Information Security

Automatisierung des Abgleichs von Software-Paketen Unix-ähnlicher Systeme mit Security-Advisory-Feeds

Ein Werkzeug zur effizienten Feststellung von Schwachstellen in der IT-Infrastruktur eines Unternehmens

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

eingereicht von

Alexander Weidinger, BSc
is101514

im Rahmen des
Studienganges Information Security an der Fachhochschule St. Pölten

Betreuung
Betreuer: Dipl.-Ing. Dr. Paul Tavolato

St. Pölten, 2. August 2012

(Unterschrift des Verfassers)

(Unterschrift des Betreuers)

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Bachelorarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Bachelorarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.
- Ich räume hiermit der Fachhochschule St. Pölten das ausschließliche und räumlich unbeschränkte Werknutzungsrecht für alle Nutzungsarten an dieser Diplomarbeit ein, und behalte das Recht, als Urheber dieses Werkes genannt zu werden.

St. Pölten, 2. August 2012

(Unterschrift des Verfassers)

Proudly made with Vim and LaTeX

Zusammenfassung

Die Selektion für die eigene IT-Infrastruktur relevanter Hinweise aus der Fülle sicherheitsbezogener Meldungen ist eine ebenso verantwortungsvolle wie ermüdende Aufgabe. Ein Unternehmen hat üblicherweise kein gesteigertes Interesse daran eine/n hochbezahlte/n Spezialisten/-in tagtäglich sämtliche Informationsquellen nach solchen Meldungen durchsuchen zu lassen. Ähnlich verhält es sich mit der (manuellen) Pflege des dazu benötigten Software-Inventars.

Hier zeigt die vorliegende Diplomarbeit Wege auf, wie die Erstellung des Software-Inventars und die Vorauswahl der sicherheitsrelevanten Meldungen automatisiert stattfinden können. Dabei werden die technischen wie auch die rechtlichen Rahmenbedingungen analysiert und als Ausgangspunkte für die Implementierung einer konkreten Lösung herangezogen. Diese besteht zum einen aus einem zentralen Server, welcher das Inventar bereitstellt und die Korrelation von Paket-Daten und Sicherheitsmeldungen durchführt. Zum anderen liefern Web-Crawler und auf den zu überwachenden Hosts installierte Agenten dem Server die dazu benötigten Daten. Durch die Modularisierung der beteiligten Komponenten lässt sich die vorgestellte Lösung leicht an verschiedene Datenquellen anpassen.

Bei der erwähnten Korrelation werden Beziehungen zwischen Softwarepaketen und sicherheitsrelevanten Meldungen geknüpft, aus welchen sich der jeweilige Bedrohungsgrad eines Pakets oder eines Hosts ableitet. Diese Bedrohungsgrade werden anhand eines Farbverlaufs entsprechend ihres Wertes visualisiert. Im Endergebnis kann die Bedrohungslage der eigenen IT-Infrastruktur in der Web-Oberfläche des erwähnten Servers eingesehen und ggf. korrigiert werden.

Abstract

The selection of security advisories which are relevant to one's own IT infrastructure is a responsible but tedious task. A company is usually not interested in paying a specialist to search all sources of information for such reports. Manually maintaining the software inventory which is needed for these searches is of no interest either.

This work shows ways to automatically create the software inventory and select the relevant security advisories. Technical and legal aspects are analyzed and form the basis of the implementation of an actual software solution. This solution consists of a central server, web crawlers and agents which are installed on the monitored hosts. The server holds the inventory and correlates package data with security advisories. The web crawler and the agent support the server with the needed input. Because the participating components are separated into modules the introduced solution can be adapted to different sources of information.

Relations between software packages and relevant security advisories are set during the mentioned correlation. The threat level of a host or package is deduced from this relations. A color gradient is used to visualize the threat level's value. In effect the current threat situation of one's own IT infrastructure can be viewed and (if necessary) corrected in the server's web interface.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation für das Thema . . . 1
1.2 Erkenntnisgegenstand . . . 1
1.3 Forschungsleitende Fragestellung . . . 2
1.4 Aufbau dieser Arbeit / Forschungsstrategie . . . 2

2 Einführung in die Thematik **3**
2.1 Paketmanager . . . 5
2.1.1 Debian Package Manager (dpkg) . . . 5
2.1.2 Advanced Packaging Tool (apt) . . . 6
2.1.3 *BSD Ports . . . 6
2.2 Security Advisory Feeds . . . 6
2.2.1 Debian Security Advisories . . . 9
2.2.2 Ubuntu security notices . . . 10
2.2.3 NIST / National Vulnerability Database . . . 12
2.2.4 US-CERT / Vulnerability Notes Database . . . 13
2.2.5 Open Source Vulnerability Database . . . 15
2.2.6 Red Hat . . . 17
2.2.7 CentOS . . . 17
2.3 Fazit der einführenden Überlegungen . . . 18

3 Aufgabenstellung **19**

4 Architektur und Design **20**
4.1 Bedrohungsmodell (Threat Model) und Sicherheitsfunktionen . . . 22
4.1.1 WizzanAgent . . . 24
4.1.2 WizzanCrawler . . . 24

4.1.3 Security-Advisory-Feeds . . . 25
4.1.4 WizzanServer . . . 25

5 Kommunikation zwischen den Komponenten 27
5.1 Verteilung und Verifikation des Schlüsselmaterials . . . 28
5.2 Deployment-Phase . . . 28
5.3 Operativer Einsatz . . . 28
5.4 Nachrichtenformat . . . 29

