm zugewiesene Ressourcen ansprechen und ggf. Einstellungen anpassen. Als zusätzliche Dienstleistung wird durch Provider oft ein regelmäßiges Backup des Inhaltes im Hintergrundspeicher angeboten. Zur einfacheren Einrichtung können für Kunden Vorlagen erstellt werden, welche ein gewisse Grundfunktionalität zur Verfügung stellen.

### 2.4.2 Betreutes Hosting

Beim betreuten Hosting wird dem Kunden ein Datenspeicher auf einem Webserver zur Verfügung gestellt. Der Kunde hat lediglich eingeschränkten Zugriff auf die Konfiguration des Webserver. Dies ist nötig, damit der Kunde durch Konfigurationsfehler keinen Ausfall des Dienstes hervorrufen kann.

Technisch realisiert der Provider ein solches betreutes Hosting zumeist dadurch, dass er seine Infrastruktur mandantenfähig konfiguriert und sich somit in der Regel mehrere Kunden einen einzelnen Server teilen (sog. Shared Hosting). Die einzelnen Kunden können die entsprechenden Daten dann über ein Dateiübertragungsprotokoll wie z.B. FTP auf den entsprechenden Webserver hochladen.

Im Vergleich zu virtuellen Maschinen benötigt der Kunde keine detaillierten Kenntnisse für die Konfiguration und Wartung eines Betriebssystems. Ebenso liegt die Zuständigkeit bei der Konfiguration und Wartung der Laufzeitumgebung beim entsprechenden Provider. Beim betreuten Hosting ist es für den Provider nicht immer möglich, individuelle Anforderungen des Kunden zu berücksichtigen, denn durch die Shared Hosting Umgebung würden ggf. die Dienstleistungen für andere Kunden eingeschränkt. Aus diesem Grund können beim betreuten



Hosting nicht alle Webanwendungen ausgeführt werden, weil der Provider die entsprechend nötigen Laufzeitumgebungen nicht unterstützt. Es sollte daher, bevor ein Hostingvertrag zustande kommt, die Kompatibilität der Webanwendung beim entsprechenden Provider erfragt werden.

### 2.4.3 Betreute Webanwendungen

Bei betreuten Webanwendungen erhält der Kunde eine bereits vom Provider installierte Webanwendung. Der Provider kümmert sich um deren regelmäßige Wartung und ein durchgängiges Monitoring. Dadurch kann der Provider dem Kunden je nach den Anforderungen bestimmte Verfügbarkeiten der Webanwendung zusichern. Der Kunde ist letztlich für die Konfiguration der Webanwendung selbst, sowie für die Einpflegung von eigenen Inhalten zuständig. Der Provider unterstützt den Kunden in der Regel dabei durch entsprechende Supportangebote.

Wenn man den Markt für gehostete Webanwendungen betrachtet, so fällt auf, dass sich einzelne Provider zumeist nur auf eine kleine Anzahl von Webanwendungen spezialisieren. Dadurch muss der Kunde für jede einzelne Webanwendung, die er betreiben möchte, einen geeigneten Provider finden.

## 2.5 IT-Sicherheit

Als IT-Sicherheit bezeichnet man den Schutz von Informationen auf technischen Systemen. Dieser Schutz soll dabei Vertraulichkeit, Verfügbarkeit und Integrität der entsprechenden Informationen sicherstellen. Da die entsprechenden Systeme Bedrohungen und Schwachstellen enthalten können, muss dieser Schutz durch angemessene Maßnahmen sichergestellt werden. [BSI13]

Damit die IT-Sicherheit von einzelnen System und Organisationen besser betrachtet und verglichen werden kann, wurden verbindliche Standards etabliert. Die im Rahmen dieser Arbeit relevanten werden in den nachfolgenden Abschnitten vorgestellt.

### 2.5.1 ISO/OSI Sicherheitsarchitektur

Auf Basis des ISO/OSI Referenzmodells für Netzprotokolle wird eine Sicherheitsarchitektur eingeführt. Das Referenzmodell unterteilt dabei Netzprotokolle in sieben Schichten. Diese folgen streng aufeinander und besitzen dabei nur Schnittstellen zu den direkt benachbarten Schichten. Die einzelnen Schichten sind nach dem ISO/OSI Referenzmodell: [ITU94]

- **Schicht 7: Anwendung**

Die Anwendungsschicht stellt die oberste Schicht dar. Sie verbindet das Netzprotokoll mit der dazugehörigen Anwendung.

- **Schicht 6: Darstellung**

Die Darstellungsschicht wandelt zu versendende Daten, die ggf. in einem hostspezifischen Format vorliegen, in ein allgemein gültiges um, so dass diese vom Zielsystem verarbeitet werden können.

- **Schicht 5: Sitzung**

In dieser Sitzungsschicht wird eine logische Verbindung erzeugt. Sollte die Transport-

Verbindung unterbrochen werden, kann durch diese Schicht die logische Verbindung (Sitzung) fortgesetzt werden.

- **Schicht 4: Transport**

Die Transportschicht stellt den oberen Schichten Schnittstellen zur Verfügung, um einen Datenstrom über das Netz zu transportieren. Diese Schicht segmentiert die Daten und verhindert durch geeignete Maßnahmen eine Überlastung.

- **Schicht 3: Vermittlung**

Die Vermittlungsschicht stellt eine Übertragung von Daten zwischen verschiedenen Teilnehmern des Netzes zur Verfügung. Insbesondere kümmert sich diese Schicht um die Auswahl des Weges (Routing) zum anderen Teilnehmer.

- **Schicht 2: Sicherung**

Die Sicherungsschicht regelt den Zugriff auf das Übertragungsmedium (Media Access Control) und versucht die Datenübertragung durch Prüfsummen resistenter gegenüber Fehlern zu machen.

- **Schicht 1: Bitübertragung**

Die niedrigste Schicht ist verantwortlich für die physikalische Übertragung der Daten. Die Bitübertragungsschicht moduliert dafür entsprechende Daten über ein Medium.

Für das ISO/OSI Referenzmodell wird in ITU X.800 eine Sicherheitsarchitektur definiert. Diese sieht Sicherheitsdienste und -mechanismen vor, welche die Sicherheit der Kommunikation eines verteilten Systems gewährleisten sollen. Dazu werden folgende Dienste, welche jeweils auf verschiedenen Schichten zum Einsatz kommen, festgelegt:

- **Authentisierung**

Verifiziert die Identität eines Teilnehmers am verteilten System. Dazu ist ein sog. Verifier nötig, wie z.B. ein Passwort oder ein privater Schlüssel.

- **Zugriffskontrolle**

Die Zugriffskontrolle stellt sicher, dass nur autorisierte Operationen durch einen Netzteilnehmer ausgeführt werden können.

- **Verbindlichkeit**

Stellt sicher, dass stattgefundenere Ereignisse zu einem späteren Zeitpunkt verifizierbar sind. Beteiligte Entitäten können das jeweilige Ereignis nicht leugnen.

- **Vertraulichkeit**

Die übertragenen Daten sollen nur den autorisierten Entitäten zugänglich sein.

- **Integrität**

Eine Veränderung von übertragenen Daten durch unautorisierte Entitäten kann erkannt werden. Ebenso müssen unautorisiert wiederholte Daten erkannt werden können.

- **Sicherheitsaudit und -alarme**

Ein weiterer Dienst stellt Sicherheitsaudits zur Verfügung. Es werden die Aktivitäten des verteilten System überprüft. Sollten dabei Auffälligkeiten festgestellt werden, wird durch einen Sicherheitsalarm eine dafür verantwortliche Stelle informiert. Ggf. kann durch den Alarm auch bereits eine Reaktion ausgelöst werden.

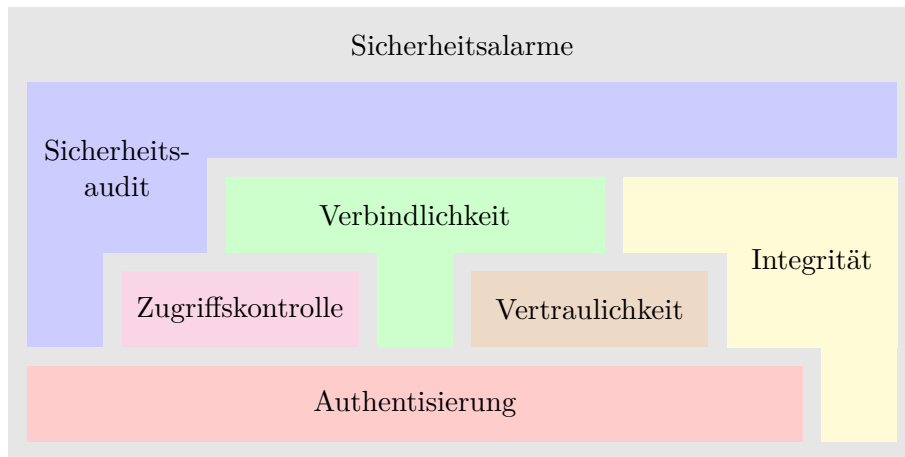


Abbildung 2.4: Hierarchie der OSI-Sicherheitsdienste (nach [Rei02])

Diese OSI-Sicherheitsdienste bilden eine Hierarchie, wie man in Abbildung 2.4 erkennen kann. [Rei08a]

Ein Sicherheitsdienst setzt sich dabei aus Sicherheitsmechanismen zusammen. Dabei unterscheidet man zwischen spezifischen und durchgängigen Mechanismen. Spezifische Mechanismen, wie z.B. digitale Signatur oder Verschlüsselung, arbeiten auf bestimmten Schichten des Referenzmodells. Mechanismen wie Auditing und Logging sind dagegen generisch einsetzbar und daher kompatibel mit allen Schichten und Diensten. [Rei08b] [ITU91]

### 2.5.2 ISO/IEC 27001:2005

Während sich die OSI-Sicherheitsarchitektur nur auf die Kommunikation von verteilten Systemen beschränkt, zielt die internationale Norm ISO/IEC 27001:2005 auf eine generellere Lösung: Sie legt Anforderungen an ein Information Security Management System (ISMS) fest. Ein solches ISMS versucht Mechanismen aus der Qualitätssicherung auf die IT-Sicherheit zu übertragen. Die Norm ist Teil der ISO/IEC 27000-Reihe, in der auch weitere Prozesse in der IT-Sicherheit normiert sind.

Ein ISMS nach ISO/IEC 27001:2005 basiert auf dem Management von Geschäftsrisiken. Es wird also anhand des ISMS auf bekannte Risiken reagiert. Dabei zielt es auf eine Wahrung der IT-Sicherheit in der entsprechenden Organisation ab. Es beachtet dabei die Entwicklung, Implementierung, Durchführung, Überwachung, Überprüfung, Instandhaltung und Verbesserung der Informationssicherheit. Dabei muss es die Struktur und die Ressourcen der entsprechenden Organisation berücksichtigen. [KRS13]

Es werden alle Werte der Organisation kategorisiert, so dass diese entsprechend durch die Geschäftsführung ausgegebener Leitlinien gesichert werden. Die Einführung eines ISMS ist der Prozess zur Entwicklung dieser Leitlinien. Durch Verwendung des sog. Demingkreis bei der Einführung, sowie später zur kontinuierlichen Verbesserung, kann auf etwaige Veränderungen dynamisch reagiert werden. Beim Demingkreis handelt es sich um einen Zyklus, in welchem diese Phasen durchlaufen werden:

- **Plan**

In dieser Phase soll das ISMS oder falls es schon existiert, mögliche Schritte zur dessen Verbesserung konzipiert werden.

- **Do**

Die Planungen sollen in kleinen Bereichen oder auch in Testumgebungen umgesetzt werden. Dies soll nur wenige Ressourcen binden und somit prototypisch durchgeführt werden.

- **Check**

Die Ergebnisse der vorherigen Phase sollen ausgiebig analysiert werden und falls diese als erfolgreich angesehen werden, sollen die Planungen als künftige Leitlinien festgelegt werden.

- **Act**

Die Leitlinien aus dem vorherigen Schritt werden nun umfassend implementiert und deren Einhaltung durch regelmäßige Audits sichergestellt.

### 2.5.3 IT-Grundschutz

Eine weitere Quelle für Vorgehensweisen zur Verbesserung der IT-Sicherheit für Betreiber von IT-Systemen ist das Konzept des IT-Grundschutzes, welches vom Bundesamt für Sicherheit in der Informationstechnik (BSI) entwickelt wurde. Das BSI hat das ausführliche Handbuch „IT-Grundschutz-Kataloge“ erstellt. Der Anspruch des Handbuches wird derart beschrieben: „Die IT-Grundschutz-Kataloge beschreiben detailliert diese Standard-Sicherheitsmaßnahmen, die praktisch für jedes IT-System zu beachten sind.“ [BSI13]

Die enthaltenen Kataloge sind intern durch ein Schichtenmodell gegliedert und ordnen sich diesen Bereichen zu:

- **Bausteinkatalog**

Ein Baustein ist das zentrale Element des IT-Grundschutzes. Um auf die weitreichenden Unterschiede in der IT-Landschaft einzugehen, kann sich jedes Unternehmen, die nötigen Bausteine herausuchen und so nach dem Baukasten-Prinzip den IT-Grundschutz individuell zusammenstellen.

Jeder Baustein enthält dabei die zugehörige Gefährdungslage. Das umfasst sowohl die zutreffenden Gefährdungen, als auch deren Eintrittswahrscheinlichkeit. Als Reaktion darauf wird ein entsprechend angepasster Maßnahmenkatalog vorgeschlagen.

Die Bausteine werden dabei in diese Schichten eingeteilt: „Übergreifende Aspekte“, „Infrastruktur“, „IT-Systeme“, „Netze“ und „Anwendungen“.

- **Gefährdungskataloge**

In den Gefährdungskatalogen wird jede Gefährdung beschrieben und zusätzliches Hintergrundwissen vermittelt. Dies dient dem besseren Verständnis des Sachverhaltes. Die einzelnen Gefährdungen werden dabei einer Schicht der folgenden Schichten zugewiesen: „Elementare Gefährdungen“, „Höhere Gewalt“, „Organisatorische Mängel“, „Menschliche Fehlhandlungen“, „Technisches Versagen“ und „Vorsätzliche Handlungen“.

- **Maßnahmenkataloge**

Dort werden Maßnahmen gesammelt, mit denen Gefährdungen begegnet werden kann.

Es wird dabei die Maßnahme selbst beschrieben. Außerdem werden die jeweils verantwortlichen Rollen bei der Initiierung sowie Umsetzung benannt. Zur Überprüfung der konkreten Implementierung werden Kontrollfragen angeboten. Auch hier werden die Maßnahmen wiederum in Schichten eingeteilt: „Infrastruktur“, „Organisation“, „Personal“, „Hardware und Software“, „Kommunikation“ und „Notfallvorsorge“.

Alle Elemente dieses IT-Grundschutzes können über eine eindeutige Notation referenziert werden. Diese besteht aus dem Anfangsbuchstaben (X) des jeweiligen Bereiches, der Nummer der entsprechenden Schicht (Y) und einer laufenden Nummer je Schicht (Z): X Y.Z.

### 2.6 Angriffsvektoren

In diesem Abschnitt werden die möglichen Angriffsvektoren auf Web- und Applikationsserver besprochen. Dies ist zur Entwicklung von Anforderungen für den sicheren Betrieb eben dieser Server nötig.

#### 2.6.1 DoS & DDoS

Wie alle über das Internet erreichbaren Dienste sind auch Web- und Applikationsserver gegen *Denial of Service (DoS)* Angriffe nicht gefeit. Dabei wird der jeweilige Dienst durch eine Vielzahl an fingierten Anfragen überlastet, so dass er legitime Anfragen nicht mehr beantworten kann. Werden diese Anfragen von einer größeren Anzahl unterschiedlicher Systeme versandt, so spricht man von einem *Distributed Denial of Service (DDoS)*. [MPR02]

In der Praxis werden für solche DDoS-Angriffe häufig Botnetze gemietet. Diese bestehen aus einer großen Anzahl an verteilten Rechnern, welche Aufgaben des Mieters ausführen. Häufig bekommen die wahren Eigentümer der beteiligten Systeme gar nicht mit, dass sie Teil eines Botnetzes sind, da die Rechner durch entsprechende Malware dem Botnetz beitreten. Ein Botnetz kann eine beachtliche Größe erreichen: Das Botnetz Chameleon wurde im Februar 2013 entdeckt und bestand aus mindestens 120.000 Systemen. [spi13]

Eine weitere Verstärkung der Bandbreite des Angriffes wird durch sog. Amplification Attacks erreicht. Dabei macht sich der Angreifer zunutze, dass bei Diensten, die UDP verwenden, gefälschte IP-Adressen als Absendeadresse nicht erkannt werden können (Spoofing). Er lässt also durch ein Botnetz Anfragen an UDP-Dienste schicken. Als Absendeadresse wird dabei die des Opfers angegeben. Die Antwort des UDP-Dienstes wird an das Opfer geschickt. Wenn der Angreifer nun den verwendeten UDP-Dienst so wählt, dass auf eine kleine UDP Anfrage eine sehr große Antwort erfolgt, kann er seine für den Angriff verwendete Bandbreite vervielfachen. Dieser Ablauf wird durch Abbildung 2.5 dargestellt. Oftmals wird dafür das *Domain Name System (DNS)* eingesetzt, da es insbesondere bei Verwendung der Sicherheitserweiterung DNSSEC auf kleine Anfragen mit sehr großen Paketen antwortet. Desweiteren gibt es durch die herausragende Bedeutung von DNS für das Internet eine Vielzahl an DNS-Servern, welche über gewaltige Ressourcen verfügen. [McN13]

Als Betreiber von Web- und Applikationsservern muss man daher im Vorfeld eines DDoS-Angriffes eine Erkennungs- und Abwehrstrategie entwickeln. Eine wichtige Grundregel ist, dass die eigene Netzanbindung über genügend Kapazität verfügen sollte, damit ein DDoS diese nicht überlasten kann. Ferner gilt es alle Komponenten gegenüber eines solchen Angriffes abzusichern. Denn ein abgesicherter Webserver ist bei einem DDoS auf die DNS-Server trotzdem nicht erreichbar.

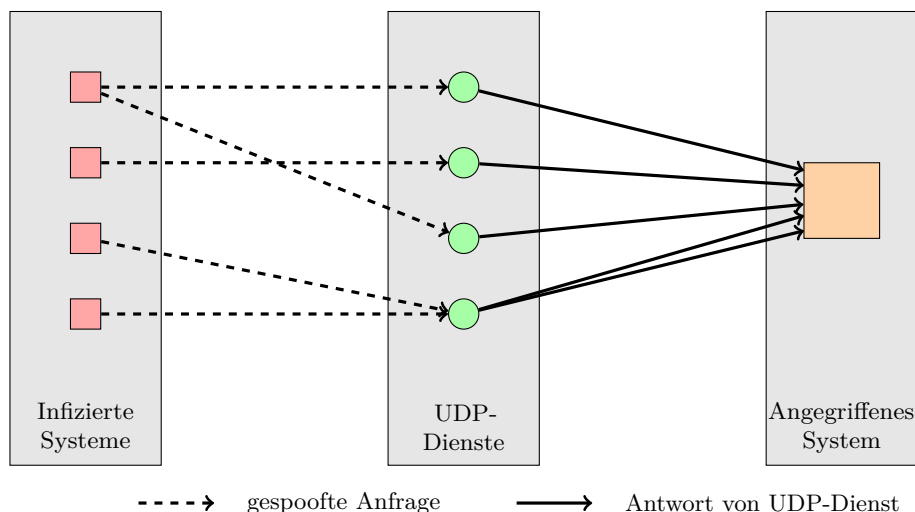


Abbildung 2.5: Ablauf einer UDP-Amplification-Attack

### 2.6.2 Unsichere bzw. bekannte Passwörter

Um unberechtigten Zugriff auf ein fremdes System erhalten zu können, reicht es dem Angreifer in vielen Fällen aus, an das Passwort eines legitimen Nutzers zu gelangen. Eine mögliche Variante ist ein sog. Wörterbuch-Angriff. Dafür wird eine bestehende Datenbank mit besonders häufig verwendeten Passwörtern einfach ausprobiert. Dieser Angriff ist erfolgreich, wenn am entsprechenden System das voreingestellte Passwort nicht geändert wurde oder der Nutzer bei der Änderung ein sehr schwaches Passwort verwendet hat. Tipps zur Auswahl eines starken und somit sicheren Passwortes gibt das BSI: [BSI11]

- Es sollte mindestens 8 Zeichen lang sein.
- Es sollte nicht (auch nur in Teilen) in Wörterbüchern vorkommen.
- Es sollten keine Namen verwendet werden.
- Man kann sich ein komplexes Passwort durch einen Merksatz relativ einfach konstruieren und somit besser merken, indem man von den Wörter des Merksatzes lediglich das erste Zeichen im Passwort verwendet.
- Es sollten Groß- und Kleinbuchstaben sowie Sonderzeichen und Ziffern enthalten sein.
- Es dürfen keine gängigen Standardpasswörter oder absehbare Folgen verwendet werden.
- Es sollte für jeden Dienst ein anderes Passwort verwendet werden.
- Ein Passwort sollte in regelmäßigen Abständen geändert werden.

Auch wenn der Nutzer ein sicheres Passwort gewählt hat, bleiben dem Angreifer weitere Möglichkeiten um an das Passwort zu gelangen. Er kann versuchen, dass der legitime Nutzer

ihm das Passwort durch sog. Phishing mitteilt. Dabei gibt der Angreifer vor, dass es sich bei ihm um einen vertrauenswürdigen Ansprechpartner handelt und er bittet in täuschend echt aussehenden E-Mails und anderen Nachrichten den Nutzer um Herausgabe des Passwortes. Meist wird darin der Nutzer durch eine fingierte Geschichte auf eine Webseite gelotst, wo er dann sein Passwort bestätigen soll. Grundsätzlich sollte man bei solchen Bitten sehr misstrauisch werden und niemals das Passwort auf anderen als den gewohnten Webseiten eingeben.

Ein weitere Quelle für Passwörter kann ein infizierter Rechner eines Nutzers sein. Dazu überwacht der Angreifer die Eingaben auf bestimmten Webseiten und liest ggf. auf dem Rechner abgespeicherte Passwörter aus. Dadurch bekommt der Angreifer die korrekten Benutzernamen und Passwörter für bestimmte Dienste.

### 2.6.3 Bekannte Sicherheitslücken veralteter Software

Ein weiterer Ansatzpunkt für Angriffe kann der Einsatz von veralteter Software durch Provider bzw. Kunde auf den entsprechenden Web- und Applikationsserver sein. Wird eine neue Sicherheitslücke in einer Software aufgespürt, hängt es vom Entdecker ab, wie er mit dem Fund umgeht. Im Sinne der IT-Sicherheit wäre es das Verantwortungsvollste, wenn der Entdecker eine „Responsible Disclosure“ ausführen. Dabei wird der Hersteller bzw. Urheber der Software vertraulich auf das Sicherheitsproblem hingewiesen. Es wird sich dann auf einen Zeitpunkt geeinigt, zu dem das Sicherheitsproblem mit allen Details veröffentlicht wird. Während dieses Zeitraums sollte der Hersteller das Problem durch Aktualisierungen für die Software beheben und entsprechend die Anwender der Software über das Update informieren. Für den Fall, dass der Hersteller/Autor der Software nicht auf die Meldung reagiert, kann der Entdecker die Sicherheitslücke dennoch veröffentlichen (sog. „Full Disclosure“) und dadurch weiteren Druck auf den Autor ausüben, damit dieser die Lücke behebt. [Sch07]

Leider handeln nicht alle Entdecker von Sicherheitslücken so verantwortungsbewusst. Denn neuentdeckte Sicherheitsprobleme mit dazugehörigem Schadcode (sog. Zero-Day-Exploits) lassen sich auf einschlägigen Seiten verkaufen. Der Preis richtet sich dabei nach der Verbreitung der entsprechenden Software. Neben kriminellen Akteuren beteiligen sich mutmaßlich auch Regierungorganisationen an diesem Markt, welche die Schwachstellen zur Spionage und Cyber-Abwehr einsetzen könnten. Der Markt an geheimen Sicherheitslücken verhindert jedoch, dass der Hersteller davon erfährt und diese Lücken schließen kann. Dadurch wird die gesamte IT-Sicherheit negativ beeinflusst. Einige Hersteller sind bereits dazu übergegangen, für eine „Responsible Disclosure“ entsprechende Prämien an die Entdecker zu bezahlen, um damit die Attraktivität eines Verkaufs der Sicherheitslücke zu senken. [Kas12]

Durch viele Organisationen werden sog. Computer Emergency Response Teams (CERTs) betrieben. Deren Zweck ist es, bei der Entdeckung von Hard- und Softwareschwachstellen eine Warnung herauszugeben und gleichzeitig Empfehlungen zur Schadensbeseitigung und -begrenzung zu geben. Die Warnung wird zumeist über Mailinglisten den jeweiligen Abonnenten eines CERTs mitgeteilt. Bekannte CERTs aus Deutschland sind das *DFN-CERT* des Deutschen Forschungsnetzes (DFN), *CERT-Bund* des BSIs und *RUS-CERT* der Universität Stuttgart. [RFC2350]

Damit Sicherheitslücken eindeutig referenziert werden können, wird durch das Forschungsinstitut MITRE Corp. eine CVE-ID (Common Vulnerabilities and Exposures Identifier) vergeben. Eine CVE-ID enthält die Jahreszahl der Entdeckung und eine fortlaufende Nummer. Man kann anhand dieser CVE-ID die vielzähligen Quellen von Exploit- und IT-

Sicherheitsmeldungen wie z.B. von CERTs besser verknüpfen und somit einfacher Informationen zu bestimmten Sicherheitslücken finden. [Cor13]

Dank entsprechender Datenbanken ist es für einen Angreifer ein Leichtes anhand der Versionsnummern einer Anwendung entsprechende, öffentlich bekannte Lücken ausfindig zu machen. Daher sollten sämtliche Komponenten von Web- und Applikationsserver regelmäßig aktualisiert werden.

### 2.6.4 Verwundbare Konfiguration

Ein weiteres Einfalltor für Angreifer stellt eine mangelhafte Konfiguration von Anwendungen auf Web- und Applikationsserver dar. Zumeist werden diese durch den Kunden selbst installiert. Dabei wird auch oft eine Installation und Grundkonfiguration dieser Anwendungen durch Laien vorgenommen. Dadurch können durch nicht angepasste Standardeinstellungen die Anwendungen bereits von Beginn an in einem unsicheren Zustand sein. Besonders häufige Probleme sind diesem Umfeld:

- Installations- und Konfigurationshilfsmittel der Anwendung werden nach deren Gebrauch nicht deaktiviert.
- Die standardmäßig vergebenen Passwörter und Accounts werden nicht verändert.
- Vertrauliche Datenverzeichnisse der Anwendung sind ungeschützt aufrufbar.

Dabei kann auch dem Laien die Einrichtung einer Anwendung bis zu einem lauffähigen Zustand ohne weiteres gelingen. Insbesondere bei Verwendung der im Internet zahlreich verfügbaren Anleitungen ist das mit etwas Recherche kein Problem. Jedoch werden dann häufig die Wartungsaufgaben vernachlässigt.

### 2.6.5 Verwundbare Webapplikationen

Eine weitere mögliche Verwundbarkeit kann direkt aus der eingesetzten Webapplikation stammen. Obwohl keine Sicherheitslücken der aktuellen Version bekannt sind, können wesentliche Sicherheitskriterien bei der Entwicklung nicht berücksichtigt worden sein. Insbesondere bei wenig eingesetzten Webapplikationen und Eigenentwicklungen sollte diese Gefahr nicht vernachlässigt werden, da dort Fehler durch die geringere Verbreitung wesentlich schwieriger entdeckt werden.

Eine gute Übersicht über mögliche Gefahren bietet die Top 10 Liste des *Open Web Application Security Project (OWASP)*. Nach dieser Liste sind dies die dringlichsten Gefahren: [OWASP13]

- **Injection**

Die Webanwendung gibt nicht ausreichend validierte Eingabedaten an nachgelagerte Anwendungen/Systeme, wie z.B. Datenbanken und Betriebssysteme, weiter. Durch nicht vorhergesehene Eingabedaten können dort unerwünschte Operationen ausgeführt werden.

- **Fehlerhafte Authentisierung und Sitzungsverwaltung**

Durch eine fehlerhafte Implementierung kann die Authentisierung umgangen werden. Dies ist z.B. möglich, wenn die Identifier der Sitzung vorhersehbar sind oder innerhalb der URL übergeben werden.



- **Cross-Site Scripting (XSS)**

Es werden Eingabedaten nicht validiert und anschließend an den Client ausgeliefert. Ein Angreifer ist damit in der Lage Code, der im Browser ausgeführt wird, einzuschleusen. Dadurch kann beispielsweise der Sitzung-Identifizierer des Clients gestohlen werden.
- **Unsichere direkte Objekt-Referenzierung**

Daten der Webapplikation werden über einen direkten Parameter in der URL referenziert. Durch Abänderung dieses Parameters durch einen Angreifer können ohne passende Validierung Daten ungewollt ausgeliefert werden.
- **Mangelnde IT-Sicherheit der Umgebung**

Durch mangelnde Sicherheit des Web- und Applikationsservers wird auch die Sicherheit der Applikation selbst beeinträchtigt. Das ist genau das Thema, mit dem sich diese Arbeit beschäftigt.
- **Verlust der Vertraulichkeit sensibler Daten**

Durch unzureichenden Schutz von sensiblen Daten wie Passwörtern und Abrechnungsdaten werden diese unautorisierten Nutzern zugänglich.
- **Mangelnde Zugriffskontrolle**

Es findet keine ausreichende Zugriffskontrolle auf geschützte Inhalte statt. Dies ist z.B. der Fall, wenn Daten lediglich für den Client versteckt werden, denn der Zugriff muss immer auch serverseitig verhindert werden.
- **Cross-Site Request Forgery (CSRF)**

Der Angriff erfolgt auf einen authentisierten Client einer Webapplikation. Der Angreifer verleitet den Client durch eine von ihm kontrollierte Webseite dazu, dass dieser eine Aktion auf der Webapplikation ausführt. Die Webapplikation prüft dabei nicht, ob die Aktion wirklich durch den Benutzer beabsichtigt war.
- **Verwendung von Komponenten mit bekannten Sicherheitslücken**

Werden externe Komponenten in einer Webapplikation verwendet, so wirken sich Sicherheitslücken der Komponenten auch auf die gesamte Applikation aus. Es sollte daher regelmäßig die Sicherheit der Komponenten überprüft werden.
- **Ungeprüfte Weiterleitungen**

Oftmals werden Weiterleitungen auf externe Seite in Webapplikationen eingebaut. Dabei ist es wichtig, dass die verwendete URL validiert wird, weil ansonsten ein Angreifer eigene URLs dort verwenden könnte und so die Clients auf eigene Seiten leiten könnte.



## 3 Anforderungsanalyse

Am Ende dieses Kapitels sollen die Anforderungen an einen sicheren Betrieb von Web- und Applikationsservern aufgestellt werden. Dazu wird in Abschnitt 3.1 der aktuelle Zustand der Web- und Applikationsserver am LRZ vorgestellt. Basierend auf dem am LRZ vorgefundenen Szenario werden dann in Abschnitt 3.2 allgemeingültige Anforderungen an die Absicherung von Web- und Applikationsserver abgeleitet.

### 3.1 Situation am LRZ

Dieser Abschnitt betrachtet die derzeitige Situation bezüglich Web- und Applikationsserver am LRZ. Es werden in diesem Bereich die Produkte „Virtuelle Maschinen“ und „Betreutes Hosting“ angeboten. Das Produkt „Betreute Webanwendungen“ steht gegenwärtig nicht zur Verfügung.

Die Kundenstruktur umfasst vor allem wissenschaftliche Einrichtungen im Großraum München. Der Träger des LRZs ist die Bayerische Akademie der Wissenschaften (BAdW). Um Zugriff auf die Produkte erhalten zu können, benötigt der Kunde eine sog. LRZ-Kennung. Es wird dabei zwischen persönlichen Kennungen und Funktionskennungen unterschieden. Während jeder Einzelbenutzer des LRZs eine persönliche Kennung erhält, werden Funktionskennungen nur an Lehrstühle, Institute, Einrichtungen und Projekte vergeben. [LRZ13b][LRZ13c]

#### 3.1.1 Virtuelle Maschinen

Das LRZ bietet seinen Kunden virtuelle Maschinen (vgl. Abschnitt 2.4.1) an. Diese hochverfügbaren virtualisierten Serverinstanzen werden auf LRZ-eigener Server-Hardware betrieben. Als Virtualisierungssoftware wird dabei *VMware* eingesetzt. Die Vergabe der VMs erfolgt projektbezogen, daher benötigt man bei der Beantragung eine LRZ-Projektnummer.

Prinzipiell werden virtuellen Maschinen mit zwei verschiedenen Service-Varianten angeboten: „Attended Hosting“ und „Unattended Hosting“. Beim „Attended Hosting“ kümmert sich das LRZ um die Ersteinrichtung und Pflege des Betriebssystems, es stehen dabei diese Systeme zur Auswahl: *SuSE Linux Enterprise 11 SP3 für 64bit-Architektur*, *Windows Server 2008 R2 für 64bit-Architektur* und *Windows Server 2012 R2 für 64bit-Architektur* (Stand 12/2013). Zur Pflege des Betriebssystems gehört u.a. das automatisierte Einspielen von Sicherheitsupdates. Zusätzlich wird ein grundlegendes Monitoring installiert, so dass „Attended Hosting“ VMs durch die Monitoring Systeme des LRZs erfasst werden können.

Für den Fall, dass der Kunde ein nicht angebotenes Betriebssystem einsetzen muss, erhält er die Service-Variante „Unattended Hosting“. Er muss sich dann selbst um die Ersteinrichtung und Pflege des gewünschten Betriebssystems kümmern. In den Nutzungsrichtlinien verpflichtet das LRZ den Kunden einer solchen VM, dass er verfügbare Sicherheitsupdates für das Betriebssystem binnen sieben Tagen einspielt.

Für alle Varianten können dem Kunden Speicher- und Rechenressourcen entsprechend seinen Forderungen flexibel bereitgestellt werden. Für den Hintergrundspeicher wird optional ein zentrales Datensicherungs- und Wiederherstellungssystem auf Basis des *Tivoli Storage*

*Manager (TSM)* angeboten. Bei der Netzanbindung der VM kann der Kunde auswählen, ob diese nur innerhalb des Münchner Wissenschaftsnetz (MWN) oder weltweit erreichbar sein soll.

Die folgenden Nutzungsrichtlinien müssen die Kunden von virtuellen Maschinen einhalten, ansonsten kann die Netzanbindung der jeweiligen VM durch das LRZ unterbunden werden: [LRZ13a]

- Der Kunde trägt die Verantwortung für das Einspielen von Sicherheitsupdates der Anwendungssoftware. Binnen sieben Tagen nach Erscheinen des Updates muss dieses installiert sein.
- Vom LRZ installierte Management-Anwendungen dürfen nicht durch technische Maßnahmen deaktiviert werden.
- Für das jeweilige System muss eine Kennung mit administrativen Rechten für einen Zugriff durch das LRZ angelegt werden.
- Die sog. *VMware-Tools* müssen auf dem System installiert sein.
- Wird Windows als Betriebssystem eingesetzt, muss die lokale Firewall aktiviert sein und die Antiviren-Software des LRZs installiert und aktiviert sein.

#### 3.1.2 Betreutes Hosting

Beim Produkt „Webhosting“ des LRZs wird zwischen den zwei Kennungstypen unterschieden:

Für persönliche Kennungen wird ein Hosting von statischen Webinhalten angeboten. Der Einzelbenutzer kann dabei eine Domain mit der Endung „.userweb.mwn.de“ wählen. Der Speicherplatz ist auf zwei GB begrenzt und wird durch ein Backup täglich gesichert. Dieses Produkt lässt sich nicht bezüglich individueller Ansprüche des Kunden erweitern. [LRZ12b]

Das Angebot für Funktionskennungen dagegen besteht in der Basisvariante aus zehn GB Speicherplatz. Die Domain, unter welcher die dazugehörige Webseite erreichbar ist, kann durch den Kunden entsprechend seiner Zugehörigkeit zu einer Institution im MWN gewählt werden. Die Webinhalte können dabei im Gegensatz zu dem obigen Angebot auch dynamisch generiert werden. Das Basispaket kann je nach Anforderungen der Kunden angepasst werden. Sollte der Speicherplatz nicht ausreichen, kann entsprechend mehr bereitgestellt werden. Ebenso ist es möglich, eine für viele Webanwendungen obligatorische Datenbank einzurichten. Ist sich der Kunde unsicher, ob die gewünschte Webanwendung mit der Konfiguration am LRZ kompatibel ist, kann er dies bereits im Vorfeld der Einrichtung durch eine individuelle Beratung mit dem LRZ abstimmen. [LRZ13c]

Für die weitere Arbeit wird das Webhosting für persönliche Kennungen nicht betrachtet, da zum einen durch die lediglich statischen Inhalte viele Gefährdungen bei Webservern nicht auftreten können und zum anderen alle Aspekte auch bei dem erweiterten Angebot für Funktionskennungen auftreten.

Es wird nun das Augenmerk auf die Infrastruktur geworfen, die beim Produkt „Betreutes Hosting“ am LRZ zum Einsatz kommt. Eine Übersicht dieser Struktur ist in Abbildung 3.1 dargestellt. Es wird nun im Folgenden auf die einzelnen Komponenten eingegangen.

Der meist genutzte Dienst beim Webhosting ist natürlich HTTP, sowie dessen verschlüsselte Variante HTTPS. Damit die Webseiten von jedermann aufgerufen werden können, sind diese HTTP(S)-Dienste aus dem öffentlichen Internet erreichbar. Dabei läuft die Kommunikation

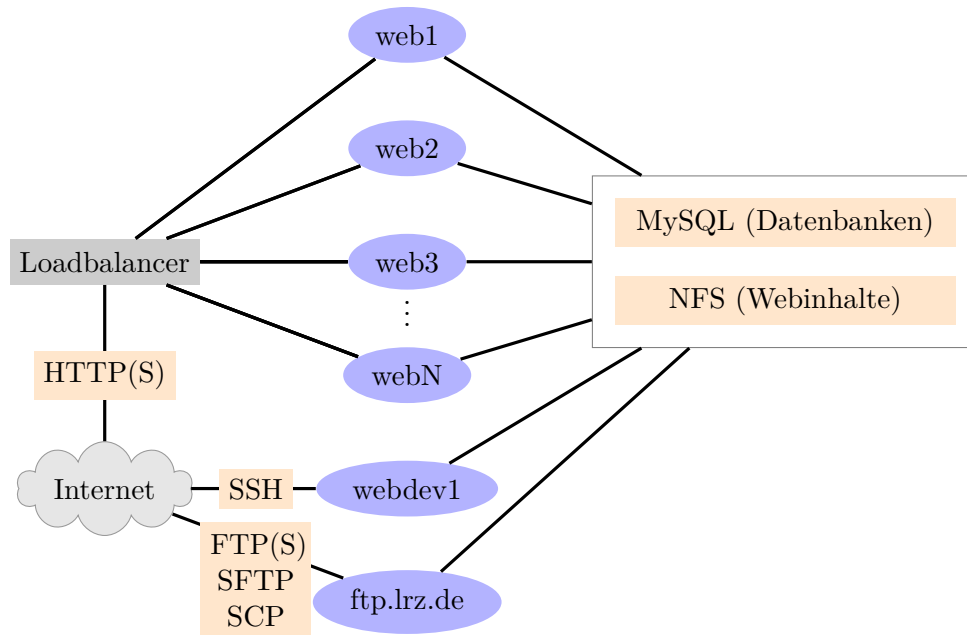


Abbildung 3.1: Topologie des betreuten Hostings am LRZ

bei einem Aufruf aus dem Internet über sog. Loadbalancer, welche die Anfragen auf mehrere Webserver verteilen. Man bezeichnet diese Anordnung auch als Webfarm. Dadurch wird die Ausfallsicherheit erhöht und die Last auf mehrere Webserver-Instanzen verteilt. Die Webserver werden teils auf virtuellen Maschinen, teils auf dedizierter Hardware betrieben. Jedoch sollen mittelfristig alle Webserver auf entsprechende VMs migriert werden, da dies den Wartungsaufwand verringert und eine höhere Verfügbarkeit ermöglicht.