6 Agent & Crawler: Separated at birth 32

7 Auswertung von Hosts und Paketmanagern 34
7.1 Systeminformationen . . . 35
7.2 Paketinformationen . . . 36
7.3 Debian Package Manager . . . 38
7.4 Advanced Packaging Tool . . . 39

8 WizzanServer 42
8.1 Framework . . . 43
8.2 Datenbanken . . . 44
8.3 Schema der Wizzan-Datenbank . . . 45

9 Automatisierte Erstellung des Paket-Inventars 49
9.1 Verarbeitung empfangener Paketlisten . . . 49
9.2 Entfernen alter Pakete . . . 50
9.3 Paket-Objekt . . . 50
9.4 Fazit zum Inventar . . . 51

10 Auswertung von Security-Advisory-Feeds 52
10.1 Technische Aspekte . . . 55
10.2 Rechtliche Aspekte . . . 56
10.3 Anfordern von Paketlisten . . . 57
10.4 Matching . . . 59
10.5 Limitierungen und Wiederaufnahme . . . 60
10.6 Normalisierung von Security-Advisories . . . 61
10.7 Extraktion von Versionsinformationen . . . 63

10.8 Aufbereiten von Versionsinformationen . . . 64
10.9 Implementierungsbeispiele . . . 66
10.9.1 Debian Security Advisories . . . 66
10.9.2 Ubuntu security notices . . . 67
10.9.3 NIST / National Vulnerability Database . . . 68
10.9.4 US-CERT . . . 69

11 Korrelation von Paketen und Advisories 71
11.1 Advisory-Queue und Advisory-Dispatcher . . . 72
11.2 Korrelation der Paketnamen . . . 73
11.2.1 Die Datenbankabfrage . . . 73
11.2.2 Aliases . . . 74
11.3 Korrelation der Paketversionen . . . 74
11.3.1 Ignorieren von Prä- und Postfix . . . 75
11.3.2 Branches . . . 76
11.3.3 Definition verwundbarer Versionen . . . 77
11.3.4 Vergleichen von Versionsnummern . . . 80
11.4 Fallback für die Korrelation von Paketversionen . . . 80
11.5 Ermittlung des Bedrohungsgrades . . . 81
11.6 Aussortieren falsch-positiver Relationen . . . 81
11.7 Erkennen problembehebender Paket-Objekte . . . 82

12 Darstellung der Ergebnisse der Korrelation 83
12.1 Lizenzrechtliche Aspekte . . . 84

13 Lizenzierung von Wizzan 85

14 Bekannte Probleme, Weiterentwicklung 86

15 Fazit 88

Literaturverzeichnis 89

Abbildungsverzeichnis 96

Einleitung

1.1 Motivation für das Thema

Die Diplomarbeit dient als Startpunkt für ein Open-Source-Projekt, dessen Ziel es ist, Überblick über aktuelle Patchstände, verfügbare Updates und aktuelle Bedrohungen der eigenen IT-Infrastruktur zu schaffen.

1.2 Erkenntnisgegenstand

Ab einer gewissen Anzahl von Rechnern, die innerhalb eines (Unternehmens-)Netzwerks betrieben werden, ist es nicht mehr trivial, die auf ihnen installierten Software-Versionen mit den gegenwärtigen Bedrohungen zu korrelieren und ggf. Aktualisierungen vorzunehmen.
Hieraus lassen sich die folgenden drei Missstände formulieren, die es in Unternehmen zu verhindern gilt: [1]

- Es ist unbekannt, welche Software-Versionen auf welchen Rechnern installiert sind.
- Es ist unbekannt, ob und welche Aktualisierungen für bestimmte Software-Pakete verfügbar sind.
- Es ist unbekannt, welche Bedrohungen auf die eigene IT-Infrastruktur aufgrund ungepatchter Sicherheitslücken einwirken.

Für Windows existieren bereits Lösungen, die die oben beschriebenen Missstände begrenzen bzw. beheben. Während der Recherche wurden keine vergleichbaren Lösungen für unixoide Systeme gefunden. [2]

1.3 Forschungsleitende Fragestellung

Wie können die Informationen aus Paket-Managern unixoider Systeme (z.B. apt, yum, etc.) mit den Daten von Security-Advisory-Feeds (RSS, Websites, etc.) abgeglichen werden, sodass eine Übersicht über die aktuelle Bedrohungslage und die notwendigen Patches/Changes der unternehmenseigenen Infrastruktur entsteht?

1.4 Aufbau dieser Arbeit / Forschungsstrategie

Zu Beginn werden im **einführenden Teil** (*theoretische Analyse mittels Literaturrecherche*) die Gemeinsamkeiten und Eigenheiten sowohl der gängigen Paket-Manager (z.B. *apt*, *yum*, *FreeBSD-Ports*) als auch populärer *Security-Advisory-Feeds* (z.B. von *NIST*, *Ubuntu*, *Debian*, *RedHat*, etc.) ermittelt. Dabei ist wichtig, dass die Betreiber dieser Feeds maschinelle Auswertungen der Inhalte nicht untersagen. Die aus diesen Quellen gewonnenen Daten müssen in eine einheitliche Form gebracht, d.h. normalisiert werden, um eine sinnvolle Weiterverarbeitung zu ermöglichen.

Dies ergibt die Ausgangsbasis für das Konzept der Web-Applikation, die den in der forschungsleitenden Fragestellung beschriebenen Abgleich durchführt. Das Konzept wird mit Hilfe geeigneter Darstellungsmethoden veranschaulicht. Es orientiert sich klarerweise am eingesetzten Framework (s.u.), stellt dieses aber nicht in voller Detailtiefe dar, um übersichtlich zu bleiben.