Die Aufteilung der insgesamt rund 1100 virtuellen Hosts (vHosts) auf einzelne Webserver erfolgt dabei je nach Zugehörigkeit des Kunden zu einer dieser drei Institutionen: Ludwig-Maximilians-Universität München (LMU), Technische Universität München (TUM) und übriges Münchener Wissenschaftsnetz (MWN). Dabei sind alle vHosts einer Institution unter derselben IP-Adresse erreichbar und der Webserver unterscheidet lediglich anhand des übermittelten Hostnamens, welche Webpräsenz ausgeliefert werden soll. Für jede Institution werden mindestens zwei Webserver-Instanzen betrieben, damit beim Ausfall eines Webserver alle Anfragen durch den verbleibenden beantwortet werden können.

Auf den Webservern wird das Betriebssystem *SuSE Linux Enterprise 11* verwendet. Und darauf wird als Webserver-Software der *Apache HTTP Server* eingesetzt. Für die dynamische Generierung von Webinhalten stehen sowohl die Skriptsprache *PHP* als auch CGI-Skripte zur Verfügung. Es können dabei verschiedene *PHP*-Versionen je Kunde eingesetzt werden, welche mittels FastCGI eingebunden werden (vgl. dazu Abschnitt 2.3.1).

Da die Webinhalte der Benutzer aus einem Storage-System über das Protokoll *NFS* abgerufen werden, stehen jeder Webserver-Instanz dieselben Webinhalte zur Verfügung. Die Logdateien der Webserver, welche Zugriffe und etwaig aufgetretene Fehler enthalten, werden zuerst lokal auf den jeweiligen Webservern abgelegt. Periodisch erfolgt dann eine Zusammenführung der

einzelnen Dateien aller Webserver und diese wird dann den entsprechenden Kunden zur Analyse zur Verfügung gestellt.

Für Kunden wird auf der VM „webdev1“ ein SSH-Dienst angeboten. Dieser stellt nach Login eine ähnliche Umgebung wie auf den Webservern mit einer interaktive Shell zur Verfügung. Es kann darüber z.B. relativ einfach eine Webapplikation installiert werden. Derzeit wird der SSH-Dienst für alle Kunden auf einer einzelnen VM erbracht. Zukünftig ist es angedacht hier je Institution des MWNs eine eigene VM anzubieten, um, wie auch auf den Webservern, die Kunden der verschiedenen Institutionen zu trennen.

Eine weitere Möglichkeit Webinhalte aufzuspielen bzw. zu verändern, besteht über „ftp.lrz.de“. Dort werden die Dateiübertragungsprotokolle FTP, dessen verschlüsselte Variante FTPS, SFTP (FTP über SSH) und SCP angeboten. Auf einen Zugriff via FTP sollte man möglichst verzichten, weil FTP das Passwort im Klartext überträgt.

Viele Webapplikationen benötigen für die Persistierung von Inhalten eine Datenbank. Eine gängige Datenbank, die oftmals mit der Skriptsprache *PHP* zum Einsatz kommt, ist dabei *MySQL*. Daher werden durch das LRZ für das Webhosting ebenfalls solche Datenbankserver betrieben. Die *MySQL*-Server werden dabei einschließlich ihrer Daten repliziert, um eine höhere Ausfallsicherheit zu erreichen. Auch hier werden die Daten mindestens einmal täglich gesichert. Zur Verwaltung der Datenbanken wird den Kunden die Verwaltungsoberfläche *phpMyAdmin* bereit gestellt.

Im Rahmen dieser Arbeit wurde das Tool `webapp_discover` entwickelt (vgl. Abschnitt 5.2), welches installierte Webanwendungen anhand einer Durchsuchung des Dateisystems erkennen kann. Das Tool unterstützt die Webanwendungen *Drupal*, *Joomla*, *phpMyAdmin*, *Typo3* und *WordPress*. Mit `webapp_discover` kann festgestellt werden, ob im untersuchten Verzeichnis eine unterstützte Webanwendung installiert ist. Falls eine Webanwendungen gefunden wurde, werden weitere Informationen wie z.B. die verwendete Version und installierte Plugins der Webanwendung ermittelt. Am 3. Februar 2014 wurden die Webpräsenzen der 415 Nutzerkennungen, welche beim Webhosting des LRZs aktiv sind, durchsucht. Bei 132 Nutzerkennungen (entspricht 32,8% aller Nutzerkennungen) wurde mindestens eine der fünf Webapplikationen erkannt. Die genaue Verteilung der Nutzerkennungen auf die einzelnen Institutionen kann aus Abbildung 3.2 entnommen werden. Es wurden 272 Installationen der unterstützten Webanwendungen ermittelt. Die am häufigsten eingesetzte Webanwendung am LRZ ist demnach *Joomla*, gefolgt von *Drupal* und *Typo3*. Die Verteilung der erkannten Webanwendungen ist in Abbildung 3.3 dargestellt. Eine Aufschlüsselung nach Versionen der einzelnen Webanwendungen ist in Abschnitt 5.2.3 zu finden.

Für das oben beschriebene Webhosting wird durch das LRZ den Kunden die Dienstgüte „Best Effort“ zugesichert. Dies entspricht einer minimalistischen Zusicherung, dass Anfragen lediglich im Rahmen der verfügbaren Ressourcen beantwortet werden. Dies ist darauf zurückzuführen, dass sich in der Webfarm mehrere Kunden die zur Verfügung stehenden Ressourcen teilen müssen. [LRZ13a]

#### 3.1.3 Analyse der Sicherheitsvorfälle im Jahr 2013

Dieser Abschnitt betrachtet die Sicherheitsvorfälle beim Webhosting am LRZ im Jahr 2013. Die Vorfälle wurden durch das Webhosting-Team des LRZs bearbeitet und im internen LRZ-Wiki dokumentiert. Insgesamt wurden im Jahr 2013 20 Vorfälle durch das LRZ registriert. Der zeitliche Verlauf wird durch die Abbildung 1.1 dargestellt. Vermutlich sind dem LRZ nicht alle Vorfälle bekannt geworden und daher ist von einer Dunkelziffer an Sicherheitsvorfällen

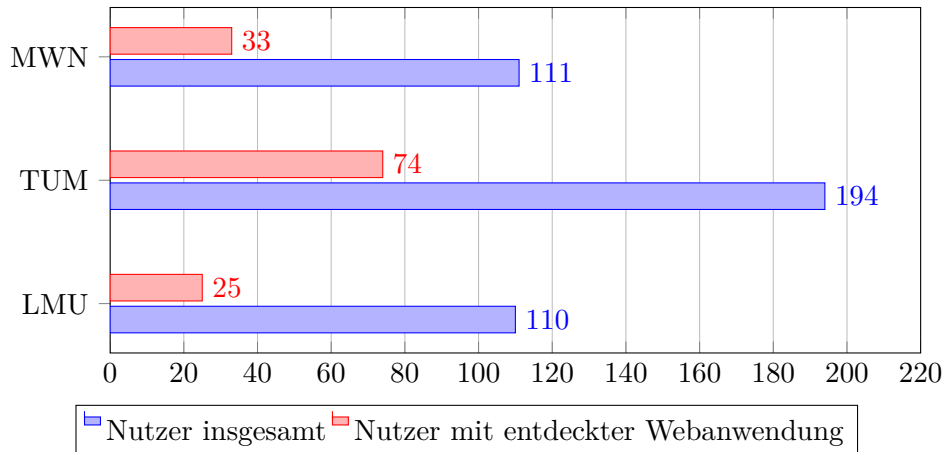


Abbildung 3.2: Verteilung der Nutzerkennungen nach Institutionen

auszugehen.

Die aufgetretenen Sicherheitsvorfälle wurden dabei folgendermaßen entdeckt:

- **Meldung durch Kunden**

In diesen Fällen wurde die Webseite durch den Angreifer verändert und der entsprechende Kunde hatte sich daraufhin an das Webhosting-Team des LRZs gewandt.

- **Meldung durch Dritte**

Auch durch externe Dritte wurden Sicherheitsvorfälle an das LRZ gemeldet. Die Meldungen wurden von *SpamCop* und *CleanMx* an die Abuse-Adresse des LRZs gerichtet ([abuse@lrz.de](mailto:abuse@lrz.de)). [Cis14] [CMX14]

- **Feststellung durch das LRZ**

Ebenso konnten Sicherheitsvorfälle durch Mitarbeiter oder Systeme des LRZs erkannt werden. Da Webanwendungen nach einer Infektion oftmals versuchen über den E-Mail-Dienst des Webservers Spam-Nachrichten zu verschicken, konnten darüber zwei Sicherheitsvorfälle entdeckt werden.

Weitere Infektionen konnten durch eine gezielte Suche nach verwundbaren Systemen entdeckt werden. Nachdem eine durch Dritte gemeldete Infektion bearbeitet worden war, wurde nach Webanwendungen gesucht, die ähnliche Veränderungen aufzeigen. Es konnten dadurch fünf weitere Vorfälle entdeckt werden. Da sich dieser Vorgang im März 2013 abgespielt hat, kann damit der deutlich erhöhte Wert an Vorfällen in Abbildung 1.1 erklärt werden.

Nach dem Eingang der Meldungen wurden betroffene Webapplikationen unverzüglich deaktiviert und der jeweilige Kunde über die Kompromittierung informiert. Die Kunden wurden aufgefordert, zur Abstimmung von weiteren Schritten das LRZ zu kontaktieren. Hier eine Auswahl der durchgeführten Maßnahmen:

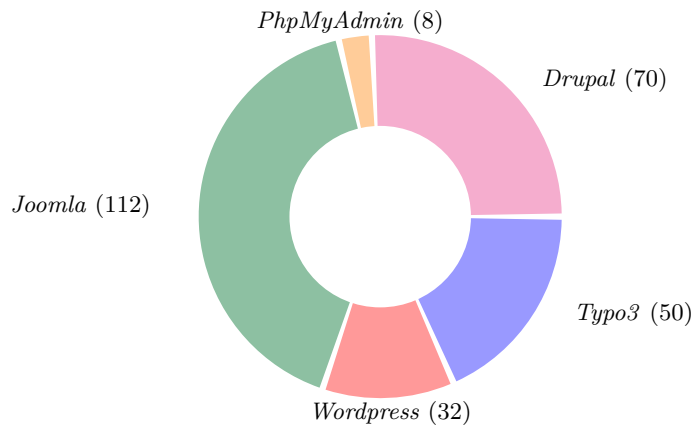


Abbildung 3.3: Verteilung von erkannten Webanwendungen beim Webhosting am LRZ

- **Wiederherstellung der Webanwendung**

1. Sofern vorhanden, wurde eine Wiederherstellung der Webanwendung aus einer nicht infizierten Kopie (vom Kunden bzw. aus dem Backup) durchgeführt. Ansonsten mussten infizierte Dateien manuell bereinigt werden.
2. Es wurde eine Änderung aller verwendeter Passwörter vorgenommen.
3. Durch eine Aktualisierung der Webanwendung wurden die Sicherheitslücken, welche vermutlich zum Angriff geführt haben, behoben.

- **Dauerhafte Deaktivierung der Webanwendung**

- **Neuinstallation einer alternativen (sicheren) Webanwendung**

Zusätzlich wurde durch das Webhosting-Team des LRZs versucht die Ursache der Kompromittierung herauszufinden. In 18 der 20 Sicherheitsvorfälle im Jahr 2013 wurde vermutet, dass eine veraltete Version der Webanwendung den Angriff ermöglicht hatte. Dabei ist in 15 dieser Fälle die Webanwendung *Joomla* betroffen gewesen. Die verbleibenden beiden Fälle sind mutmaßlich auf ausgespähte Passwörter zurückzuführen. [LRZ13d]

## 3.2 Anforderungen

Innerhalb dieses Abschnitts werden Anforderungen vorgestellt, deren Einhaltung die Sicherheit beim Betrieb von Web- und Applikationsservern verbessern. Diese Anforderungen werden in vier Kategorien eingeteilt, welche Aussagen über deren Hauptzweck treffen. In dem nachfolgenden Kapitel 4 werden dann für die Erfüllung dieser Anforderungen konkrete technische und organisatorische Lösungsvorschläge erarbeitet.

Zur Herleitung der Anforderungen wurden neben den Sicherheitsvorfällen am LRZ auch die allgemeinen Beschreibungen von Angriffsvektoren in Abschnitt 2.6 verwendet. Durch die Betrachtung sowohl von konkreten Sicherheitsvorfällen bei einem Providern als auch von



theoretischen denkbaren Angriffsvektoren konnte sichergestellt werden, dass alle notwendigen Anforderungen ermittelt wurden.

### 3.2.1 Prävention

Bei der Prävention wird durch vorbeugende Maßnahmen versucht, Risiken bezüglich der Sicherheit des System bereits im Vorfeld eines konkreten Sicherheitsereignisses soweit als möglich zu minimieren. Dabei wird den Herausforderungen durch Verbesserung der technischen und administrativen Rahmenbedingungen begegnet.

#### 3.2.1.1 Sichere Passwortrichtlinien

Ein Provider sollte in einer für alle Mitarbeiter verbindlichen Richtlinie festlegen, wie der Umgang mit Passwörtern auf seinen Systemen zu erfolgen hat. Dort sollte die minimale Komplexität von Passwörtern festgelegt werden. Ebenso soll durch entsprechende Regeln bestimmt werden, welche Besonderheiten bei der Übertragung und Speicherung von Passwörtern berücksichtigt werden müssen. Die Richtlinien sollten zudem festlegen, dass Passwörter nach einem bestimmten Zeitraum geändert werden müssen. Da auch Kunden innerhalb ihres Verantwortungsbereiches Passwörter nutzen, sollte eine entsprechende Richtlinie den Kunden zur Anwendung empfohlen werden.

#### 3.2.1.2 Regelmäßige Wartung von Webanwendungen

Ein Provider sollte kontrollieren, ob der Kunde gewisse Wartungsaufgaben, welche bei Webanwendungen anfallen, regelmäßig durchführt. Zu diesen Aufgaben gehört u.a. auch das Einspielen von Aktualisierungen bei Webanwendungen, da durch den Einsatz von nicht aktuellen Softwarekomponenten der Betrieb der Webanwendung gefährdet wird (vgl. Abschnitte 2.6.3 und 3.1.3). Stellt der Provider fest, dass ein Kunde diese Wartungsaufgaben nicht durchführt, sollte er den Kunden darüber informieren und eine entsprechende Wartung von ihm fordern. Ggf. kann der Provider für den Kunden entsprechende Aufgaben automatisiert übernehmen.

#### 3.2.1.3 Automatisierte Installation und Aktualisierung

Der Provider stellt dem Kunden entsprechende Methoden zur Verfügung, anhand derer der Kunde die Webanwendung auf den Produkten „Virtuelle Maschinen“ und „Betreutes Hosting“ automatisiert installieren und konfigurieren kann. Dabei wird darauf geachtet, dass schon bei der Installation sicherheitsrelevante Einstellungen entsprechend den Vorstellungen des Providers gesetzt werden.

Zudem sollte regelmäßig überprüft werden können, ob die Konfiguration immer noch gemäß den Vorgaben des Providers festgelegt ist. Dabei wird ebenfalls sichergestellt, dass die verwendeten Versionen der eingesetzten Software aktuell sind. Falls eine veraltete Version erkannt wird, sollte die Einspielung der Aktualisierung automatisiert ablaufen.

#### 3.2.1.4 Dokumentation & Schulung

Der Provider stellt für seine Kunden entsprechende Dokumentation zur Verfügung, um auf die möglichen Gefahren beim Betrieb von Webanwendungen hinzuweisen. Er geht dabei

auf die Verantwortungsbereiche des Kunden ein, wie z.B. Wahl eines sicheren Passwortes und regelmäßige Wartungsaufgaben bei Webanwendungen. Dieses Angebot kann durch entsprechende Schulungen unterstützt werden.

#### **3.2.1.5 Verhinderung von unberechtigten Datenverkehr**

Der Provider kontrolliert den ein- und ausgehenden Datenverkehr zu entsprechenden Webapplikationen und sorgt dafür, dass nur für den Betrieb notwendiger Datenaustausch erlaubt wird. Jeglicher anderweitiger Datenverkehr sollte durch entsprechende Maßnahmen unterbunden werden.

#### **3.2.1.6 Minimale Berechtigungen für Webanwendungen**

Eine Webanwendung sollte möglichst nur diejenigen Berechtigungen auf dem System erhalten, die zur Ausführung der Webanwendung unmittelbar nötig sind. Die zur Webanwendung gehörenden Dateien sollten durch entsprechende Berechtigungen vor unerwünschtem Zugriff geschützt werden.

### **3.2.2 Detektion von Verwundbarkeiten**

Durch die Detektion von verwundbaren Webanwendungen versucht ein Provider möglichen Angreifern zuvorzukommen. Er durchsucht dazu die auf seinen Systemen installierten Webanwendungen nach entsprechenden Schwachstellen.

#### **3.2.2.1 Entdeckung von anfälliger/veralteter Software**

Der Provider sollte die durch den Kunden installierten Webanwendungen regelmäßig inspizieren. Technisch ist dies für den Provider nur bei dem Produkt „Betreutes Hosting“ möglich, da er bei „Virtuellen Maschinen“ nicht feststellen kann, ob und wo Webanwendungen durch den Kunden installiert wurden.

Bei dieser regelmäßigen Überprüfung sollen Webanwendungen, welche Sicherheitslücken aufweisen, entdeckt werden. Die Feststellung von Sicherheitslücken, kann entweder anhand von konkreten Tests an der Webanwendung oder durch Rückschluss aus den eingesetzten Versionen erfolgen.

#### **3.2.2.2 Ermittlung von schwachen Passwörtern**

Der Provider kontrolliert, dass durch die Kunden keine schwachen Passwörter in Webanwendungen verwendet werden. Den dazu nötigen Zugriff hat der Provider jedoch nur bei dem Produkt „Betreute Webanwendung“. Bei den anderen beiden Produkten könnte lediglich eine kleine Anzahl an schwachen Passwörtern durch simples Ausprobieren getestet werden. Daher kann diese Anforderung für diese Produkte nicht umgesetzt werden.

### **3.2.3 Detektion von Angriffen**

Für Provider ist eine zeitnahe Erkennung von infizierten Webanwendungen essentiell. Denn je früher durch entsprechende Gegenmaßnahmen das Ausmaß der Beeinträchtigungen auf eigene und fremde Systeme minimiert wurde, desto besser. Die nachfolgenden Anforderungen legen

mögliche Quellen fest, anhand derer ein Provider einen stattgefundenen Angriff erkennen kann.

### 3.2.3.1 Meldung durch Dritte

Der Provider sollte externen Dritten die Möglichkeit geben, dass sie infizierte Webanwendungen auf den Systemen des Providers melden können. Dazu wird oftmals eine sog. Abuse-Kontaktadresse zur Verfügung gestellt. Externe Dritte stellen eine Infektion möglicherweise noch vor dem jeweiligen Kunden oder dem Provider fest, da sie ggf. durch Auswirkungen der Infektionen direkt beeinträchtigt werden. Denkbare Auswirkungen sind in diesem Zusammenhang beispielsweise, dass ein Dritter durch eine infizierte Webanwendungen angegriffen wird. Ebenso kann er feststellen, dass eine von ihm benutzte Webanwendung Schadcode verteilt. Eine weitere Anforderung an den Provider ist, dass er öffentlich verfügbare Datenbanken, welche infizierte Systeme beinhalten, auf eigene Systeme überwacht.

### 3.2.3.2 Auffälligkeiten im Datenverkehr

Durch die Überwachung des Datenverkehrs aller drei Produkte kann ein Provider Rückschlüsse auf ggf. angegriffene Webanwendungen ziehen. Der Provider analysiert dazu den Datenverkehr nach gewissen Gesichtspunkten. Fällt dabei eine Webanwendung bzw. ein System negativ auf, muss der Provider davon ausgehen, dass dieses einem Angriff zum Opfer gefallen ist.

### 3.2.3.3 Beobachtung des Dateisystems

Durch eine laufende Kontrolle der Inhalte des Dateisystems bei Web- und Applikationsservern sollte ein Provider nach Anzeichen für eine Infektion von Webanwendungen suchen. Bei einer Infektion können beispielsweise Programmdateien der Webanwendungen durch Schadcode ersetzt werden. Solche Hinweise soll ein Provider durch geeignete Maßnahmen erkennen. Da der Provider nur bei den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“ Zugriff auf das Dateisystem hat, ist diese Anforderung nur dort umsetzbar.

## 3.2.4 Reaktion auf Angriffe

Um die negativen Auswirkungen einer Infektion von Webanwendungen auf andere Systeme des Providers zu minimieren, setzt ein Provider Maßnahmen ein, welche die nachfolgenden Anforderungen erfüllen sollten. Dadurch soll verhindert werden, dass eine infizierte Webanwendung weitere Webanwendungen und Dienste behindert.

### 3.2.4.1 Isolation von Mandanten

Der Provider stellt sicher, dass eine Produkt-Instanz eines Kunden keinerlei Zugriff auf die Instanzen anderer Kunden erhält. Insbesondere bei den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“, wo in der Regel mehrere Instanzen unter Verwaltung eines Betriebssystems ausgeführt werden, ist es wichtig, dass weder über das Dateisystem noch über Funktionen des Betriebssystems auf Daten einer anderen Instanz zugegriffen werden kann. So kann verhindert werden, dass sich eine Infektion einer Instanz auf dem System weiterverbreiten kann.

#### 3.2.4.2 Limitierung von Ressourcen

Damit eine angegriffene Instanz nicht die gesamten Ressourcen seines Hostsystemes beanspruchen kann, müssen die durch eine Instanz allozierbaren Ressourcen limitiert sein. Dadurch kann erreicht werden, dass für die restlichen Instanzen auf dem Hostsystem noch ausreichend Ressourcen zur Verfügung stehen, obwohl eine Webanwendung alle Ressourcen beansprucht, die ihr zu geteilt werden. Es sollten dabei u.a. die Ressourcen CPU-Zeit, Arbeitsspeicher, Netzbandbreite und Dateizugriffe limitiert werden.

#### 3.2.4.3 Automatisierte Deaktivierung

Nachdem ein System des Providers erkannt hat, dass eine Infektion stattgefunden hat (vgl. Abschnitt 3.2.3), wird eine automatisierte Deaktivierung der entsprechenden Instanz durchgeführt. Die größte Problematik bei dieser Anforderung liegt darin, dass weitgehend ausgeschlossen werden muss, dass eine Fehldiagnose zur Abschaltung einer nicht infizierte Instanz führt.

#### 3.2.5 Überblick

In Tabelle 3.1 finden sich nochmals alle Anforderungen im Überblick. Es ist dabei jeweils angegeben, auf welche Produkte die entsprechenden Anforderungen zielen. Im nachfolgenden Kapitel 4 werden für diese Anforderungen passende Lösungsansätze vorgestellt.

	Virtuelle Maschinen	Webhosting	Webanwendung
<b>Prävention (3.2.1)</b>			
Sichere Passwortrichtlinien (3.2.1.1)	ja	ja	ja
Regelmäßige Wartung von Webanwendungen (3.2.1.2)	ja	ja	ja
Automatisierte Installation und Aktualisierung (3.2.1.3)	ja	ja	nein
Dokumentation & Schulung (3.2.1.4)	ja	ja	ja
Verhinderung von unberechtigten Datenverkehr (3.2.1.5)	ja	ja	ja
Minimale Berechtigungen für Webanwendungen (3.2.1.6)	ja	ja	ja
<b>Detektion von Verwundbarkeiten (3.2.2)</b>			
Entdeckung von anfälliger/veralteter Software (3.2.2.1)	nein	ja	nein
Ermittlung von schwachen Passwörtern (3.2.2.2)	nein	nein	ja
<b>Detektion von Angriffen (3.2.3)</b>			
Meldung durch Dritte (3.2.3.1)	ja	ja	ja
Auffälligkeiten im Datenverkehr (3.2.3.2)	ja	ja	ja
Beobachtung des Dateisystems (3.2.3.3)	nein	ja	ja
<b>Reaktion auf Angriffe (3.2.4)</b>			
Isolation von Mandanten (3.2.4.1)	ja	ja	ja
Limitierung von Ressourcen (3.2.4.2)	ja	ja	ja
Automatisierte Deaktivierung (3.2.4.3)	ja	ja	ja

Tabelle 3.1: Anforderungen im Überblick



## 4 Lösungsmöglichkeiten

Dieses Kapitel stellt Lösungen vor, welche die Sicherheit von Web- und Applikationsservern verbessern können. Die Lösungen werden dazu erläutert und deren Vor- und Nachteile diskutiert. Zudem werden die durch die Lösung erfüllten Anforderungen aus Abschnitt 3.2 erfasst.

Die Lösungen werden zusätzlich nach den Bereichen kategorisiert, in denen sie eingesetzt werden sollen. Der Abschnitt 4.1 stellt organisatorische Lösungsansätze vor. Daran anschließend zeigt Abschnitt 4.2 welche Maßnahmen zur Verbesserung der Sicherheit am Betriebssystem getätigt werden können. Die Abschnitte 4.3 und 4.4 befassen sich mit der Absicherung von Web- bzw. Applikationsservern. Daran schließt sich in Abschnitt 4.5 die Betrachtung von Maßnahmen an der Infrastruktur eines Providers an.

In Abschnitt 4.6 werden die Lösungen und entsprechend erfüllte Anforderungen in einer Tabelle übersichtlich aufgeführt. Aus der Betrachtung der Lösungen ergeben sich schließlich in Abschnitt 4.7 die am LRZ durchzuführenden Maßnahmen.

### 4.1 Organisatorische Maßnahmen

Mit der Durchführung von bestimmten organisatorischen Maßnahmen bei Providern lässt sich die Sicherheit von Web- und Applikationsservern erhöhen. Entsprechende Maßnahmen werden in den nachfolgenden Abschnitten samt zu erwartender Vor- und Nachteile diskutiert.

#### 4.1.1 Festlegung einer Passwortrichtlinie

Bei Providern werden für die Authentisierung von Kunden, welche privilegierte Operationen ausführen wollen, zumeist Passwörter verwendet. Dabei sollte der Provider eine verbindliche, sichere Passwortrichtlinie für seine Dienste festsetzen. Die Einhaltung dieser Richtlinie ist möglichst durch entsprechende, technische Maßnahmen zu erzwingen. Neben den Empfehlungen zur Wahl eines komplexen Passwortes aus Abschnitt 2.6.2 sind diese Regeln für Provider relevant: (vgl. M 2.11 [BSI13])

1. Die Übertragung von Passwörtern sollte immer verschlüsselt ablaufen. Dies gilt insbesondere auch dann, wenn für diese Übertragung ein vermeintlich sicheres, internes Netze benutzt wird.
2. Erfolgreiche Anmeldeversuche dürfen keinen Rückschluss erlauben, ob eine fehlerhafte Kennung, ein fehlerhaftes Passwort oder ggf. beides ursächlich waren. Desweiteren sollte eine Mehrzahl von erfolglosen Versuchen binnen kurzer Zeit eine Sperrung der Kennung auslösen. Diese Sperrung sollte dabei nicht bei dem entsprechenden Versuch angezeigt werden, sondern sofern möglich dem Nutzer Out of Band (z.B. über E-Mail) mitgeteilt werden.
3. Der Benutzer sollte zu Beginn ein Initialpasswort erhalten, welches er bei der ersten Verwendung ändern muss. Wie bei jeder Änderung des Passwortes sollte die Wahl eines unsicheren Passwortes durch entsprechende Prüfungen verhindert werden.

4. Der Benutzer sollte an die regelmäßige Änderung des Passwortes erinnert werden, falls er sein Passwort über einen längeren Zeitraum nicht geändert hat.
5. Die Passwortdatenbanken sollte nicht im Klartext gespeichert werden. Damit kein Rückschluss auf den Klartext eines Passwortes möglich ist, sollte eine kryptographische Hashfunktion zum Einsatz kommen. Damit selbst bei einer Offenlegung der Datenbank in Folge eines Angriffes die Klartexte der Passwörter weiter geschützt bleiben, empfiehlt sich der Einsatz von sog. Salt bzw. Pepper Zeichenketten, welche ebenfalls in die Hashfunktion miteinbezogen werden.
6. Nutzer sollten die Passwörter nur an gesicherten Orten im Klartext abspeichern bzw. hinterlegen. Der Zugang durch Dritte soll dabei weitestgehend ausgeschlossen werden können.

Eine entsprechend der obigen Angaben umgesetzte Passwortrichtlinie trägt dazu bei, die Angriffe auf sämtliche Dienste zu erschweren, welche Passwortauthentisierung nutzen und unter der Verwaltung des Providers sind. Die Erfolgchancen von einfachen Angriffen (wie z.B. Ausprobieren von weitverbreiteten Passwörter) werden durch diese Richtlinien weitestgehend gemindert.

Die aktuell gültige Passwortrichtlinie am LRZ sieht bereits die Regeln 1., 4. und 6. um. Es sollte überlegt werden, diese Richtlinien um die fehlenden Regeln zu erweitern. [LRZ12a]

Diese Maßnahme ist anwendbar bei den drei in dieser Arbeit betrachteten Produkten „Virtuelle Maschinen“, „Webhosting“ und „Webanwendungen“. Bei den ersten beiden Produkten werden zumeist auch vom Kunden Dienste mit Passwortauthentisierung betrieben. Dabei sollte der Provider den Kunden zur Einhaltung der Empfehlungen dieses Abschnittes auffordern.

##### 4.1.2 Zeitliche Befristung von Produkten

Eine Möglichkeit, um die Anzahl an ungepflegten Instanzen der jeweiligen Produkte eines Providers zu senken, wäre diese zeitlich zu befristen. Vor Ablauf der Frist wird der Kunde durch eine entsprechende Benachrichtigung darauf hingewiesen, dass sein gebuchtes Produkt in Kürze abläuft und er es verlängern muss, sofern er das entsprechende Produkt weiter nutzen möchte. Zusätzlich könnte der Kunde in dieser Benachrichtigung darauf hingewiesen werden, dass ggf. installierte Webanwendungen regelmäßig aktualisiert werden sollten. Somit wird der Kunde regelmäßig an die Existenz der ihm bereitgestellten Produkte und seine Verantwortung für die Pflege derer erinnert.

Es wird damit verhindert, dass eine größere Anzahl an ungepflegten „Karteileichen“ die Sicherheit der Systeme eines Providers negativ beeinflusst. Um den Verwaltungsaufwand bei dieser Lösung zu minimieren, sollten die dazugehörigen Prozesse soweit wie möglich automatisiert werden. Die Verlängerung eines Produktes sollte durch den Kunden selbst in einer Webanwendung vorgenommen werden können. Existiert bereits ein solches Kundenportal, hält sich der Aufwand für die Implementierung dieser Lösung in Grenzen. Ein praktikable Laufzeit für Produkte könnte beispielsweise zwei Jahre betragen. Somit wird regelmäßig sichergestellt, dass die entsprechenden Produkte weiterhin betrieben werden sollen.

Eine solche Befristung bringt jedoch auch Nachteile mit sich. Neben dem einmaligen Aufwand zur Implementierung muss auch im laufenden Betrieb mit Komplikationen gerechnet werden: Ist z.B. die Adresse des verantwortlichen Betreuers nicht mehr aktuell, kann es



passieren, dass das Überschreiten der Frist vom Kunden nicht registriert wird und es dadurch zu einer Abschaltung kommt, die nicht im Sinne des Kunden ist. Ebenso ist es möglich, dass die Bedeutung der E-Mail durch den Kunden falsch eingeschätzt wird und er diese daher ignoriert. Durch den mehrmaligen Versand der E-Mail, insbesondere in kürzeren Zeitabständen je näher der Termin der Deaktivierung kommt, sollte dem entgegengewirkt werden. Zudem sollte die verbleibende Zeit bis zur Deaktivierung dem Kunden direkt ins Auge stechen, indem diese in den E-Mails beispielsweise im Betreff aufgeführt ist.

Trotz dieser Nachteile kann eine solche Lösung gerade bei Providern, welche von den Kunden keine direkten Gebühren erheben, wie das LRZ beim Produkt „Betreutes Hosting“, helfen, die Anzahl ungepflegter Webanwendungen zu minimieren. Gerade weil im universitären Umfeld Ansprechpartner häufig wechseln, geschieht des Öfteren, dass von ehemaligen Mitarbeitern betreute Produkte am entsprechenden Lehrstuhl in Vergessenheit geraten.

Diese Lösung ist der Anforderung nach regelmäßiger Aktualisierung (vgl. Abschnitt 3.2.1.2) zuzuordnen. Auch wenn es keine erzwungene Kontrolle auf Aktualisierung gibt, können zumindest Webanwendungen ganz ohne Pflege nach spätestens einer Laufzeit entdeckt werden.

### 4.1.3 Dokumentation für Kunden

Durch eine umfassende Dokumentation kann ein Provider den Kunden dabei unterstützen, auf den Produkten Webanwendungen sicher zu konfigurieren und zu betreiben. Daher sieht diese Lösung vor, dass ein Provider auf den eigenen Seiten entsprechende Anleitungen anbietet bzw. geprüfte Anleitungen von Dritten verlinkt, wie das Produkt mit häufig durch die Kunden eingesetzten Anwendungen genutzt werden kann. Auch der Einsatz eines sog. Wiki-Systems ist denkbar. Dort können Kunden des Providers solche Anleitungen und Links kollaborativ erstellen und pflegen. Dies setzt jedoch eine gewisse Anzahl an Kunden voraus.

In einer solchen Anleitung sollen die Schritte zur Einrichtung einer Webanwendung beschrieben werden. Insbesondere sollte auf sicherheitskritische Einstellungen und Konfigurationen eingegangen werden. Bei den meisten Webanwendungen ist es beispielsweise nötig, dass in der Konfiguration standardmäßig vergebene Passwörter abgeändert werden müssen. Sinnvoll ist auch in dieser Anleitung auf regelmäßige Wartungsaufgaben, wie z.B. Prüfung von Logdateien oder Aktualisierung der Webanwendung, einzugehen.

Zusätzlich zur Bereitstellung von Dokumentation kann ein Provider seinen Kunden Schulungen anbieten. Dort können dem Kunden die notwendigen Kenntnisse zur Einrichtung, Konfiguration und Wartung von Webanwendungen praxisorientiert vermittelt werden.

Nachteil dieser Maßnahmen ist, dass sie sehr viel Arbeitsaufwand beim Provider bei Erstellung und laufender Aktualisierung erfordern. Außerdem müssen die Kunden das Angebot auch annehmen, damit die Sicherheit der Webanwendungen davon profitiert. Sofern die Diversität der eingesetzten Webanwendungen nicht sonderlich hoch ist, kann es sich bei der Lösung, um ein effektives Mittel handeln, die Sicherheit der entsprechenden Webanwendungen zu erhöhen. Außerdem wird der Kunde dadurch für die Gefahren, die beim Betrieb von Webanwendungen bestehen können, sensibilisiert. Diese Lösung erfüllt die Anforderungen nach Dokumentation und Schulung für Kunden, welche in Abschnitt 3.2.1.4 aufgestellt wurde.

### 4.1.4 Berücksichtigung von externen Abuse-Datenbanken

Damit zwischen Angriff und dessen Entdeckung möglichst wenig Zeit verstreicht, sollten auch frei verfügbare, externe Datenquellen von solchen infizierten Webanwendungen und Servern ausgewertet werden. Diese verzeichnen auffälliges Verhalten von bestimmten Webseiten. Auch das Betrachten von Spam-Blacklisten kann hilfreich sein, um infizierte virtuelle Maschinen zu entdecken.

Die folgende Aufstellung zeigt Dienste an, welche für die Auswertung durch Provider geeignet sind, da man dort Einträge anhand der Nummer des autonomen Systems (AS) durchsuchen kann. Da eine solche ASN alle IP-Netze eines Providers zusammenfasst, kann bei diesen Diensten der Infektionsstatus auch von Providern ohne übermäßigen Aufwand abgefragt werden.

- *Google Safe Browsing*

*Google* untersucht bei der Indizierung von Webseiten für deren Suchmaschine auch, ob die jeweilige Seite Schadcode enthält oder ggf. Anzeichen für einen sog. Phishing-Angriff. Ist das der Fall wird die entsprechende URL in eine Datenbank aufgenommen, welche infizierte Webseiten enthält. Diese Datenbank kann über eine öffentliche API abgefragt werden und wird hauptsächlich durch Browser genutzt, die während des Aufrufes einer Webseite durch den Nutzer, deren Sicherheit darüber überprüfen. [Goo14]

Stellt *Google* eine Infektion fest, versucht es den sog. Abuse-Kontakt, welcher für die betroffene IP-Adresse zuständig ist, zu benachrichtigen. Damit dies zuverlässig funktioniert, kann sich der Provider unter <http://www.google.com/safebrowsing/alerts/> mit seiner ASN registrieren. Um die derzeit infizierten Webseiten aus einem bestimmten AS anzuzeigen, muss an diese Adresse die ASN angehängt und die entsprechende Seite aufgerufen werden:

<http://safebrowsing.clients.google.com/safebrowsing/diagnostic?site=AS:>

- *StopBadware*

Einen ähnlichen Ansatz wie *Google* verfolgt *StopBadware*: Es betreibt ebenfalls eine Datenbank mit Webseiten, welche Schadcode verbreiten. Die erforderlichen Daten erhält die gemeinnützige Organisation von den Partnern *Google*, *ThreatTrack Security* und *NSFOCUS* und aggregiert diese. Eine Suche nach Infektionen innerhalb eines bestimmten AS ist unter <https://www.stopbadware.org/clearinghouse/search/> möglich. Dort wird entsprechenden Zuständigen auch Unterstützung per E-Mail angeboten, um die aufgetretenen Infektionen zu beheben. [SBw14]

- *UCEprotect.net*

Die DNS-basierten Blacklists von *UCEprotect.net* enthalten IP-Adressen, welche durch den Versand von Spam aufgefallen sind. Dazu werden E-Mail-Adressen als sog. Spam-Traps betrieben. Weil diese E-Mail-Adressen ansonsten nicht verwendet werden, kann davon ausgegangen werden, dass es sich bei eingehenden E-Mails um Spam handelt. Solche Blacklists werden durch viele Betreiber angeboten (vgl. Übersicht [Wiki14]). Eine solche DNS-Anfrage an eine Blacklist, muss immer für jede IP-Adressen einzeln gestellt werden. Dies ist für die großen Netze von Providern nicht praktikabel. Eine Besonderheit von *UCEprotect.net* ist, dass dort unter <http://www.uceprotect.net/en/rblcheck.php> eine Anfrage auch nach der ASN gestellt werden kann. Zusätzlich

stellt der Anbieter seine komplette Datenbank an Spam-Versendern kostenlos zum Download zur Verfügung. [UCE14]

Mit diesen externen Quellen kann die Erkennung von angegriffenen Kunden-Instanzen der drei Produkte „Virtuelle Maschinen“, „Betreutes Webhosting“ und „Betreute Webanwendungen“ für einen Provider vereinfacht werden (vgl. Abschnitt 3.2.3.1).

## 4.2 Betriebssystem

In diesem Abschnitt werden Maßnahmen vorgestellt, die auf Ebene des Betriebssystems die Sicherheit verbessern. Es werden aufgrund der weitreichenden Verbreitung von Linux bei Web- und Applikationsserver nur konkrete Lösungen für dieses Betriebssystem betrachtet. Die dahinterstehenden Konzepte sind jedoch auch in anderen Betriebssystemen anwendbar.

### 4.2.1 Einschränkung von Dateiberechtigungen

Die Berechtigungen des Dateisystems bei Linux werden auf Web- und Applikationsserver oftmals weitreichender vergeben, als für den Betrieb des Systems nötig ist. Zumeist wird es einer Webapplikation erlaubt, die eigenen Programmdateien zu bearbeiten. Sollte eine Sicherheitslücke einer Webapplikation bekannt werden, erlaubt das einem Angreifer beliebigen Programmcode mit den Rechten der Webapplikation auszuführen. Dies muss durch entsprechende Berechtigungen verhindert werden. Bei Linux wird jedem Prozess, jedem Ordner und jeder Datei sowohl genau ein Benutzer als auch eine Gruppe zugewiesen. Bei Dateien und Ordnern werden die Berechtigungen dann für den Benutzer, für die Gruppe und für alle anderen vergeben. Nur der Benutzer, welcher auch Besitzer genannt wird, kann diese Berechtigungen der Dateien verändern.

Für das Beispiel eines Webservers, welcher dynamische Webseiten mit *PHP* über *FastCGI* (vgl. Abschnitt 2.3.1) ausliefert, wird folgende Benutzer/Gruppen-Struktur vorgeschlagen:

- **Benutzer *www*:**  
Unter diesem Benutzer läuft der Webserver, d.h. dieser Benutzer benötigt Lesezugriff auf alle Dateien, die durch den Webserver ausgeliefert werden. Ein Webserver liefert in der Regel Webseiten für mehreren Kunden aus, daher ist es nötig, dass der Kunde bzw. dessen Programmcode zu keiner Zeit Rechte als Benutzer *www* erhält.
- **Benutzer *phpX*:**  
Der Benutzer *phpX* existiert genau einmal je Kunde, wobei das *X* durch eine eindeutige Kennung des Kunden ersetzt wird. Der Benutzer wird für die Ausführung von *PHP*-Skripten verwendet.
- **Benutzer *kundeX*:**  
Der Benutzer *kundeX* existiert analog zu *phpX* genau einmal je Kunde und unter diesem Benutzer kann der Kunde sich über Dienste wie z.B. SSH, FTP und SCP auf dem Rechner anmelden und darüber alle Dateien der Webapplikation verändern.
- **Gruppe *kundeX*:** Die Gruppe *kundeX* fasst nun die Benutzer *kundeX*, *phpX* und *www* zusammen. Auch diese Gruppe existiert genau einmal je Kunde. Damit lassen sich Berechtigungen für die Gruppe definieren.

## 4 Lösungsmöglichkeiten

Anhand dieser Benutzer und Gruppen kann nun das Dateisystem eines Webservers mit minimalen Berechtigungen versehen werden. Das Ergebnis findet man dabei im Listing 4.1. Dort werden spaltenweise die Dateiberechtigungen, der Besitzer, die Gruppen und der Name der Dateien bzw. der Verzeichnisse angegeben.

Im vorliegenden Beispielfall würden im Kundenverzeichnis mit Pfad `/kunden` die einzelnen Homeverzeichnisse zu finden sein. Die Berechtigungen gewähren dabei den Besitzer `root` alle Rechte, während die Gruppe, welche die jeweiligen Kunden-Gruppe `kundeX` ist, nur lesen darf. Desweiteren wird rekursiv bei allen Verzeichnissen das sog. Setgid-Bit gesetzt, welches die Gruppe für neu erstellte Dateien bzw. Verzeichnisse vererbt. Andere Benutzer erhalten keinerlei Rechte.

Im Homeverzeichnis der Benutzer existieren vier Verzeichnisse: `data` ist für Daten des Kunden, welche unter keinen Umständen über den Webserver ausgeliefert werden sollen. In `tmp` kann der Kunde bzw. dessen Webapplikation temporäre Daten ablegen. Der Ordner `logs` ist für Protokolldateien des Webservers gedacht und im sog. Webroot `htdocs` liegen die Webapplikationen selbst. Bei allen Verzeichnissen ist wieder die entsprechende Kunden-Gruppe als Gruppe eingestellt. Während die Gruppe bei `tmp` Lese- und Schreib-Rechte besitzt, kann sie in `htdocs` und `logs` nur lesen und erhält überhaupt keinen Zugriff auf `data`. Der Besitzer ist bei `logs` und `tmp` `root`, während er bei `data` und `htdocs` `kunde1` ist, da die Daten dort unter der Verwaltung des Kunden und seiner Nutzerkennung `kunde1` stehen. Dem Besitzer werden volle Berechtigungen zugewiesen, da er als Besitzer ohnehin die Berechtigungen der Datei ändern kann.

Im Webroot `htdocs` ist der Besitzer und die Gruppe aller Dateien und Verzeichnisse auf `kunde1` festgelegt. Sämtliche zur Webapplikation gehörenden Dateien sollten dabei für die Gruppe nur lesbar sein. Schreib-Rechte für die Gruppe sollten nur auf Dateien und Verzeichnisse angewendet werden, welche wirklich von der Webapplikation beschrieben werden müssen. Oftmals sind das Konfigurationsdateien, Upload-Verzeichnisse und Datenbanken. Die Ausführung von *PHP*-Skripten sollte in diesen beschreibbaren Ordnern durch die Webserverkonfiguration unterbunden werden. Weitergehend sollte der Webserver durch eine Prüfung des Besitzers von *PHP*-Skripten sicherstellen, dass diese durch den Benutzer `kunde1` erstellt wurden und nur dann ausführen.

```
# Kundenverzeichnis
> ls /kunden/
drwxr-s--- root   kunde1 kunde1
drwxr-s--- root   kunde2 kunde2

# Homeverzeichnis des Kunden
> ls /kunden/kunde1/
drwx--S--- kunde1 kunde1 data
drwxr-s--- kunde1 kunde1htdocs
drwxr-s--- root   kunde1 logs
drwxrws--- root   kunde1 tmp

# Webroot des Kunden
> ls /kunden/kunde1/htdocs/
-rw-r----- kunde1 kunde1 index.php
drwxr-s--- kunde1 kunde1 lib
drwxrws--- kunde1 kunde1 upload
```

## Listing 4.1: Verzeichnisse mit minimalen Berechtigungen für Webserver

Bei Zugriff der Nutzer sollte die Einstellung einer `umask` vorgenommen werden. Sie gibt an, welche Berechtigung neu erstellte Dateien und Verzeichnisse erhalten sollen. Standardmäßig ist diese `umask` meist auf `0022` eingestellt. Damit erhalten neue Dateien bzw. Verzeichnisse volle Berechtigungen für Besitzer, und Lese-Berechtigung für die Gruppe und andere Benutzer. Besser wäre eine Einstellung von `0026` bzw. `0006`, da diese andere Nutzer ausschließt und der Gruppe Lese- bzw. Lese-Schreib-Berechtigung verschafft.

Diese Lösung könnte ein Provider auf die Produkte „Betreutes Hosting“ und „Webanwendungen“ implementieren. Damit würden minimale Rechte für Webanwendungen vergeben werden (vgl. Abschnitt 3.2.1.6). Außerdem dient strikte Anwendung von Dateiberechtigungen der Isolierung von einzelnen Kunden auf Servern, welche mehrere Kunden beherbergen (vgl. Abschnitt 3.2.4.1).

Solch strikt vergebene Dateiberechtigungen bringen jedoch auch betriebliche Nachteile mit sich. Durch die Trennung des Kunden in zwei Benutzer, wird die Komplexität auch für den Endkunden erhöht. Es kann sein, dass der Kunde über Dienste wie SSH, FTP keinen Zugriff mehr auf bestimmte Daten erhält, welche von der Webanwendung geschrieben wurden. Das kann z.B. durch eine falsche bzw. nicht beachtete `umask`-Einstellung verursacht werden.

Ein erheblicher Nachteil dieser strikten Dateiberechtigungen sind die Einschränkungen bei Funktionalität von Webanwendungen. Bedingt durch die verweigerten Schreib-Rechte kann sich eine Webapplikation beispielsweise nicht mehr durch ein ggf. implementiertes Auto-Update selbst aktualisieren. Durch veraltete Webanwendungen können Sicherheitslücken entstehen, welche schwerwiegendere Auswirkungen als zu lasch vergebene Dateiberechtigungen haben können. Daher sollte je nach Anwendungsfall ein Kompromiss zwischen der vollen Implementierung dieser Vorgaben und der einfachen Möglichkeit zur Aktualisierung von Webanwendungen getroffen werden.

Aus diesen Gründen werden die Dateiberechtigungen beim Webhosting am LRZ derzeit ohne Trennung in zwei verschiedene Benutzer je Kunden vergeben. Es werden also die Webanwendungen unter demselben Benutzer `kunde1` ausgeführt, der sich auch über SSH bzw. FTP einloggt. Auf die Verwendung des Benutzers `php1` wird verzichtet. Damit wird den Webanwendungen die Möglichkeit gegeben, eigene Programmdateien zu bearbeiten, um damit automatische Aktualisierungen der Webanwendung nicht zu verhindern.

#### 4.2.2 Ausführung von Diensten in einem Chroot

Eine Möglichkeit verschiedene Dienste auf einem Linux System zu isolieren ist ein sog. Chroot. Dabei handelt es sich um einen Systemaufruf von Unix-Betriebssystemen, welcher erlaubt, dass der Pfad des Rootverzeichnisses für den aufrufenden Prozess geändert wird. Auf zuvor geöffnete Filedeskriptoren des entsprechenden Prozesses kann dieser auch nach dem Systemaufruf von `chroot` zugreifen, selbst wenn diese außerhalb des neuen Rootverzeichnis liegen. Nach dem Wechsel geöffnete Dateien müssen sich jedoch innerhalb des neuen Rootverzeichnisses befinden. Der `chroot`-Systemaufruf kann nur durch den `root`-Benutzer erfolgen. Ein Prozess, welcher unter diesem Benutzer läuft, kann nicht sicher in einem „Chroot Jail“ eingesperrt werden, da er mit Hilfe seiner erweiterten Rechte wieder ausbrechen kann. [MAN10]

Einige Anwendungen implementieren den `chroot`-Systemaufruf bereits in deren Programmcode, um gewisse Unterprozesse vom restlichen System zu isolieren. Der quellenoffene SSH-Dienst *OpenSSH* kann den Aufruf beispielsweise verwenden, um bestimmte Benutzer

nach dem Login in eine solche Umgebung einzusperren. Prinzipiell lässt sich auf einem Unix-Betriebssystem jeder Prozess in einer Chroot-Umgebung betreiben. Hierfür muss der entsprechende Prozess jedoch alle notwendigen Dateien, Verzeichnisse und Devices im Chroot vorfinden, welche er zur Ausführung benötigt.

Um nun von den Vorteilen einer Chroot-Umgebung bei Web- und Applikationsserver zu profitieren, wäre es nötig, entweder den Server selbst oder gewisse Unterprozesse von Laufzeitumgebungen wie z.B. *PHP* mit *FastCGI*-Anbindung in ein solches Chroot einzusperren. Da jedoch diese Anwendungen weitreichende Abhängigkeiten zu anderen Anwendungen und Bibliotheken haben, würde die Erstellung einer entsprechenden Umgebung den Rahmen dieser Arbeit sprengen. Eine Beschreibung der notwendigen Vorgehensweise ist unter [Fro09] zu finden. Neuere *PHP* Versionen ( $\geq 5.3$ ) unterstützen mit dem *FastCGI Process Manager (FPM)* eine vergleichsweise einfache Möglichkeit *PHP*-Webanwendungen in einem Chroot auszuführen. [PHP14]

An dem Beispiel in Listing 4.2 wird jedoch die Erzeugung eines Chroots, welches die minimale Shellumgebung *Busybox* enthält, verdeutlicht. Die Erzeugung wird durch den `root`-Benutzer vorgenommen und mittels des `chroot`-Hilfskommando wird anschließend das Rootverzeichnis geändert. Diesem Kommando wird zusätzlich noch der gewünschte Benutzername – in diesem Fall `kunde1` – übergeben, welcher nach dem Wechsel des Rootverzeichnisses verwendet werden soll, sowie der auszuführende Prozess mit dessen Pfad innerhalb des Chrootes. Anhand der Beispieldatei `test123.txt` kann man erkennen, dass sich wirklich das Rootverzeichnis der Umgebung geändert hat.

```
# Anlegen von Verzeichnissen
> mkdir /_chroot_test/ /_chroot_test/dev/ /_chroot_test/bin

# Installieren von Busybox im Chroot
> /bin/busybox --install /_chroot_test/bin/

# Beispieldatei anlegen
> touch /_chroot_test/test123.txt

# Wechsel in Chroot
> chroot --userspec=kunde1 /_chroot_test/ /bin/sh
BusyBox v1.20.2 (Ubuntu 1:1.20.0-8) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# Zeige Root Verzeichnis
> ls /
bin          dev          test123.txt
```

Listing 4.2: Chroot in ein Verzeichnis mit *Busybox*-Umgebung

Diese Lösung ist einsetzbar für die Produkte „Betreutes Webhosting“ und „Betreute Webanwendungen“. Sie sichert dabei die Anforderungen nach Isolation von Mandaten (vgl. Abschnitt 3.2.4.1) bei einer Reaktion auf einen Angriff ab, indem es Mandanten und sogar Prozesse innerhalb eines Mandaten in unterschiedlichen Chroot-Umgebungen voneinander isoliert. Innerhalb von Chroots können Laufzeitumgebungen und Anwendungssoftware weitgehend unabhängig vom eigentlichen Betriebssystem ausgeführt werden, da alle erforderlichen Anwendungen und Bibliotheken in das Chroot kopiert werden. Somit können Änderungen und Aktualisierungen des Betriebssystems im tatsächlichen Rootverzeichnis des jeweiligen Servers

die Anwendung im Chroot nicht wesentlich beeinflussen. Es wird dadurch die Stabilität der Umgebung aus Anwendungssicht erhöht. Ein weiterer Vorteil ist, dass der Provider nun verschiedenste Chroot-Umgebungen für seine Kunden anbieten kann. Der Inhalt einer solchen Umgebung kann an die Erfordernisse des Kunden flexibler angepasst werden, da eine Änderung im einzelnen Chroot nur den jeweiligen Kunden betrifft.

Allerdings kann eine komplette Isolierung dadurch nicht realisiert werden, da sich die verschiedenen Chroots eine Linux-Kernel-Instanz teilen. Dadurch kann es auch bei Einsatz von unterschiedlichen Chroots zu Konflikten auf den durch den Kernel bereitgestellten Ressourcen kommen. Dies betrifft z.B. das Prozessmanagement und den Netzstack. Im nachfolgenden Kapitel wird eine Lösung vorgestellt, welche diese Mängel behebt.

Ein weiterer Nachteil besteht darin, dass es ebenso wie im Hostbetriebssystem nötig ist, die Anwendungen und Bibliotheken in einem Chroot laufend zu aktualisieren. Durch die feingranulare Aufteilung einzelner Webanwendungen in eigene Chroots entsteht ein nicht zu unterschätzender Aufwand, entsprechende Update-Maßnahmen zu implementieren.

### 4.2.3 Linux Containers (LXC)

Bei den Linux Containers (LXC) handelt es sich um eine Container-Virtualisierung, welche auf Funktionen von aktuellen Linux-Kerneln basiert. Bei der Container-Virtualisierung werden durch eine einzige Kernel-Instanz mehrere virtuelle Userspace-Umgebungen betrieben. Der Kernel versucht sich gegenüber diesen Userspace-Umgebungen möglichst so zu verhalten wie eine dedizierter Linux-Kernel.

Mit der sog. Namespace-Isolation des Kernels kann LXC Kernelfunktionalitäten und -ressourcen, wie z.B. Prozess-Identifizierer, Netzadapter und Interprozesskommunikation für eine bestimmte Untergruppe an Prozessen zur Verfügung stellen. Innerhalb eines Namespaces verhält sich der Kernel gegenüber den Prozessen so, als ob es nur diesen einen Namespace gäbe. Die Limitierung, die Priorisierung und das Accounting der Hardware-Ressourcen zwischen unterschiedlichen Containern wird durch Control-Groups (cgroups) durchgeführt. Diese bestehen aus einer Gruppe von Prozessen, für die gewisse Richtlinien im Bezug auf die Hardware gelten. Außerdem setzt LXC Chroots (vgl. Abschnitt 4.2.2) und Mandatory Access Control (vgl. Abschnitt 4.2.5) ein. [LXC14] [Men13]

LXC stellt eine Weiterentwicklung der Chroots dar, weil diese Container neben der durch Chroots erreichten Isolation des Dateisystems auch die Funktionalitäten des Linux-Kernel isoliert anbieten können. Es eignet sich daher ebenso wie Chroots für die Produkte „Betreutes Webhosting“ und „Betreute Webanwendungen“ eines Providers und erfüllt folglich die Anforderung nach Isolation der Mandanten (vgl. Abschnitt 3.2.4.1). Als Erweiterung dazu kann LXC auch die beanspruchten Ressourcen durch den Einsatz von einzelnen Containern beschränken und implementiert damit die Anforderung nach Ressourcen Limits (vgl. Abschnitt 3.2.4.2). Weiterhin können in einem solchen Container auch Applikationen ausgeführt werden, die zur Ausführung `root`-Rechte benötigen, da LXC die Rechte soweit einschränkt, dass trotz der Verwendung des privilegierten `root`-Benutzers kein Ausbruch aus dem Container möglich ist. Das Namespacing des Netzstacks durch den Linux-Kernel verhindert Überschneidung mit anderen Mandanten und so können z.B. für jeden Kunden individuelle Firewallregeln angewandt werden.

Wie bei Chroots besteht ein großer Nachteil darin, dass durch die große Anzahl an Chroots, die gepflegt werden müssen, ein deutlich erhöhter Aufwand zu erwarten ist, wenn diesen mit dem einer Lösung ohne den Einsatz von Containern für jeden einzelnen Mandanten vergleicht.

Ein Ansatz wie dieses Problem gelöst werden könnte, wird mit dem LXC-Verwaltungstool *Docker* vorgestellt.

#### 4.2.4 Docker

Mit dem ersten Release im März 2013 ist das Open-Source Projekt *Docker* noch relativ jung. Auch die Entwickler empfehlen ihre Software noch nicht zum Einsatz auf produktiven Systemen. Trotzdem ist die Betrachtung von *Docker* im Rahmen dieser Arbeit interessant, da es die in den vorherigen Abschnitten über LXC und Chroots aufgezeigten Nachteile im Bereich der Verwaltung und Pflege von verschiedenen Chroots beseitigt. Dazu nutzt es die durch den Linux-Kernel und LXC bereitgestellten Funktionalitäten und ermöglicht Anwendungen in einer einheitlichen Umgebung mit sehr geringem Virtualisierungs-Overhead auszuführen. [Dock14a] [Dock14d]

Obwohl *Docker* noch nicht für den produktiven Einsatz vorgesehen ist, wird die Lösung nachfolgend genauer betrachtet, da sie für Provider die Bereitstellung von den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“ erheblich vereinfachen kann.

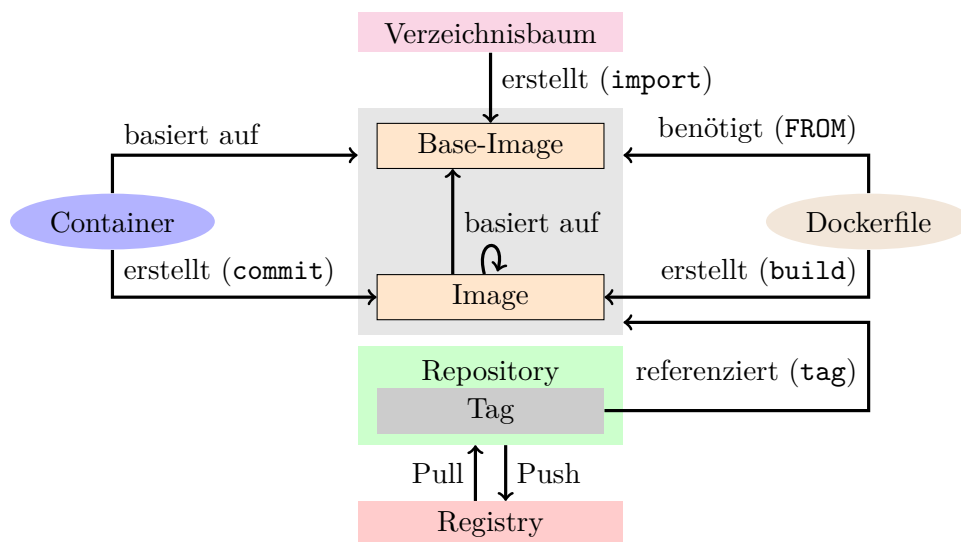


Abbildung 4.1: *Docker* Komponenten im Überblick

Die Struktur von *Docker* ist in Abbildung 4.1 dargestellt, sie besteht aus den folgenden Komponenten:

- **Container**

Ein Container bei *Docker* entspricht im Wesentlichen einem Container bei LXC. Bei *Docker* basiert ein Container immer auf einem Image. Für ein bestimmtes Image können beliebig viele Container gestartet werden.

Es werden dabei nur die Veränderungen, welche in den jeweiligen Containern vorgenommen wurden, gespeichert. Es muss also nicht der Inhalt eines Images bei Erstellung eines Containers kopiert werden, sondern erst wenn Änderungen daran vorgenommen



werden. Das wird durch sog. Copy-On-Write (COW) ermöglicht, welches durch eines der Storage-Backends verwaltet wird: AUFS, LVM-Snapshots oder BTRFS-Snapshots.

Wird ein Container gestartet, erhält dieser automatisiert eine IP-Adresse aus einem privaten Adressbereich (nach [RFC1918]). Die private Adresse eines Containers wird durch Network-Address-Translation (NAT) auf diejenige des Hosts umgeschrieben, sobald der Container auf externe Netze zugreift. Soll ein Dienst des Containers öffentlich verfügbar sein, so wird dieser über ein Port-Forwarding unter einer IP-Adresse des Hosts erreichbar gemacht. Standardmäßig unterbindet *Docker* jegliche Kommunikation zwischen zwei Containern.

Um die Rechte eines Containers weiter einzuschränken nutzt *Docker* das Mandatory Access Control System *AppArmor* (vgl. Abschnitt 4.2.5).

- **Image**

Ein Image stellt die Dateien und Verzeichnisse zur Verfügung, welche ein Container bei der Neuerstellung enthält. Ein Image selbst ist nicht veränderlich, es kann also nur lesend darauf zugegriffen werden. Jedes Image erhält einen eindeutigen Identifier, welcher 64 Bit lang ist und in der Regel im hexadezimalen Format angegeben wird. Ein Image kann auf einen sog. Parent-Image basieren, dann enthält das jeweilige Image nur die veränderten Daten zusammen mit der Referenz auf den Identifier des Parent-Images. Besitzt ein Image kein Parent-Image, so spricht man auch von einem Base-Image. [Dock14e]

Ein neues Image kann durch diese drei Möglichkeiten erstellt werden:

- Erstellung eines neuen Base-Images durch Import eines Verzeichnisbaumes.
- Mit dem sog. Commit eines gestoppten Containers wird dessen Dateisystem in ein Image umgewandelt.
- Durch Dockerfiles kann aus einem bestehenden Image ein Neues erstellt werden. Das Dockerfile enthält die entsprechend im Image durchgeführten Veränderungen.

- **Repository**

Ein Image wird ausschließlich über den Image-Identifier referenziert. Weist man einem Image einen bestimmten Namen zu, so wird es dann als Repository bezeichnet. Dieser Name besteht nur aus den Zeichen **A-Za-z0-9-** und enthält ggf. zur Trennung / bzw. :. Der Name ist nach folgenden Schema aufgebaut:

(PRÄFIX/)NAME(:TAG)

Die eingeklammerten Bereiche sind dabei optional. Wird kein Tag angegeben, wird das Standard-Tag `latest` verwendet. Das Präfix kann ggf. durch weitere / noch öfters unterteilt werden. Als Repository werden alle Tags, die in Präfix und Name übereinstimmen zusammengefasst.

Dieses Namensschema dient dazu, den Umgang mit den etwas unhandlichen Image-Identifiern zu vereinfachen, denn ein solcher Name kann einfacher gemerkt und eingegeben werden als eine langer hexadezimaler Identifier. Außerdem ermöglichen Repositories, immer die aktuellste Version eines Images zur Verfügung zu stellen, da die Referenzen der Repositories im Gegensatz zu den Inhalten eines Images geändert werden können.

- **Registry**

Über Registries können bereits erstellte Images ausgetauscht werden. Dazu erlaubt eine Registry, entsprechend autorisierten Benutzern eigene Images anderen Nutzern zur Verfügung zustellen. Die Entwickler von *Docker* betreiben eine zentrale Registry als Cloud-Dienst, welche alle hochgeladenen Images öffentlich zur Verfügung stellt. Soll der Zugriff auf die Images restriktiver gehandhabt werden, muss dafür eine private Registry betrieben werden. Die dafür notwendigen Anwendungen sind als Open-Source-Software verfügbar.

Soll nun ein lokales Repository hochgeladen werden, so wird durch *Docker* ein sogenannter Push ausgeführt und das Repository samt referenzierter Images zu den Registry-Severn übertragen. Soll ein Image von der Registry heruntergeladen werden, so wird das durch einen Pull erledigt.

- **Volume**

Als Volumes werden bestimmte Verzeichnisse innerhalb eines Containers deklariert, die nicht Teil des Copy-On-Write Filesystems im Container sind. Diese Volumes können zwischen mehreren Containern geteilt werden. Durch den fehlenden Overhead des Copy-On-Write Mechanismus ist zudem die Performance höher.

- **Dockerfile**

In einem Dockerfile wird beschrieben, welche Kommandos und Veränderungen an einem bestehenden Image durchgeführt werden sollen. Dies dient dazu, das Image später erneut zu generieren, falls das jeweilige Parent-Image aktualisiert worden ist.

Im Dockerfile können diese Kommandos verwendet werden:

- Das FROM-Kommando gibt an auf welchem Repository das Dockerfile basiert.
- Die MAINTAINER-Anweisung enthält die E-Mail-Adresse von Verantwortlichen für das Dockerfiles.
- Mittels RUN werden Befehle in dem jeweiligen Image ausgeführt. Das ist das am häufigsten verwendete Kommando, denn damit können beispielsweise Pakete installiert werden, Anwendungen heruntergeladen werden und vieles mehr.
- Mit dem ADD-Kommando kann eine Datei aus dem Verzeichnis des Dockerfiles in das Image kopiert werden. Häufig wird diese Anweisung dafür verwendet, um längere Skripte in das Image zu kopieren und anschließend auszuführen.
- Durch die EXPOSE-Anweisung werden im Dockerfile diejenigen Ports deklariert, welche durch ein Port-Forwarding von außen erreichbar sein sollen.
- Mit dem VOLUME-Kommando werden die Pfade des Images festgelegt, welche durch ein Volume abgespeichert werden sollen.
- Durch CMD wird der Standard-Befehl definiert, welcher bei Start des Images im jeweiligen Container ausgeführt werden soll.

Mit Unterstützung dieser Komponenten kann *Docker* die Produkte „Betreutes Hosting“ und „Betreute Anwendungen“ für Provider einfacher handhabbar machen. Die mächtige Verwaltung von Images durch Repositories und Registries ermöglicht eine einfache Versionierung, Verteilung und Aktualisierung von verschiedensten Laufzeitumgebungen und Webanwendungen. Ein detailliertes Beispiel wie *Wordpress* als „Betreute Webanwendung“ mit *Docker*

angeboten werden kann, ist unter Abschnitt 5.3 zu finden. *Docker* kann die Anforderungen nach regelmäßiger Wartung (vgl. Abschnitt 3.2.1.2) und automatisierter Installation und Konfiguration (vgl. Abschnitt 3.2.1.3) von Webanwendungen erfüllen. Durch die Verwendung von LXC werden auch die Anforderungen nach Isolation (vgl. Abschnitt 3.2.4.1) und Ressourcenlimitierung (vgl. Abschnitt 3.2.4.2) im Falle einer Kompromittierung berücksichtigt. Durch den obligatorischen Einsatz von *AppArmor* und entsprechende Rechte-Vergabe bei Erzeugung von Images kann die Anforderung erreicht werden, dass die Rechte der Webanwendung möglichst minimal sind (vgl. Abschnitt 3.2.1.6).

#### 4.2.5 Mandatory Access Control mit *AppArmor*

Dieser Abschnitt beschäftigt sich mit der weitergehenden Absicherung von Linux-Systemen durch den Einsatz von Mandatory Access Control (MAC). Mit MAC kann ein Linux-System bezüglich der Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit zusätzlich zu den Standard-Mechanismen abgesichert werden. Diese standardmäßige Rechteverwaltung implementiert Discretionary Access Control (DAC), welche Rechte ausschließlich auf Grund der Identität eines Akteurs zuteilt. Um die Rechte zu beschreiben wird zwischen Subjekten und Objekten unterschieden. Als Subjekte werden Akteure wie z.B. Prozesse und Benutzer bezeichnet, die ein gewisses Recht anfordern, Operationen auf Objekten wie z.B. Dateien und Ressourcen durchzuführen. Bei DAC wird nun für eine beabsichtigte Operation Subjekt und Objekt betrachtet und aufgrund dessen die Entscheidung getroffen, ob diese durchgeführt werden darf oder eben nicht.

Um nun Dateien und Verzeichnisse eines Benutzers vor anderen Benutzern zu schützen, können die entsprechenden Regeln in diesem Modell umgesetzt werden. Jedoch gibt es Fälle, welche nicht durch DAC abgebildet werden können. Ein solcher ist die Änderung des Passworts bei Linux: Um sein Passwort zu ändern, benötigt der Benutzer Schreib-Rechte für die Datei `/etc/shadow`. Diese können dem Benutzer jedoch nicht zugeteilt werden, da er ansonsten beliebige Passwörter anderer Nutzer ändern könnte. Man behilft sich damit, dass es einen Befehl `passwd` gibt, welcher zur Änderung benutzt werden muss. Dieser wird durch das sog. Setuid-Bit immer mit den Rechten des `root`-Benutzers ausgeführt, unabhängig davon, welcher Benutzer den Befehl aufruft. Das Problem dieser Lösung besteht darin: Sobald eine Sicherheitslücke im `passwd`-Befehl bekannt wird, kann jeder beliebige Nutzer über den Befehl die Rechte des `root`-Benutzers erlangen.

Das Modell von DAC kann bei komplexeren Szenarien die Schutzziele nicht immer optimal gewährleisten. Mandatory Access Control (MAC) geht dabei einen konträren Weg: Zu jedem Subjekt und Objekt existieren festgelegte Richtlinien, die entsprechend Operationen zulassen bzw. verweigern. Dazu muss eine MAC an zentraler Stelle die Einhaltung dieser Richtlinien durchsetzen und dabei die Operationen aller Prozesse überwachen. Bei Linux ermöglicht das Framework Linux Security Modules (LSM) an entsprechender Stelle im Betriebssystem-Code das MAC auszuführen, damit die Einhaltung der Richtlinien sichergestellt werden kann. Grundsätzlich kommt die MAC bei Linux zusätzlich zum bereits bestehenden DAC zum Einsatz. Sollte eine Aktion schon durch die Richtlinien des DAC verhindert werden, kommt MAC gar nicht erst zur Anwendung.

Damit bei MAC die Zuweisung von Rechten auf Objekten für bestimmte Subjekte erfolgen kann, wird durch sog. Type-Enforcement jedem Subjekt und jedem Objekt ein Typ (alternative Bezeichnungen: Labels, Domain) zugewiesen. Die Richtlinien definieren dann die Berechtigungen anhand des Typs. Die Richtlinien sollten dabei möglichst so gestaltet werden,

dass ein Prozess genau diejenigen Berechtigungen erhält, die zu seiner vollen Funktionalität notwendig sind. [Spe07]

Die beiden MAC-Systeme *AppArmor* (*Application Armor*) und *SELinux* (*Security-Enhanced Linux*) haben sich im Laufe der letzten Jahre aus einer größeren Menge an MACs hervorgehoben, dadurch dass sie den Einzug in die großen Linux-Distributionen geschafft haben. Die beiden unterscheiden sich vor allem in der Art, wie Richtlinien konfiguriert und zur Anwendung gebracht werden:

- Definition der Richtlinien

Richtlinien bei *SELinux* definieren ausschließlich auf Basis der entsprechend vergebenen Typen für Subjekt und Objekt. Diese Zuordnung der einzelnen Dateien zu den verschiedenen Typen wird auf Dateisystemebene durchgeführt. In den Richtlinien wird nun jedem Typ die entsprechende Berechtigung zugewiesen. Diese Richtlinien müssen, bevor sie eingesetzt werden können, kompiliert werden. Die durch *SELinux* aufgestellten Richtlinien gelten für alle laufenden Prozesse.

*AppArmor* verfolgt bei der Zuordnung von Berechtigungen in den Richtlinien einen anderen Ansatz: Anstatt sich auf eine Typenzuordnung durch das Dateisystem zu verlassen unterscheidet es anhand des Pfads von Subjekt und Objekt. Je nach Anwendung muss dabei ein sog. Profil angelegt werden. Existiert kein Profil, so wird diese Anwendung nicht durch *AppArmor* limitiert. Die zu vergebenden Berechtigungen sind nicht so detailliert abgestuft wie bei *SELinux*. Das hat jedoch auch den Vorteil, dass *AppArmor*-Richtlinien meist deutlich kürzer sind als die entsprechenden von *SELinux*. Zusätzlich benötigen diese keine explizite Kompilierung, bevor diese angewandt werden können.

- Aktivierung des MAC-Systems

*SELinux* unterscheidet systemweit nur zwischen zwei Betriebsmodi: Im Modus „Permissive“ werden Verletzungen der Richtlinien nur gemeldet, jedoch nicht unterbunden. Der Modus „Enforcing“ dagegen unterbindet Verletzungen der Regeln. Sobald man also *SELinux* aktivieren will, müssen die Richtlinien für das gesamte System korrekt sein. Denn ansonsten werden ggf. Dienste durch *SELinux* blockiert, die berechtigterweise Operationen im Rahmen ihrer Aufgaben durchführen, jedoch nicht in den Richtlinien enthalten sind.

Bei *AppArmor* wird die Aktivierung je definiertem Profil in den Richtlinien umgesetzt. Somit können Anwendungen Schritt für Schritt durch *AppArmor* abgesichert werden. Sollen Regeln erst getestet werden, so bietet sich dafür der „Complain“-Modus an. Ein Profil in diesem Modus meldet lediglich eine Überschreitung von Berechtigungen, sie wird dadurch aber nicht verhindert. Das kann bei der Entwicklung der Richtlinien eingesetzt werden. Soll ein Profil durch *AppArmor* durchgesetzt werden, muss es im „Enforce“-Modus sein. Damit werden die definierten Berechtigungen erzwungen.

- Labeling von Subjekten und Objekten

Bei *SELinux* ist es nötig, dass alle Dateien und Verzeichnisse dem entsprechenden Typ auf Dateisystemebene zugewiesen bekommen. Das setzt voraus, dass sowohl das eingesetzte Dateisystem als auch Befehle, die Operationen auf den Dateien ausführen, diese entsprechenden Label unterstützen. Des Weiteren sind Anpassungen bei einigen

Standard-Befehlen von Linux nötig, um die entsprechenden *SELinux*-Informationen, wie z.B. Typ und Kontext anzuzeigen.

Bei *AppArmor* erfolgt die Zuordnung von Typen anhand des entsprechenden Pfads der Dateien. Es sind keine Modifikationen auf Dateisystemebene oder an den Standard-Befehlen nötig.

[MAN05][MAN13]

Nach dem Vergleich der beiden MAC-Systeme stellt sich *AppArmor* als geeigneter für den Einsatz bei Providern heraus, da es auf dessen Systemen durch die Unterteilbarkeit in verschiedene Profile sukzessive für gefährdete Dienste implementiert werden kann. Durch die pfadbasierte Zuordnung des Typs bei Subjekten und Objekten wird der Aufwand bei Implementierung und Pflege des MAC-Systems im Vergleich zu der Zuordnung durch das Dateisystem bei *SELinux* deutlich minimiert. Die Verwendung von Pfaden bei der Definition von Richtlinien dient außerdem der besseren Lesbarkeit.

Durch MAC kann die Anforderung nach möglichst minimaler Rechtezuteilung an Webanwendungen erfüllt werden (vgl. Abschnitt 3.2.1.6). Außerdem dient eine MAC auch der besseren Isolation von einzelnen Mandanten bei gemeinsamer Nutzung von Ressourcen (vgl. Abschnitt 3.2.4.1).

Ein *AppArmor*-Profil für den *Apache Tomcat*-Applikationsserver wird im Abschnitt 4.4.2 entwickelt. Dieses Profil wird dann im Abschnitt 5.1.4 durch *Puppet* auf einem Server installiert.

## 4.3 Webserver

Dieser Abschnitt behandelt Lösungsvorschläge, die sich speziell auf die Absicherung von Sicherheitsproblemen bei Webservern beziehen. Sofern Applikationsserver Dienste über das HTTP(S)-Protokoll anbieten sind die vorgestellten Lösungen dort ebenfalls anwendbar.

### 4.3.1 Einsatz einer Web Application Firewall (WAF)

Eine Web Application Firewall (WAF) kontrolliert auf HTTP(S)-Ebene die Datenkommunikation zwischen Client und Server. Dabei versucht es, mögliche Angriffe auf Webapplikationen zu erkennen und anschließend zu verhindern. Eine verwundbare Webapplikation (vgl. Abschnitt 2.6.5) kann somit durch eine WAF geschützt werden. Dies gilt insbesondere dann, wenn eine Verwundbarkeit einer Webapplikation bekannt ist, jedoch keine Möglichkeit besteht, diese durch Aktualisierung oder Änderung des Codes der Webapplikation zu beheben. Denn durch speziell darauf abzielende Regeln einer WAF kann die Ausnutzung solcher Verwundbarkeiten unterbunden werden. Grundsätzlich muss eine WAF händisch gepflegt werden, da das standardmäßige Regelwerk inkompatibel mit der zu sichernden Webapplikation sein kann. Zusätzlich müssen aufgetretene Regelverstöße zeitnah analysiert werden, so dass es zu keiner Einschränkung im Betrieb der Webapplikation kommt. [OWASP08]

Software-WAFs werden meist direkt auf dem Webserver installiert, auf dem auch die Webanwendung zur Ausführung kommt. Die Regeln müssen dementsprechend auch nur die Besonderheiten der auf diesem Webserver ausgeführten Webanwendungen berücksichtigen. Entweder wird eine solche Lösung als sog. Reverse-Proxy zwischen den eigentlichen Webserver und den zugreifenden Client positioniert oder die Software-WAF wird direkt als Modul durch den Webserver geladen.

Das Open-Source-Projekt *ModSecurity* ist eine Software-WAF, die sich als Modul in einem Webserver einbinden lässt. Es werden dabei die Webserver *Apache HTTP*, *Nginx* und *IIS* unterstützt. Durch das Open Web Application Security Project (OWASP) wird das sog. Core Rule Set (CRS) angeboten, welches vorgefertigte Regeln gegenüber Schwachstellen in Webapplikationen bietet. Eine Auswahl dieser Regeln kann dann für eine bestimmte Webanwendung eingesetzt werden. [OWASP14][PK09]

Eine weitere Möglichkeit zur Realisierung einer WAF ist durch sog. Hardware Appliances gegeben. Diese stellen eine deutlich kostspieligere Variante dar und bieten dafür einiges an zusätzlichen Funktionen, wie beispielsweise Beschleunigung von SSL-Verbindungen und Load-Balancing. Da eine Appliance häufig für eine Vielzahl an Webservern verwendet wird, ist eine relativ komplexe Konfiguration nötig, um die große Anzahl an betriebenen Webapplikationen abzusichern. Hardware-Appliances werden immer als Reverse-Proxy für die eigentlichen Webserver verwendet. [Mar09]

Eine weitere Alternative ist eine sog. Cloud WAF. Diese wird häufig in Verbindung mit Content Delivery Network (CDN)-Diensten angeboten. Dort tritt eine Cloud als Reverse-Proxy für die eigene Webanwendung auf. Durch Caching und regionale Verteilung der Cloud-Standorte werden die Ladezeiten für die entsprechenden Seiten minimiert. Als zusätzlichen Service wird ggf. eine WAF angeboten. Ein solches Produkt wird z.B. von *Cloudflare* angeboten. Die nötigen Einstellungen lassen sich dort über ein Webinterface tätigen. Ebenso kann darüber eine Historie mit den aufgetretenen Regelverstößen eingesehen werden. [CF114]

WAF sind für einen Provider nur im Kontext des Produkts „Betreute Webanwendungen“ relevant, da für jede Webanwendung ein eigenes speziell an die individuellen Erfordernisse angepasstes Regelwerk nötig ist. Somit ist ein Betrieb einer WAF bei dem Produkt „Betreutes Hosting“ nicht praktikabel, da dort durch Kunden die verschiedensten Webanwendungen ohne Rücksprache mit dem Provider betrieben werden.

Beim Produkt „Betreute Webanwendungen“ dient eine WAF zur präventiven Absicherung von Webanwendungen gegenüber unberechtigten Anfragen (vgl. Abschnitt 3.2.1.5).

### 4.3.2 Einsatz von HTTPS

Sobald zu einer Webanwendung sensible Daten übertragen werden, sollte die Vertraulichkeit dieser Übertragung mittels Verschlüsselung sichergestellt werden. Da die meisten Webanwendungen über einen passwortgeschützten Bereich verfügen, trifft diese Forderung für einen Großteil der Nutzer von den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“ zu. Daher sollte ein Provider bei jedem dieser Produkte eine verschlüsselte Übertragung anbieten.