Im (umfangreicheren) **praktischen Teil** (*empirisches Verfahren: Experiment*) erfolgt die Implementierung eines Prototypen. Als Ausgangsbasis für die Entwicklung wird ein PHP-Framework (konkret: *Kohana*) eingesetzt, um eine belastbare Grundlage (Benutzerverwaltung, Zugriffskontrolle, Security, schnellere Entwicklung, etc.) zu haben.

Die Software bzw. deren Module lösen ausschließlich die oben beschriebenen (Teil-)Aufgaben, dies jedoch besonders gut. Ein Kernsystem mit Benutzer-Authentifikation korreliert die abstrahierten Daten und stellt diese dar. Ausgegliederte Teilsysteme zapfen die verschiedenen Datenquellen an. Dieser modulare Aufbau soll in Zukunft eine leichte Erweiterbarkeit gewährleisten.

Die Benutzeroberfläche konzentriert sich primär auf eine tabellarische Darstellung der wichtigsten Daten der überwachten Systeme inkl. einer farblichen Kodierung (Ampel) der aktuellen Bedrohung.

Der Umfang der Implementierung beschränkt sich auf das Kernsystem, zwei Feed-Module und zwei Paket-Manager-Module. Die Dokumentation und die Code-Verwaltung erfolgen vollständig in *Redmine* (Ticketing-System inkl. Wiki mit Fokus auf Software-Entwicklung) bzw. mit Hilfe von *git* (Versionskontrollsystem).

Einführung in die Thematik

Dieses Kapitel behandelt die Grundlagen und Rahmenbedingungen maschineller Unterstützung des Menschen bei der Auswahl (möglicherweise) relevanter Security-Advisories. Es werden organisatorische und technische Aspekte genannt, die die Ermittlung der Bedrohungslage eines Unternehmensnetzwerks beeinflussen.

Die Behandlung der Abwehr von Angriffen, die auf noch unbekannten bzw. gerade erst bekannt gewordenen Sicherheitsschwachstellen basieren (sog. *Zero-Day-Exploits*), ist ausdrücklich nicht Ziel dieser Diplomarbeit. Sinnvolle maschinelle Unterstützung ist zum genannten Zeitpunkt des Bekanntwerdens einer Schwachstelle auch nur schwer möglich, da meist Unklarheit über die genauen Details herrscht.

Erst nach einigen Tagen bzw. erst bei Erscheinen aktualisierter Software über die entsprechende Sicherheitslücke sowie mögliche Gegenmaßnahmen unterrichtet zu werden ist laut Microsoft Security Intelligence Report Volume 11 [3, S. 14] vergleichsweise unproblematisch. Der Report lässt den Schluss zu, dass Sicherheitsschwachstellen erst rund zwei Monate nach Erscheinen des Patches verstärkt ausgenutzt werden. [3, S. 15f.] Die beiden Abbildungen 2.1 und 2.2 stellen diesen Sachverhalt am Beispiel zweier unterschiedlicher Sicherheitsschwachstellen exemplarisch dar.

Aus (naiver) Sicherheitssicht wäre es wünschenswert alle Patches unmittelbar nach ihrem Erscheinen einzuspielen, um die eigenen Systeme in einem möglichst sicheren Zustand zu halten. Dieser Wunsch steht jedoch der Anforderung der Aufrechterhaltung der Services und der Kosteneffizient diametral entgegen. FIRST (Forum of Incident Response and Security Teams) schlägt in einem Beispiel einer auf CVSS basierenden Patch-Policy [4, S. 3] eine vom Bedrohungsgrad abhängige maximale Wartezeit bis zum Einspielen eines Patches vor. Abbildung 2.3 stellt diese Wartezeiten dar.

Um die oben exemplarisch angeführte Patch-Policy anwenden zu können, muss zuerst erkannt werden, *ob* Software-Komponenten der eigenen IT-Infrastruktur von einer Sicherheitsschwachstelle betroffen sind. Damit an dieser Stelle maschinelle Unterstützung erfolgen kann, müssen die Daten über installierte Software, verfügbare Aktualisierungen und Security-Advisories ermittelt und miteinander in Verbindung gesetzt werden.