Technisch kann diese Forderung mit dem HTTPS-Protokoll umgesetzt werden, welches HTTP durch Transport Layer Security (TLS / vormals Secure Sockets Layer (SSL)) um Verschlüsselung und Authentifizierung erweitert. Die Authentifizierung wird dabei zumeist durch den Client vorgenommen, welcher das Serverzertifikat mittels Public-Key-Infrastruktur verifiziert. Damit der Provider nicht für jeden Kunden ein Zertifikat vorhalten muss, kann durch sog. Wildcard-Zertifikate eine generische Domain für verschlüsselte Verbindungen eingesetzt werden. Dazu erhält jeder Kunde eine individuelle Subdomain, unter welcher ein eigener vHost betrieben wird.

Bei der Konfiguration von HTTPS bei Webservern müssen verschiedene Parameter beachtet werden, da ansonsten ggf. Sicherheitslücken bei TLS ausgenutzt werden könnten. Es sollten dabei die folgenden Empfehlungen beachtet werden:

- **Versionen von SSL / TLS**

Die TLS Versionen TLS-1.1 und TLS-1.2 sollten unterstützt und aktiviert sein, da für diese beiden Versionen noch keine Sicherheitsrisiken bekannt sind. Aus Gründen der Kompatibilität sollten TLS-1.0 und SSL-3.0 ebenfalls aktiviert sein, obwohl bei diesen Versionen Sicherheitsprobleme bekannt sind. Noch ältere Version dürfen nicht aktiviert sein.

- **Auswahl der verwendeten Verschlüsselungsalgorithmen**

Um Schwachstellen bei bestimmten Algorithmen zu begegnen, sollte bei der Auswahl des zur Verschlüsselung verwendeten Algorithmus die Präferenz des Servers beachtet werden und die Reihenfolge der Algorithmen auf dem Server entsprechend konfiguriert sein.

Der Algorithmus RC4 sollte nicht mehr verwendet werden, da dieser ernsthafte Sicherheitslücken besitzt. Obwohl der Einsatz von RC4 die einzige serverseitige Maßnahme ist, um die Attacken BEAST und Lucky-13 auf Cipher-Block-Chaining (CBC) bei TLS-1.0 zu verhindern. Da diese Attacken jedoch auch durch einen gepatchten Client verhindert werden können, ist eine komplette Deaktivierung von RC4 sinnvoller. [AP13][AIF13]

Ebenso sollte von der Verwendung von Data Encryption Standard (DES) abgesehen werden, da dieser lediglich eine Schlüssellänge von 56bit verwendet.

Stattdessen sollte auf Algorithmen zurückgegriffen werden, die Perfect Forward Secrecy (PFS) unterstützen. Denn dabei wird ein Sitzungsschlüssel vereinbart, welcher später selbst bei aufgezeichneter Datenübertragung und bekannt gewordenem privatem Schlüssel nicht mehr rekonstruiert werden kann. Zur Erzeugung solcher Schlüssel wird bei TLS beispielsweise das Diffie-Hellman-Verfahren eingesetzt. [Ber12]

- **Kompression abschalten**

Über den sog. CRIME-Angriff (Compression Ratio Info-leak Made Easy) können aus der Kompressionsrate Informationen zu der verschlüsselten Übertragung gewonnen werden. Daher sollte die Kompression bei TLS vorsorglich deaktiviert werden.

- **Schutz des privaten Schlüssels**

Die verwendeten privaten Schlüssel auf Webservern sollen eine Mindestlänge von 2048bit aufweisen. Zudem ist es notwendig, dass diese Schlüssel durch entsprechende Berechtigung vor dem Auslesen durch Dritte geschützt sind.

[Ris13]

Um diese Anforderungen bei einem *Apache HTTP*-Webserver umzusetzen, sollte die Konfiguration im Listing 4.3 angewandt werden. Damit die angegebenen Algorithmen und TLS-Versionen auch unterstützt werden, ist eine ausreichend aktuelle Version von *OpenSSL* ( $\geq 1.0.0h$ ) und *Apache HTTP* ( $\geq 2.2.26$ ) nötig. [OSS14][Apa13a]

Die Konfiguration der TLS-Parameter eines Webserver kann mit dem *SSL Server Test* (<https://www.ssllabs.com/ssltest>) von *Qualys SSL Labs* anschließend getestet werden. Dort werden die Einstellungen des Webserver auf Kompatibilität und Sicherheit geprüft. Dazu wird untersucht für welche Algorithmen sich gängige Browser entscheiden würden. Die Zusammenfassung für die untenstehende Konfiguration ist in Abbildung 4.2 zu entnehmen.

## 4 Lösungsmöglichkeiten

```
# Aktiviere die Protokolle SSLv3 TLSv1 TLSv1.1 TLSv1.2
SSLProtocol ALL -SSLv2

# Berücksichtige die Reihenfolge der Algorithmen
SSLHonorCipherOrder On

# Reihenfolge verwendeter Algorithmen
SSLCipherSuite ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES
: ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS

# Deaktiviere SSL Kompression
SSLCompression off
```

Listing 4.3: Sichere Konfiguration von TLS beim *Apache HTTP*-Webserver (vgl. [Sch14])

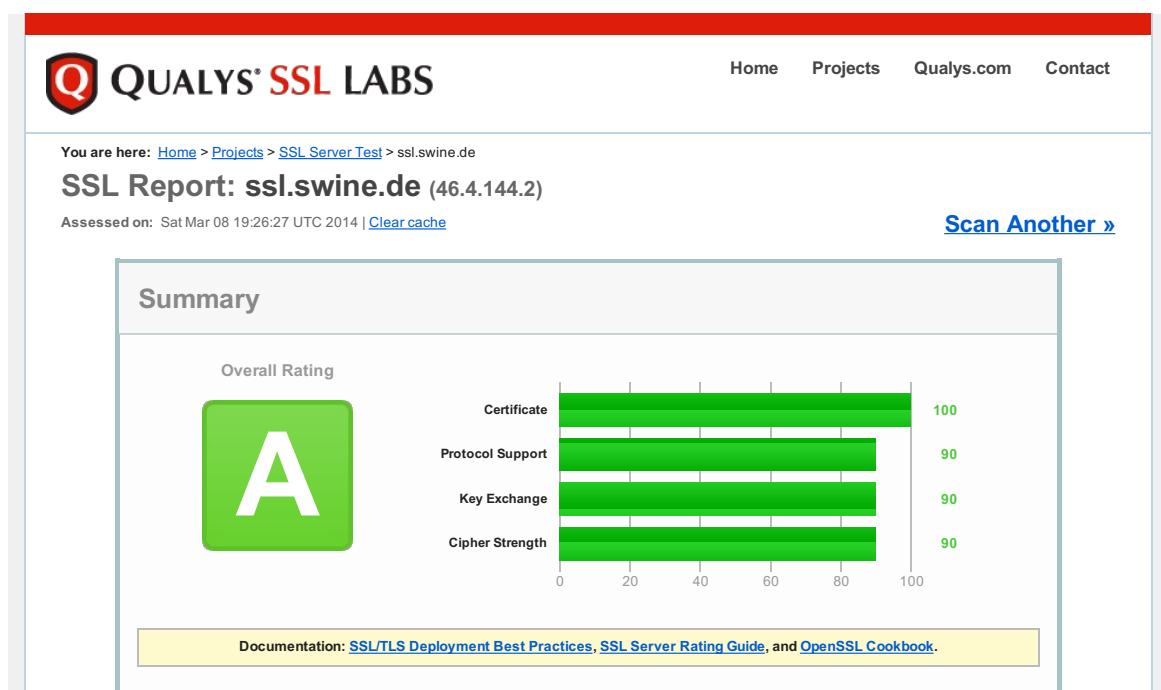


Abbildung 4.2: Zusammenfassung des Ergebnisses des *SSL Server Test* der vorgeschlagenen Konfiguration

Die Aktivierung von TLS bei der Übertragung von sensiblen Daten ist für einen Provider bei den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“ realisierbar. Dadurch wird u.a. auch die Sicherheit übertragener Passwörtern 3.2.1.1 gewährleistet.

### 4.3.3 Sicherheitsrisiken bei Webanwendungen

Damit ein Provider potentiell gefährliche Webanwendungen erkennen kann, ist es nötig, dass er die Inhalte seiner Webserver regelmäßig kontrolliert. Dabei stehen die Inhalte selbst



zumeist nicht unter der Verwaltung des Providers, sondern werden durch den Kunden betrieben. Prinzipiell kann zwischen zwei Techniken unterschieden werden, wie ein solcher Scan ausgeführt werden kann: Da der Provider Zugriff auf das Dateisystem, auf welchen die Webanwendungen abgespeichert sind, hat, kann er diese direkt dort untersuchen. Eine weitere Möglichkeit ist die Webanwendungen, wie normale Besucher, über das HTTP-Protokoll abzurufen und dort mögliche Schwachstellen auszumachen. Diese beiden Varianten werden in den nun nachfolgenden Ausführungen behandelt und entsprechende Lösungen vorgestellt.

Ein Scan auf Dateisebene hat den Vorteil, dass dieser einen wesentlich geringeren Aufwand an Ressourcen erfordert, da das langwierige Ausprobieren von möglichen Pfaden, wie bei der Durchsuchung über HTTP, durch den direkten Zugriff auf die Verzeichnisstruktur entfällt. Zudem liegen im Dateisystem die direkten Programmdateien, auf welche ein Scan über HTTP in der Regel keinen Zugriff erhält, da dieser nur die ausgelieferten (öffentlichen) Inhalte, der jeweiligen Webanwendung beurteilen kann. Ein Scan über das Dateisystem kann jedoch Schadcode, welcher an anderer Stelle, wie z.B. der Datenbank, hinterlegt ist, nicht erkennen. Außerdem kann es sein, dass eine Untersuchung des Verzeichnisbaumes veraltete Installationen mit einschließt, welche nur noch als Backup fungieren und bereits deaktiviert sind. Dort werden dann Sicherheitsrisiken entdeckt, welche in Wirklichkeit nicht gegeben sind, da die Webapplikationen dort nicht mehr ausgeführt werden.

#### 4.3.3.1 Lösungen auf Dateisebene

Eine denkbare Lösung für eine Durchsuchung des Verzeichnisbaumes sind gängige Virens Scanner. Diese versuchen anhand von Signatur-Datenbanken und bestimmter Heuristiken Schadsoftware zu erkennen. Allerdings sind diese Lösungen spezialisiert auf den Schutz von Endbenutzer-Systemen und daher nur bedingt geeignet, Schadcode zu erkennen, welcher Sicherheitslücken bei Webservern ausnutzt. Eine Untersuchung durch *R-fx Networks* kam zu dem Ergebnis, dass über 80% einer Signatur-Datenbank, welche auf Schadcode für Webserver spezialisiert ist, von den 30 Virenscannern nicht als Schadcode erkannt wurde. Bei erkanntem Schadcode wurde dieser durchschnittlich von nur 48% dieser 30 Virens Scanner erkannt. Bei dieser Untersuchung wurde auf Daten der Malware Hash Registry (MHR) von *Team Cymru* zurückgegriffen. Dieses Projekt testet Schadsoftware mit 30 gängigen Virenscannern und veröffentlicht die Resultate. [Cym14][Rfx13]

Aufgrund der schlechten Leistungen von regulären Virens Scanner hat *R-fx Networks* einen auf Hosting-Umgebungen spezialisierten Scanner *Linux Malware Detect* entwickelt und stellt diesen als Open-Source-Software zu Verfügung. Die Signatur-Datenbank dieser Software war Grundlage für die Untersuchung von regulären Virenscannern. Diese Software unterstützt neben der signaturbasierten Erkennung auch einen Algorithmus, der mittels Pattern Matching versucht, Varianten von Schadcode zu entdecken. Zusätzlich wird durch statistische Analyse versucht, verschleierte Schadcode zu entdecken. *Linux Malware Detect* kann zur Überprüfung sowohl periodisch auf dem gesamten Verzeichnisbaum ausgeführt werden, als auch die Änderungen an Dateien unterhalb eines Pfades laufend überwachen. Dafür muss das verwendete Dateisystem die `inotify`-Funktion des Linux-Kernels unterstützen. Eine Überprüfung aller Shared-Hosting Accounts der LMU beim LRZ ist in Listing 4.4 dargestellt. Sofern *Linux Malware Detect* auf die Scan-Engine *ClamScan* zurückgreifen kann, ist der erforderliche Zeitaufwand des Scans auch bei einer großen Anzahl an Dateien praktikabel. So dauerte ein Scan aller Hosting-Accounts am LRZ ca. 20 Minuten. Dabei wurde in keiner der über 4 Millionen durchsuchten Dateien eine Auffälligkeit registriert.

```

# Scan aller Hosting-Kennungen der LMU am LRZ
> maldet -a /nfs/web_lmu/www/
Linux Malware Detect v1.4.2
      (C) 2002-2013, R-fx Networks <proj@r-fx.org>
      (C) 2013, Ryan MacDonald <ryan@r-fx.org>
inotifywait (C) 2007, Rohan McGovern <rohan@mcgovern.id.au>
This program may be freely redistributed under the terms of the GNU GPL v2

{scan} signatures loaded: 11574 (9689 MD5 / 1885 HEX)
{scan} building file list for /nfs/web_lmu/www/, this might take awhile...
{scan} file list completed, found 1035870 files...
{scan} found ClamAV clamscan binary, using as scanner engine...
{scan} scan of /nfs/web_lmu/www/ (1035870 files) in progress...

{scan} scan completed on /nfs/web_lmu/www/: files 1035870, malware hits 0,
      cleaned hits 0
{scan} scan report saved, to view run: maldet --report 022614-2340.38638

# Vorgehen für /nfs/web_tum/www und /nfs/web_mwn/www wiederholen

```

Listing 4.4: Überprüfung aller Hosting-Accounts der LMU beim LRZ mit *Linux Malware Detect*

Eine interessante Funktionalität für Scannern auf Dateisystemebene wäre die Möglichkeit, installierte Webapplikationen samt Version festzustellen, da nicht nur von Schadcode, sondern auch von veralteten Webapplikationen Gefahren ausgehen. Leider konnte selbst nach umfangreicher Recherche keine bereits bestehende Lösung gefunden werden. Deshalb wurde beschlossen, im Rahmen dieser Arbeit eine entsprechende Lösung zu entwerfen.

Zuerst wurde dabei ein Ansatz favorisiert, welcher ähnlich wie die zuvor vorgestellten Scanner die gewünschten Informationen anhand einer Signatur-Datenbank ermittelt. Dabei sollte der Open-Source Virens scanner *ClamAV* als Basis verwendet werden. Diese Überlegungen wurden jedoch bald darauf wieder verworfen, da eine entsprechende Datenbank sehr viel Aufwand bei der Erstellung und laufenden Pflege verursacht hätte. Außerdem bestehen Webanwendungen in der Regel aus mehreren hundert Dateien, welche alle in die Datenbank aufgenommen worden wären. Da die Einträge je Webanwendung und jeweiliger Version angefallen wären, hätte diese enorme Menge an Datensätzen die Leistung des Scanners verringert.

Stattdessen wurde eine einfachere Strategie gewählt: Die verschiedenen Versionen einer Webapplikation gleichen sich in den meisten Fällen deutlich in ihrer Verzeichnisstruktur. Daher wurde ein Scanner entworfen, der diese Struktur analysiert und mit Profilen von bekannten Webanwendungen vergleicht. Dieser Vergleich gibt die Wahrscheinlichkeiten an, dass in dem untersuchten Verzeichnis eine bestimmte Webapplikation betrieben wird. Übersteigt diese Wahrscheinlichkeit einen bestimmten Schwellenwert, wird die Version anhand der im jeweiligen Profil hinterlegten Methode festgestellt. Meist werden dafür bestimmte Dateien mittels regulären Ausdrücken untersucht. Im Profil können weitere Funktionalitäten implementiert werden, wie z.B. eine Erkennung, welche Erweiterungen einer Webapplikation installiert sind. Die Implementierung dieser Lösung wird in Abschnitt 5.2 in der Skriptsprache *Python* unter dem Namen `webapp_discover` prototypisch durchgeführt. Dabei werden entsprechende Profile für Webapplikationen, welche häufig am LRZ eingesetzt werden, erstellt. Die Resultate eines Scanvorganges werden dort anschließend graphisch aufbereitet.

Diese beiden vorgestellten Lösungen können die Produkte „Betreutes Webhosting“ und

„Betreute Webanwendungen“ absichern. Es wird dabei die Anforderungen nach Detektion veralteter Webanwendungen erfüllt (vgl. Abschnitt 3.2.2.1). Außerdem kann mit *Linux Malware Detect* ein stattgefundener Angriff erkannt werden, sofern dadurch entsprechender Schadcode hochgeladen wurde (vgl. Abschnitt 3.2.3.3).

#### 4.3.3.2 Lösungen mit Zugriff über HTTP

Bei der Detektion von Sicherheitsrisiken bei Webanwendungen über das HTTP-Protokoll, greift der Scanner über dieselbe Schnittstelle auf die Webanwendung zu wie mögliche Angreifer. Ein solcher Scanner kann daher auch durch Angreifer benutzt werden. Im Rahmen dieser Arbeit wurden dazu die Produkte *Nikto2* und *Skipfish* betrachtet.

Es wurden mit beiden Scannern Probeläufe durchgeführt, jedoch erwiesen sich beide Programme als untauglich, um damit Webserver, die für das Produkt „Betreutes Webhosting“ verwendet werden, zu scannen. Dies liegt vor allem daran, dass die durchzuführenden Tests, sehr lange dauern und die Ergebnisse sehr viele falsch positive Ergebnisse liefern. Beide Programme richten sich eher an Entwickler dieser Webanwendungen, um diese sicherer zu gestalten. Soll eine selbst entwickelte Webanwendung als Produkt „Betreute Webanwendungen“ angeboten werden, so kann eine Untersuchung mit einem solchen Scanner sinnvoll sein. Dabei würden die Anforderungen nach Entdeckung von Schwachstellen in Webanwendungen (vgl. Abschnitt 3.2.2.1) und die Erkennung von schwachen Passwörtern (vgl. Abschnitt 3.2.2.2) ermöglicht. Ansonsten sind diese Lösungen für die Betrachtungen in dieser Arbeit nicht weiter relevant.

## 4.4 Applikationsserver

Dieser Abschnitt schlägt konkrete Lösungsmöglichkeiten für Applikationsserver auf Basis von *Apache Tomcat* vor. Dazu werden bereits zuvor betrachtete Lösungen speziell an die Umgebung des *Tomcat*-Servers angepasst.

### 4.4.1 Absicherung der Konfiguration von *Apache Tomcat*

Der Applikationsserver *Apache Tomcat* für Java-Webanwendungen kann auf den meisten Linux-Distributionen innerhalb kürzester Zeit installiert werden. Die standardmäßig verwendete Konfiguration ist dabei in den Betriebssystemen als sicher zu betrachten. Allerdings wird dazu die sog. Manager-Webanwendung deaktiviert, was die Benutzerfreundlichkeit deutlich verringert. Als Konsequenz daraus wird deshalb diese oftmals direkt nach der Installation von *Apache Tomcat* wieder aktiviert. Dabei werden häufig wesentliche Kriterien für den sicheren Betrieb dieser Anwendung missachtet.

Zum Aktivieren der Manager-Webanwendung muss in der Benutzerdatenbank `tomcat-users.xml` ein Benutzer eingetragen werden und diesem entsprechende Rollen, die zum Zugriff auf die Manager-Webanwendung berechtigen, zugewiesen werden. Die standardmäßig installierte Benutzerdatenbank hält dabei die nötigen Elemente bereit, allerdings in auskommentierter Form. Dies lädt Benutzer dazu ein, durch einfaches Entfernen der Kommentar-Bezeichner die Manager-Webapplikation funktionsfähig zu machen. Dann wird allerdings auch das standardmäßige Passwort verwendet, was einen Angriff über diese Manager-Webanwendung sehr vereinfacht.

Eine weitere Schwäche der Standard-Konfiguration ist, dass die Passwörter im Klartext in der Benutzerdatenbank hinterlegt werden. Da der *Tomcat*-Server installierte Webanwendungen mit denselben Berechtigungen ausführt, die er selbst besitzt, ist es somit für die Webanwendungen ein Leichtes, die Passwörter der Benutzerdatenbank auszulesen. Es wird daher empfohlen, dass das Passwort-Format der Benutzerdatenbank als Hash abgespeichert wird. *Tomcat* unterstützt dabei lediglich die Hashingverfahren SHA1, MD2 und MD5, welche allesamt als veraltet und angreifbar gelten. Da keine moderneren Verfahren wie SHA-2 oder SHA-3 implementiert sind, wird auf SHA-1 zurückgegriffen, da SHA-1 die größere Hashlänge besitzt. [Rei13][KM13]

Zur Festlegung des Hashingverfahren der Benutzerdatenbank auf SHA-1 muss in der Konfigurationsdatei `server.xml` beim Element `Realm` der Parameter `digest` hinzugefügt werden:

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase"
        digest="SHA"/>
```

Eine entsprechende Benutzerdatenbank `tomcat-users.xml` ist im Listing 4.5 dargestellt. Das Passwort sollte dabei den Regeln aus Abschnitt 2.6.2 genügen und kann dann mit dem Befehl `echo -n "<Passwort>" | sha1sum -` gehasht werden.

```
<?xml version="1.0" encoding="utf-8"?>
<tomcat-users>
  <role rolename="manager-gui" />
  <role rolename="manager" />
  <role rolename="admin" />
  <user username="admin"
        password="51b948072c72e746f1e9da87bf529f63fd18b639"
        roles="admin,manager,manager-gui" />
</tomcat-users>
```

Listing 4.5: Benutzerdatenbank `tomat-users.xml` von *Apache Tomcat*

Eine weitere Schwachstelle in der Standard-Konfiguration ist, dass ausschließlich HTTP angeboten wird. Um auch HTTPS zu unterstützen, muss ein entsprechendes Zertifikat und der dazugehörige Schlüssel vorhanden sein (vgl. Abschnitt 4.3.2). Im Listing 4.6 wird gezeigt, wie ein Zertifikat erstellt werden kann. Dabei wird der Befehl `keytool` aus der Java-Laufzeitumgebung verwendet, um einen privaten Schlüssel des asymmetrischen Kryptosystems RSA mit der Schlüssellänge von 4096 Bit zu erzeugen. Dieser wird in einem sog. Keystore abgelegt. Damit dieser geschützt ist, wird er mit einem Passwort verschlüsselt und die Dateiberechtigungen angepasst.

Sollte für den Anwendungszweck ein selbst signiertes Zertifikat ausreichen, ist damit die Erstellung des Zertifikates abgeschlossen. Wenn dagegen das Zertifikat von einer Zertifizierungsstelle (engl. Certificate authority (CA)) signiert werden soll, muss weiterhin ein Zertifikatsantrag (engl. Certificate Signing Request (CSR)) erstellt werden, welcher dann bei der gewünschten CA eingereicht wird. Sofern die Vorgaben der CA erfüllt sind, erhält man dann ein signiertes Zertifikat zurück. Dieses muss schließlich in den Keystore importiert werden.

Da das LRZ am Dienst *DFN-PKI* des Deutschen Forschungsnetzes (DFN) teilnimmt, hat es die Möglichkeit durch diesen Public Key Infrastruktur (PKI)-Dienst eine eigene CA zu

betreiben. Daher sollten für Systeme am LRZ Zertifikate mit der *DFN-PKI* erstellt werden. Weitere Informationen hierüber finden sich auf den entsprechenden Webseiten des LRZs (vgl. [LRZ14]).

```
# Erzeugen eines privaten Schlüssels samt Keystore
> keytool \
  -genkey \
  -alias tomcat \
  -keyalg RSA \
  -keysize 4096 \
  -keystore /etc/tomcat7/tomcat.keystore \
  -keypass <PASSWORT> \
  -storepass <PASSWORT>
What is your first and last name?
> tomcat1.lrz.de # Hier entsprechenden Hostnamen des Servers verwenden
What is the name of your organizational unit?
> Leibniz-Rechenzentrum
What is the name of your organization?
> Bayerische Akademie der Wissenschaften
What is the name of your City or Locality?
> Garching b. Muenchen
What is the name of your State or Province?
> Bayern
What is the two-letter country code for this unit?
> DE
Is CN=tomcat1.lrz.de, OU=Leibniz-Rechenzentrum,O=Bayerische Akademie der
Wissenschaften,L=Garching b. Muenchen,ST=Bayern,C=DE correct?
> yes

# Sicherung des Keystore vor fremden Zugriff
> chmod 640 tomcat.keystore
> chown root:tomcat7 tomcat.keystore

# Soll nur ein selbst-signiertes Zertifikat verwendet werden,
# so sind keine weiteren Schritte nötig.

# Erzeugung eines CSR für Einreichung bei einer CA
> keytool \
  -certreq \
  -alias tomcat \
  -keyalg RSA \
  -file /tmp/cert.csr \
  -keystore /etc/tomcat7/tomcat.keystore \
  -keypass <PASSWORT> \
  -storepass <PASSWORT>

# Die Datei /tmp/cert.csr muss bei der CA eingereicht werden.

# Anschließend erhält man das signierte Zertifikat von der CA
# und speichert es unter /tmp/cert.crt

# Import des Zertifikates
> keytool \
  -import \
  -alias tomcat \
  -trustcacerts \
```

```
-file /tmp/cert.crt \  
-keystore /etc/tomcat7/tomcat.keystore \  
-storepass <PASSWORT>
```

Listing 4.6: Erzeugung eines Zertifikates bei *Apache Tomcat*

Nun muss die HTTPS-Verbindung noch in der Konfigurationsdatei des Servers `server.xml` aktiviert werden. Da die TLS-Implementierung des *Tomcat*-Servers auf der Java Secure Socket Extension (JSSE) basiert, muss mindestens Java 7.0 eingesetzt werden, um neuere TLS-Versionen als 1.0 zu unterstützen. Die dafür notwendige Konfiguration sieht folgendermaßen aus:

```
<Server>  
  <Service>  
    <Connector  
      port="8443"  
      protocol="HTTP/1.1"  
      maxThreads="200"  
      scheme="https"  
      secure="true"  
      SSLEnabled="true"  
      clientAuth="false"  
      sslProtocol="TLSv1.2"  
      keyAlias="tomcat"  
      keystoreFile="/etc/tomcat7/tomcat.keystore"  
      keystorePass="<PASSWORT>" />  
    </Service>  
</Server>
```

Listing 4.7: Konfiguration von HTTPS bei *Apache Tomcat*

Um schließlich den Zugriff auf den Manager nur noch über HTTPS zu erlauben, müssen die folgenden Elemente in der Konfiguration `web.xml` der Manager-Webanwendung eingefügt werden:

```
<security-constraint>  
  <user-data-constraint>  
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>  
  </user-data-constraint>  
</security-constraint>
```

Durch die in diesem Abschnitt vorgeschlagenen Änderungen wird die Sicherheit der Manager-Webanwendung erheblich verbessert. Anhand dieser Lösung kann ein Provider „Betreutes Webhosting“ und „Betreute Webanwendungen“ auf Basis des Applikationsserver *Apache Tomcat* anbieten. Dabei wird die Anforderung der Vertraulichkeit des Passwortes (vgl. Abschnitt 3.2.1.1) erfüllt. Eine praktische Umsetzung dieser Vorschläge findet in Abschnitt 5.1 statt. Der nachfolgende Abschnitt beschäftigt sich mit der Absicherung von Tomcat durch ein MAC-System.

### 4.4.2 Absicherung von *Apache Tomcat* mit *AppArmor*

Dieser Abschnitt zeigt, wie ein *Apache Tomcat* durch die Verwendung von Mandatory Access Control abgesichert werden kann. Es kommt die in Abschnitt 4.2.5 bereits vorgestellte Technik

bei einer konkreten Anwendung zum Einsatz. Für die Betrachtungen in diesem Abschnitt wurde die Linux-Distribution *Debian Wheezy* verwendet, da dieses Betriebssystem auch später bei der prototypischen Implementierung u.a. dieser Lösung zum Einsatz kommt. Die Gründe, die zu dieser Entscheidung geführt haben, werden in Kapitel 5 dargestellt.

Da *Debian Wheezy* keine vorgefertigten Richtlinien für den Applikationsserver *Tomcat* mitliefert, wird im Laufe dieses Abschnittes die Entwicklung eines geeigneten Profils aufgezeigt. Dazu wird zuerst sichergestellt, dass die Pakete von *AppArmor* und *Apache Tomcat* ordnungsgemäß installiert wurden. Anschließend muss ein ggf. ausgeführter *Tomcat*-Server beendet werden. Dann kann mittels des Hilfsprogramms **aa-gen** ein neues Profil für den Applikationsserver erstellt werden. Dazu führt man dieses Programm aus und übergibt als ersten Parameter den Pfad der Anwendung, für die ein Profil erstellt werden soll. Ab dann beobachtet *AppArmor* alle von der Anwendung (Subjekt) angeforderten Dateien und Verzeichnisse (Objekte). Man startet nun den *Tomcat*-Server wie gewohnt und führt für den Betrieb des Applikationsserver typische Operationen aus wie z.B.: Aufrufen von Webapplikationen, Bereitstellen einer neuen Webapplikation, Anmeldung an einer Webapplikation usw.. Anschließend kann **aa-gen** beendet werden. Dabei schreibt es aus den angeforderten Berechtigungen ein entsprechendes Profil, welches alle aufgetretenen Zugriffe auf Objekte erlaubt.

Das dabei erzeugte Profil weist nun erheblich mehr Regeln auf als eigentlich notwendig wären, da **aa-gen** den Zugriff auf jedes einzelne Objekt in dem Profil mit eigenen Regeln versieht, welche dessen absoluten Pfad enthalten. Da jedoch für Dateien, die sich unterhalb eines bestimmten Pfades befinden, dieselben Regeln gelten sollen, kann nun eine Vielzahl an Regeln reduziert werden. So würde das automatisiert erstellte Profil die folgenden Regeln enthalten, welche besser durch die einzelne Regel in der letzten Zeile ersetzt werden können:

```
/var/log/tomcat7/catalina.out rw,
/var/log/tomcat7/localhost.2014-02-16.log rw,
/var/log/tomcat7/localhost_access_log.2014-02-16.txt rw,
/var/log/tomcat7/catalina.2014-02-16.log rw,
# Diese Regel ersetzt die vier vorhergehenden
/var/log/tomcat7/** rw,
```

Dabei handelt es sich scheinbar um Regeln, welche den Lese- und Schreib-Zugriff auf die Logdateien des *Tomcat*-Servers erlauben. Durch dieses Zusammenfassen von Pfaden, für die dieselben Regeln gelten, kann die Länge des Profils deutlich verkürzt werden. Ein Blick in die Dokumentation des *Tomcat*-Server verrät dabei die Funktionen der entsprechenden Pfade.

Eine weitere Möglichkeit, die Länge des Profils zu verkürzen, stellen sog. Includes dar. Diese erlauben, die Regeln anderer Profile ins das jeweilige Profil mit aufzunehmen. Dabei werden bei *AppArmor* standardmäßig für bestimmte Anwendungsfälle Profile mitgeliefert. Diese werden auch als Abstractions bezeichnet. Für den *Tomcat*-Server wurde die folgende Aufteilung in Teilprofile durchgeführt, um die Regeln besser gruppieren zu können:

- Hauptprofil `usr.share.tomcat7.bin.catalina.sh`

Das Hauptprofil fasst alle Teilprofile zusammen. Zusätzlich werden dort Bereiche einer Anwendung definiert (sog. Hats). Für jeden Bereich können separate Richtlinien festgelegt werden. In die entsprechenden Bereiche kann die Anwendung, sofern sie eine Unterstützung dafür hat, wechseln. Beim *Tomcat*-Server wird immer dann in einen solchen Bereich gewechselt, wenn der Programmcode einer Webapplikation ausgeführt wird. Es kann dabei für jede Webanwendung des Applikationsservers ein entsprechender

Bereich definiert werden. Das im Zuge dieses Abschnittes erstellte Hauptprofil (vgl. Listing 4.8) unterstützt lediglich den Bereich `DEFAULT`, welcher für alle Applikationen gilt.

Im Bereich `DEFAULT` werden die Regeln der Abstractions `<abstractions/tomcat>` und `<abstractions/tomcat-webapp>` angewandt. Die Regeln des Standard-Bereiches, welche für den *Tomcat*-Server selbst gelten, erhalten die beiden Abstractions und zusätzlich die standardmäßig mitgelieferte Abstraction `<abstractions/bash>` und einzelne weitere Regeln, die nötig sind, damit der *Tomcat*-Server die zum Betrieb notwendigen Berechtigungen erhält. Darin enthalten sind beispielsweise auch die Berechtigungen, um die Konfigurationsdateien des *Tomcat*-Servers zu lesen oder zu bearbeiten. Somit kann einer Webapplikation der Zugriff auf diese Konfigurationsdateien verweigert werden, obwohl der Applikation dieser nach der DAC von Linux zustehen würde, da die Webapplikation mit demselben Benutzer wie der *Tomcat*-Server ausgeführt wird.

Da beim Startvorgang des Servers weitere Befehle ausgeführt werden, muss für diese ebenfalls eine entsprechende Berechtigung im Profil existieren. Dabei werden die Rechte `rx` vergeben: Während `r` den Lesezugriff und `x` die Ausführung erlaubt, erreicht die Berechtigung `i`, dass für den jeweiligen ausgeführten Befehl dasselbe Profil gilt, wie für die ursprüngliche Anwendung. Dadurch wird erreicht, dass ein aufgerufener Befehl keine unvorhergesehene Erweiterung der Rechte nach sich zieht.

- Teilprofil `<abstractions/tomcat>`  
Dieses Teilprofil umfasst diejenigen Regeln, welche sowohl für den *Tomcat*-Server als auch für dessen Applikationen gelten. Dort wird u.a. der Zugriff auf Logdateien, Anwendungsdaten der Webapplikationen und temporäre Dateien und Verzeichnisse erlaubt. Zusätzlich muss ein Lesezugriff auf Dateien, die zur Java-Laufzeitumgebung gehören, möglich sein. Die entsprechenden Regeln werden durch das Teilprofil `<abstractions/java>` abgebildet.
- Teilprofil `<abstractions/java>`  
In diesem Teilprofil sind die Regeln zusammengefasst, die nötig sind, um Java-Applikationen auszuführen. Da diese auch für weitere Anwendungen wiederverwendet werden, die eine Java-Laufzeitumgebung verwenden, wurde ein eigenes Teilprofil dafür erstellt. Die Berechtigungen erlauben den Zugriff auf Bibliotheken und weitere Dateien, die für die Ausführung der Java-Laufzeitumgebung nötig sind. Zusätzlich werden die standardmäßigen Abstractions `<abstractions/base>` und `<abstractions/namespace>` inkludiert, welche den Zugriff auf essentielle Funktionen wie z.B. dynamisch geladene Bibliotheken und Namensauflösung durch DNS erlauben.
- Teilprofil `<abstractions/tomcat-webapp>`  
Dieses Teilprofil ist standardmäßig leer. Sollte eine Applikation des *Tomcat*-Servers besondere Berechtigungen außerhalb der gängigen Pfade benötigen, können diese dort eingetragen werden. Sie gelten dann sowohl für die Applikationen als auch den *Tomcat*-Server selbst.

```
/usr/share/tomcat7/bin/catalina.sh {
# Includes
```



```

#include <abstractions/bash>
#include <abstractions/tomcat>
#include <abstractions/tomcat-webapp>

# Zugriff auf Terminalemulation
/dev/tty rw,

# Tomcat Konfiguration lesen/bearbeiten
/etc/tomcat7/** rw,

# Hilfsanwendungen für Startskript
/bin/dash rxi,
/bin/touch rxi,
/bin/rm rxi,
/bin/uname rxi,
/usr/bin/tty rxi,
/usr/bin/dirname rxi,

# PID file
/run/tomcat7.pid rw,

# Schreibzugriff auf Webanwendungen
/var/lib/tomcat7/webapps/** rw,

# Webapp special paths

## Default HAT
^DEFAULT {

    # Includes
    #include <abstractions/tomcat>
    #include <abstractions/tomcat-webapp>

}
}

```

Listing 4.8: *AppArmor* Hauptprofil für *Apache Tomcat*

Diese Profile befinden sich im Pfad `/etc/apparmor.d` und werden von nun an beim Systemstart geladen. Um das eben erstellte Profil zu testen kann es mit dem Befehl `aa-complain` im sog. „Complain“-Modus aktiviert werden. Dabei werden die Regeln nicht durchgesetzt, jedoch erzeugen Verletzungen dieser eine Warnmeldung in der Logdatei `audit.log`. Sofern die Regeln vollständig sind, dürften keinerlei solcher Meldungen erzeugt werden. Dann kann das Profil mittels des Kommandos `aa-enforce` in den „Enforce“-Modus versetzt werden, welcher die Regeln zur Anwendung bringt und somit nicht explizit erlaubte Zugriffe unterbindet.

Sollte beim Betrieb von Applikationen durch den *Tomcat*-Server ein unerwartetes Problem auftreten, kann dies an einem zu restriktiven *AppArmor*-Profil liegen. Man sollte dann zuerst das `audit.log` auf dementsprechende Meldungen untersuchen. Sollte das Problem auch im „Complain“-Modus weiterhin bestehen, ist es nicht durch *AppArmor* verursacht, da dieser Modus von den Auswirkungen auf die Anwendung einer kompletten Deaktivierung des Profils gleicht.

Um für den *Tomcat*-Server andere Regeln als für seine Applikationen gelten zu lassen, muss die oben beschriebene Unterteilung des Profils in mehrere Unterbereiche (Hats) durchgeführt werden. Zudem muss der jeweilige *Tomcat*-Server über ein entsprechendes Modul verfügen, das

die Bereichswechsel (sog. Change Hat) durchführt. Bei der Linux-Distribution *Debian Wheezy* wird dieses Modul nicht als Paket angeboten, daher musste es aus den Quellcode-Archiv von *AppArmor* kompiliert werden. Anschließend wurde durch das folgende Element in der Server-Konfiguration `server.xml` das Modul beim Starten von *Tomcat* aktiviert:

```
<Server>
  <Service>
    <Engine>
      <Valve className="com.novell.apparmor.catalina.valves.ChangeHatValve"
            mediationType="ServletPath" />
    </Engine>
  </Service>
</Server>
```

Das entwickelte *AppArmor*-Profil kommt im Abschnitt 5.1.4 bei der automatisierten Konfiguration eines *Tomcat*-Server mittels *Puppet* zur Anwendung. Dieses Profil kann bei Providern die Produkte „Betreutes Webhosting“ und „Betreute Webanwendungen“ absichern. Es erfüllt dabei die Anforderungen nach minimaler Rechtezuteilung an Webanwendungen (vgl. Abschnitt 3.2.1.6). und dient der besseren Isolation von einzelnen Mandanten bei gemeinsamer Nutzung eines *Tomcat*-Servers (vgl. Abschnitt 3.2.4.1).

### 4.5 Infrastruktur

Innerhalb dieses Abschnittes werden Maßnahmen vorgestellt, welche sich auf die gesamte Infrastruktur eines Providers auswirken. Durch die Einführung einer dieser Lösungen sind mehrere Systeme betroffen.

#### 4.5.1 Konfigurationsmanagement mit *Puppet*

Auf Konfigurationsmanagement von Betriebssystemen aus dem Unix-Bereich sind im wesentlichen drei Anwendungen spezialisiert: *Chef*, *Puppet* und *Bcfg2*. Alle Lösungen unterstützen dabei den Betrieb als Client-Server-System. Der Server generiert dabei die entsprechenden Konfigurationen für die Clients durch Auswertung der auf ihm abgelegten Vorlagen. Ein Client setzt dann die erhaltene Konfiguration auf dem eigenen System um. Anschließend erstattet er dem Server Bericht darüber, ob die gewünschten Änderungen erfolgreich verlaufen sind. Die Vorlagen für die Konfigurationen werden dabei in einer Sprache verfasst, welche an die jeweilige Lösung angepasst ist. Daher sind die erstellten Vorlagen nicht untereinander kompatibel. Für die weitere Betrachtung wird das Augenmerk auf *Puppet* gelegt. Viele der Betrachtungen können auch auf *Chef* und *Bcfg2* angewandt werden.

*Puppet* ist in der Skriptsprache *Ruby* implementiert und nutzt daher zur Spezifizierung der Vorlagen eine von *Ruby* abgeleitete Domain-Specific Language (DSL). Diese Vorlagen werden auch als Manifeste bezeichnet. Zur besseren Abstrahierung werden mehrere solcher Manifeste als Modul zusammengefasst. Ein Modul ist häufig auf eine bestimmte Anwendung limitiert, welche konfiguriert werden soll. Zur Spezifizierung der Vorlage werden in den Manifesten sog. Ressourcen instanziiert. Dabei erhält jede Instanz einen pro Ressource eindeutigen Identifier und es können zusätzlich optionale Parameter übergeben werden. Eine Ressource löst auf dem Zielsystem durch den Client eine bestimmte Aktion aus. Nachfolgend wird eine Auswahl von häufig verwendeten Ressourcen vorgestellt:

- **File**

Die **File**-Ressource repräsentiert entweder eine Datei, ein Verzeichnis oder einen Dateisystem-Link. Der Identifier der Ressource gibt den Pfad zum jeweiligen Dateisystem-Objekt an. Über verschiedene Parameter kann der Zustand der Ressource näher festgelegt werden. Der Parameter **ensure** bestimmt dabei den Typ des Objekts (**file**: Datei, **directory**: Verzeichnis und **link**: Link) bzw. kann durch Angabe von **absent** sichergestellt werden, dass sich am angegebenen Pfad kein Objekt befindet.

Als weitere Parameter können mittels **owner**, **group** und **mode** die Berechtigungen des entsprechenden Objekts verändert werden. Der Inhalt einer Datei kann mittels dem Parameter **content** festgelegt werden. Der Inhalt wird zumeist direkt als Zeichenkette angegeben. Alternativ kann auch ein Funktionsaufruf die gewünschten Inhalte der Datei generieren. Häufig wird dabei die **template()**-Funktion verwendet, welche Templates der in *Ruby* integrierten Sprache **ERB** verarbeiten kann. Alternativ dazu kann auch eine statische Datei über den Parameter **source** als Quelle des Inhaltes angegeben werden.

- **Exec**

Mit der **Exec**-Ressource können beliebige Befehle ausgeführt werden. Im Identifier dieser Ressource wird das auszuführende Kommando angegeben. Dabei muss beachtet werden, dass entweder der Befehl mit vollständiger Pfadangabe verwendet wird oder die entsprechenden Such-Pfade über den Parameter **path** angegeben werden. Da *Puppet* diesen Befehl bei jeder Anwendung der Konfiguration ausführt, muss dieser idempotent sein. Es muss also auch bei mehrmaliger Ausführung immer der Zustand nach dem ersten Aufruf erhalten bleiben. Sollte das nicht möglich sein, kann mit dem Parameter **onlyif** ein Befehl angegeben werden, welcher prüft, ob eine Ausführung nötig ist.

- **Package**

Über die **Package**-Ressource können Paket-Verwaltungen unter Linux wie z.B. *Apt* und *Yum* angesprochen werden. Als Identifier wird dabei der Name eines entsprechenden Paketes erwartet. Der Parameter **ensure** definiert dabei den gewünschten Zustand des Pakets. Entweder wird dort die gewünschte Version des Pakets angegeben oder eines dieser Stichworte:

- **latest** installiert ggf. das Paket in der neusten Version oder führt, sofern es bereits in einer älteren Version installiert ist, ein Update durch.
- **present** installiert das Paket in der neusten Version, falls es nicht vorhanden ist.
- **absent** deinstalliert ein Paket.

- **Service**

Die **Service**-Ressource repräsentiert einen konfigurierten Dienst eines Systems. Der Identifier gibt dabei den Namen des Dienstes an. Der gewünschte Zustand kann hier wiederum mit **ensure** angegeben werden. Wird dort der Wert **running** gesetzt, wird bei Anwendung der Konfiguration überprüft, ob der entsprechende Dienst auch aktuell läuft. Ebenso kann mit dem Wert **stopped** sichergestellt werden, dass der Dienst gestoppt ist. Über den Parameter **enable** wird eingestellt, ob der Dienst beim Systemstart automatisch gestartet werden soll.

- **Augeas**

Durch die Integration der Anwendung *Augeas* in *Puppet* stellt es die Funktionalität dieses Hilfsprogramms über die gleich bezeichnete Ressource **Augeas** zur Verfügung. *Augeas* kann mit entsprechenden Modulen eine große Anzahl der unter Linux verwendeten Konfigurationsdateien derart parsen, dass diese bequem in einem XML-Baum bearbeitet werden können. Damit können einzelne Parameter in diesen Konfigurationsdateien bearbeitet werden, ohne dass die komplette Datei über eine **File**-Ressource gepflegt werden muss.

Der Identifier dieser Ressource dient lediglich zur eindeutigen Bezeichnung dieser. Die durchzuführenden Veränderungen am XML-Baum werden dann mittels des Parameters **changes** angegeben. Diese Operationen werden im XPath-Format angegeben. [Aug14]

[Pup14c]

Oftmals bestehen Abhängigkeiten zwischen den definierten Ressourcen. Damit diese berücksichtigt werden, gibt es Parameter, die für alle Ressourcen verwendet werden können. Mit den Parameter **require** bzw. **before** werden benötigte Ressourcen bzw. nachfolgende Ressourcen übergeben. Eine Erweiterung dieser Abhängigkeit durch sog. Notifications ist mit den Parametern **subscribe** und **notify** möglich. Sie definieren implizit eine Abhängigkeit wie bei **require** und **before**, jedoch werden zusätzlich bestimmte Ressourcen erneuert, falls durch Ressourcen, zu denen eine Abhängigkeit besteht, eine Änderung vorgenommen wurde. Unter den zuvor vorgestellten Ressourcen-Typen führen **Exec** und **Service** eine besondere Aktion aus, sobald sie eine solche Notification erhalten. Die **Exec**-Ressource führt ihren zugehörigen Befehl nur aus, wenn sie eine Notification empfängt. Die Ressource **Service** startet dagegen den durch sie verwalteten Dienst neu. Dies dient dazu, um eine ggf. veränderte Konfiguration einzulesen. [Pup14b]

Um das Zusammenwirken der besprochenen Ressourcen zu verdeutlichen, wird im Listing 4.9 ein SSH-Server mit *Puppet* installiert und konfiguriert. Zuerst wird dazu die **Package**-Ressource ausgeführt, da diese keinerlei Abhängigkeiten besitzt. Diese installiert das Paket **openssh-server** über die Paket-Verwaltung. Anschließend ist die über **require** definierte Abhängigkeit der Ressourcen **File** und **Augeas** erfüllt. Die **File**-Ressource kopiert nun das Konfigurationsfile **known\_hosts** vom Puppet-Server in den angegeben Pfad und stellt die entsprechenden Dateiberechtigungen sicher. Diese Datei enthält vertrauenswürdige öffentliche Schlüssel anderer SSH-Server. Die **Augeas**-Ressource dient dazu, die Konfiguration des SSH-Servers so abzuändern, dass der **root**-Benutzer sich nicht mehr über ein Passwort einloggen kann. Erst nachdem diese Ressource angewandt worden ist, kann die **Service**-Ressource abgearbeitet werden, da sie über den Parameter **subscribe** von ihr abhängig ist. Sie startet den SSH-Server, sofern dieser noch läuft, und richtet den automatischen Start des Dienstes während des Bootvorganges ein. Wenn der Server bereits läuft und die mittels **subscribe** beobachtete Ressource **Augeas** eine Änderung vornehmen musste, wird der SSH-Dienst neugestartet, um die neue Konfiguration zu verwenden.

```
# Paket wird benötigt
package { 'openssh-server':
  ensure => present,
}

# Erlaube root-Login nur mit Schlüssel
```

```

augeas { 'ssh_config_permit_root_keyonly':
  require => Package['openssh-server'],
  context => '/files/etc/ssh/sshd_config/',
  changes => 'set PermitRootLogin without-password',
}

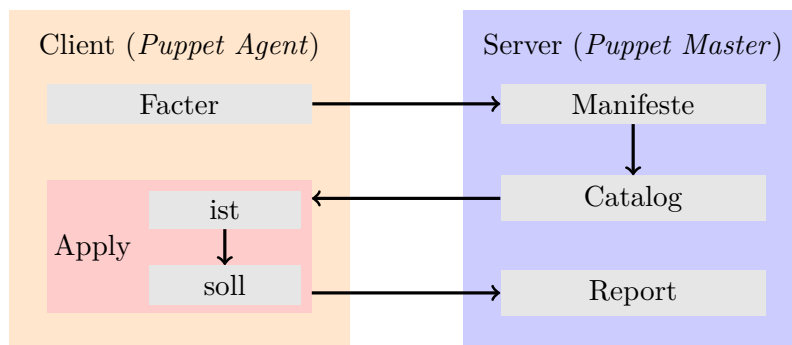
# Installiere bekannte Hostschlüssel
file { '/etc/ssh/known_hosts':
  ensure => file,
  mode   => 644,
  require => Package['openssh-server'],
  source => 'puppet://sshd/known_hosts',
}

# Aktiviere den SSH-Dienst
service { 'ssh':
  ensure => running,
  enable => true,
  subscribe => Augeas['ssh_config_permit_root_keyonly'],
}

```

Listing 4.9: Konfiguration eines SSH-Servers mit Puppet

Der Ablauf von *Puppet* im Client-Server-Modus ist in Abbildung 4.3 dargestellt. Dabei ist der erste Schritt, dass sich der Client, welcher auch als *Puppet Agent* bezeichnet wird, zum Server, dem *Puppet Master*, verbindet. Hier teilt der Client dem Server sog. Facts mit, welche weitere Informationen zum System des Clients liefern. Der *Puppet Master* hat zudem die Möglichkeit, in dieser Phase weitere Facts anzufordern, welche durch Module auf dem *Puppet Master* definiert werden. Diese Verbindung läuft in der Regel verschlüsselt über das HTTPS-Protokoll ab. Zur Authentifizierung von Clients betreibt der Server eine eigene CA und bei jeder aufgenommenen Verbindung eines Agents wird dessen Identität durch sein entsprechendes Zertifikat verifiziert.

Abbildung 4.3: *Puppet* im Client-Server-Modus (vgl. [Pup14a])

Nun kompiliert der *Puppet Master* aus den Manifesten und den vom Client übermittelten Facts den sog. Catalog, der die gewünschte Konfiguration des Clients enthält. Dieser Catalog wird nun zum Agent übermittelt und dieser vergleicht im Schritt Apply den aktuellen Zustand

seines Systems mit dem gewünschten Zustand aus dem Catalog. Bei Diskrepanzen zwischen den Zuständen führt der *Puppet Agent* die entsprechenden Änderungen am System durch. Anschließend sendet der Client die Ergebnisse des Durchlaufes zur Auswertung an den Server. Der *Puppet Agent* erhält dabei lediglich Einblick in die eigene Konfiguration, da der Catalog ausschließlich seine eigene Konfiguration enthält.

*Puppet* kann auch im Modus Stand-Alone ausgeführt werden. Dann wird durch die lokale Installation aus den Manifesten der Catalog generiert und schließlich auf dem lokalen System angewandt. Diese Funktionalität ist beim Testen von Manifesten sehr hilfreich.

Provider können *Puppet* für alle Produkte einsetzen. Insbesondere kann dadurch eine regelmäßige Wartung für virtuelle Maschinen angeboten werden. Damit können durch *Puppet* diese Anforderungen zur Prävention erfüllt werden: Die automatisierte Konfiguration und Aktualisierung von Webanwendungen (vgl. Abschnitt 3.2.1.3). Über *Puppet* können zudem regelmäßige Wartungen (vgl. Abschnitt 3.2.1.2) durchgeführt werden. Ebenso ist es möglich minimale Berechtigung für Webanwendungen (vgl. Abschnitt 3.2.1.6) anzuwenden und laufend zu kontrollieren.

Ein Beispiel, wie eine solche Vorlage für die sichere Konfiguration des *Apache Tomcat*-Applikationsserver erstellt werden kann, ist in Abschnitt 5.1 beschrieben.

### 4.5.2 Firewall

Der Einsatz von Firewalls sollte bei einem Provider obligatorisch sein. Er unterbindet damit unerwünschten ein- und ausgehenden Datenverkehr auf seinen Systemen. Wird eine Firewall z.B. auf einem Router implementiert, kann dadurch ein ganzer Netzbereich mit einer Vielzahl an Systemen abgesichert werden. Zudem ist auch der Einsatz auf einem Endsystem möglich. Die beste Absicherung wird durch eine Kombination der beiden Möglichkeiten erreicht. Eine Firewall zieht zur Entscheidung, ob ein Paket erwünscht ist, die folgenden Kriterien heran: Quell- und Zieladresse, Quell- und Zielport, Protokolltyp und Zustand der Verbindung.

Bei dem Produkt „Virtuelle Maschinen“ ist die Konfiguration der Firewall auf dem Endsystem im Zuständigkeitsbereich des Kunden. Obwohl der Provider für den entsprechenden Netzbereich, welcher diese VMs beinhaltet, eine Firewall implementieren könnte, sollte er dort keine Einschränkungen vornehmen, damit die Kunden das Produkt vorbehaltlos nutzen können.

Die Systeme, welche für die Produkte „Betreutes Hosting“ und „Betreute Webanwendungen“ betrieben werden, können dagegen durch eine Firewall umfassend geschützt werden, indem nur erwünschte Kommunikation erlaubt wird. Bei der Entwicklung der Firewall-Regeln sollten die folgenden Aspekte berücksichtigt werden:

- **Eingehender Datenverkehr**

Jeder Dienst eines Systems der von externen Netzen aus erreicht werden soll, sollte explizit in den Firewallregeln des Endsystems freigegeben werden. Das verhindert, dass Dienste, welche nur für das lokale System betrieben werden, von Dritten genutzt werden können.

- **Ausgehender Datenverkehr**

Auch der vom Web- und Applikationsserver ausgehende Datenverkehr sollte durch die Firewall reglementiert werden. Webanwendungen sollten sich zu Diensten außerhalb des Providernetzes uneingeschränkt verbinden können. Einzige Ausnahme sollte für den SMTP-Dienst bestehen, da über diesen von angegriffenen Webanwendungen oftmals

Spam verschickt wird. Daher sollten Webanwendungen die Verbindung zu externen SMTP-Diensten verweigert werden.

Innerhalb des Providernetzes sollte jeder erlaubte Dienst explizit freigegeben werden, da diese internen Verbindungen von anderen Diensten ggf. als privilegiert betrachtet werden. Erlaubt werden sollten durch die Webanwendungen benötigte Dienste wie z.B. die Verbindung zu Datenbankservern.

- **Deaktivierung von UDP**

Ein Web- bzw. Applikationsserver muss nicht zwingend über das Transportprotokoll UDP erreichbar sein, da er Dienste wie HTTP(S) über TCP abwickelt. Daher sollte für entsprechende Server der komplette eingehende UDP-Datenverkehr verworfen werden, da damit die Möglichkeit von (D)DoS-Attacken verringert wird. Denn die sog. Amplification Attacks funktionieren nur mit UDP (vgl. Abschnitt 2.6.1) Die UDP-Pakete sollten, sobald diese das Providernetz erreichen, verworfen werden, so dass die Auswirkungen auf das Netz der Providers möglichst gering sind.

Damit eine Namensauflösung über DNS und Zeitsynchronisation über NTP, welche beide UDP einsetzen, weiterhin auf den Web- und Applikationsservern funktioniert, sollten dafür eigene Dienste im Providernetz verwendet werden. Diese können nach wie vor über UDP mit den Web- und Applikationsservern kommunizieren.

Durch geeignete Firewallregeln kann der Datenverkehr bei den Produkten „Betreutes Hosting“ und „Betreute Webanwendungen“ entsprechend der Anforderung aus Abschnitt 3.2.1.5 beschränkt werden.

### 4.5.3 Intrusion-Detection-System (IDS)

Ein netzbasiertes Intrusion-Detection-System (IDS) beobachtet den Datenverkehr, um auffällige Aktivitäten, die auf einen Angriff oder eine Infektion schließen lassen, zu entdecken. Dazu wird der Datenverkehr an zentraler Stelle wie z.B. dem Internet-Uplink ausgewertet. Sollte die Leistung des IDS nicht ausreichen, um den kompletten Datenverkehr zu beobachten, kann es vorkommen, dass ein Teil des Datenverkehrs nicht analysiert wird.

Zur Erkennung von Angriffen dienen bestimmte Muster, die entweder anhand von Signaturen oder durch statistische Methoden erkannt werden. Erkennt ein IDS einen Angriff, so erzeugt es eine Alarmmeldung und benachrichtigt die Verantwortlichen über den Angriff. Ein IDS löst also keine aktiven Gegenmaßnahmen aus. Aktive Gegenmaßnahmen werden durch sog. Intrusion-Prevention-Systeme (IPS) realisiert, welche zusätzlich die Möglichkeit haben, den Datenverkehr zu beeinflussen. Dazu interagiert das IPS mit einer Firewall und kann nach Identifizierung eines Angriffes den dazugehörigen Datenverkehr sperren. Auch andere Aktionen, die den Angriff unterbinden, können ggf. durch IPS ausgelöst werden. Wenn ein System des Providers infiziert wurde und daher Angriffe durchführt, könnte dieses beispielsweise automatisch deaktiviert werden. [Sou14a]

Am LRZ wird als IDS-System die Open-Source-Lösung *Suricata* eingesetzt, welche am dortigen Internet-Uplink den Datenverkehr untersucht. Damit Angriffe auf Web- und Applikationsserver erkannt werden können, müssen für deren Netzbereiche entsprechende Regeln aktiviert werden, welche die zugehörigen Angriffsmuster definieren. Derartige Regeln werden laufend durch das *Sourcefire Vulnerability Research Team (VRT)* herausgegeben.

Während der sofortige Zugriff auf das aktuelle Regelwerk kostenpflichtig ist, kann das entsprechende Regelwerk mit 30-tägiger Verzögerung kostenlos genutzt werden. Die für Web- und Applikationsserver nützlichen Regeln sind innerhalb dieser Sammlung in den Dateien `server-webapp.rules` und `malware-backdoor.rules` zusammengefasst. [OISF14] [Sou14b]

Ein IDS kann für alle Produkte eingesetzt werden, da es nicht aktiv in den Datenverkehr eingreift. Es dient dazu Angriffe zu erkennen und darüber zu benachrichtigen (vgl. Abschnitt 3.2.3.2). Automatische Gegenmaßnahmen werden dann durch ein IPS bei Erkennung eines Angriffes durchgeführt. Dadurch kann dynamisch auf einem Angriff reagiert werden und entsprechender Datenverkehr unterbunden werden (vgl. Abschnitt 3.2.1.5). Wurde dabei eine Kunden-Instanz infiziert, kann diese entsprechend der Anforderung aus Abschnitt 3.2.4.3 automatisiert deaktiviert werden. Es sollte dabei sichergestellt sein, dass der gewöhnliche Datenverkehr der jeweiligen Produkte nicht als Angriff bewertet wird.

### 4.5.4 Zentrale Logauswertung

Eine wichtige Informationsquelle bei der Suche nach Hinweise zu Angriffen und Infektionen auf Web- und Applikationsserver sind die Logdateien der benutzten Infrastruktur. Dabei ist eine Betrachtung dieser Quellen interessant:

- Das **Access-Log** eines Web- bzw. Applikationsservers enthält u.a. Informationen darüber, welche IP-Adressen zu welcher Zeit welche Webseiten aufgerufen haben. Falls der Datenverkehr für bestimmte Webseiten signifikant zunimmt, kann das ein Hinweis auf eine Infektion sein.

Auch das **Error-Log** dieser Server sollte erfasst werden, denn dort sind Zugriffe vermerkt, welche auf dem Server einen Fehler verursacht haben. Eine Häufung von solchen Fehlern kann ein Anzeichen dafür sein, dass der Server gerade auf Schwachstellen untersucht wird.

Bei den meisten Web- und Applikationsservern sind diese Logdateien ähnlich aufgebaut und enthalten annähernd dieselben Informationen. Daher lassen sie sich gut automatisiert auswerten.

- Die Protokolle der **Webapplikationen** selbst können Hinweise zu Infektionen und Angriffen enthalten. Dort finden sich beispielsweise Informationen über Anmeldeversuche an der Webanwendung oder durch Benutzer ausgeführte Operationen. Da sich die Webanwendungen beim Format der Logdateien oftmals erheblich unterscheiden, ist eine generelle Analyse der Logdateien schwierig zu implementieren. Sofern eine große Anzahl der betriebenen Webanwendungen vom gleichen Typ sind, sollte der Einsatz eines darauf angepassten Parsers erwogen werden.
- Bei Einschränkung bzw. Kontrolle der Netzanbindung durch **Firewalls, Intrusion-Detection- und Intrusion-Prevention-Systemen** (vgl. Abschnitte 4.5.2 und 4.5.3) sollten entsprechende Anomalien protokolliert werden. Anhand der Protokollmeldungen können dann diese Netzaktivitäten analysiert werden, ob dort Hinweise auf Angriffe bzw. Infektionen zu finden sind.
- Für den Fall das eine **Web Application Firewall** eingesetzt wird, können deren Logdateien wertvolle Hinweise zur Erkennung von infizierten bzw. angegriffenen Webappli-



kationen enthalten. Daher sollte auch diese Quelle bei der Auswertung herangezogen werden.

- Als weitere Datenquelle sollte die **Auslastung der Ressourcen** von den eingesetzten Systemen betrachtet werden. Anhand des zeitlichen Verlaufs dieser Daten lassen sich damit Angriffe und Infektionen erkennen und einzelnen Systemen zuordnen. Dabei sollte u.a. die Auslastung der Ressourcen Prozessor, Arbeitsspeicher, Hintergrundspeicher und Netzbandbreite beobachtet werden.
- Die Logdateien weiterer Dienste im Umfeld der Web- und Applikationsservern sollten ebenfalls in die Auswertung einbezogen werden. So können beispielsweise Änderungen an den Webanwendungen über die Protokolldateien des FTP-Dienstes nachvollzogen werden.
- Die Programmdateien von Webanwendungen könnten bei Linux mit dem `Inotify`-Mechanismus laufend auf Veränderungen überwacht werden. Diese Änderung werden protokolliert und können anschließend ebenfalls als Quelle für ein zentrales Logsystem dienen.

Die Informationen der vorgestellten Datenquellen liefern jedoch nur Indizien, welche auf einen Angriff bzw. eine Infektion hinweisen könnten. Damit diese Indizien aus den verschiedenen Quellen zu einem Gesamtbild zusammengefügt werden können, müssen diese an einer zentralen Stelle zur Verfügung stehen und dort ausgewertet und miteinander verknüpft werden.

Lösungen zur Bewältigung dieser Aufgabe folgenden diesem Aufbau: Ein Parser liest die einzelnen Logdateien auf den verschiedenen Systemen und bringt sie für die weitere Analyse in eine besser verarbeitbare Format. Anschließend werden diese Daten an einen zentralen Such- und Indizierungsdienst weitergereicht. Der Dienst wird zumeist durch ein verteiltes System bereitgestellt, da ein solches sich leichter skalieren lässt. Durch Anfragen an den Such-Dienst werden die Meldungen von einer entsprechenden Anwendung schließlich visualisiert und korreliert. Folgende Lösungen stehen dafür zur Verfügung:

- *Splunk*  
Bei *Splunk* handelt es sich um eine kommerzielle Lösung für zentralisiertes Log-Management. Es ist eine kostenlose Version verfügbar, welche bis zu 500 MByte an Eingabedaten je Tag verarbeitet. *Splunk* unterstützt von Haus aus bereits viele Anwendungsfälle. So sind entsprechende Parser und Tools zur Visualisierung und Auswertung für die gängigsten Quellen bereits integriert. Sollen damit individuelle Anwendungsfälle abgebildet werden, kann *Splunk* durch sog. Apps dementsprechend erweitert werden. [Spl14]
- *Logstash/Fluentd*, *Elasticsearch* und *Graylog2/Kibana*  
Um eine zu *Splunk* vergleichbare Lösung mit Open-Source-Software zu implementieren ist ein wesentlich größerer Aufwand nötig, da diese nur aus einem Zusammenspiel von entsprechenden Komponenten realisiert werden kann. Als Parser der Logs aus den verschiedenen Quellen kann entweder *Logstash* oder *Fluentd* eingesetzt werden. Zudem kümmern sich diese Lösungen um den Transport zum Such- und Indizierungsdienst *Elasticsearch*. Zur Auswertung und Visualisierung der Daten aus dem Index kann die Oberfläche *Kibana* zum Einsatz kommen. Ein weiteres Tool zur Visualisierung

und Auswertung der Meldungen stellt *Graylog2* dar. Es bietet zudem Funktionen zur Korrelation von verschiedenen Meldungen und Ereignissen. Mit *Graylog2* lassen sich bei bestimmten Bedingungen Alarmer auslösen, welche dann weitere Aktionen nach sich ziehen können. [McD12][Tor13]

Ein solche Auswertung könnte beispielsweise beim FTP-Dienst helfen Zugänge mit bekanntem Passwort zu erkennen. Am LRZ wurde bereits öfters beobachtet, dass in einem solchen Fall der Angreifer eine größere Anzahl an Dateien innerhalb einer kurzen Zeitspanne heruntergeladen und kurz darauf mit Modifikationen, welche sich durch eine geringfügig veränderte Dateigröße zeigten, wieder hochgeladen hat. Alle dafür nötigen Informationen finden sich im Transferprotokoll `xferlog` des FTP-Dienstes. Sofern dieses Protokoll dem zentralen Logsystem zur Verfügung steht, kann es nun die Häufigkeit solcher Vorkommnisse ermitteln. Wird für eine FTP-Kennung ein bestimmter Schwellenwert überschritten sollte durch das Logsystem ein Alarm ausgelöst werden. [LRZ13d]

Durch eine solche Lösung lassen sich Angriffe und Infektionen detektieren, da die aus verschiedenen Quellen stammenden Indizien damit aggregiert werden können. Dies führt zu einer verlässlicheren Aussage darüber, ob eine Infektion oder ein Angriff vorliegt. Es ist damit realisierbar, dass bei ausreichend vorliegenden Indizien ein solches System automatisiert Gegenmaßnahmen wie z.B. die Deaktivierung der betroffenen Webanwendung auslöst (vgl. Abschnitt 3.2.4.3). Bei Verwendung entsprechender Datenquellen kann mit dieser Lösung das Dateisystem auf etwaige Angriffe überwacht werden (vgl. Abschnitt 3.2.3.3).

### 4.6 Gegenüberstellung mit Anforderungen

Anhand der Tabellen 4.1 und 4.2 werden die Lösungsansätze mit den jeweilig erfüllten Anforderungen dargestellt. Zudem geht aus der farblichen Markierung hervor, für welche Produkte von Provider sich der entsprechende Lösungsansatz eignet.





















	Prävention (3.2.1)	Detektion von Verwundbarkeiten (3.2.2)	Detektion von Angriffen (3.2.3)	Reaktion auf Angriffe (3.2.4)
<b>Lösungsansatz anwendbar auf:</b>				
	Virtuelle Maschinen			
	Betreutes Hosting			
	Betreute Webanwendung			
	Virtuelle Maschinen & Betreutes Hosting			
	Virtuelle Maschinen & Betreute Webanwendung			
	Betreutes Hosting & Betreute Webanwendungen			
	Alle Produkte			
<b>Prävention (3.2.1)</b>				
	Sichere Passwortrichtlinien (3.2.1.1)			
	Regelmäßige Wartung von Webanwendungen (3.2.1.2)			
	Automatisierte Installation und Aktualisierung (3.2.1.3)			
	Dokumentation & Schulung (3.2.1.4)			
	Verhinderung von unberechtigten Datenverkehr (3.2.1.5)			
	Minimale Berechtigungen für Webanwendungen (3.2.1.6)			
<b>Detektion von Verwundbarkeiten (3.2.2)</b>				
	Entdeckung von anfälliger/veralteter Software (3.2.2.1)			
	Ermittlung von schwachen Passwörtern (3.2.2.2)			
<b>Detektion von Angriffen (3.2.3)</b>				
	Meldung durch Dritte (3.2.3.1)			
	Auffälligkeiten im Datenverkehr (3.2.3.2)			
	Beobachtung des Dateisystems (3.2.3.3)			
<b>Reaktion auf Angriffe (3.2.4)</b>				
	Isolation von Mandanten (3.2.4.1)			
	Limitierung von Ressourcen (3.2.4.2)			
	Automatisierte Deaktivierung (3.2.4.3)			
<b>Organisatorische Maßnahmen (4.1)</b>				
	Festlegung einer Passwortrichtlinie (4.1.1)			
	Zeitliche Befristung von Produkten (4.1.2)			
	Dokumentation für Kunden (4.1.3)			
	Berücksichtigung von externen Abuse-Datenbanken (4.1.4)			
<b>Betriebssystem (4.2)</b>				
	Einschränkung von Dateiberechtigungen (4.2.1)			
	Ausführung von Diensten in einem Chroot (4.2.2)			
	Linux Containers (LXC) (4.2.3)			
	Docker (4.2.4)			
	Mandatory Access Control mit <i>AppArmor</i> (4.2.5)			

Tabelle 4.1: Lösungen im Überblick (1/2)

	Prävention (3.2.1)	Detektion von Verwundbarkeiten (3.2.2)	Detektion von Angriffen (3.2.3)	Reaktion auf Angriffe (3.2.4)
<b>Lösungsansatz anwendbar auf:</b>	<ul style="list-style-type: none"> <li><span style="color: red;">■</span> Virtuelle Maschinen</li> <li><span style="color: blue;">■</span> Betreutes Hosting</li> <li><span style="color: green;">■</span> Betreute Webanwendung</li> <li><span style="color: red;">■</span> Virtuelle Maschinen &amp; Betreutes Hosting</li> <li><span style="color: red;">■</span> Virtuelle Maschinen &amp; Betreute Webanwendung</li> <li><span style="color: blue;">■</span> Betreutes Hosting &amp; Betreute Webanwendungen</li> <li><span style="color: black;">■</span> Alle Produkte</li> </ul>			
<b>Webserver (4.3)</b>	Sichere Passwortrichtlinien (3.2.1.1)	Regelmäßige Wartung von Webanwendungen (3.2.1.2)	Automatisierte Installation und Aktualisierung (3.2.1.3)	Dokumentation & Schulung (3.2.1.4)
Einsatz einer Web Application Firewall (WAF) (4.3.1)				
Einsatz von HTTPS (4.3.2)	<span style="color: blue;">■</span>			
<b>Sicherheitsrisiken bei Webanwendungen (4.3.3)</b>				
Lösungen auf Dateisystemebene (4.3.3.1)			<span style="color: blue;">■</span>	<span style="color: green;">■</span>
Lösungen mit Zugriff über HTTP (4.3.3.2)			<span style="color: green;">■</span>	
<b>Applikationsserver (4.4)</b>				
Absicherung der Konfiguration von <i>Apache Tomcat</i> (4.4.1)	<span style="color: blue;">■</span>			
Absicherung von <i>Apache Tomcat</i> mit <i>AppArmor</i> (4.4.2)		<span style="color: blue;">■</span>		<span style="color: blue;">■</span>
<b>Infrastruktur (4.5)</b>				
Konfigurationsmanagement mit <i>Puppet</i> (4.5.1)	<span style="color: black;">■</span>	<span style="color: red;">■</span>	<span style="color: black;">■</span>	
Firewall (4.5.2)		<span style="color: black;">■</span>		
Intrusion-Detection-System (IDS) (4.5.3)		<span style="color: black;">■</span>		<span style="color: black;">■</span>
Zentrale Logauswertung (4.5.4)				<span style="color: blue;">■</span>

Tabelle 4.2: Lösungen im Überblick (2/2)

## 4.7 Auswahl von Lösungsansätzen für das LRZ

Nach der ausführlichen Betrachtung der einzelnen Lösungen werden in diesem Abschnitt nun geeignete Maßnahmen ausgewählt, welche sich für die Implementierung am LRZ eignen. Diese Auswahl erfolgt unter Berücksichtigung der individuellen Situation am LRZ, welche im Abschnitt 3.1 besprochen wird.

### 4.7.1 Virtuelle Maschinen

Die Unterscheidung der Service-Varianten „Attended Hosting“ und „Unattended Hosting“ beim Produkt „Virtuelle Maschinen“ am LRZ wirkt sich auch auf die Auswahl der Maßnahmen aus.

Für beide Varianten sollte ein Befristung der Dienstleistung entsprechend Abschnitt 4.1.2 eingeführt werden, da man dadurch VMs, für die kein Ansprechpartner erreichbar ist, nach einem gewissen Zeitraum erkennen kann. Ebenso sollten externe Daten zur Entdeckung von angegriffenen VMs (vgl. Abschnitt 4.1.4) bei beiden Varianten berücksichtigt werden.

Die am LRZ bestehende Passwortrichtlinie ist für das Produkt „Virtuelle Maschinen“ ausreichend, da die in Abschnitt 4.1.1 angesprochenen zusätzlichen Regeln bei der Beantragung von VMs berücksichtigt werden. Spezialisierte Regeln für VMs am Intrusion-Detection-System (IDS) (vgl. Abschnitt 4.5.3) vom LRZ sind nicht praktikabel, denn sie würden eine hohe Anzahl an Fehlalarmen erzeugen, weil die auf den VMs betriebenen Dienste individuell vom Kunden gewählt werden und dem LRZ nicht bekannt sind. Aus demselben Grund ist der Einsatz von Firewallregeln durch das LRZ (vgl. Abschnitt 4.5.2) nicht sinnvoll.

Stattdessen sollte den Kunden entsprechende Dokumentation (vgl. Abschnitt 4.1.3) zur Verfügung gestellt werden, welche die Konfiguration einer Firewall und anderer sicherheitsrelevanter Einstellungen an der VM erläutert.

Bei VMs mit „Attended Hosting“ sollte auf längerfristige Sicht das Konfigurationsmanagement-Tool *Puppet* verwendet werden (vgl. Abschnitt 4.5.1). Im Zuge der Einführung von *Puppet* können ggf. neben *SuSE Linux Enterprise* weitere Linux-Betriebssysteme bei der Service-Variante *Attended Hosting* unterstützt werden.

Zudem sollte ein zentrales Logging-System für „Attended Hosting“-VMs etabliert werden (vgl. Abschnitt 4.5.4). Da am LRZ *Splunk* bereits für andere Dienste eingesetzt wird, sollte diese Lösung zur Analyse der VM-Logdateien verwendet werden.

Die für das Produkt „Virtuelle Maschinen“ am LRZ empfohlenen Maßnahmen sind hier nochmals übersichtlich zusammenfasst:

- „Virtuelle Maschinen“ (generell)
  - Verträge zeitlich befristen (vgl. Abschnitt 4.1.2).
  - Dokumentation für sichere Basis-Konfiguration von VMs erstellen (vgl. Abschnitt 4.1.3).
  - Berücksichtigung von externen Abuse-Datenbanken (vgl. Abschnitt 4.1.4).
- „Virtuelle Maschinen“ mit „Attended Hosting“
  - Konfigurationsmanagement von VMs mit *Puppet* (vgl. Abschnitt 4.5.1). Ggf. mit Unterstützung weiterer Betriebssysteme.
  - Zentrale Logauswertung mit *Splunk* (vgl. Abschnitt 4.5.4).

### 4.7.2 Betreutes Hosting

Da beim Produkt „Betreutes Hosting“ ein Provider für den Betrieb von mehr Komponenten als beim Produkt „Virtuelle Maschinen“ verantwortlich ist, bieten sich dort beim LRZ auch mehr Lösungsansätze zur Umsetzung an.

Ebenso wie bei VMs sollte beim Webhosting eine Befristung der Dienstleistung eingeführt werden (vgl. Abschnitt 4.1.2). Auch ist eine Einbeziehung von externen Abuse-Datenbanken zur Entdeckung von angegriffenen Webanwendungen hilfreich (vgl. Abschnitt 4.1.4). Gleichfalls sollte für alle anfallenden Logdateien eine zentrale Lösung durch *Splunk* implementiert werden (vgl. Abschnitt 4.5.4).

Da das LRZ keine Hosting-Dienstleistungen mit Applikationsservern anbietet, ist eine Umsetzung der dazugehörigen Lösungen aus den Abschnitten 4.4.1 und 4.4.2 am LRZ nicht notwendig.

Die Konfiguration der am Webhosting des LRZs beteiligten Systeme – wie z.B. der Webserver-Instanzen „webX“ und des Entwicklungsservers „webdev1“ – sollte das Konfigurationsmanagement-Tool *Puppet* zum Einsatz kommen. Damit können die Einstellungen zentralisiert in entsprechenden Manifesten für die gesamte Webhosting-Infrastruktur getroffen werden (vgl. Abschnitt 4.5.1).

Durch Dokumentationen und ggf. Schulungen (vgl. Abschnitt 4.1.3) sollte das LRZ seinen Kunden die nötigen Kenntnisse vermitteln, welche sie zur Installation, Konfiguration und Pflege von Webanwendungen beim Webhosting des LRZs benötigen. Dabei sollte auf speziell auf die Konfiguration von häufig eingesetzten Webanwendungen eingegangen werden. Diese Maßnahmen würde zu einer Sensibilisierung der Kunden für mögliche Gefahren, welche beim Betrieb von Webanwendungen drohen, führen.

Die Passworrichtlinie des LRZs sollte für das Webhosting in gewissen Aspekten verschärft werden (vgl. Abschnitt 4.1.1). Um ein Abfangen von Kunden-Passwörter zu erschweren, sollte dort die unverschlüsselte Übertragung von Passwörter, wie sie derzeit beim FTP-Dienst praktiziert wird, ausgeschlossen werden. Damit die von Kunden eingesetzten Passwörter für Webapplikationen auch verschlüsselt übertragen werden können, sollte entsprechend Abschnitt 4.3.2 für jede Kennung ein Zugriff auf die Webanwendungen mittels HTTPS bereitgestellt werden.

Die derzeitig auf den LRZ-Webservern verwendeten Dateiberechtigungen sollte so beibehalten werden. Eine Verschärfung dieser durch Separierung der Berechtigungen von Webanwendung und Kundenzugriff, würde zu viele Nachteile in der praktischen Umsetzung mit sich bringen (vgl. Abschnitt 4.2.1). Insbesondere würden solche Berechtigungen dazu führenden, dass eine automatische Selbstaktualisierung von Webanwendungen nicht mehr möglich ist. Um die Berechtigungen von Webanwendungen auf den Webservern weitergehend einzuschränken, sollte stattdessen die MAC-Lösung *AppArmor* eingesetzt werden (vgl. Abschnitt 4.2.5).

Als weitere Einschränkung sollte der Datenverkehr für die Webanwendungen gemäß Abschnitt 4.5.2 reglementiert werden. Die Beobachtung des Datenverkehrs mit dem IDS des LRZs sollte um zusätzliche Regeln zur Erkennung von Angriffen auf Webanwendungen erweitert werden (vgl. Abschnitt 4.5.3).

Um infizierte Webanwendungen besser erkennen zu können, bietet sich der Einsatz von *Linux Malware Detect* an. Zusätzlich dazu sollten veraltete Webanwendungen mit `webapp_discover` identifiziert werden. Die dazugehörigen Kunden sollten anschließend durch eine E-Mail über dieses Sicherheitsrisiko informiert werden und zu einer Aktualisierung der entsprechenden Webanwendung angeregt werden (vgl. Abschnitt 4.3.3.1).

Auf längerfristige Sicht sollte eine Migration des LRZ-Webhostings auf die containerbasierte Lösung *Docker* angestrebt werden (vgl. Abschnitt 4.2.4). Durch die Verwendung von Containern für jede Produkt-Instanz kann eine individuelle Laufzeitumgebung für Webanwendungen bereitgestellt werden. Diese Umgebungen lassen sich zudem auch für das Produkt „Betreute Webanwendungen“ einsetzen, welches am LRZ ggf. für häufig eingesetzte Webanwendungen eingeführt werden könnte. Bis *Docker* produktiv eingesetzt werden kann, wird wohl noch einige Zeit vergehen. Ebenfalls ist noch nicht abzusehen, wann *Docker* die Linux-Distribution *SuSE Linux Enterprise*, welche am LRZ vorwiegend eingesetzt wird, unterstützt.

Die in diesem Abschnitt vorgeschlagenen Maßnahmen sind in der nachfolgenden Übersicht nach deren geschätzten Realisierungszeitraum gruppiert:

- kurzfristig
  - Kunden, welche laut `webapp_discover` nicht aktualisierte Webanwendungen einsetzen, darüber informieren und zum Upgrade auffordern (vgl. Abschnitte 4.3.3.1 und 5.2).
  - Regelmäßigen Scan nach Malware auf Webserver-Dateisystem durchführen (vgl. Abschnitt 4.3.3.1).
  - Beschränkung der Netzanbindung durch Firewall (vgl. Abschnitt 4.5.2).
- mittelfristig
  - Verschärfung der Passwortrichtlinien am LRZ (vgl. Abschnitt 4.1.1).
  - Webhosting-Verträge zeitlich befristen (vgl. Abschnitt 4.1.2).
  - Dokumentation für sichere Basis-Konfiguration von häufig verwendeten Webanwendungen (vgl. Abschnitt 4.1.3).
  - Berücksichtigung von externen Abuse-Datenbanken (vgl. Abschnitt 4.1.4).
  - HTTPS mit Wildcard-Domain für alle Kunden anbieten (entsprechend Abschnitt 4.3.2).
  - Mit *AppArmor* die Rechte von Webanwendungen beschränken (vgl. Abschnitt 4.2.5).
  - Konfigurationsmanagement von Webservern mit *Puppet* (vgl. Abschnitt 4.5.1).
  - Zentrale Logauswertung mit *Splunk* etablieren (vgl. Abschnitt 4.5.4).
- langfristig
  - Spezialisierte IDS-Regeln für Web- und Applikationsserver (vgl. Abschnitt 4.5.3).
  - Webhosting mit *Docker* realisieren (vgl. Abschnitt 4.2.4).
  - Das Produkt „Betreute Webanwendung“ für Webanwendungen einführen, welche am LRZ häufig eingesetzt werden (vgl. Vorgehen für *Wordpress* in Abschnitt 5.3).





## 5 Prototypische Implementierung am LRZ

Dieses Kapitel soll die Umsetzbarkeit der im Laufe der Arbeit vorgestellten Konzepte in der Praxis nachweisen. Dafür werden einzelne Maßnahmen ausgewählt und prototypisch implementiert.

In Abschnitt 5.1 wird gezeigt, wie ein *Apache Tomcat*-Applikationsserver unter Verwendung von *Puppet* installiert und konfiguriert werden kann. Dieser Prototyp kann zur Pflege von entsprechenden Kunden-VMs mit Service-Variante „Attended Hosting“ am LRZ verwendet werden. Desweiteren ist eine Weiterverwendung des Prototypen für das Produkt „Betreutes Hosting“ denkbar, falls diese Dienstleistung zukünftig auch Java-Web-Applikationen ausgeweitet werden soll. Das dabei erstellte Manifest ist so angelegt, dass es prinzipiell verschiedenste Linux-Distributionen unterstützt. Bei der von *SuSE Linux Enterprise 11* mitgelieferten *Augeas*-Version trat ein Problem mit der Verarbeitung von XML-Dateien bereits bei ersten Tests auf, daher wurde das entsprechende *Puppet*-Manifest für die Distribution *Debian Wheezy* entwickelt und getestet. Einer späteren Anpassung an eine neuere Version von *SuSE Linux Enterprise* steht jedoch nichts im Weg.

Als zweite Maßnahme wird die prototypische Implementierung von `webapp_discover` in Abschnitt 5.2 diskutiert. Zudem werden die Ergebnisse eines Testlaufes auf den Daten des Webhostings am LRZ gezeigt.

In Abschnitt 5.3 wird unter Verwendung von *Docker* gezeigt, wie das Produkt „Betreute Webanwendungen“ durch Container-Virtualisierung realisiert werden kann. Durch die fehlende Unterstützung von *Docker* bei *SuSE Linux Enterprise 11* musste hier ebenfalls auf eine alternative Linux-Distribution zurückgegriffen werden. Während auf dem Hostsystem *Ubuntu Linux 12.04* zum Einsatz kam, wurde für die Container als Basis *Debian Wheezy* eingesetzt.

### 5.1 Sichere Konfiguration von *Apache Tomcat* durch *Puppet*

In diesem Abschnitt soll aufgezeigt werden, wie ein *Apache Tomcat*-Applikationsserver durch die Konfigurationsmanagement-Software *Puppet* installiert und mit einer initialen sicheren Konfiguration versehen werden kann. Da einige Dienste am LRZ bereits mit *Puppet* konfiguriert werden, wurden die im Rahmen dieses Abschnittes erstellten Manifeste in das bestehende Regelwerk des LRZs integriert. Die dazugehörigen Dateien befinden sich unter Versionsverwaltung durch *Git* und sind für entsprechend autorisierte LRZ-Mitarbeiter unter `gitosis@git.lrz.de:puppet-websec.git` zu erreichen.

Zur Aufteilung des im Rahmen dieses Abschnittes erstellten *Puppet*-Moduls `lrz_kom_tomcat` wurden die jeweiligen Ressourcen in Klassen eingeteilt. Diese Struktur folgt gängigen Patterns bei der Erstellung von *Puppet*-Modulen. Es konnte daher mit einer Vorlage (Quelle: [Rus14]) für *Puppet*-Module gestartet werden. Entsprechend dieser Vorlage wurden folgende Klassen zur Aufteilung des Manifests benutzt:

- `lrz_kom_tomcat::params` dient zur Definition der Standard-Parameter. Zusätzlich wird in dieser Klasse die Unterteilung in verschiedene Betriebssysteme vorgenommen, da sich beispielsweise die Namen der nötigen Pakete von Distribution zu Distribu-

tion unterscheiden. Derzeit sind nur die Linux-Betriebssysteme *Debian* und *Ubuntu* unterstützt.

- `lrz_kom_tomcat::install` installiert die nötigen Pakete.
- `lrz_kom_tomcat::config` konfiguriert den *Tomcat* hinsichtlich verschiedener Sicherheitsaspekte (vgl. dazu die Abschnitte 4.4.1 und 4.4.2). Zur besseren Übersichtlichkeit wurde diese Konfiguration in Unterklassen aufgeteilt: Die Klasse `lrz_kom_tomcat::config::userdb` konfiguriert dabei die Benutzerdatenbank. Die HTTPS Konfiguration des *Tomcat*-Server findet in der Klasse `lrz_kom_tomcat::config::tls` statt. Und schließlich wird durch die Klassen `lrz_kom_tomcat::config::apparmor` und `lrz_kom_tomcat::config::apparmor_changehat` eine MAC für *Tomcat* durch *AppArmor* eingerichtet.
- `lrz_kom_tomcat::service` sorgt dafür, dass der *Tomcat*-Dienst gestartet wird.

[Pie12]

Die Definition des Moduls selbst ist in Listing 5.1 dargestellt. Darin werden die oben vorgestellten Unterklassen aufgerufen und die Abhängigkeiten zwischen den einzelnen Klassen definiert. Außerdem werden diese Argumente für das Modul festgelegt:

- Mit `ensure` gibt man an, ob der *Tomcat*-Server installiert oder ggf. entfernt werden soll. Wird der Parameter auf `present` oder `latest` gesetzt, wird sichergestellt, dass die nötigen Pakete installiert sind. Ein Wert von `absent` entfernt dagegen den *Tomcat*-Server. Der Standardwert des Parameters ist `present`.
- `manager_password` definiert das zu verwendende Passwort für die Manager-Webapplikation, welche die Verwaltung des *Tomcat*-Servers vereinfacht. Wird kein Kennwort angegeben, wird nach der Installation ein zufälliges Kennwort generiert.
- Über den Parameter `apparmor` lässt sich die *AppArmor*-Unterstützung aktivieren. Standardmäßig wird *AppArmor* nur dann aktiviert, wenn es bereits auf dem System eingerichtet ist. Der Parameter erwartet boolesche Eingabewerte.
- `apparmor_changehat` legt fest, ob der *Tomcat*-Server mit dem sog. Change-Hat in andere Bereiche des Profils wechseln soll (vgl. Abschnitt 4.4.2). Diese Einstellung ist standardmäßig aktiviert und erwartet als Eingabe boolesche Werte.
- Der Parameter `tls` aktiviert die HTTPS-Unterstützung des *Tomcat*-Server. Standardmäßig ist diese mit dem Wert `true` aktiviert.