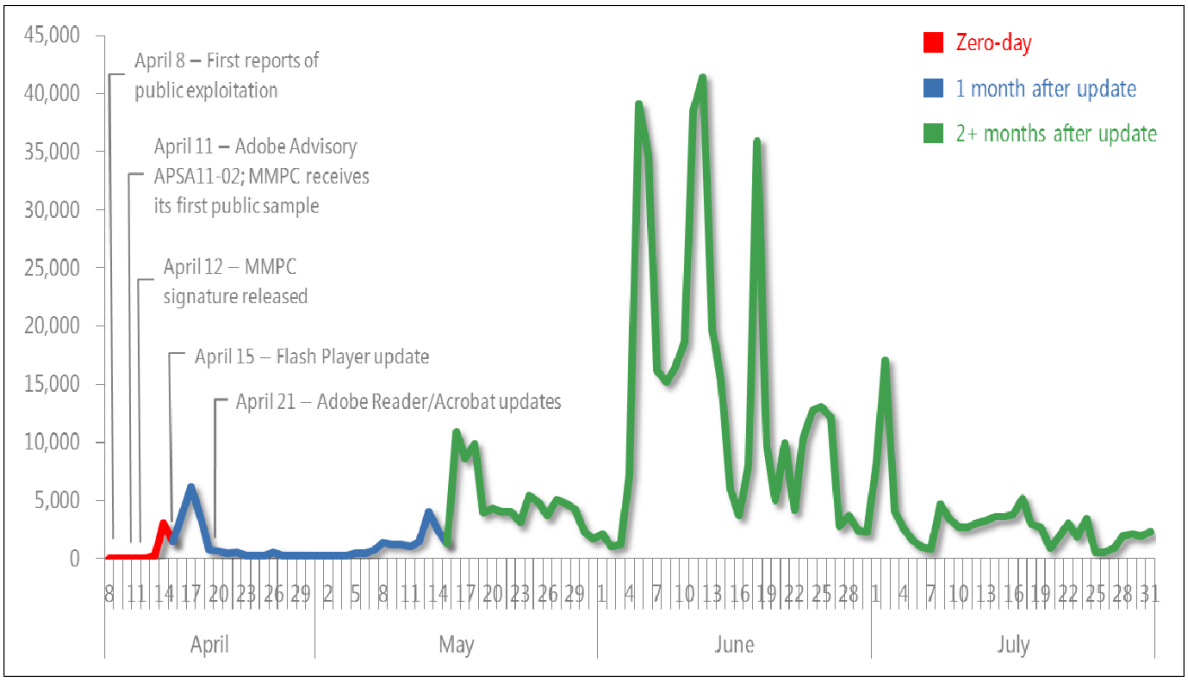

Abbildung 2.1: Detections of exploits targeting CVE-2011-0611, April-July, 2011 [3, S. 15]

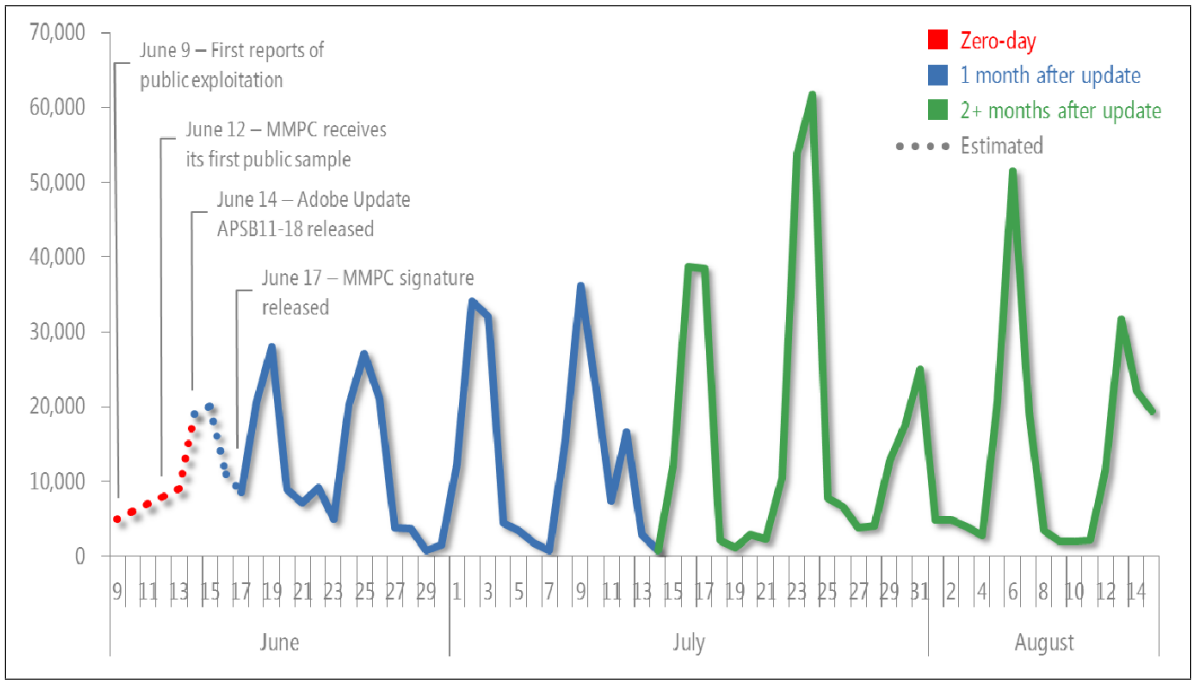

Abbildung 2.2: Detections of exploits targeting CVE-2011-2110, June-August, 2011 [3, S. 16]

CVSS Score	Priority	Patch SLA
0	P4	Discretionary
.1-3	P3	Next Patch Cycle (3-6 months)
4-6	P2	4 Weeks Max
7-10	P1	2 Weeks Max

Abbildung 2.3: CVSS Vulnerability Assessment Results [4]

2.1 Paketmanager

Die meisten Linux-Distributionen und auch die verschiedenen BSD-Systeme stellen mittlerweile Software in Form von Paketen zur Verfügung. An die Stelle großer, monolithischer Applikationen, die ihrerseits aus verschiedenen Teilkomponenten zusammengesetzt sind, treten Pakete überschaubarer Größe, in deren Meta-Informationen u.a. die Abhängigkeiten zu anderen Paketen definiert sind. [5] Anstatt monolithische Produkte übereinander zu stapeln bis daraus eine lauffähige Applikation entsteht, lösen Paketmanager die Abhängigkeiten der gewünschten Software auf und installieren anschließend alle benötigten Pakete.
Aus einer langwierigen Installation wird so z.B. im Fall der Distribution *Debian* ein einzelnes Kommando: apt-get install *whatever*

Jede Software(-Komponente), die per Paketmanager installiert wurde, kann *inklusive ihrer genauen Versionsnummer* aufgelistet werden. Dies ist genau jener Punkt, an dem die automatisierte Erstellung des Software-Inventars ansetzt.