```
# Modul zur Konfiguration von Apache Tomcat
class lrz_kom_tomcat (
    $ensure = $lrz_kom_tomcat::params::ensure ,
    $manager_password = undef ,
    $apparmor = $lrz_kom_tomcat::params::apparmor ,
    $apparmor_change_hat = true ,
    $tls = true ,
) inherits lrz_kom_tomcat::params {
```

```

# Validate ensure
case $ensure {
  # Installieren
  'present', 'latest' : {
    class { 'lrz_kom_tomcat::install': } ->
    class { 'lrz_kom_tomcat::config': } ~>
    class { 'lrz_kom_tomcat::service': } ->
    Class ['lrz_kom_tomcat']
  }
  # Entfernen
  'absent' : {
    class { 'lrz_kom_tomcat::install': } ->
    Class ['lrz_kom_tomcat']
  }
  default : {fail("ensure = '${ensure}' not supported")}
}
}

```

Listing 5.1: Definition des Moduls `lrz_kom_tomcat` in der Datei `init.pp`

Um dieses *Puppet*-Modul nun zu verwenden, muss lediglich die Klasse `lrz_kom_tomcat` eingebunden werden. Damit automatisiert Updates installiert werden und eine Verwendung von *AppArmor* erzwungen wird, sollten dabei die entsprechenden Parameter so gesetzt werden:

```

class { 'lrz_kom_tomcat':
  ensure => latest,
  apparmor => true,
}

```

Bei der Entwicklung des Modules wurde ein test-getriebener Ansatz verfolgt. Dazu wurden zuerst gewünschte Testszenarien definiert. Diese geben an, wie sich das Manifest bei bestimmten Eingabewerten zu verhalten hat. Anschließend wurde das Manifest entwickelt und laufend die Szenarien getestet. Diese Tests werden mit dem Hilfsprogramm *RSpec-Puppet* durchgeführt. Dazu muss folgender Befehl im Modulverzeichnis ausgeführt werden: `bundle exec rake test` [Sha12]

### 5.1.1 Installation der nötigen Pakete

Die Installation der benötigten Pakete findet in der Unterklasse `lrz_kom_tomcat::install` statt. Diese ist im Listing 5.2 dargestellt. Dort werden drei `Package`-Ressourcen verwendet, um die für den Betrieb des *Tomcat*-Server nötigen Pakete zu installieren. Die Namen der Pakete werden durch die Unterklasse `lrz_kom_tomcat::params` bereitgestellt.

Es wird dabei die Java-Laufzeitumgebung *OpenJDK 7* installiert, da erst ab der Version 7 moderne SSL-Standards TLS 1.1 und 1.2 unterstützt werden. Das spielt bei der sicheren Konfiguration von HTTPS eine Rolle (vgl. Abschnitt 5.1.3).

Die anderen beiden zu installierenden Pakete sind der *Tomcat*-Server selbst, sowie die Webapplikation zur Administration. Dabei gilt es zu beachten, dass bei der Entwicklung des Manifests ein Fehler im Hilfsprogramm *Augeas* entdeckt wurde: Es konnten keine XML-Dateien eingelesen werden, welche innerhalb der XML-Deklaration, die sich in der ersten Zeile einer XML-Datei befindet, einfache Anführungszeichen verwendeten. Daher werden als

Workaround mit einer `Exec`-Ressource nach der Installation die einfachen Anführungszeichen durch doppelte ersetzt.

```
# Installiert notwendige Pakete für Apache Tomcat
class lrz_kom_tomcat::install {

  # Verwende Standard-Parameter
  include lrz_kom_tomcat::params

  # OpenJDK7
  package { $lrz_kom_tomcat::params::java_package_name :
    ensure => $lrz_kom_tomcat::ensure,
  }

  # Tomcat
  package { $lrz_kom_tomcat::params::package_name :
    ensure => $lrz_kom_tomcat::ensure,
  }

  # Tomcat Manager Webapplikation
  package { $lrz_kom_tomcat::params::manager_package_name :
    ensure => $lrz_kom_tomcat::ensure,
  }

  # Workaround: Augeas-Bug XML single quotes
  exec { "${module_name}_workaround_augeas_bug_wheezy":
    refreshonly => true,
    subscribe   => Package[$lrz_kom_tomcat::params::package_name],
    command     => "/bin/sed -i \"s/<?xml version='1.0' encoding='utf-8'?>/<?
      xml version=\\\\"1.0\\\\" encoding=\\\\"utf-8\\\\"?>/g\" ${lrz_kom_tomcat
        ::params::config_server_path} ${lrz_kom_tomcat::params::
          config_tomcat_users_path}\"",
  }
}
```

Listing 5.2: Installation der benötigten Pakete in der Datei `install.pp`

### 5.1.2 Erstellung eines Zugangs zur Administration

In diesem Abschnitt wird die Klasse `lrz_kom_tomcat::config::userdb` besprochen. Der Inhalt dieser ist aus Listing 5.3 zu entnehmen. Es wird damit sichergestellt, dass die notwendigen Rollen und Benutzer in der Nutzerdatenbank des *Tomcat*-Servers existieren, so dass die Webapplikation zur Administration verwendet werden kann.

Dazu werden mit einer `Augeas`-Ressource ggf. fehlende Rollen hinzugefügt. Ein weitere `Augeas`-Ressource fügt den Benutzer `admin` hinzu, sofern dieser nicht existiert. Dieser erhält dann ein zufällig generiertes Passwort, welches durch die SHA1-Hashfunktion abgespeichert wird. Der Klartext des Passwortes wird in einer Datei, welche nur für den privilegierten `root`-Benutzer lesbar ist, mit einer `Exec`-Ressource abgelegt. Zuletzt konfiguriert eine weitere `Augeas`-Ressource in Konfiguration vom *Tomcat*-Server, dass SHA1-Hashes in der Nutzerdatenbank verwendet werden.

```

# Konfiguration der Nutzerdatebank
class lrz_kom_tomcat::config::userdb {

  # Füge Rollen hinzu
  augeas { "${module_name}_add_roles":
    incl    => $lrz_kom_tomcat::params::config_tomcat_users_path,
    lens    => 'Xml.lns',
    require => File[$lrz_kom_tomcat::params::config_tomcat_users_path],
    changes => [
      'set tomcat-users/role[1]/#attribute/rolename manager-gui',
      'set tomcat-users/role[2]/#attribute/rolename manager',
      'set tomcat-users/role[3]/#attribute/rolename admin-gui',
      'set tomcat-users/role[4]/#attribute/rolename admin',
    ],
  }

  # Wenn kein Benutzer admin existiert, erzeuge neuen mit zufälligen Passwort
  $password=random_password(14)
  $password_hash=shasum($password)
  augeas { "${module_name}_add_admin_user":
    incl    => $lrz_kom_tomcat::params::config_tomcat_users_path,
    lens    => 'Xml.lns',
    require => File[$lrz_kom_tomcat::params::config_tomcat_users_path],
    onlyif  => 'match tomcat-users/user[#attribute/username=\`admin\`] size
      ==0',
    changes => [
      'set tomcat-users/user[last()+1]/#attribute/username admin',
      "set tomcat-users/user[#attribute/username='admin']/#attribute/password
        ${password_hash}",
      'set tomcat-users/user[#attribute/username=\`admin\`]/#attribute/roles
        admin,admin-gui,manager,manager-gui',
    ],
    notify  => Exec["${module_name}_echo_password"],
  }

  # Schreibe Passwort in die Datei .tomcat_manager_password
  exec {"${module_name}_echo_password":
    command    => "/bin/sh -c \"umask 0066 && /bin/echo '${password}' > ${
      lrz_kom_tomcat::params::config_path}/.tomcat_manager_password\"",
    refreshonly => true,
  }

  # Einstellung dass die Benutzerdatenbank SHA1 Hashes verwendet
  augeas { "${module_name}_config_realm_digest":
    incl    => $lrz_kom_tomcat::params::config_server_path,
    lens    => 'Xml.lns',
    changes => [
      'set //Realm[#attribute/className=\`org.apache.catalina.realm.
        UserDatabaseRealm\`]/#attribute/digest SHA'
    ],
  }
}

```

Listing 5.3: Einrichtung der Benutzerdatenbank von *Tomcat*

### 5.1.3 Aktivierung des HTTPS-Zugriffes

Die Klasse `lrz_kom_tomcat::config::tls` aktiviert nun HTTPS für den *Apache Tomcat*-Server. Das zugehörige *Puppet*-Manifest ist in dem Listing 5.4 abgedruckt. Darin wird ein sog. Keystore erzeugt, welcher den privaten Schlüssel für das selbst-signierte Zertifikat enthält. Dazu ruft die `Exec`-Ressource das `keytool` entsprechend den Anweisungen im Abschnitt 4.3.2 auf. Mit der `File`-Ressource werden anschließend die Berechtigungen dieser Datei angepasst, so dass sie ausschließlich vom *Tomcat*-Server gelesen werden kann. Über die `Augeas`-Ressource wird die Konfiguration des *Tomcat*-Servers, wie in Abschnitt 4.3.2 besprochen, abgeändert, damit dieser auch HTTPS anbietet.

Um nun alle Verbindungen zur Manager-Webanwendung nur noch über HTTPS abzuwickeln, werden die dafür zuständigen Konfigurationsdateien mit zwei weiteren `Augeas`-Ressourcen entsprechend angepasst.

```
# Konfiguriere HTTPS
class lrz_kom_tomcat::config::tls {

  # Pfad zum Keystore
  $keystore = "${lrz_kom_tomcat::params::config_path}/tomcat.keystore"

  # Setze Berechtigung des Keystores
  file {[$keystore] :
    owner => root,
    group => $lrz_kom_tomcat::params::service_group,
    mode  => '0640',
  }

  # Generiere Passwort für den Keystores
  $keystore_pass = random_password(14)

  # Erzeuge selbstsigniertes Zertifikat & Schlüssel
  exec{"${module_name}_create_self_signed_cert":
    refreshonly => true,
    command     => "/usr/bin/keytool -genkey -alias tomcat -keyalg RSA -
      keysize 4096 -keystore ${keystore} -keypass \"${keystore_pass}\" -
      storepass \"${keystore_pass}\" -dname \"cn=$(hostname --fqdn)\",
    creates     => $keystore,
    notify     => [
      File[$keystore],
      Class['lrz_kom_tomcat::service'],
    ]
  }

  # Konfiguriere TLS im Tomcat
  Augeas { "${module_name}_config_tls":
    incl => $lrz_kom_tomcat::params::config_server_path,
    lens => 'Xml.lns',
    require => File[$lrz_kom_tomcat::params::config_server_path],
    onlyif => 'match Server/Service/Connector[#attribute/port="8443"] size
      ==0',
    notify => Exec["${module_name}_create_self_signed_cert"],
    changes => [
      'set Server/Service/Connector[last()+1]/#attribute/port 8443',
      'set Server/Service/Connector[last()]/#attribute/protocol HTTP/1.1',
    ]
  }
}
```

```

    'set Server/Service/Connector[last()]/#attribute/maxThreads 200',
    'set Server/Service/Connector[last()]/#attribute/scheme https',
    'set Server/Service/Connector[last()]/#attribute/secure true',
    'set Server/Service/Connector[last()]/#attribute/SSLEnabled true',
    'set Server/Service/Connector[last()]/#attribute/clientAuth false',
    'set Server/Service/Connector[last()]/#attribute/sslProtocol TLSv1.2',
    'set Server/Service/Connector[last()]/#attribute/keyAlias tomcat',
    "set Server/Service/Connector[last()]/#attribute/keystoreFile ${
        keystore}",
    "set Server/Service/Connector[last()]/#attribute/keystorePass ${
        keystore_pass}",
  ],
}

# Erzwingen TLS bei Manager
augeas { "${module_name}_config_manager_force_tls":
  incl => $lrz_kom_tomcat::params::config_admin_host_manager_path,
  lens => 'Xml.lns',
  changes => [
    'set web-app/security-constraint[1]/user-data-constraint/transport-
      guarantee/#text "CONFIDENTIAL"',
    'set web-app/security-constraint[2]/user-data-constraint/transport-
      guarantee/#text "CONFIDENTIAL"',
  ]
}

augeas { "${module_name}_config_host_manager_force_tls":
  incl => $lrz_kom_tomcat::params::config_admin_manager_path,
  lens => 'Xml.lns',
  changes => [
    'set web-app/security-constraint[1]/user-data-constraint/transport-
      guarantee/#text "CONFIDENTIAL"',
    'set web-app/security-constraint[2]/user-data-constraint/transport-
      guarantee/#text "CONFIDENTIAL"',
  ]
}
}
}

```

Listing 5.4: Aktivierung von HTTPS beim *Tomcat*-Server

#### 5.1.4 *AppArmor*-Profil für den *Tomcat*-Server aktivieren

Die Unterklasse `lrz_kom_tomcat::config::apparmor` wendet das im Abschnitt 4.4.2 entwickelte *AppArmor*-Profil auf den *Tomcat*-Server an. Das hierfür nötige Manifest ist im Listing 5.5 dargestellt. Es benutzt über die `include`-Anweisung das *Puppet*-Modul `lrz_kom_apparmor`, welches die Installation und Aktivierung von *AppArmor* durchführt. Diese Funktionalität wurde in ein eigenes Modul ausgelagert, damit es dann unabhängig vom Modul `lrz_kom_tomcat` benutzt werden kann. Nach der erstmaligen Aktivierung von *AppArmor* muss das entsprechende System neugestartet werden, damit der dabei gesetzte Kernel-Parameter `security=apparmor` aktiv wird.

Um anschließend das Profil auf dem System aktivieren zu können, müssen zuvor die nötigen Dateien mit jeweils einer `File`-Ressource dorthin kopiert worden sein. Diese `File`-Ressourcen benachrichtigen dann über den Parameter `notify` die `Service`-Ressource von *AppArmor*. Sie startet daraufhin den zugehörigen Dienst neu und dieser lädt dabei das neu erstellte Profil.

```

# Konfiguriert das AppArmor Profil für Tomcat
class lrz_kom_tomcat::config::apparmor {

# Lade AppArmor Modul
include lrz_kom_apparmor

$base_name = $lrz_kom_tomcat::params::base_name

# Tomcat Profile
file {"${lrz_kom_apparmor::params::config_path}/usr.share.${base_name}.bin.
  catalina.sh":
  ensure => file,
  mode   => '0644',
  owner  => 'root',
  group  => 'root',
  content => template("${module_name}/apparmor/catalina.sh.erb"),
  notify => Class['lrz_kom_apparmor::service']
}

# Tomcat Base
file {"${lrz_kom_apparmor::params::config_path}/abstractions/tomcat":
  ensure => file,
  mode   => '0644',
  owner  => 'root',
  group  => 'root',
  content => template("${module_name}/apparmor/abstractions-tomcat.erb"),
  notify => Class['lrz_kom_apparmor::service']
}

# Tomcat Wepapp für extra Verzeichnisse
file {"${lrz_kom_apparmor::params::config_path}/abstractions/tomcat-webapp"
  :
  ensure => file,
  mode   => '0644',
  owner  => 'root',
  group  => 'root',
  notify => Class['lrz_kom_apparmor::service']
}

# Java Base
file {"${lrz_kom_apparmor::params::config_path}/abstractions/java":
  ensure => file,
  mode   => '0644',
  owner  => 'root',
  group  => 'root',
  content => template("${module_name}/apparmor/abstractions-java.erb"),
  notify => Class['lrz_kom_apparmor::service']
}
}

```

Listing 5.5: Installation der *AppArmor*-Profile von *Tomcat*

Damit der *Tomcat*-Server die Change-Hat Funktionalität unterstützt, wird in der Unterklasse `lrz_kom_tomcat::config::apparmor_change_hat` das zugehörige Plugin installiert (vgl. Listing 5.5). Dieses Plugin ist Teil von *AppArmor*, wird jedoch bei *Debian Wheezy* nicht als Paket angeboten. Über zwei File-Ressourcen werden die beiden Bibliotheken in die ent-



sprechenden Pfade kopiert. Während sich die Java-Bibliothek `changeHatValve.jar` um die Einbindung in den *Tomcat*-Server kümmert, greift die native Bibliothek `libJNIChangeHat.so` auf die entsprechenden Schnittstellen von *AppArmor* zu.

Damit das Plugin erfolgreich geladen wird, muss zum einen der Pfad zu der nativen Bibliothek beim Start des *Tomcat*-Servers übergeben werden und zum anderen die Java-Bibliothek in der Konfiguration des Servers als Plugin festgelegt werden. Dies wird durch zwei *Augeas*-Ressourcen realisiert.

```
# Konfiguriert Tomcat für Webapps verschiedene Hats zu benutzen
class lrz_kom_tomcat::config::apparmor_change_hat {

    $base_name = $lrz_kom_tomcat::params::base_name

    # Installiere Change Hat Plugin
    file {"/usr/share/${base_name}/lib/changeHatValve.jar":
        ensure => file,
        mode    => '0644',
        owner   => 'root',
        group   => 'root',
        source  => "puppet:///modules/${module_name}/apparmor/changeHatValve.jar",
        before  => Augeas["${module_name}_enable_change_hat"],
    }

    file {'/usr/lib/libJNIChangeHat.so':
        ensure => file,
        mode    => '0644',
        owner   => 'root',
        group   => 'root',
        source  => "puppet:///modules/${module_name}/apparmor/libJNIChangeHat.so",
        before  => Augeas["${module_name}_enable_change_hat"],
    }

    # Korrigiere Pfad zu nativen Library
    Augeas {"${module_name}_add_lib_path":
        before => Augeas["${module_name}_enable_change_hat"],
        context => "/files/etc/default/${base_name}",
        changes => 'set JAVA_OPTS "\'-Djava.awt.headless=true -Xmx128m -XX:+
            UseConcMarkSweepGC -Djava.library.path=/usr/lib\'"',
    }

    # Aktiviere Change Hat in Tomcat Konfiguration
    Augeas {"${module_name}_enable_change_hat":
        incl    => $lrz_kom_tomcat::params::config_server_path,
        lens    => 'Xml.lns',
        require => File[$lrz_kom_tomcat::params::config_server_path],
        onlyif  => 'match Server/Service/Engine/Host/Valve[#attribute/className =
            "com.novell.apparmor.catalina.valves.ChangeHatValve"] size==0',
        notify  => Class[lrz_kom_tomcat::service],
        changes => [
            'set Server/Service/Engine/Host/Valve[last()+1]/#attribute/className "
                com.novell.apparmor.catalina.valves.ChangeHatValve"',
            'set Server/Service/Engine/Host/Valve[last()]/#attribute/mediationType
                "ServletPath"',
        ]
    }
}
}
```

Listing 5.6: Aktivierung des Change-Hat Plugins beim *Tomcat*-Server

### 5.1.5 Test des *AppArmor*-Profils

Um das ordnungsgemäße Zusammenspiel von *Tomcat* und *AppArmor* zu testen, wird eine einfache Webanwendung entwickelt, welche versucht die Passwortdatei `tomcat-users.xml` auszulesen. Da für Webanwendungen durch die Change-Hat Funktionalität andere *AppArmor*-Regeln wie für den *Tomcat*-Server gelten, sollte dieser Zugriff durch *AppArmor* unterbunden werden, wenn die Einrichtung korrekt vollzogen wurde.

Wie Listing 5.7 zu entnehmen ist, verfolgt die Webanwendungen zwei verschiedene Ansätze, um die Passwortdatei zu lesen: Beim ersten Versuch wird direkt über ein `FileReader`-Objekt gelesen. Im zweiten Versuch wird das Kommando `cat` zum Anzeigen der Inhalte der Passwortdatei benutzt. Bei fehlerfreier Einrichtung von *AppArmor* sollten beide Varianten fehlschlagen. Wenn die erste Variante die Passwortdatei anzeigt, jedoch die zweite nicht, so ist *AppArmor* für den *Tomcat*-Server aktiv, allerdings schlägt der Wechsel des Bereichs mit dem Change-Hat Aufruf fehl. Zeigen beide Varianten die Inhalte der Passwortdatei, so ist *AppArmor* für den *Tomcat*-Server komplett deaktiviert.

```
public class AppArmor {

    String tomcat_users_path = "/etc/tomcat7/tomcat-users.xml";

    /* Lese Inhalt von Reader */
    public String getContent(Reader reader_unbuf) throws IOException
    [...]

    /* Lese tomcat-users.xml über FileReader */
    public String getTomcatUsersFopen() throws IOException {

        return getContent(new FileReader(this.tomcat_users_path));
    }

    /* Lese tomcat-users.xml über Ausführung von cat */
    public String getTomcatUsersExec() throws IOException {

        /* Führe Kommando aus */
        Runtime rt = Runtime.getRuntime();
        String[] cmd = {"/bin/cat", this.tomcat_users_path};
        Process pr = rt.exec(cmd);

        return getContent(new InputStreamReader(pr.getInputStream()));
    }
}
```

Listing 5.7: Webanwendung zum Testen der *AppArmor*-Konfiguration des *Tomcat*-Servers

Um die korrekte Konfiguration von *Tomcat* und *AppArmor* zu testen, wird die Webanwendung auf einen fertig eingerichteten *Tomcat*-Server installiert. Wenn die *AppArmor*-Regeln wirksam sind, sollten beim Aufruf der Webanwendung die Inhalte der Passwortdatei bei beiden Varianten nicht angezeigt werden (wie in Abbildung 5.1).

## Lese tomcat-users.xml via FileReader

An exception occurred: `/etc/tomcat7/tomcat-users.xml` (Keine Berechtigung)

## Lese tomcat-users.xml via exec

An **exception** occurred: Cannot run program `"/bin/cat"`: error=13, Keine Berechtigung

Abbildung 5.1: Zugriff der Webanwendung auf `tomcat-users.xml` wird blockiert

## 5.2 Installierte Webanwendungen detektieren mit *webapp\_discover*

Da keine bereits existierende Lösung gefunden werden konnte, welche die Versionen von installierten Webanwendungen aus einem Verzeichnisbaum ermittelt, wurde beschlossen diese Lösung im Rahmen der Arbeit zu implementieren. Es wurde sich für die Skriptsprache *Python* entschieden, da diese von vielen Betriebssystemen und Distributionen unterstützt wird.

Bei der Suche wird ein Verzeichnisbaum rekursiv durchlaufen, dabei wird die Struktur jedes Ordners betrachtet und mit vorgegebenen Strukturen von den unterstützten Webanwendungen verglichen. Dabei wird der Prozentsatz der Übereinstimmung berechnet. Übersteigt die Übereinstimmung einen Schwellenwert, wird angenommen, dass es sich dabei um die entsprechende Webanwendung handelt. Anschließend können weitere Informationen über die Webanwendung ermittelt werden, wie z.B. die eingesetzte Version oder installierte Plugins.

Die Ausgabe von *webapp\_discover* zeigt dann die gefundenen Webanwendungen im JSON-Format an. Eine Beispiel einer solchen Ausgabe ist im Listing 5.8 zu finden. Dort wurde jeweils eine Installation der Webanwendung *Wordpress* und *Joomla* gefunden. Anhand der Parameter können weitere Informationen entnommen werden. Der Wert von `path` gibt dabei den Fundort der Installation an. Der Parameter `score` enthält die Übereinstimmung der Verzeichnisstruktur. Während bei einem Wert von 1, die komplette erwartete Verzeichnisstruktur existierte, wurde bei einem Wert von 0 keinerlei Übereinstimmung gefunden. Die Ausgabe erfolgt jedoch nur, wenn der Wert von `score` größer ist als der Schwellenwert, welcher standardmäßig 0,7 beträgt.

```
{
  "webapps": [
    {
      "path": "/var/www/wordpress",
      "version": "3.7.1",
      "name": "Wordpress",
      "score": 0.98333333333333328,
      "plugins": [
        {
          "version": "2.5.9",
          "name": "akismet"
        }
      ]
    }
  ]
}
```

```

    },
    {
      "path": "/var/www/joomla_alt",
      "version": "1.0.15",
      "name": "Joomla",
      "score": 0.96875,
      "plugins": null
    }
  ]
}

```

Listing 5.8: Ausgabe der Suchergebnisse von `webapp_discover`

Von `webapp_discover` werden standardmäßig die Webanwendungen *Drupal*, *Joomla*, *php-MyAdmin*, *Typo3* und *WordPress* erkannt. Zusätzliche Webanwendungen können ohne großen Aufwand zu `webapp_discover` hinzugefügt werden. Die dazu nötigen Schritte werden in Abschnitt 5.2.4 erläutert.

### 5.2.1 Implementierung von `webapp_discover`

Da die Implementierung objektorientiert erfolgen sollte, wurde zuerst die Problemstellung durch folgende Klassen modelliert:

- **Explorer**

Die Klasse `Explorer` ist die zentrale Komponente von `webapp_discover`. Dort werden nach dem Start die Definitionen für Webanwendungen (Unterklassen von `WebApp`) geladen. Anschließend wird der zu untersuchende Verzeichnisbaum rekursiv durchlaufen und für jedes Verzeichnis die Übereinstimmung mit den Profilen verglichen.

Der Quellcode der Klasse ist im Listing 5.9 in Auszügen abgedruckt. Die Methode `__init_webapps()` wird bei Instanziierung der Klasse aufgerufen und liest die Definitionen der Webapplikationen, welche sich im Ordner `webapps` befinden müssen. Diese Definitionen werden in Klasse festgelegt, welche Nachfahre der Klasse `WebApp` sein muss.

Der eigentliche Suchvorgang wird in der Generator-Methode `detect(path, level, ratio)` durchgeführt. Als Argumente erhält die Methode den Pfad `path`, welcher durchsucht werden soll, die Tiefe der Rekursion `level` und den Schwellenwert `ratio`, der angibt ab wann eine Webanwendung als erkannt gilt. Anschließend werden für jedes Verzeichnis, welches bis zur angegebenen Tiefe liegt, alle definierten Webanwendungen getestet. Überschreitet das Ergebnis den Schwellenwert, werden die Funktionen zur Feststellung von Version und installierten Plugins ausgeführt und die entsprechenden Informationen durch den Generator zurückgegeben.

- **FileTree**

Die `FileTree`-Klasse modelliert die Verzeichnis- und Dateistruktur einer Webanwendung. Sie liest dazu die existierenden Pfade aus einer Liste und wandelt diese in ein hierarchisch aufgebautes Dictionary um. In der Instanz-Methode `check(path, ratio)` ist der Algorithmus, welcher ein real existierende Verzeichnisstruktur mit dem Dictionary vergleicht, implementiert. Diese Methode wird durch die `Explorer`-Klasse beim Durchsuchen benutzt.

- **WebApp**

Die abstrakte Klasse `WebApp` entspricht einer Webanwendung. Sie vererbt den groben Rahmen an die Unterklassen, welche dann die Definitionen der unterstützten Webanwendungen enthalten. Diese Klasse definiert die Stumpf-Methoden `get_version()` und `get_plugins()`, welche in den Unterklassen dann die Versionsnummer und eingesetzte Plugins ermitteln soll.

- **PhpWebApp**

Die abstrakte Klasse `PhpWebApp` ist eine Unterklasse von `WebApp` und soll für alle PHP-Webanwendungen benutzt werden. Diese Klasse implementiert Methoden, die über einen PHP-Interpreter PHP-Code auswerten können. Das dient dazu Konfigurationsdateien von Webanwendungen zu lesen, welche im PHP-Format verfasst sind.

```
class Explorer(object):

    webapps_directory = './webapps'
    [...]

    # Lade Definitionen von Webapps
    def __init_webapps(self):

        # Return value
        webapp_classes = []

        # Root dir of module
        root_dir = os.path.dirname(os.path.realpath(__file__))

        # Verzeichnis für Module
        webapps_dir = os.path.abspath(
            os.path.join(
                root_dir,
                Explorer.webapps_directory,
            )
        )

        # Lese Definitionen für Webapp-Module
        for f in os.listdir(os.path.abspath(webapps_dir)):

            # Trenne in Name und Erweiterung
            module_name, ext = os.path.splitext(f)

            # Nur *.py Dateien als Module lesen
            if ext == '.py' and not module_name.startswith("__"):
                # Lese Modul
                module_classes = pyclbr.readmodule(
                    module_name,
                    path=[webapps_dir]
                )

                # Schleife über Klassen
                for module_class in module_classes.keys():
                    # Prüfe ob Klasse vom entsprechende Modul ist
                    if module_classes[module_class].module == module_name:
```

```

        # Lade Modul
        module_webapps = __import__(
            'webapp_discover.webapps',
            fromlist=[module_name],
            level=0
        )

        # Hole Modul-Instanz
        module_instance = getattr(module_webapps, module_name
            )

        # Hole Klassen-Instanz
        class_instance = getattr(module_instance,
            module_class)

        # Prüfe ob die jeweilige Klasse Unterklasse von
        # Webapp ist
        if isinstance(class_instance, WebApp):
            webapp_classes.append(class_instance())

    return webapp_classes

# Durchsuche Pfad path bis zur Tiefe level mit dem Schwellenwert ratio
def detect(self, path, level, ratio):

    # Schleife über Verzeichnisse
    for root in Explorer.walk_dirs(path, level):
        for webapp in self.webapps:
            # Schleife über Webanwendungen
            val = webapp.webapp_filetree.check(root)

            # Nur wenn Schwellenwert überschritten
            if val >= ratio:
                yield {
                    'path': root,
                    'name': webapp.webapp_name,
                    # Bestimme Version
                    'version': webapp.get_version(root),
                    # Finde Plugins der Webapp
                    'plugins': webapp.get_plugins(root),
                    'score': val
                }

[...]
```

Listing 5.9: Implementierung der Klasse Explorer (in Auszügen)

Nachdem die vorgestellten Klassen implementiert worden waren, wurden die von der `PhpWebApp`-Klasse abgeleiteten Definitionen der Webanwendungen erstellt. Dabei wurden die Webanwendungen *Drupal*, *Joomla*, *phpMyAdmin*, *Typo3* und *Wordpress* implementiert. Das entsprechende Vorgehen bei der Entwicklung der einzelnen Definitionen wird in Abschnitt 5.2.4 am Beispiel der Webanwendung *Drupal* erläutert.

Mit dem Testing-Framework *nose* wurden für die Klassen `Explorer` und `FileTree` entsprechende Tests implementiert, welche die Ergebnisse einzelner Methoden mit den erwarteten vergleichen. Damit Probleme mit der lokalen Python-Installation ausgeschlossen werden können, sollte ein Testlauf durchgeführt werden. Dieser Verlauf zeigt, dass alle Test erfolgreich abgeschlossen wurden: [Nos14]

```
# Tests durchführen
> nosetests -w test
.....
-----
Ran 12 tests in 0.143s

OK
```

Der Code dieser Implementierung steht unter der *GNU GPL v3* (vgl. [GPLv3]) und ist damit Open-Source. Das Projekt kann unter der Adresse [https://github.com/simonswine/webapp\\_discover](https://github.com/simonswine/webapp_discover) eingesehen und heruntergeladen werden.

### 5.2.2 Durchführung eines Scans beim LRZ

Um die Suche nach Webanwendungen unter realistischen Bedingungen zu testen, wurde die Verzeichnisstruktur des Webhostings am LRZ untersucht. Die eigens dafür eingerichtete VM `websec.dev.lrz.de` erhielt lesenden Zugriff auf das NFS-Dateisystem, das die durch Kunden des LRZs installierten Webanwendungen enthält (vgl. Abschnitt 3.1.2).

Diese Webanwendungen befinden sich je nach Zugehörigkeit des entsprechenden Kunden in einem der Verzeichnisse `/nfs/web_lmu/www`, `/nfs/web_tum/www` oder `/nfs/web_mwn/www`. Nachdem die neuste Version von `webapp_discover` installiert worden war, wurden durch eine Schleife die drei Verzeichnisse gescannt. Die standardmäßig verwendete maximale Suchtiefe von 4 wurde dabei nach und nach erhöht und die Anzahl der Ergebnisse verglichen. Bei der Erhöhung der Suchtiefe von 6 auf 7 wurde kein signifikanter Anstieg der Suchergebnisse wie zuvor festgestellt. Daher wurde dann der finale Suchlauf mit der Tiefe 7 durchgeführt (vgl. Listing 5.10). Dieser Suchlauf dauerte dann rund 15 Minuten. Der Schwellenwert bei dessen Überschreitung eine Webanwendung als erkannt gilt, wurde beim Standardwert von 0.7 belassen, da diese Einstellung plausible Resultate lieferte.

Die Ergebnisse aus dem finalen Suchlauf, werden im nachfolgenden Abschnitt 5.2.3 ausgewertet und visualisiert.