2.1.1 Debian Package Manager (dpkg)

Die Debian-Pakete bündeln Dateien, die benötigt werden, um ein bestimmtes Kommando oder eine bestimmte Funktion zur Verfügung zu stellen. Es werden zwei Typen unterschieden: *Binärpakete* enthalten ausführbare Programme, Konfigurationsdateien, Handbuch-Dateien und urheberrechtliche Informationen, wohingegen *Quelltextpakete* den zugehörigen Original-Quelltext samt Debian-spezifischer Patches bündeln. [6]

> *„dpkg [...] ist die Basis der Debian Paketverwaltung. [sic!] [...] Die wichtigste Funktion von dpkg besteht darin, das System in einem stabilen Zustand zu halten. Da die Werkzeuge zur Paketverwaltung in anderen Disrtibutionen [sic!] nicht den Anforderungen der Debian Entwickler [sic!] genügten, wurde dpkg entworfen. Das Paketformat RPM, das u.a. von RedHat/Fedora- und Novell/SuSE-Distributionen verwendet wird, prüft lediglich, ob eine benötigte Datei vorhanden ist. Ob diese Datei in der gewünschten oder gar benötigten Version vorliegt, wird nicht geprüft. [7]*

Mit dpkg ist es möglich *installierte* Pakete inkl. Versionsnummer aufzulisten. *Verfügbare Aktualisierungen* werden von dpkg hingegen *nicht* erfasst. [7]

2.1.2 Advanced Packaging Tool (apt)

dpkg stellt zwar sicher, dass zu einem Paket gehörige Dateien in der richtigen Version installiert sind bzw. entfernt wurden, jedoch löst es keine Abhängigkeiten auf.
Das Debian-Projekt schuf APT als *„schnelles, praktisches und effizientes Mittel, um Pakete zu installieren, das Abhängigkeiten automatisch behandeln und [...] Konfigurationsdateien während des Aktualisierens berücksichtigen würde.“* [8]

apt-get ermöglicht die Simulation einer Systemaktualisierung. Der Output enthält die Namen und Versionsnummern der *verfügbaren Aktualisierungen*. Für die Auflistung installierter Software ist dpkg aufgrund höherer Performance besser geeignet. [9]

2.1.3 *BSD Ports

Anders als die zuvor genannten Paketverwaltungssysteme RPM und DPKG transportiert das Ports-System *nicht primär Binärdateien* bereits vorkompilierter Software, sondern den Quelltext selbst. [10] D.h. ein Port ist üblicherweise ein Verzeichnis, das alle für das Kompilieren erforderlichen Daten sowie das *Makefile* enthält.

Ports haben aber auch Einzug in die „Linux-Welt“ gehalten: Die Distributionen Gentoo [11] und Arch Linux [12] beispielsweise verwenden ports-ähnliche Paketverwaltungssysteme.

Im Gegensatz zu dpkg oder RPM existieren bei ports-basierten Systemen vielerlei Varianten, die oft auch mit unterschiedlichen Kommandozeilen-Befehlen bedient werden. Für die Mehrzahl der Ports-Implementierungen auf Basis ihrer Gemeinsamkeiten eine generische Lösung zu implementieren ist praktisch nicht durchführbar. [10] [11] [12]

2.2 Security Advisory Feeds

Ein *Security-Advisory-Feed* ist im Kontext dieser Diplomarbeit im weitesten Sinne als Datenquelle sicherheitsrelevanter Handlungshinweise zu verstehen. Diese bündeln Merkmale einer Sicherheitsschwachstelle bzw. eines potentiellen Angriffs sowie mögliche Gegenmaßnahmen. Beispiele für derartige Merkmale sind:

- Verwundbare Software-Versionen
- Nicht verwundbare bzw. ausgebesserte Software-Versionen
- Schweregrad der Schwachstelle (Als objektivstes Maß hat sich der sog. CVSS-Score [13] etabliert, der jedoch nicht von allen Herausgebern gleichermaßen berechnet und angegeben wird.)
- Datum des öffentlichen Bekanntwerdens
- Referenzen zu weiteren Informationsquellen

Für die maschinelle Verwertbarkeit derartiger Informationen sind die folgenden Aspekte besonders ausschlaggebend. Sie ermöglichen eine erste, grobe Einschätzung, ob es ökonomisch sinnhaft ist, einen Security-Advisory-Feed maschinell zu verarbeiten.

- Die Regelmäßigkeit in der die Daten eines Security-Advisory in der selben Form und Struktur vorliegen. (Ständig wechselnde Formate erschweren es zukünftige Meldungen zeitnah zu verarbeiten.)
- Die Kongruenz des Begriffs „Softwarepaket“ auf Seiten des Herausgebers von Security-Advisories und auf Seiten der eventuell betroffenen Distribution. (Wenn ein Security Advisory eine größere Applikation als verwundbar bezeichnet, kann dies je nach Präzision der Angaben auch mehrere Pakete betreffen.)
- Im Sinne der Maschinenlesbarkeit stellen entsprechende Formate wie z.B. XML erhebliche Erleichterungen dar.
- Manche Online-Ressourcen reagieren mit der Sperrung der anfragenden IP-Adresse, wenn zu viele Anfragen pro Zeiteinheit gesendet werden. [14] Deshalb ist es nötig zwischen Anfragen künstlich ein Intervall zu erzeugen. Manche Websites geben dieses Intervall auch explizit über einen Parameter in der Datei `robots.txt` an. [15]
- Verglichen mit dem *rechtlichen Aspekt* sind die oben genannten Punkte relativ „weiche“ Kriterien. Manche Herausgeber [16] [17] hervorragend aufbereiteter Security-Advisories verbieten jedwede Nutzung ihrer Inhalte um ihr eigenes Geschäftsmodell nicht zu gefährden. Andere erlauben wiederum einen relativ freien Umgang mit ihren Informationen. [18] Somit ist vor jeder ernsthaften Beschäftigung mit einer interessanten Datenquelle die rechtliche Sachlage abzuklären.

Zusätzlich zur Maschinenlesbarkeit der aktuellen Security-Advisories ist das Vorhandensein eines **Archivs** wünschenswert, mit dessen Hilfe auch auf historische Sicherheitsschwachstellen zurückgegriffen werden kann.

Dies ist vor allem dann interessant, wenn der Bedrohungsgrad (eigentlich veralteter) *Legacy-Systeme* bestimmt werden soll.

Die oben beschriebenen Aspekte bergen auch Ursachen für **Ungenauigkeiten** in sich. Je geringer die Korrelation zwischen den verwendeten Software-Bezeichnern (z.B. Paketname) der Paketverwaltung eines Systems und jenen veröffentlichter Security-Advisories ist, desto höher wird die Wahrscheinlichkeit für falsch-positive bzw. falsch-negative Aussagen darüber, ob nun ein bestimmtes Advisory ein bestimmtes Paket betrifft oder nicht.
Eine falsch-positive Zuordnung nervt den Menschen, der die Ergebnisse durchgeht. Eine falsch-negative Zuordnung würde bedeuten, dass eine Bedrohung nicht erkannt wird und einem potentiellen Angreifer eine Chance geboten wird. Die obige Überlegung führt zu dem Schluss, dass es sinnvoll ist, die Gewichtung der Korrelationsalgorithmen bei verminderter Übereinstimmung der Software-Bezeichner in Richtung vermehrter *False-Positives* zu verschieben.

Security-Advisory-Feeds von Herstellern bzw. Distributoren enthalten üblicherweise die exaktesten Angaben zu Paketbezeichnungen und Versionsnummern. [19] [20] Allerdings erscheint ein Security-Advisory als Hinweis auf eine Sicherheitsschwachstelle üblicherweise erst, sobald auch eine Lösung für das zugrunde liegende Problem erarbeitet wurde. (*Responsible Disclosure*) Das Advisory transportiert meist nur die Information in welcher Version eines bestimmten Pakets der Fehler behoben worden ist.

Wie bereits zu Beginn der Einführung (Kapitel 2) dargelegt, erreicht die Zahl der Angriffe, die eine konkrete Schwachstelle ausnutzen, ihren Höhepunkt etwa drei Monate nach Veröffentlichung des bzw. der Patches. Trotz der Tatsache, dass hier vielfach der spätestmögliche Zeitpunkt für einen konkreten Hinweis auf ein Sicherheitsproblem gewählt wird, bleibt also noch genügend Zeit, um die aktuellen Paketversionen rechtzeitig einzuspielen.

An dieser Stelle sei auch angemerkt, dass zumindest im deutschsprachigen Raum Open-Source-Hersteller und -Distributoren generell ein gesteigertes Interesse daran haben sollten Mängel ihrer Software offenzulegen. Nach deutschem Recht gilt unentgeltlich zur Verfügung gestellte Software als Schenkung, wobei eine Haftung für arglistig verschwiegene Mängel besteht. [21]

Von Herstellern bzw. Distributoren unabhängige **Sicherheitsdienstleister** konzentrieren ihre Benennungen üblicherweise weniger stark auf die Pakete der vielen Linux- und BSD-Derivate, als viel mehr auf die Bezeichnung des Programms dessen Quelltext Fehler aufweist. [22] Viele Sicherheitsdienstleister untersagen zudem die Nutzung ihrer Inhalte. Die Recherche ergab, dass das NIST [23] und das US-CERT [24] die Nutzung ihrer Daten gestatten. An dieser Stelle sei angemerkt, dass die Nutzungsbedingungen zumeist für die jeweilige Website als ganzes festgelegt werden. Nur das NIST definiert Nutzungsbedingungen [23] speziell für seine

National Vulnerability Database.
Secunia [16] und VUPEN [17] hingegen verbieten generell die Nutzung ihrer Inhalte.

Die Auswirkungen der verschiedenen Lizenzen und Nutzungsbedingungen auf die Implementierung der Extraktion und Verarbeitung von Daten aus Security-Advisories sind in den Abschnitten 10.2 und 12.1 dargelegt. Grundsätzlich muss die (lizenz-)rechtliche Situation für jeden Security-Advisory-Feed, der neu integriert werden soll, gesondert bewertet werden. Wie aus den nachfolgenden Unterabschnitten hervorgeht, beziehen sich die wenigsten Nutzungsbedingungen bzw. Lizenzen direkt auf die Security-Advisory-Feeds. Meist wird die Website als Ganzes angesprochen, wodurch erkennbar ist, dass die Security-Advisory-Feeds sich nicht im Fokus der Verfasser der Nutzungsbedingungen befanden. Dieser Umstand trägt nicht notwendigerweise zu einer erleichterten Beurteilung der jeweiligen rechtlichen Rahmenbedingungen bei.