```
# Installiere Programmcode in Home-Verzeichnis
> git clone https://github.com/simonswine/webapp_discover.git ~/webapp_discover

# Durchlauf in den entsprechende Verzeichnissen mit einer max. Tiefe von 7
> for dir in mwn lmu tum; do
  ~/webapp_discover/webapp_discover.py \
    --depth 7 \
    /nfs/web_${dir}/www \
  > ~/report_${dir}_depth7.json
done
Running discovery in /nfs/web_mwn/www
Running discovery in /nfs/web_lmu/www
Running discovery in /nfs/web_tum/www
```

Listing 5.10: Installation und Ausführung von `webapp_discover`

### 5.2.3 Statistiken zum Webhosting am LRZ

Um die Ergebnisse des finalen Suchlaufes aus dem vorherigen Abschnitt auswerten zu können, wurde `webapp_discover` mit dem Skript `create_report_latex.py` erweitert. Dieses Skript fasst die Resultate aus den drei Durchläufen von `webapp_discover` zusammen und erstellt den Quellcode für die Diagramme, welche in den entsprechenden Abbildungen gezeigt werden.

Wie bereits in Abschnitt 3.1.2 besprochen, wurden beim Suchlauf insgesamt 272 Installationen von Webanwendungen entdeckt. Betrachtet man in Abbildung 5.2 die Verteilung nach Webanwendung. Die beim LRZ am häufigsten eingesetzte Webanwendung ist demnach *Joomla* gefolgt von *Drupal* und *Typo3*. Die Abbildungen 5.3 bis 5.7 zeigen die Verteilung der Versionen für jede Webanwendung. Während sich Major-Versionen durch verschiedene Grundfarben unterscheiden, wird bei den Minor-Versionen nur nach der Sättigung der Farbe unterschieden.

Da aus den Informationen zur konkreten Version von Webanwendungen nicht sonderlich viel Aussage darüber getroffen werden kann wie aktuell die Webanwendungen sind, wurde das Release-Datum der jeweiligen Versionen ermittelt. Als Quelle dienten dabei meist die Versionsverwaltung-Systeme der Webanwendungen. War darüber keine Aussage möglich, wurden entsprechende Informationen aus den Changelog-Dateien gewonnen. Darüber konnte das Alter der entsprechenden Installationen festgestellt werden, welches angibt wie lange das Release der eingesetzten Version zurückliegt.

Das durchschnittliche Alter der 272 Webanwendungen ist nach diesen Untersuchungen 2,5 Jahre. Die genaue Verteilung der Altersstruktur ist in Abbildung 5.8 gezeigt. Diese Untersuchung offenbart, dass ein großer Anteil der Webanwendungen nicht regelmäßig aktualisiert wird. 87 Webanwendungen sind älter als 3 Jahre. Die entsprechenden Kunden sollten wie in Abschnitt 4.7.2 vorgeschlagen über eine Benachrichtigung zu einer Aktualisierung aufgefordert werden.

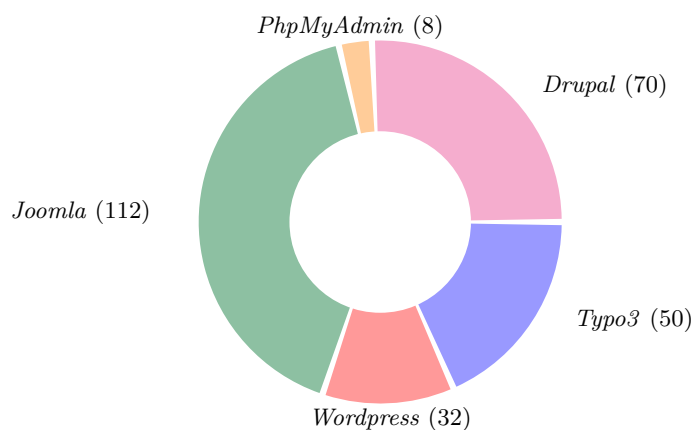


Abbildung 5.2: Verteilung von erkannten Webanwendungen beim Webhosting am LRZ



5.2 Installierte Webanwendungen detektieren mit *webapp\_discover*

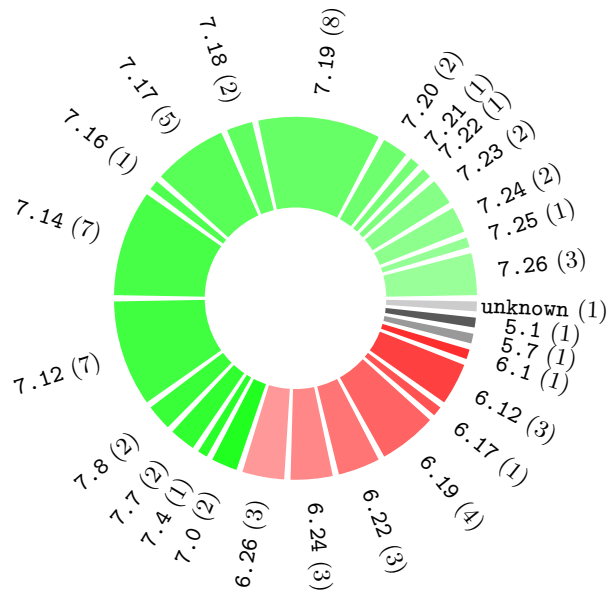


Abbildung 5.3: Versionsverteilung bei *Drupal*

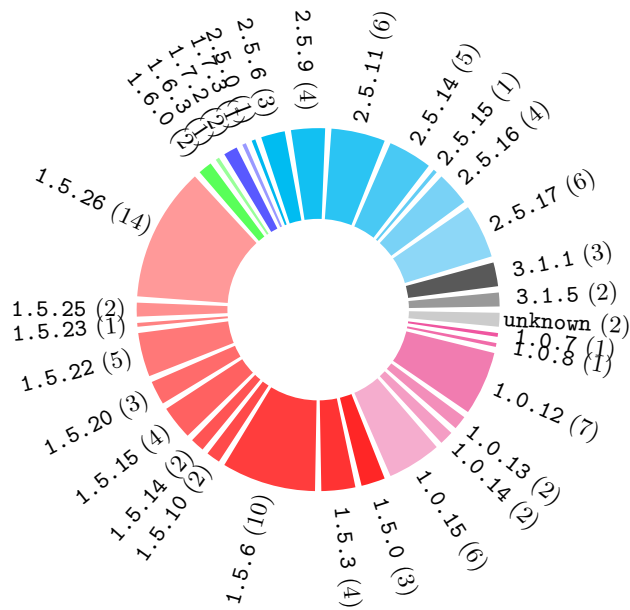


Abbildung 5.4: Versionsverteilung bei *Joomla*

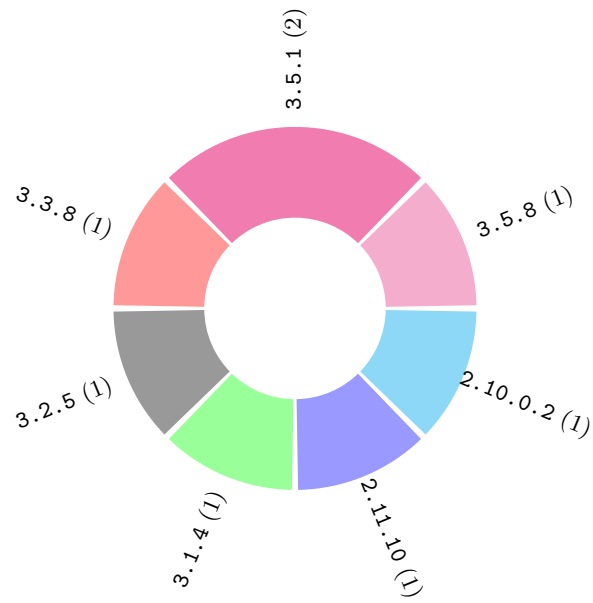


Abbildung 5.5: Versionsverteilung bei *phpMyAdmin*

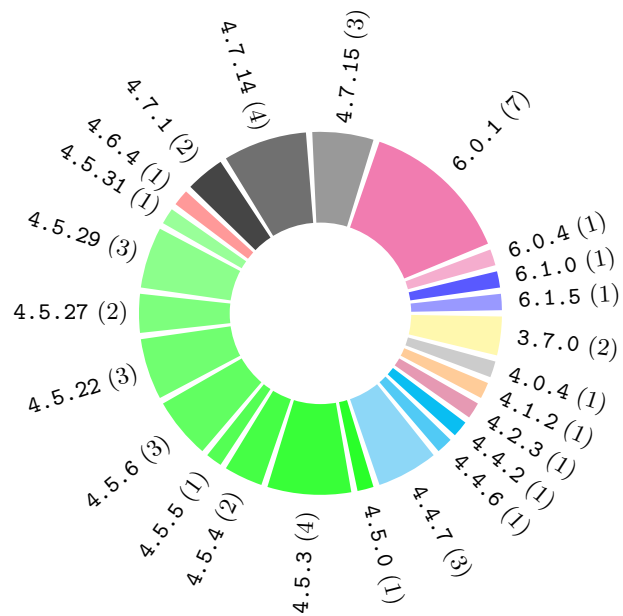


Abbildung 5.6: Versionsverteilung bei *TYPO3*

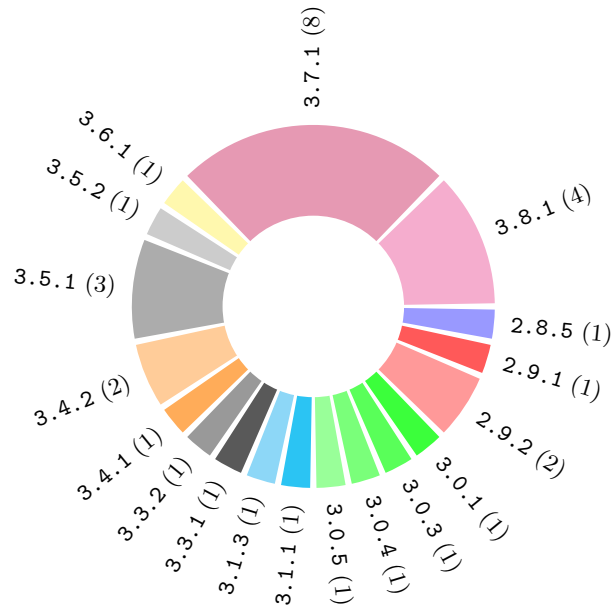


Abbildung 5.7: Versionsverteilung bei *WordPress*

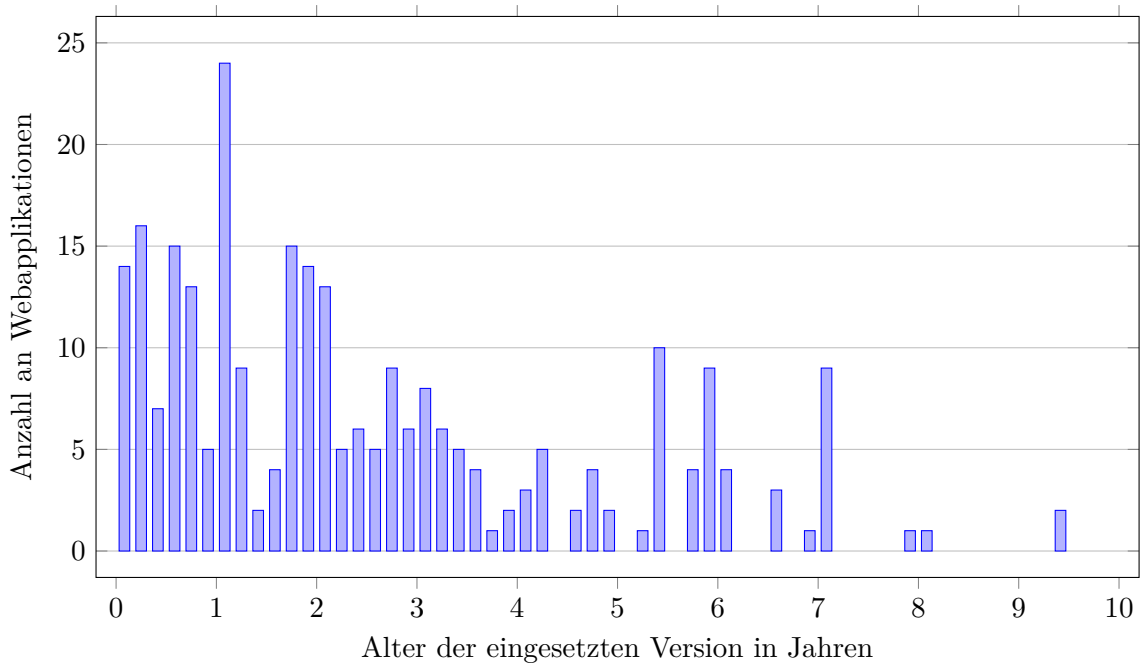


Abbildung 5.8: Überblick über das Alter der jeweils eingesetzten Webapplikations Version

### 5.2.4 Erzeugen der Definition einer Webanwendung

Dieser Abschnitt zeigt die Erstellung einer Definition für Webanwendungen bei `webapp_discover`. Es wird das Vorgehen für *Drupal* gezeigt. Eine Definition für andere Webanwendungen kann analog dazu erstellt werden.

Als Basis für eine Definition sollte die Datei `webapps/example.py.ex` dienen. Dazu wird die Datei in den Pfad `webapps/drupal.py` kopiert. Anschließend muss der Name der Klasse von `ExampleWebApp` in `DrupalWebApp` umbenannt werden. Ebenso sollte der Name der Webapplikation in die Klassen-Variable `webapp_name` eingetragen werden.

Daraufhin ist nötig, eine für die Webanwendung charakteristische Verzeichnisstruktur festzustellen. Dazu wird die Verzeichnisstruktur der Webanwendung in mindestens zwei Versionen untersucht. Die Versionen sollte dabei einige Zeit auseinander liegen. Im Falle von *Drupal* wurden die Versionen 4.0 und 7.26 in lokale Verzeichnisse installiert. Der Vergleich der beiden Strukturen wird mit dem dazu erstellten Hilfsprogramm `tools/find_matching_files.py` durchgeführt. Das Programm erhält die Pfade zu mindestens zwei Verzeichnissen als Parameter. Beim Aufruf vergleicht es die angegebenen Pfade und gibt diejenigen Dateien und Verzeichnisse zurück, die in allen Pfaden vorkommen. Diese Ausgabe entspricht der für eine Webanwendung charakteristischen Verzeichnisstruktur und wird daher als Liste in der Klassen-Variable `webapp_files` abgespeichert. Die für *Drupal* ermittelte Struktur ist in der kompletten Definition der Webanwendung im Listing 5.11 zu finden.

Damit die verwendete Version von *Drupal* durch `webapp_discover` ermittelt werden kann, muss in der Klasse `DrupalWebApp` die Methode `get_version()` überschrieben werden. Bei *Drupal* kann die Version bei neueren Installationen (Version  $\geq 6.0$ ) durch das Parsen von bestimmten Quelltext-Dateien erkannt werden. Bei den älteren Versionen ist es nötig, dass die Versionsinformation aus dem neusten Eintrag des Changelogs ermittelt wird. Bei beiden Varianten wird mit einem regulären Ausdruck die verwendete Version aus den jeweiligen Dateien gewonnen.

Auf eine Suche nach installierten *Drupal*-Modulen wurde aufgrund des hierfür nötigen Aufwandes bei der prototypischen Implementierung verzichtet.

```
# Importiere nötige Module
from webapp_discover.php_webapp import PhpWebApp
import re
import os

## Reguläre Ausdrücke zur Ermittlung der Version

# aus Code für Drupal >= 6.0
RE_VERSION = re.compile("""["']VERSION["']\s*,\s*["'](\d+(\.\d+)+)["']""")

# aus Changelog für Drupal < 6.0
RE_VERSION_CHANGELOG = re.compile("""drupal\s+(\d+(\.\d+)+),""")

# Klasse für Drupal -> Unterklasse von PhpWebApp
class DrupalWebApp(PhpWebApp):

    # Name der Webanwendung
    webapp_name = "Drupal"
```

```

# Verzeichnisstruktur (Versionen 4.0 und 7.26)
webapp_files = [
    '.htaccess',
    'cron.php',
    'includes/common.inc',
    'includes/module.inc',
    'includes/theme.inc',
    'includes/xmlrpc.inc',
    'includes/xmlrpcs.inc',
    'index.php',
    'misc/druplicon.png',
    'scripts/code-clean.sh',
    'scripts/cron-lynx.sh',
    'update.php',
    'xmlrpc.php'
]

def get_version(self, path):

    ## Version aus Code, Drupal >= 6.0
    # Versuche zwei Dateien
    for conf in [
        'includes/bootstrap.inc',
        'modules/system/system.module',
    ]:
        # Pfad zur entsprechenden Datei
        conf_path = os.path.join(path, conf)

        # Nur wenn Datei existiert
        if os.path.exists(conf_path):
            # Lese Inhalt
            cont = open(conf_path).read()
            # Wende regulären Ausdruck an
            m = RE_VERSION.search(cont)
            # Wenn Version gefunden, diese zurückgeben
            if m is not None:
                return (m.group(1))

    ## Version aus Changelog, Drupal < 6.0
    conf_path = os.path.join(path, 'CHANGELOG')
    # Pfad des Changelogs
    if os.path.exists(conf_path):
        # Lese Inhalt
        cont = open(conf_path).read()
        # Wende regulären Ausdruck an
        m = RE_VERSION_CHANGELOG.search(cont)
        # Wenn Version gefunden, diese zurückgeben
        if m is not None:
            return (m.group(1))

```

Listing 5.11: Definition der Webanwendung *Drupal*

Das hier vorgestellte Vorgehen bei *Drupal* wurde in ähnlicher Art und Weise für die Webanwendungen *Joomla*, *phpMyAdmin*, *Typo3* und *Wordpress* wiederholt. Für *Typo3* und *Wordpress* konnte ein Algorithmus gefunden werden, welcher aktivierte bzw. installierte Module ermitteln kann. Dieser ist in den Definitionen im Quelltext von `webapp_discover` zu finden.

### 5.3 „Betreute Webanwendungen“ mit *Docker*

Dieser Abschnitt befasst sich mit der Container-Virtualisierung durch *Docker* (vgl. Abschnitt 4.2.3) und wie man als Provider damit das Produkt „Betreute Webanwendungen“ anbieten kann. Dazu werden *Docker*-Images zuerst für den LAMP-Stack und darauf aufbauend für *WordPress* erstellt.

Die im Rahmen dieses Abschnittes erstellten Images und Container sind in Abbildung 5.9 abgebildet. Images sind dort rechteckig und Container oval dargestellt. Die Pfeile geben an welche Abhängigkeiten ein Container bzw. Image hat. Lediglich das sog. Base-Image `leibnizrz/wheezy-base:import` hat keinerlei Abhängigkeiten. Während die fertigen Images über die Registry von *Docker* (<https://index.docker.io/u/leibnizrz/>) herunterladbar sind, stehen deren zugehörige Dockerfiles samt Skripte unter Versionierung und sind auf *GitHub* (<https://github.com/simonswine/docker-leibnizrz-images>) zu finden.

Zuerst wurde versucht, als Betriebssystem das am LRZ übliche *SuSE Linux Enterprise 11* zu verwenden. Jedoch war darauf *Docker* nicht lauffähig. Aus diesem Grund wurde schließlich *Ubuntu Linux* eingesetzt. Auch wurde versucht *SuSE Linux Enterprise 11* als Base-Image einzusetzen, jedoch machte hier die nötige Registrierung, bevor das Paketmanagement benutzt werden konnte, Probleme beim automatisierten Setup. Daher wurde stattdessen *Debian Wheezy* als Base-Image verwendet.

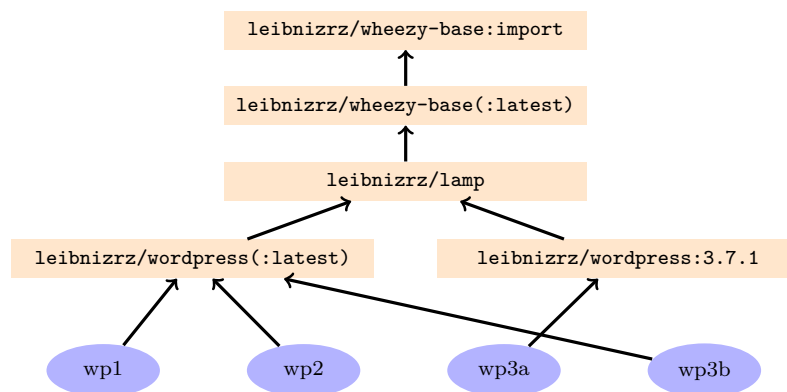


Abbildung 5.9: Abhängigkeiten der *Docker*-Images und Container

#### 5.3.1 *Docker* installieren

Für die Installation von *Docker* empfiehlt sich der Einsatz der Linux-Distribution *Ubuntu Linux*. Ab Version 12.04 werden alle nötigen Komponenten unterstützt. Die Installation selbst wird entsprechend der Empfehlungen der Dokumentation von *Docker* durchgeführt. Die nötigen Schritte wurden in einer VM mit *Ubuntu Linux 12.04* durchgeführt:

```

# Kernel > 3.2 wird installiert
> apt-get install linux-image-generic-lts-raring linux-headers-generic-lts-raring
# Reboot (!!) der VM
  
```

```

> reboot

# Docker Paketquellen hinzufügen
> apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 36
  A1D7869245C8950F966E92D8576A8BA88D21E9
> echo "deb http://get.docker.io/ubuntu docker main" > /etc/apt/sources.list.
  d/docker.list
> apt-get update

# Docker installieren
> apt-get install lxc-docker

```

Listing 5.12: Installation von *Docker* auf *Ubuntu Linux 12.04* (vgl. [Dock14c])

### 5.3.2 Erstellung eines Base-Images für *Debian Wheezy*

In diesem Abschnitt soll ein Basis-Image für *Debian Wheezy* erstellt werden. Dazu installiert man zuerst ein minimales Basissystem mit Hilfe des Tools *debootstrap* in den Unterordner *wheezy\_minimal*. Anschließend wird aus den Dateien und Verzeichnissen mittels *docker import* das Basis *Docker*-Image erstellt und im Repository *leibnizrz/wheezy-base* unter dem Tag *import* abgespeichert. [Dock14b]

```

# Mit debootstrap ein minimales debian wheezy erstellen
> debootstrap \
  wheezy \                               # Version von Debian
  wheezy_minimal \                       # Zielverzeichnis
  http://ftp.de.debian.org/debian/      # Packet-Mirror

# Debian Wheezy importieren in Docker
> tar -C wheezy_minimal -c . |
  docker import \                         # Docker-Befehl für Import
  - \                                     # Quelle Standard-Input
  leibnizrz/wheezy-base:import           # Repository und Tag

# Neues Image anzeigen lassen
> docker images
REPOSITORY          TAG          IMAGE ID
leibnizrz/wheezy-base  import      ae0099bc2ba0

```

Listing 5.13: Erstellung eines minimalen *Debian Wheezy* Chroots

Da nun ein Basis-Image für *Debian Wheezy* existiert, können auf diesem aufbauend mittels *Dockerfile* weitere Images erzeugt werden. Die Möglichkeit wird genutzt, um das eben erstellte Basis-Image mit häufig verwendeten Anwendungen zu erweitern und verfügbare Aktualisierungen einzuspielen. Ebenso können über dieses *Dockerfile* generell gültige Einstellungen wie z.B. die Zeitzone eingestellt werden. Die einzelnen Anweisungen finden sich im *Dockerfile* in Listing 5.14:

```

# Parent-Image festlegen
FROM leibnizrz/wheezy-base:import
MAINTAINER Christian Simon <simonc@campus.lmu.de>

```

```

# Füge Debian Paketquellen hinzu
ADD ./sources.list /etc/apt/sources.list
# Update der Paketlisten
RUN apt-get update
# Upgrade des Basissystems
RUN apt-get -y upgrade && apt-get clean

# Installiere Tools
RUN apt-get -y install vim git procs psmisc wget supervisor nano pwgen
    tzdata rsync && apt-get clean

# Setze deutsche Zeitzone
RUN echo "Europe/Berlin" > /etc/timezone && dpkg-reconfigure -f
    noninteractive tzdata

```

Listing 5.14: Dockerfile für leibnizrz/wheezy-base(:latest)

Um die Anweisungen des Dockerfiles nun anzuwenden und das entsprechende Image zu bauen, muss das Kommando `docker build` folgendermaßen ausgeführt werden:

```

# Baue Dockerfile in Verzeichnis ./wheezy-base
> docker build \
    -t leibnizrz/wheezy-base:latest \ # Repository für erfolgreichen Build
    wheezy-base/                    # Pfad zum Dockerfile
[...]
# Kontrolle ob das Image existiert
> docker images
REPOSITORY          TAG          IMAGE ID
leibnizrz/wheezy-base latest      415a8a9eff8c
leibnizrz/wheezy-base import      ae0099bc2ba0

```

Listing 5.15: Dockerfile für leibnizrz/wheezy-base(:latest) bauen

### 5.3.3 Erstellung eines Images für den LAMP-Stack

Dieser Abschnitt installiert auf dem grundlegenden Minimal-Image für *Debian Wheezy* einen kompletten LAMP-Stack (**L**inux, **A**pache, **M**ySQL und **P**HP vgl. Abschnitt 2.2.3). Das entsprechende Dockerfile ist im Listing 5.16 in ganzer Länge abgedruckt. Die Aktionen des Dockerfiles werden jetzt in den nächsten Absätzen besprochen:

Zuerst wird das Verzeichnis `./docker/` angelegt, in welchem Daten abgelegt werden, die zum Management des jeweiligen Containers dienen. Das Unterverzeichnis `data` ist dabei Ablage für Informationen, die den Zustand eines Container beschreiben. Der derzeit einzige dort abgespeicherte Zustandswert besagt, ob der jeweilige Container bereits initialisiert wurde. Eine bereits durchgeführte Initialisierung wird durch Existenz der Datei `./docker/data/.docker_init` angezeigt.

Ob eine Initialisierung notwendig ist, überprüft das standardmäßig ausgeführte Start-Skript `./docker/run.sh` (vgl. Kommando `CMD` im Dockerfile 5.16). Falls eine Initialisierung nötig ist, führt es dazu die Skripte im Verzeichnis `./docker/init/` aus. Zum Abschluss startet `run.sh` den Verwaltungsdaemon `supervisord`. Dieser kümmert sich um die Verwaltung der für den Betrieb des Containers notwendigen Dienste. Das ist nötig, da *Docker* darauf verzichtet ein sonst bei Linux übliches Init-System zu verwenden. Die Konfigurationen, der



zu startenden Dienste werden durch das Dockerfile entsprechend den Anforderungen im Verzeichnis `/etc/supervisor/conf.d/` abgelegt.

Anschließend werden die notwendigen Pakete installiert, damit der Webserver *Apache 2.2* dynamische Seiten mittels *PHP* ausliefern kann. Die Integration wird durch Einsatz des Apache-Moduls *Mod-PHP* ermöglicht (vgl. 2.3.1). Die Standard-Konfiguration von Apache wird durch Deaktivierung der *CGI*-Funktionalität abgeändert, da diese für den Einsatz von *PHP* nicht benötigt wird. Der Webroot des Webserver im Pfad `/var/www` wird als *Docker*-Volume definiert, da es sich dabei um ein Datenverzeichnis handelt.

Daraufhin wird ein *MySQL*-Datenbankserver eingerichtet. Nach der Installation ist die Datenbank ohne entsprechendes Passwort geschützt, daher ist für *MySQL* eine Initialisierung notwendig, welche ein sicheres Passwort für den administrativen Benutzer `root` vergibt. Dieser Vorgang wird durch das Skript `/.docker/init/050-mysql` beim erstmaligen Start eines solchen Containers vorgenommen. Das Prefix `050` legt dabei die Ausführungsreihenfolge der Skripte bei der Initialisierung fest, da die Skripte nach lexigraphischer Ordnung ausgeführt werden. Als GUI für *MySQL*-Operationen wird zusätzlich die Webapplikation *phpMyAdmin* installiert, welche ebenfalls eine Initialisierung benötigt, da zum Betrieb eine entsprechende Datenbank samt gültigen Zugängen auf dem *MySQL*-Server eingerichtet sein müssen. Diese Aufgabe wird durch das Skript `/.docker/init/025-phpmyadmin` durchgeführt. Das niedrigere Prefix von `025` sorgt dafür, dass dies noch vor Änderung des *MySQL*-root Passwortes geschieht.

Aus diesem Dockerfile kann wieder mittels `docker build` ein Image generiert werden. Es wird dann im Repository `leibnizrz/lamp` bereitgestellt.

```
# Parent-Image festlegen
FROM leibnizrz/wheezy-base
MAINTAINER Christian Simon <simonc@campus.lmu.de>

## Docker Verwaltungsverzeichnis
RUN mkdir -p /.docker/ && mkdir -p /.docker/init
# Füge Startskript hinzu
ADD run.sh /.docker/run.sh
RUN chmod +x /.docker/run.sh
# Docker Data Volume
VOLUME ["/.docker/data"]

## Apache & PHP
# Installiere Apache und PHP 5.4
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install mysql-client apache2
  libapache2-mod-php5 php5-mysql php5-gd php5-mcrypt && apt-get clean
# Apache Startskript
ADD ./apache.foreground.sh /etc/apache2/foreground.sh
# Apache Startkonfig
ADD ./supervisord.apache.conf /etc/supervisor/conf.d/apache.conf
# Apache Standard vHost Konfiguration
ADD ./apache.sites.default.conf /etc/apache2/sites-available/default
# Webroot als Volume
VOLUME ["/var/www"]

## MySQL
# Füge MySQL User/Group hinzu um stabile uid/gid zu erhalten
RUN addgroup --gid 200 mysql
```

```

RUN adduser --gid 200 --uid 200 mysql
# Installiere MySQL und phpMyAdmin
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install mysql-server phpmyadmin
  && apt-get clean
# Volume für MySQL
VOLUME ["/var/lib/mysql"]
# MySQL Startkonfig
ADD ./supervisord.mysql.conf /etc/supervisor/conf.d/mysql.conf
# MySQL Initialisierung (Ändert root Passwort)
ADD init_050-mysql.sh /.docker/init/050-mysql
RUN chmod +x /.docker/init/050-mysql
# Entferne bereits bestehende MySQL Datenbanken
RUN rm -rf /var/lib/mysql/*
# Aktiviere phpMyAdmin
RUN echo "phpmyadmin phpmyadmin/reconfigure-webserver multiselect apache2" |
  debconf-set-selections -v && dpkg-reconfigure -f noninteractive
  phpmyadmin
# phpMyAdmin Initialisierung (Ändert MySQL-Passwort / legt Datenbankschema an
)
ADD init_025-phpmyadmin.sh /.docker/init/025-phpmyadmin
RUN chmod +x /.docker/init/025-phpmyadmin

# Port 80 soll veröffentlicht werden
EXPOSE 80

# Standardmäßig run.sh ausführen
CMD ["/.docker/run.sh"]

```

Listing 5.16: Dockerfile für leibnizrz/lamp

### 5.3.4 Erstellung des *Wordpress* Images

In diesem Abschnitt wird nun gezeigt, wie auf dem zuvor erstellten LAMP-Stack-Image die Webapplikation *Wordpress* hinzugefügt werden kann. Die dazu notwendigen Aktionen sind im Listing 5.17 abgedruckt. Diese Aktionen können für die meisten Webapplikationen, welche den LAMP-Stack benutzen, analog verwendet werden:

Zuerst werden die notwendigen Ordner und Dateien der Webapplikation in einem Ordner außerhalb des Webroots installiert. Für *Wordpress* wird der Ordner `/usr/local/share/wordpress` gewählt. Dort wird die zum Zeitpunkt der Arbeit neuste Version 3.8.1 mit deutschen Sprachanpassungen installiert. Die installierten Ordner und Dateien werden für den `root`-Benutzer beschreibbar gemacht, während alle anderen Benutzer Lese-Rechte erhalten. Ein zusätzlicher Link der Konfigurationsdatei von Wordpress in das Webroot-Verzeichnis `/var/www` ist nötig, damit jeder Container eine eigenständige Konfiguration erhält.

Anschließend werden für *Wordpress* notwendige Konfigurationen an dem Container vorgenommen. Während alle obligatorischen Pakete bereits im Image `leibnizrz/lamp` installiert wurden, muss das Apache Module *Mod-Rewrite* noch aktiviert werden, damit die Anforderungen von *Wordpress* erfüllt sind. [Wor14]

Zuletzt wird noch das Skript zur Initialisierung hinzugefügt. Während der Ausführung des Skriptes wird die für *Wordpress* notwendige Datenbank und passende Zugänge eingerichtet. Anschließend werden die notwendigen Dateien und Verzeichnisse der Webapplikation aus `/usr/local/share/wordpress` mittels symbolischen Links in das Webroot referenziert. Lediglich das Verzeichnis mit den individuellen Daten je *Wordpress*-Instanz `wp-content` wird in

das Webroot kopiert und mit Schreib-Rechten für *PHP* versehen. Während die Dateien und Verzeichnisse, die mit symbolischen Links eingebunden werden, durch ein Update des Images verändert werden können, bleiben die Daten im `wp-content`-Ordner unangetastet. Zuletzt legt das Skript noch die Konfigurationsdatei `wp-config.php` in das Webroot. Dort werden automatisiert individuelle Parameter eingetragen, wie z.B. Nutzer und Passwort für die Datenbank und Parameter für ein sicheres Hashing der Passwortdatenbank von *WordPress*.

Die Operationen des Dockerfiles werden wiederum mittels `docker build` ausgeführt und das entsprechende Image als Standard im Repository `leibnizrz/wordpress` abgespeichert. Damit im weiteren Verlauf eine Aktualisierung von *WordPress* getestet werden kann, wird das Dockerfile leicht modifiziert, so dass es die veraltete Version 3.7.1 installiert. Auch aus diesem Dockerfile wird ein Image gebaut, welches im Repository `leibnizrz/wordpress:3.7.1` gespeichert wird.

```
# Parent-Image festlegen
FROM leibnizrz/lamp
MAINTAINER Christian Simon <simonc@campus.lmu.de>

## Wordpress
# Download und entpacken von Wordpress 3.8.1 (de_DE)
RUN mkdir -p /usr/local/share/wordpress
RUN wget --progress=dot:mega http://de.wordpress.org/wordpress-3.8.1-de_DE.tar.gz -O - | tar xfz - -C /usr/local/share/wordpress --strip-component=1
# Passe Rechte an
RUN chown -cR root:root /usr/local/share/wordpress
RUN chmod -cR o-w,g-w /usr/local/share/wordpress
# Linke Config Datei
RUN ln -s ../../../../var/www/wp-config.php /usr/local/share/wordpress/wp-config.php
# Aktiviere Apache-Module rewrite
RUN a2enmod rewrite
# Wordpress Initialisierung
ADD init_025-wordpress.sh /.docker/init/025-wordpress
RUN chmod +x /.docker/init/025-wordpress
```

Listing 5.17: Dockerfile für `leibnizrz/wordpress`

### 5.3.5 Veröffentlichung von *Docker*-Images in der Registry

Die gerade erzeugten Images sollen nun in einer Registry abgelegt werden. Durch eine Veröffentlichung in der zentralen *Docker*-Registry können die entsprechenden Images auf jeder *Docker*-Installation heruntergeladen und anschließend ausgeführt werden.

Bei einem produktiven Einsatz von *Docker* ist es natürlich nicht in allen Fällen wünschenswert, dass die zugehörigen Images öffentlich verfügbar sind oder überhaupt auf externen Systemen gespeichert werden. Daher lässt sich eine *Docker*-Registry auch als private Registry auf eigenen Systemen betreiben (vgl. [Dock14f]). So können die Images In-House bei einem Provider vorgehalten werden, und über das interne Netz deutlich schneller übertragen werden. Zudem kann der Provider die Zugriffskontrolle auf die Images entsprechend seiner Bedürfnisse selbst gestalten. Da der Betrieb einer eigenen Registry den Rahmen dieser Arbeit sprengen würde, wurde sich dafür entschieden, die zentrale öffentliche Registry zu nutzen.

Damit die Veröffentlichung dort möglich ist, muss zuerst ein Konto auf der Webseite der

Registry (<https://index.docker.io/>) angelegt werden. Der Kontoname muss dabei dem Präfix der eigenen Repositories entsprechen. Er ist in diesem Fall also `leibnizrz`. Auf der Webseite können weitere Informationen zu den einzelnen Repositories hinterlegt werden, wie z.B. Hilfestellungen und das entsprechende Dockerfile, welches zum Bau verwendet wurde. Außerdem können die Images, welche durch andere Nutzer hochgeladen wurden, durchsucht und bewertet werden.

Um die derzeit nur lokal verfügbaren Images hochzuladen, muss zuerst das Konto mit dem Kommando `docker login` auf der *Docker*-Installation eingerichtet werden. Dann kann mit dem Befehl `docker push`, gefolgt von dem Namen des Repositories, die Eintragung in die Registry ausgelöst werden. Dazu werden zusätzlich alle Images hochgeladen, zu denen eine Abhängigkeit besteht. Dieser Befehl muss für jedes Repository wiederholt werden. Anschließend können die Images auf jeder beliebigen *Docker*-Installation durch den Befehl `docker pull` heruntergeladen werden. Die durchgeführten Vorgänge sind im Listing 5.18 dokumentiert.

```
# Anzeigen der erzeugten Images
> docker images
REPOSITORY          TAG          IMAGE ID
leibnizrz/wordpress 3.7.1       7e52d0e8847f
leibnizrz/wordpress latest      8a2aef8fdcf0
leibnizrz/lamp       latest      69a7f2751147
leibnizrz/wheezy-base latest      415a8a9eff8c
leibnizrz/wheezy-base import      ae0099bc2ba0

# Login in der Registry
> docker login
Username: leibnizrz
Password: <secret>
Login Succeeded

# Push Befehl je Repository
> docker push leibnizrz/wheezy-base
The push refers to a repository [leibnizrz/wheezy-base] (len: 2)
Sending image list
Pushing repository leibnizrz/wheezy-base (2 tags)
[...]
> docker push leibnizrz/lamp
The push refers to a repository [leibnizrz/lamp] (len: 1)
Sending image list
Pushing repository leibnizrz/lamp (1 tags)
[...]
> docker push leibnizrz/wordpress
docker push leibnizrz/wordpress leibnizrz/lamp
The push refers to a repository [leibnizrz/wordpress] (len: 2)
Sending image list
Pushing repository leibnizrz/wordpress (2 tags)
[...]
```

Listing 5.18: Veröffentlichung von *Docker*-Images in der Registry

### 5.3.6 Erzeugung von *Wordpress*-Container

In diesem Abschnitt werden mit Hilfe des `leibnizrz/wordpress`-Images Container erzeugt, die ein Provider an Kunden als betreute *Wordpress*-Installation vermieten kann. Dazu wird zuerst ein Datenverzeichnis angelegt und anschließend mit dem Kommando `docker run` der entsprechende Container erzeugt und gestartet. Beim ersten Startvorgang findet die Initialisierung des Containers statt und daher müssen die dabei erzeugten Passwörter über das Kommando `docker logs` ausgegeben und notiert werden. Im Listing 5.19 werden zwei Container `wp1` und `wp2` mit der neuesten *Wordpress*-Version erzeugt. Dabei wird auch angegeben, dass der Webserver auf Port 80, innerhalb des Containers auf die Ports 8001 bzw. 8002 des *Docker*-Hosts weitergereicht werden soll. Die Datenverzeichnisse befinden sich dabei im Ordner `/kunden` des Hosts. Nach ca. zehn Sekunden kann mit einem Browser die entsprechende *Wordpress*-Instanz erreicht werden. Es müssen dann lediglich noch die finalen Schritte des Installationsassistenten durchgeführt werden (vgl. Abbildung 5.10).