Die nachfolgende Betrachtung populärer Quellen von Security-Advisories erhebt keinen Anspruch auf Vollständigkeit. Sie stellt lediglich eine erste Vorauswahl des Autors dar, die sich primär daran orientiert für die zum Zeitpunkt der Entstehung der Arbeit verfügbaren Rechner relevante Security-Advisories zu extrahieren. Wie in den nachfolgenden Kapiteln dargelegt werden wird, ist das begleitend zur Arbeit entwickelte Korrelationswerkzeug darauf ausgelegt, dass in Zukunft zusätzliche Datenquellen mit geringstmöglichem Aufwand einbezogen werden können.

2.2.1 Debian Security Advisories

> *„Seit dem 25. Januar 2012 kann neues Material unter den Bedingungen der MIT (Expat) License (normalerweise unter http://www.jclark.com/xml/copying.txt verfügbar) oder, nach Ihrer Wahl, der GNU General Public License (entweder Version 2 der Lizenz oder (nach Ihrer Wahl) jeglicher späteren Version; die neueste Version ist normalerweise unter http://www.gnu.org/licenses/gpl.html verfügbar) weiterverteilt und/oder modifiziert werden.*
>
> *Es wird derzeit daran gearbeitet, das alte Material zu oben genannten Lizenzen konform zu machen. Bis dahin beziehen Sie sich bitte auf die folgenden Bestimmungen der Open Publication License.*
>
> *Dieses Material darf nur unter den Bestimmungen, die in der Open Publication License, Entwurf v1.0 oder später (Sie können unsere lokale Kopie lesen, die neueste Version ist normalerweise unter http://www.opencontent.org/openpub/ verfügbar) festgehalten sind, weitergegeben werden.“ [18]*

Für Debian existieren neben dem *RDF*-Feed (*Resource Description Framework*) seit 1999 ohne Unterbrechung Jahres-Archive von Security-Advisories. [25] Ein Archiv ist sehr simpel aufgebaut: Es ist eine HTML-Datei mit nach Datum absteigend geordneten Links zu den Advisories. Der Text des Links enthält die exakte Bezeichnung des Pakets. [26] Das „Archiv“ ist also eigentlich ein Index für alle im entsprechenden Jahr publizierten Advisories.

Die Debian Security Advisories enthalten [27] [19]

- eine Identifikationsnummer,
- den Titel,
- das Datum der Meldung der Sicherheitsschwachstelle,
- eine Liste der betroffenen Pakete,
- Referenzen zu anderen Security-Datenbanken (meist CVE-Nummern) und
- Informationen welche Paketversionen das Problem für welche Distributionsversion beheben.

Das Problem am letzten Punkt ist das Format in dem die Versionen vorliegen. Sie sind eingebettet in englische Sätze. [27] [19] Diese sind jedoch relativ leicht zu finden, da sie stets die Worte *„fixed in version“* enthalten. Ist ein solcher Satz gefunden, kann davon ausgegangen werden, dass das „Wort“ mit den meisten Ziffern, Interpunktations- und Sonderzeichen die Versionsnummer ist.

Wie die Auswertung genau funktioniert wird in Abschnitt 10.9.1 beschrieben.

2.2.2 Ubuntu security notices

> *„You are welcome to display on your computer, download and print pages from this website for personal, education and non-commercial use only. You must retain copyright, trade mark and other notices unaltered on any copies or printouts you make. Certain materials available on this site are öpen source"materials subject to the GNU General Public Licence ("GPL") or other open-source licence and are so marked. Use of those materials is governed by the individual applicable licence.“ [28]*

Während der Recherche wurden keine auf die Ubuntu security notices anwendbaren Open-Source-Lizenzen gefunden.

Für Ubuntu wird ein RSS-Feed bereitgestellt, der Security-Advisories – die sog. Ubuntu security notices – enthält. [29] Ubuntu listet ebenfalls die einzelnen Security-Advisories in einer HTML-Datei auf. Allerdings sind diese Links nicht nach Zeitabschnitten (z.B. Jahren) gruppiert. Darüber hinaus ist die Anzahl der Security-Advisories pro Seite limitiert, wodurch ein Webcrawler gezwungen wäre die Website „durchzublättern“. Am schwersten wiegt jedoch die Tatsache, dass die Links zu den Security-Advisories *nicht* den genauen Paketnamen enthalten. Für die Auswertung historischer (nicht im RSS-Feed enthaltener) USNs ist demnach die

Erstellung einer lokalen Kopie des Archivs (schon aus Gründen der Performance) sinnvoll. Basierte die Auswertung der USNs auf den HTML-Dateien der Website, müsste jede Datei einzeln unter Einhaltung des von Ubuntu geforderten Zeitintervalls heruntergeladen werden.

Glücklicherweise werden sämtliche USNs auch an eine Mailing-Liste gesandt, welche als MBOX-Datei zur Verfügung gestellt wird, wodurch das Beziehen einer lokalen Kopie wesentlich vereinfacht wird. [30]

Ubuntu Security Notices enthalten [31]

- eine Identifikationsnummer,
- den Titel,
- ein Datum,
- eine kurze Zusammenfassung,
- die Liste betroffener Pakete,
- eine nähere Beschreibung der Schwachstelle,
- Anweisungen zur Aktualisierung der betroffenen Pakete
- und Referenzen zu anderen Security-Datenbanken (meist CVE-Nummern).

Aus dem Datum geht nicht hervor, ob der Zeitpunkt der ersten Publikation oder der Zeitpunkt der letzten Modifikation gemeint ist. In der kurzen Zusammenfassung werden zugrunde liegende Sicherheitsschwachstellen näher beschrieben, jedoch ist dieser Abschnitt für eine maschinelle Auswertung eher ungeeignet, weil sein Ziel ist Verständnis beim (menschlichen) Leser zu schaffen.