```
## Definiere und erstelle Datenverzeichnis wp1
> export DATA_DIR=/kunden/wp1; mkdir -p ${DATA_DIR}

# Erzeuge Container wp1
> docker run \
  -d \                # Ausführung im Hintergrund
  -p 8001:80 \        # Port 8001 Host -> 80 Container
  --name wp1 \        # Name des Containers
  \                  # Datenverzeichnisse
  -v ${DATA_DIR}/docker-data:/.docker/data \
  -v ${DATA_DIR}/mysql:/var/lib/mysql \
  -v ${DATA_DIR}/www:/var/www \
  leibnizrz/wordpress # Image das verwendet werden soll
[...]                # Kommando gibt die Container-Id aus

# Passwörter für wp1 einsehen
> docker logs wp1
[...]

# Definiere und erstelle Datenverzeichnis wp2
> export DATA_DIR=/kunden/wp2; mkdir -p ${DATA_DIR}

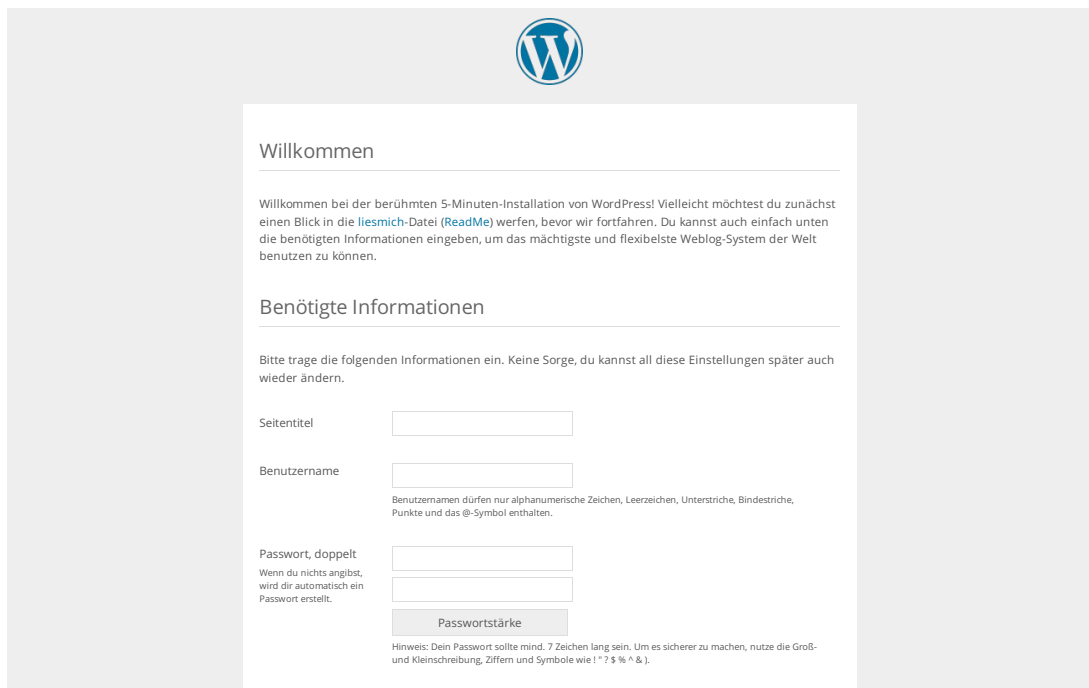
# Erzeuge Container wp2
> docker run -d -p 8002:80 --name wp2 -v ${DATA_DIR}/docker-data:/.docker/
  data -v ${DATA_DIR}/mysql:/var/lib/mysql -v ${DATA_DIR}/www:/var/www
  leibnizrz/wordpress
[...]

# Passwörter für wp2 einsehen
> docker logs wp2
[...]
```

Listing 5.19: Erzeugung von Containern aus den Images

### 5.3.7 Upgrade einer *Wordpress*-Instanz

Jetzt wird noch gezeigt, wie eine Aktualisierung von Webanwendungen mit Hilfe von *Docker* vorgenommen werden kann. Dazu wird zuerst ein neues Datenverzeichnis `/kunden/wp3`

Abbildung 5.10: Installationsassistent von *Wordpress*

angelegt. Der dazugehörige Container `wp3a` wird wie zuvor erzeugt, jedoch ist er nun auf Port 8003 erreichbar und verwendet das Image `leibnizrz/wordpress:3.7.1`. Dieses Image enthält die veraltete *Wordpress*-Version 3.7.1. Nachdem der Container gestartet ist, wird die Webanwendung durch Verwendung des Installationsassistenten grundlegend eingerichtet.

Um nun die Aktualisierung einzuspielen wird der Container `wp3a` beendet und ein Container `wp3b` auf Basis des neuesten *Wordpress* Images erstellt. Dieser verwendet dasselbe Datenverzeichnis und denselben Port wie Container `wp3a`. Nun muss nach Anmeldung im *Wordpress*-Backend noch die Datenbank aktualisiert werden. Sofern das alles erfolgreich war, kann anschließend der Container mit der älteren Version mit dem Kommando `docker rm wp3a` wieder entfernt werden.

Dieses Beispiel zeigt, dass durch *Docker* die Aktualisierung von Webanwendungen durchgeführt werden kann. Es werden dabei die Datenverzeichnisse in einem neu erstellten Container mit aktualisierten Image wiederverwendet.

```
# Definiere und erstelle Datenverzeichnis wp3
> export DATA_DIR=/kunden/wp3; mkdir -p ${DATA_DIR}

# Erstelle Wordpress Container wp3a mit Version 3.7.1
> docker run -d -p 8003:80 --name wp3a -v ${DATA_DIR}/docker-data:/.docker/
  data -v ${DATA_DIR}/mysql:/var/lib/mysql -v ${DATA_DIR}/www:/var/www
  leibnizrz/wordpress:3.7.1
[...]
```

```
# Notiere Passwörter
> docker logs wp3a
```

```
# Führe Installationsassistent aus

# Stoppe Container mit Wordpress 3.7.1
> docker stop wp3a

# Erzeuge Wordpress Container wp3b mit Version 3.8.1
docker run -d -p 8003:80 --name wp3b -v ${DATA_DIR}/docker-data:/.docker/data
-v ${DATA_DIR}/mysql:/var/lib/mysql -v ${DATA_DIR}/www:/var/www
leibnizrz/wordpress

# Login ins Wordpress-Backend und Datenbankupdate ausführen

# Wenn alles in Ordnung: Entferne Container mit Wordpress 3.7.1
> docker rm wp3a
```

Listing 5.20: Upgrade von *Wordpress* durch Austausch des Images





## 6 Fazit und Ausblick

Die in dieser Arbeit betrachteten Konzepte zur Absicherung von Web- und Applikationsservern haben gezeigt, dass ein Internetdiensteanbieter vielfältige Maßnahmen zu treffen hat, um einen möglichst hohen Grad der Sicherheit seiner Systeme zu gewährleisten. In diesem abschließenden Kapitel werden die Ergebnisse der Arbeit und die Schritte zur Umsetzung am LRZ zusammengefasst. Daran schließt ein Ausblick auf mögliche Folgearbeiten an.

### 6.1 Zusammenfassung der Ergebnisse

Aus der Betrachtung von allgemeinen Angriffsvektoren und der konkreten Situation am LRZ wurden 14 Anforderungen abgeleitet, welche nach Hauptzweck der jeweiligen Anforderung in diese Kategorien eingeteilt wurden: Prävention, Detektion von Verwundbarkeiten, Detektion von Angriffen und Reaktion auf Angriffe. Zudem wurde die Relevanz der Anforderungen für die bei Providern gängigen Produkte „Virtuelle Maschinen“, „Betreutes Hosting“ und „Betreute Webanwendungen“ angegeben. Eine Anforderung zur Prävention von Sicherheitsvorfällen ist beispielsweise, dass eine regelmäßige Wartung von Webanwendungen durchgeführt werden muss. Diese Anforderung gilt für die drei betrachteten Produkte von Providern.

Im nächsten Schritt wurden 19 Lösungsansätze ausführlich betrachtet und bezüglich der Anforderungen bewertet. Es wurden dabei sowohl organisatorische als auch technische Maßnahmen betrachtet. Die technischen Lösungsansätze erreichten dabei eine Absicherung von Betriebssystem, Webserver, Applikationsserver und Infrastruktur. Für die Lösungsansätze wurden Vor- und Nachteile diskutiert und deren Eignung für die drei untersuchten Produkte von Providern festgestellt. Um die Anforderung nach regelmäßiger Wartung von Webanwendungen zu erfüllen, kann z.B. für die Produkte „Betreutes Hosting“ und „Betreute Webanwendungen“ die Container-Virtualisierung mit *Docker* verwendet werden.

Aus diesen Lösungsansätzen wurden geeignete Maßnahmen ausgewählt, welche am LRZ umgesetzt werden sollten. Um die Realisierbarkeit der vorgeschlagenen Maßnahmen zu zeigen, wurde eine Auswahl dieser prototypisch implementiert.

### 6.2 Schritte zur Umsetzung am LRZ

Für das LRZ wurde ein Maßnahmenkatalog erstellt, der Lösungsansätze enthält, die dort zur Verbesserung der Sicherheit von Web- und Applikationsserver umgesetzt werden sollten. Dieser Katalog umfasst für die beiden am LRZ angebotenen Produkte und „Virtuelle Maschinen“ und „Betreutes Hosting“ u.a. die folgenden organisatorischen Maßnahmen:

Die Laufzeit der jeweiligen Dienstleistungen sollte zeitlich begrenzt werden, so dass durch Nutzer eine Verlängerung beantragt werden muss. Damit soll verhindert werden, dass das entsprechend vom Kunden gebuchte Produkt in Vergessenheit gerät und so von ihm nicht ausreichend gepflegt wird. Zudem sollte das LRZ seinen Kunden durch Dokumentationen und ggf. Schulungen das nötige Wissen vermitteln, um eine Webanwendung auf den Produkten sicher zu betreiben.

Für das Produkt „Virtuelle Maschinen“ mit der Service-Variante „Attended Hosting“ wurde u.a. empfohlen das Konfigurationsmanagement-Tool *Puppet* einzusetzen. Damit kann die Konfiguration von einer beliebigen Anzahl an VMs zentral in sog. Manifesten definiert werden. *Puppet* vergleicht dann den aktuellen Zustand einer VM mit dem gewünschten Zustand, welcher durch die zugehörigen Manifeste vorgegeben wurde. Treten Diskrepanzen zwischen diesen beiden Zuständen auf, versucht *Puppet* diese durch geeignete Aktionen zu beheben. Durch eine periodische Ausführung von *Puppet* auf den VMs kann sichergestellt werden, dass diese so konfiguriert sind, wie es in den Manifesten festgelegt ist. Im Rahmen der prototypischen Implementierung wurde gezeigt, dass mit *Puppet* die sichere Installation und Konfiguration eines *Apache Tomcat*-Applikationsserver auf einer VM durchgeführt werden kann. Auch beim Produkt „Betreutes Hosting“ sollte *Puppet* zur zentralen Konfiguration der beteiligten Systeme verwendet werden.

Eine weitere Lösung, die dort zum Einsatz kommen sollte, stellt ein Scanner nach potentiell verwundbaren Webanwendungen dar. Dieser untersucht durch die Kunden installierte Webanwendungen nach deren Gefährdung. Da keine bestehende Lösung zum Scannen eines Dateisystemes auf veraltete Webanwendungen ermittelt werden konnte, wurde im Rahmen der Arbeit für dieses Produkt das Skript `webapp_discover` entwickelt. Es kann durch einen Suchlauf in der Verzeichnisstruktur dort installierte Webanwendungen erkennen und gibt deren Typ und Version an. Die Erkennung basiert dabei auf einen Vergleich der existierenden mit vorgegeben charakteristischen Verzeichnisstrukturen für ausgewählte Webanwendungen. In einem Suchlauf beim Webhosting des LRZs konnten mit den fünf im Prototyp definierten Webanwendungen bereits rund ein Drittel der eingesetzten Webanwendungen erkannt werden. Dieser Anteil könnte sicherlich durch das Hinzufügen von weiteren Webanwendungen erhöht werden.

Eine weitere Lösung, die beim Webhosting verwendet werden sollte, stellt *Docker* dar. Damit können einzelne Webanwendungen durch den Einsatz von Container-Virtualisierung voneinander isoliert betrieben werden. Dabei kann für jede einzelne Container-Instanz eine bestimmte Laufzeitumgebung über Images bereitgestellt werden. Diese *Docker*-Images können durch Repositories und Registries auf verschiedene Hosts verteilt und somit gepflegt werden.

Mit *Docker* kann prinzipiell jede unter Linux ausführbare Anwendung betrieben werden, da die Images eine vollständige Linux-Umgebung zur Verfügung stellen. Das ermöglicht Providern beim Webhosting eine breite Auswahl an Laufzeitumgebungen parallel anzubieten. Dadurch können neben der häufig verwendeten *PHP*-Umgebung auch solche für weitere Sprachen wie z.B. *Java*, *Ruby*, *Scala* und *Python* zur Verfügung gestellt werden. Die Lösung *Docker* zeigt, wie möglichst individuelle Laufzeitumgebungen bei Hosting Dienstleistung bereitgestellt werden können.

Für am LRZ häufig eingesetzte Webanwendungen könnte zukünftig mit *Docker* ein entsprechendes Produkt „Betreute Webanwendung“ angeboten werden. Dadurch würde die Zuständigkeit die entsprechenden Webanwendungen zu pflegen, in den Händen des LRZs liegen. Dadurch kann beispielsweise ein rechtzeitiges Durchführen von Aktualisierungen sichergestellt werden. Die dazu notwendigen Schritte wurden im Rahmen der prototypischen Implementierung gezeigt.

### 6.3 Ausblick auf Folgearbeiten

Auf Basis der Erkenntnisse dieser Arbeit bieten sich weitere Themen zur genaueren Untersuchung an.

Mit dem Konfigurationsmanagement-Tool *Puppet* kann eine Änderung der Konfiguration in kürzester Zeit an eine große Anzahl von Systemen ausspielt werden. Durch diese automatisierte Ausspielung der Konfiguration innerhalb kurzer Zeiträume kann eine fehlerhafte Definition in den Manifesten weitreichende Folgen für eine große Anzahl an Systemen haben. Daher muss durch angepasste Arbeitsabläufe die Wahrscheinlichkeit solcher Fehler in den Manifesten auf ein Minimum reduziert werden. Dafür bieten sich entsprechende Maßnahmen zur Steigerung der Qualität aus der Softwareentwicklung, wie beispielsweise testgetriebene Entwicklung und kontinuierliche Integration, an. Diese Problematik muss bei einer Einführung von *Puppet* für Systeme des Webhostings am LRZ berücksichtigt werden und könnte durch eine Folgearbeit untersucht werden.

Eine Weiterentwicklung des Webanwendungsscanners `webapp_discover` könnte auch Gegenstand von weiteren Untersuchungen sein. Derzeit wird lediglich eine unterstützte Webanwendung samt verwendeter Version erkannt. Hilfreich wäre hierbei eine Einschätzung, ob die erkannte Version der Webanwendung durch bekannte Sicherheitslücken gefährdet ist. Durch eine Unterstützung von zusätzlichen Webanwendungen, könnte die Erkennungsrate verbessert werden. Denkbar ist dabei auch ein proaktiver Ansatz: Durch Analyse der Webserver-Logdateien könnte ermittelt werden, bei welchen Webanwendungen derzeit häufig Angriffe versucht werden. Diese Webanwendungen sollten dann bei `webapp_discover` implementiert werden. Nach einem darauffolgenden Suchlauf könnten anfällige Webanwendungen detektiert und die dazugehörigen Kunden zu entsprechenden Updates aufgefordert werden.

Lösungen wie *Docker* könnten ein Vorbote dafür sein, dass sich das klassische Webhosting auf längere Sicht in Richtung eines Cloud-Dienstes, der Platform as a Service (PaaS) bietet, entwickelt. Eine Lösung, die solche Funktionen auf Basis von *Docker* bieten will, ist das Projekt *Flynn*, welches sich noch im Entwicklungsstadium befindet (vgl. [ASo14]). Wie das bestehende Webhosting des LRZs in diese Richtung weiterentwickelt werden kann, könnte ebenfalls ein Thema von zukünftigen Folgearbeiten sein.



## Abbildungsverzeichnis

1.1	Sicherheitsvorfälle Webhosting am LRZ im Jahr 2013 (Datenquelle: [LRZ13d])	2
2.1	Sequenzdiagramm HTTP-Aufruf . . . . .	6
2.2	Überblick serverseitiger Aufbau . . . . .	7
2.3	Generierung von dynamischen Inhalten bei Webservern . . . . .	10
2.4	Hierarchie der OSI-Sicherheitsdienste (nach [Rei02]) . . . . .	15
2.5	Ablauf einer UDP-Amplification-Attack . . . . .	18
3.1	Topologie des betreuten Hostings am LRZ . . . . .	25
3.2	Verteilung der Nutzerkennungen nach Institutionen . . . . .	27
3.3	Verteilung von erkannten Webanwendungen beim Webhosting am LRZ . . . . .	28
4.1	<i>Docker</i> Komponenten im Überblick . . . . .	44
4.2	Zusammenfassung des Ergebnisses des <i>SSL Server Test</i> der vorgeschlagenen Konfiguration . . . . .	52
4.3	<i>Puppet</i> im Client-Server-Modus (vgl. [Pup14a]) . . . . .	65
5.1	Zugriff der Webanwendung auf <code>tomcat-users.xml</code> wird blockiert . . . . .	87
5.2	Verteilung von erkannten Webanwendungen beim Webhosting am LRZ . . . . .	92
5.3	Versionsverteilung bei <i>Drupal</i> . . . . .	93
5.4	Versionsverteilung bei <i>Joomla</i> . . . . .	93
5.5	Versionsverteilung bei <i>phpMyAdmin</i> . . . . .	94
5.6	Versionsverteilung bei <i>Typo3</i> . . . . .	94
5.7	Versionsverteilung bei <i>Wordpress</i> . . . . .	95
5.8	Überblick über das Alter der jeweils eingesetzten Webapplikations Version . . . . .	95
5.9	Abhängigkeiten der <i>Docker</i> -Images und Container . . . . .	98
5.10	Installationsassistent von <i>Wordpress</i> . . . . .	106



## Literatur

- [AFr14] Apache Friends. *XAMPP Apache + MySQL + PHP + Perl*. <http://www.apachefriends.org/index.html> abgerufen am 27. Februar 2014. 2014.
- [AlF13] Nadhem J. AlFardan. *On the Security of RC4 in TLS and WPA*. <http://www.isg.rhul.ac.uk/tls/> abgerufen am 24. Februar 2014. 2013.
- [AP13] Nadhem J. AlFardan und Kenneth G. Paterson. *Lucky Thirteen: Breaking the TLS and DTLS Record Protocols*. <http://www.isg.rhul.ac.uk/tls/TLStiming.pdf> abgerufen am 24. Februar 2014. 2013.
- [Apa13a] The Apache Software Foundation. *Changes with Apache 2.2.26*. [http://www.apache.org/dist/httpd/CHANGES\\_2.2.26](http://www.apache.org/dist/httpd/CHANGES_2.2.26) abgerufen am 24. Februar 2014. 2013.
- [Apa13b] The Apache Software Foundation. *Developing modules for the Apache HTTP Server 2.4*. <http://httpd.apache.org/docs/2.4/developer/modguide.html> abgerufen am 05. Dezember 2013. 2013.
- [Apa14] The Apache Software Foundation. *Apache Tomcat 7 - Security Considerations*. <http://tomcat.apache.org/tomcat-7.0-doc/security-howto.html> abgerufen am 24. Februar 2014. 2014.
- [ASo14] Apollic Software. *Flynn - Documentation*. <https://flynn.io/docs> abgerufen am 6. März 2014. 2014.
- [Aug14] Augeas Team. *Augeas - A quick tour*. <http://augeas.net/tour.html> abgerufen am 24. Februar 2014. 2014.
- [Ber12] Vincent Bernat. *SSL/TLS & Perfect Forward Secrecy*. <http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html> abgerufen am 24. Februar 2014. 2012.
- [BF11] Liana B Baker und Jim Finkle. „Sony PlayStation suffers massive data breach“. In: *Reuters, April 26* (2011).
- [Bro96] Mark R. Brown. *FastCGI Specification*. <http://www.fastcgi.com/devkit/doc/fcgi-spec.html> abgerufen am 05. Dezember 2013. Apr. 1996.
- [BSI06] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheit von Webanwendungen - Maßnahmenkatalog und Best Practices*. [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/WebSec/WebSec\\_pdf.pdf?\\_\\_blob=publicationFile](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/WebSec/WebSec_pdf.pdf?__blob=publicationFile). 2006.
- [BSI11] Bundesamt für Sicherheit in der Informationstechnik. *BSI gibt Tipps für sichere Passwörter*. [https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2011/Passwortsicherheit\\_27012011.html](https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2011/Passwortsicherheit_27012011.html) abgerufen am 28. Januar 2014. 2011.

- [BSI13] Bundesamt für Sicherheit in der Informationstechnik. „IT - Grundschutz Kataloge“. In: (2013). [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/IT-Grundschutz-Kataloge\\_2013\\_EL13\\_DE.pdf?\\_\\_blob=publicationFile](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Download/IT-Grundschutz-Kataloge_2013_EL13_DE.pdf?__blob=publicationFile).
- [Bun11] Bundeskriminalamt. *Cybercrime - Bundeslagebild 2011*. [http://www.bka.de/DE/Publikationen/JahresberichteUndLagebilder/Cybercrime/cybercrime\\_\\_node.html?\\_\\_nnn=true](http://www.bka.de/DE/Publikationen/JahresberichteUndLagebilder/Cybercrime/cybercrime__node.html?__nnn=true) abgerufen am 13. August 2013. 2011.
- [CBF13] Davide Canali, Davide Balzarotti und Aurélien Francillon. „The role of web hosting providers in detecting compromised websites“. In: (2013), S. 177–188.
- [CF114] CloudFlare Inc. *Secure, fast and easy Web Application Firewall*. <https://www.cloudflare.com/waf/> abgerufen am 24. Februar 2014. 2014.
- [Cis14] Cisco Systems Inc. *SpamCop.net - Blocking List ( bl.spamcop.net )*. <http://www.spamcop.net/bl.shtml> abgerufen am 1. März 2014. 2014.
- [CMX14] Clean MX. *Clean MX Anti-Spam Lösung*. [http://www.clean-mx.de/?1\\_fakten.htm](http://www.clean-mx.de/?1_fakten.htm) abgerufen am 1. März 2014. 2014.
- [Cor13] The MITRE Corporation. „About CVE“. In: (2013). <http://cve.mitre.org/about/index.html> abgerufen am 28. Januar 2014.
- [CS12] Commtouch und StopBadware. „Compromised Websites - An Owner’s Perspective.“ In: (2012). <http://stopbadware.org/pdfs/compromised-websites-an-owners-perspective.pdf>.
- [Cym14] Team Cymru. *The Malware Hash Registry (MHR)*. <https://www.team-cymru.org/Services/MHR/> abgerufen am 24. Februar 2014. 2014.
- [Dock14a] Docker Inc. *Docker - Changelog*. <https://github.com/docker/docker/blob/master/CHANGELOG.md> abgerufen am 24. Februar 2014. 2014.
- [Dock14b] Docker Inc. *Docker - Create a Base Image*. <http://docs.docker.io/en/latest/articles/baseimages/> abgerufen am 24. Februar 2014. 2014.
- [Dock14c] Docker Inc. *Docker - Installation - Ubuntu*. <http://docs.docker.io/en/latest/installation/ubuntu/linux/> abgerufen am 24. Februar 2014. 2014.
- [Dock14d] Docker Inc. *Docker - Learn what Docker is all about*. [http://www.docker.io/learn\\_more/](http://www.docker.io/learn_more/) abgerufen am 24. Februar 2014. 2014.
- [Dock14e] Docker Inc. *Docker - Terms - Image*. <http://docs.docker.io/en/master/terms/image/> abgerufen am 24. Februar 2014. 2014.
- [Dock14f] Docker Inc. *Registry server for Docker*. <https://github.com/docker/docker-registry> abgerufen am 5. März 2014. 2014.
- [DST13] Debian Security Team. *Debian security FAQ*. <http://www.debian.org/security/faq> abgerufen am 04. Dezember 2013. 2013.
- [DTM10] Wesam Dawoud, Ibrahim Takouna und Christoph Meinel. „Infrastructure as a service security: Challenges and solutions“. In: (2010), S. 1–8.
- [Fab09] Eberhard von Faber. „Sicherheitsaspekte beim Cloud Computing“. In: (2009).
- [Fro09] Eike Frost. *Apache httpd + suEXEC + chroot + FastCGI + PHP*. <http://e.metaclarity.org/268/httpdsuexecchrootfastcgiphp/> abgerufen am 24. Februar 2014. 2009.



- [Goo14] Google Inc. *Safe Browsing - Transparency Report - Google*. <http://www.google.com/transparencyreport/safebrowsing/?hl=en> abgerufen am 24. Februar 2014. 2014.
- [GPLv3] *GNU General Public License, version 3*. <http://www.gnu.org/licenses/gpl.html>. Juni 2007.
- [Hac11] Mark Hachman. *PlayStation Hack to Cost Sony \$171M; Quake Costs Far Higher*. <http://www.pcmag.com/article2/0,2817,2385790,00.asp> abgerufen am 13. August 2013. 2011.
- [Hir11] Kaz Hirai. *Letter to Congress on the PSN breach*. <http://www.gamesindustry.biz/articles/2011-05-05-kaz-hirais-letter-to-congress-in-full-blog-entry> abgerufen am 13. August 2013. 2011.
- [ITU91] ITU-T. „Security architecture for Open Systems Interconnection for CCITT applications“. In: X.800 (1991).
- [ITU94] ITU-T. „Open Systems Interconnection - Basic Reference Model: The basic model“. In: X.200 (Juli 1994).
- [Jur11] Nico Jurrán. *Angriff auf Playstation Network: Persönliche Daten von Millionen Kunden gestohlen*. <http://heise.de/-1233136> abgerufen am 13. August 2013. 2011.
- [Kas12] Michael Kassner. *Guess who's buying zero-day vulnerabilities?* <http://www.techrepublic.com/blog/it-security/guess-whos-buying-zero-day-vulnerabilities/> abgerufen am 28. Februar 2014. 2012.
- [KM13] Lars R Knudsen und John E Mathiassen. *Preimage and collision attacks on MD2*. <http://www.iacr.org/archive/fse2005/35570253/35570253.pdf> abgerufen am 24. Februar 2014. 2013.
- [KRS13] Heinrich Kersten, Jürgen Reuter und Klaus-Werner Schröder. *IT-Sicherheitsmanagement nach ISO 27001 und Grundschutz: der Weg zur Zertifizierung*. Bd. 4. Springer, 2013.
- [Lis07] Alexei Lisitsa. *The OSI security architecture*. [http://cgi.csc.liv.ac.uk/~alexei/COMP522/COMP522-SecurityArchitecture\\_07.pdf](http://cgi.csc.liv.ac.uk/~alexei/COMP522/COMP522-SecurityArchitecture_07.pdf) abgerufen am 27. Januar 2014. 2007.
- [LRZ12a] Leibniz-Rechenzentrum. *Richtlinie zur Wahl und Nutzung von Passwörtern und Passphrases*. <http://www.lrz.de/wir/regelwerk/passwortrichtlinien.pdf>. 2012.
- [LRZ12b] Leibniz-Rechenzentrum. *Persönliche Webseiten für LRZ-Benutzer*. <http://www.lrz.de/services/netzdienste/www/eigene-www-seiten/> abgerufen am 28. Januar 2014. 2012.
- [LRZ13a] Leibniz-Rechenzentrum. *Das Dienstleistungsangebot des LRZ*. <http://www.lrz.de/wir/regelwerk/dienstleistungskatalog.pdf>. 2013.
- [LRZ13b] Leibniz-Rechenzentrum. *Vergabe von Kennungen für LRZ-Systeme*. <https://www.lrz.de/wir/kennung/> abgerufen am 28. Januar 2014. 2013.
- [LRZ13c] Leibniz-Rechenzentrum. *WWW - Webhosting*. <http://www.lrz.de/services/netzdienste/www/> abgerufen am 28. Januar 2014. 2013.

## Literatur

- [LRZ13d] Leibniz-Rechenzentrum. *LRZ Wiki - Sicherheitsvorfälle bei Webservern*. <https://wiki.lrz.de/tiki-index.php?page=Bearbeitung+von+Sicherheitsvorfällen> abgerufen am 6. Februar 2014. 2013.
- [LRZ14] Leibniz-Rechenzentrum. *Wie man am LRZ ein Zertifikat beantragt*. <http://www.lrz.de/services/pki/wieman/> abgerufen am 3. März 2014. 2014.
- [LXC14] Serge Hallyn und Stéphane Graber. *LXC - Linux Containers*. <http://linuxcontainers.org/> abgerufen am 24. Februar 2014. 2014.
- [MAN05] Linux man-pages project. *SELinux(8) - NSA Security-Enhanced Linux (SELinux)*. <http://man7.org/linux/man-pages/man8/selinux.8.html> abgerufen am 24. Februar 2014. 2005.
- [MAN10] Linux man-pages project. *chroot(2) - change root directory*. <http://man7.org/linux/man-pages/man2/chroot.2.html> abgerufen am 24. Februar 2014. 2010.
- [MAN13] Linux man-pages project. *AppArmor(7) - kernel enhancement to confine programs to a limited set of resources*. <http://manpages.ubuntu.com/manpages/trusty/en/man7/apparmor.7.html> abgerufen am 24. Februar 2014. 2013.
- [Mar09] Stefan Marx. *Web Application Firewalls - Grundlagen und Marktübersicht*. <http://www.tecchannel.de/webtechnik/webserver/2019855> abgerufen am 24. Februar 2014. 2009.
- [McD12] Rich McDonough. *Using Graylog2 for Environment-Wide Log Collection and Correlation*. <http://blog.avidlifemedia.com/2012/06/06/using-graylog2-for-environment-wide-log-collection-and-correlation/> abgerufen am 24. Februar 2014. 2012.
- [McN13] Michael McNally. *What is a DNS Amplification Attack?* <https://deephought.isc.org/article/AA-00897/0/What-is-a-DNS-Amplification-Attack.html> abgerufen am 27. Januar 2014. 2013.
- [Men13] Paul Menage. *Kernel Documentation - Control Groups*. <https://github.com/torvalds/linux/blob/master/Documentation/cgroups/cgroups.txt> abgerufen am 24. Februar 2014. 2013.
- [MG11] Peter Mell und Timothy Grance. „The NIST definition of cloud computing“. In: (2011). <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> abgerufen am 19. November 2013.
- [Mic13] Microsoft. *Microsoft Support Lifecycle – Windows Server 2008*. <http://support.microsoft.com/lifecycle/default.aspx?LN=de&x=13&y=15&p1=12925> abgerufen am 04. Dezember 2013. 2013.
- [MPR02] Jelena Mirkovic, Gregory Prier und Peter Reiher. „Attacking DDoS at the source“. In: (2002), S. 312–321.
- [Nos14] Nose Team. *Nose - is nicer testing for python*. <https://nose.readthedocs.org/en/latest/index.html> abgerufen am 24. Februar 2014. 2014.
- [OISF14] Open Information Security Foundation. *Suricata - Features*. <http://suricata-ids.org/features/> abgerufen am 6. März 2014. 2014.

- [OSS14] OpenSSL Project. *OpenSSL Release Notes*. <https://www.openssl.org/news/openssl-notes.html> abgerufen am 24. Februar 2014. 2014.
- [OWASP08] Open Web Application Security Project German Chapter. *Best Practices: Einsatz von Web Application Firewalls*. [https://www.owasp.org/images/1/1b/Best\\_Practices\\_Guide\\_WAF.pdf](https://www.owasp.org/images/1/1b/Best_Practices_Guide_WAF.pdf) abgerufen am 24. Februar 2014. 2008.
- [OWASP13] Open Web Application Security Project. *OWASP Top Ten*. [https://www.owasp.org/index.php/Top\\_10\\_2013-Top\\_10](https://www.owasp.org/index.php/Top_10_2013-Top_10) abgerufen am 27. Januar 2014. 2013.
- [OWASP14] Open Web Application Security Project. *OWASP ModSecurity Core Rule Set (CRS)*. <http://spiderlabs.github.io/owasp-modsecurity-crs/> abgerufen am 24. Februar 2014. 2014.
- [PHP14] PHP. *PHP: FastCGI Process Manager (FPM)*. <http://www.php.net/manual/en/install.fpm.php> abgerufen am 1. März 2014. 2014.
- [Pie12] R.I. Pienaar. *Simple Puppet Module Structure Redux*. <http://www.devco.net/archives/2012/12/13/simple-puppet-module-structure-redux.php> abgerufen am 24. Februar 2014. 2012.
- [PK09] Sandeep PK. *ModSecurity - Intro*. [http://blog.supportpro.com/2009/08/mod\\_security-intro/](http://blog.supportpro.com/2009/08/mod_security-intro/) abgerufen am 24. Februar 2014. 2009.
- [PRM09] Niels Provos, Moheeb Abu Rajab und Panayiotis Mavrommatis. „Cybercrime 2.0: when the cloud turns dark“. In: 52.4 (2009), S. 42–47.
- [Pup14a] Puppet Labs Inc. *Learning Puppet — Basic Agent/Master Puppet*. [http://docs.puppetlabs.com/learning/agent\\_master\\_basic.html](http://docs.puppetlabs.com/learning/agent_master_basic.html) abgerufen am 24. Februar 2014. 2014.
- [Pup14b] Puppet Labs Inc. *Learning Puppet — Resource Ordering*. <http://docs.puppetlabs.com/learning/ordering.html> abgerufen am 24. Februar 2014. 2014.
- [Pup14c] Puppet Labs Inc. *Puppet - Type Reference*. <http://docs.puppetlabs.com/references/latest/type.html> abgerufen am 24. Februar 2014. 2014.
- [Red13] Red Hat. *Red Hat Enterprise Linux Life Cycle*. <https://access.redhat.com/site/support/policy/updates/errata/> abgerufen am 04. Dezember 2013. 2013.
- [Rei02] Helmut Reiser. *Sicherheitsarchitektur für ein Managementsystem auf der Basis Mobiler Agenten*. <http://www.nm.ifi.lmu.de/common/pub/Dissertationen/reis01/PDF-Version/reis01.pdf.gz>. 2002.
- [Rei08a] Helmut Reiser. *Ein Framework für föderiertes Sicherheitsmanagement*. <http://www.mnm-team.org/pub/Dissertationen/reis08/PDF-Version/reis08.pdf>. 2008.
- [Rei08b] Helmut Reiser. *Sicherheit vernetzter Systeme - Kapitel 2: Grundlagen*. [http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2008ws/itsec/\\_skript/itsec-k2-v4.0.pdf](http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2008ws/itsec/_skript/itsec-k2-v4.0.pdf). 2008.

- [Rei13] Helmut Reiser. *Sicherheit vernetzter Systeme - Kapitel 7: Kryptographische Hash-Funktionen*. [http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2013ws/itsec/\\_skript/itsec-k7-v9.0.pdf](http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2013ws/itsec/_skript/itsec-k7-v9.0.pdf) abgerufen am 24. Februar 2014. 2013.
- [RFC1918] Y. Rekhter u. a. *Address Allocation for Private Internets*. RFC 1918 (Best Current Practice). Updated by RFC 6761. Internet Engineering Task Force, Feb. 1996. URL: <http://www.ietf.org/rfc/rfc1918.txt>.
- [RFC2350] N. Brownlee und E. Guttman. *Expectations for Computer Security Incident Response*. RFC 2350 (Best Current Practice). Internet Engineering Task Force, Juni 1998. URL: <http://www.ietf.org/rfc/rfc2350.txt>.
- [RFC2616] R. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). Updated by RFCs 2817, 5785, 6266, 6585. Internet Engineering Task Force, Juni 1999. URL: <http://www.ietf.org/rfc/rfc2616.txt>.
- [RFC3875] D. Robinson und K. Coar. *The Common Gateway Interface (CGI) Version 1.1*. RFC 3875 (Informational). Internet Engineering Task Force, Okt. 2004. URL: <http://www.ietf.org/rfc/rfc3875.txt>.
- [Rfx13] R-fx Networks. *Linux Malware Detect*. <https://www.rfxn.com/projects/linux-malware-detect/> abgerufen am 24. Februar 2014. 2013.
- [Ris13] Ivan Ristić. *SSL/TLS Deployment Best Practices*. [https://www.ssllabs.com/downloads/SSL\\_TLS\\_Deployment\\_Best\\_Practices\\_1.3.pdf](https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices_1.3.pdf) abgerufen am 24. Februar 2014. 2013.
- [Rus14] Gareth Rushgrove. *A pretty opinionated skeleton for writing your own puppet modules*. <https://github.com/garethr/puppet-module-skeleton> abgerufen am 24. Februar 2014. 2014.
- [SBw14] StopBadware Inc. *Supporting Organizations - StopBadware*. <https://www.stopbadware.org/partners> abgerufen am 24. Februar 2014. 2014.
- [Sch07] Bruce Schneier. *Full Disclosure of Security Vulnerabilities a 'Damned Good Idea'*. <https://www.schneier.com/essay-146.html> abgerufen am 28. Januar 2014. 2007.
- [Sch09] Rene Schneider. „Absicherung eines (Web-) Applikationsservers auf Linux-Basis“. In: (2009).
- [Sch14] Hynek Schlawack. *Hardening Your Web Server's SSL Ciphers*. <https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/> abgerufen am 24. Februar 2014. 2014.
- [Sel14] Selenium. *SeleniumHQ - Browser Automation*. [http://docs.seleniumhq.org/docs/01\\_introducing\\_selenium.jsp](http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp) abgerufen am 27. Februar 2014. 2014.
- [Sey11] Patrick Seybold. *Update on PlayStation Network and Qriocity*. <http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity> abgerufen am 13. August 2013. 2011.
- [Sha12] Tim Sharpe. *RSpec tests for your Puppet manifests - Tutorial*. <http://rspec-puppet.com/tutorial/> abgerufen am 24. Februar 2014. 2012.

- [Sin02] Tony Sintes. *App server, Web server: What's the difference?* <http://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html> abgerufen am 27. Februar 2014. 2002.
- [Sou14a] Sourcefire Inc. *Snort FAQ*. <https://github.com/vrtadmin/snort-faq/blob/master/README.md> abgerufen am 24. Februar 2014. 2014.
- [Sou14b] Sourcefire. *Sourcefire Vulnerability Research Team (VRT)*. <http://www.snort.org/vrt> abgerufen am 6. März 2014. 2014.
- [Spe07] Ralf Spenneberg. *SELinux & AppArmor: Mandatory Access Control für Linux einsetzen und verwalten*. Pearson Deutschland GmbH, 2007.
- [spi13] spider.io. *Discovered: Botnet Costing Display Advertisers over Six Million Dollars per Month*. <http://www.spider.io/blog/2013/03/chameleon-botnet/> abgerufen am 27. Januar 2014. 2013.
- [Spl14] Splunk Inc. *Splunk - Get started*. <http://dev.splunk.com/view/get-started/SP-CAAAECH> abgerufen am 24. Februar 2014. 2014.
- [Tan92] Andrew S Tanenbaum. *Modern operating systems*. Bd. 2. prentice Hall Englewood Cliffs, 1992.
- [TIO13] Software BV TIOBE. *TIOBE Programming Community Index for November 2013*. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> abgerufen am 18. November 2013. 2013.
- [Tor13] TORCH GmbH. *Graylog2 architecture high level overview*. <http://support.torch.sh/help/kb/general/graylog2-architecture-high-level-overview> abgerufen am 24. Februar 2014. 2013.
- [UCE14] UCEPROTECT-Network. *UCEPROTECT-Network - SPAM-FAQ*. <http://www.uceprotect.net/en/index.php?m=2&s=0> abgerufen am 24. Februar 2014. 2014.
- [Wiki14] Wikimedia Foundation Inc. *Comparison of DNS blacklists*. [http://en.wikipedia.org/wiki/Comparison\\_of\\_DNS\\_blacklists](http://en.wikipedia.org/wiki/Comparison_of_DNS_blacklists) abgerufen am 24. Februar 2014. 2014.
- [Wit06] Bernhard C Witt. *IT-Sicherheit kompakt und verständlich: Eine praxisorientierte Einführung*. Springer DE, 2006.
- [Wor14] WordPress. *Hosting WordPress*. [http://codex.wordpress.org/Hosting\\_WordPress](http://codex.wordpress.org/Hosting_WordPress) abgerufen am 24. Februar 2014. 2014.



## Inhalt der beigelegten CD

- `Dokumentation/PDF/simo14.pdf`  
Digitale Fassung dieser Masterarbeit im PDF-Format
- `Dokumentation/Latex/`  
Latex-Quellcode der Arbeit
- `Dokumentation/Abstract.txt`  
Kurzfassung der Arbeit im Text-Format
- `Literatur/`  
Benutzte Literatur, soweit digital verfügbar
- `Prototyp/1_puppet_tomcat/`  
Quellcode zur Konfiguration von *Tomcat* mittels *Puppet* (vgl. Abschnitt 5.1)
  - `modules/`  
Module `lrz_kom_tomcat` und `lrz_kom_apparmor`
  - `webapp_test/`  
Webanwendung zum Testen von *AppArmor*
- `Prototyp/2_webapp_discover/`  
Webanwendungsscanner `webapp_discover` (vgl. Abschnitt 5.2)
  - `webapp_discover/`  
Quellcode von `webapp_discover`
  - `webapp_discover_test/`  
Verzeichnisstrukturen zum Testen von `webapp_discover`
  - `messung_2014-02-03/`  
Ergebnis des Suchlaufs vom 3. Februar 2014 am LRZ
- `Prototyp/3_docker_wordpress/`  
Dockerfiles und Skripte für die Bereitstellung von *Wordpress* mit *Docker* (vgl. Abschnitt 5.3)