Versionsnummern können aus den Aktualisierungsanweisungen extrahiert werden, welche üblicherweise eine Liste von Paketen inkl. Versionsnummern anführen, in denen das Problem behoben worden ist. Die Liste ist nach Distributionsversionen (z.B. Ubuntu 11.10 (Oneiric Ocelot), Ubuntu 12.00 (Precise Pangolin), etc.) gruppiert. Diese Strukturierung erlaubt es die Versionsinformationen (Distributionsversion, Paketname, aktuelle Versionsnummer) mit nur einem geringen Risiko von Fehlinterpretationen zu übernehmen.

Die im Abschnitt Referenzen angeführten Links sind meist mit einer CVE-Nummer bezeichnet und führen zu HTML-Dateien in einem Unterverzeichis einer Subdomain von canonical.com, deren Inhalt der Koordination der Problembehebung dient. [32] Diese Seiten enthalten eine Angabe zur Priorität sowie umfangreichere

Informationen zu Paket- und Distributionsversionen, die das zugrunde liegende Problem beheben. Die Problematik bei der Auswertung besteht darin, dass sich eine CVE-Nummer auf mehrere USNs (und umgekehrt) beziehen kann. Für die meisten der angeführten Pakete wird zugleich eine USN veröffentlicht, scheinbar aber nur für jene Distributionsversionen, für welche noch Support besteht.

Im Rahmen dieser Diplomarbeit wurde die Korrelation (siehe Kapitel 11) auf Basis von *Paket*namen und *Paket*version für ausreichend befunden. In Zukunft ist die Einbeziehung von *Distributions*versionen denkbar. (siehe Abschnitt 14)

Die Details der Auswertung werden in Abschnitt 10.9.2 behandelt.

2.2.3 NIST / National Vulnerability Database

> *„The entire NVD database can be downloaded from this web page for public use. There are no licensing restrictions on using this data, however, we would appreciate being given credit as is appropriate within products, services, and reports that use our data.“ [23]*

Das NIST (National Institute for Standards and Technology) stellt die Informationen der National Vulnerability Database in Form von XML-Dateien zur Verfügung. [23] Die Einträge werden als *Vulnerabilities* bezeichnet und entsprechen dem was im Kontext dieser Diplomarbeit als Security-Advisory bezeichnet wird. Sie entsprechen einem ebenfalls zur Verfügung gestellten XML-Schema und sind somit leicht mit einem XML-Parser zu verarbeiten, d.h. maschinenlesbar.
Eine dieser Dateien enthält alle jüngst publizierten und alle kürzlich veränderten Vulnerabilities. Ihr Inhalt entspricht in Bezug auf die Aktualität den üblicherweise in einem RSS-Feed zu erwartenden Daten.
Für jedes Jahr wird eine XML-Datei angeboten, die alle Vulnerabilities des entsprechenden Jahres enthält. [23] Somit ist das Beziehen einer lokalen Kopie des vollständigen Archivs trivial.

Einträge der NVD [33] enthalten

- eine Identifikationsnummer (CVE-Nummer),
- eine logisch verknüpfte Auflistung verwundbarer Konfigurationen,
- eine Liste verwundbarer Produkte,
- das Datum der ersten Veröffentlichung,
- das Datum der letzten Modifikation,

- die Basis-Metriken des CVSS-Score (Common Vulnerability Scoring System),
- eine standardisierte Beschreibung der Auswirkung der Sicherheitsschwachstelle,
- eine standardisierte Aufzählung der Sicherheitsschwachstellen nach dem CWE-Schema (Common Weakness Enumeration),
- kategorisierte Referenzen zu weiterführenden Informationen,
- eine kurze (nicht maschinenlesbare) Beschreibung,
- zusätzliche Referenzen zu den verwendeten Bewertungskriterien und Benennungsschemata.

Der wesentliche Vorteil der NVD ist, dass ihre Einträge maschinenlesbar sind. Zudem wird eine möglichst objektive Bewertung der Sicherheitsschwachstelle – der CVSS-Score – bereitgestellt.
Als möglicher Nachteil ist die Abweichung des Benennungsschemas von Software von den Namen der in den div. Distributionen eingesetzten Paketen zu nennen. Dies liegt jedoch in der Tatsache begründet, dass das NIST mit der NVD ein möglichst breites Publikum erreichen will und somit nicht jedes Detail der verschiedenen Benennungsschema implementieren kann. [23] Wird in Betracht gezogen, dass die Benennungsschemata standardisiert und logisch verknüpft sind, kann (bei entsprechender Auswertung) von einer präzisen Benennung der verwundbaren Software ausgegangen werden.

Die Details der Auswertung sind in Abschnitt 10.9.3 beschrieben.

2.2.4 US-CERT / Vulnerability Notes Database

> *„At this time, you may not reproduce or distribute documents available on US-CERT's website, in whole or in part, for commercial use. For noncommercial purposes or solely internal distribution, you may reproduce documents from US-CERT's website provided you include a produced by statement." [34]*

Das US-CERT (United States Computer Emergency Readiness Team) bietet einen Atom-Feed der neuesten Vulnerability Notes an. [35] Dabei ist es möglich die Anzahl der Einträge zu spezifizieren. [36]
Dies würde es eigentlich ermöglichen sämtliche Vulnerability-Notes herunterzuladen. Allerdings stellte sich während der Recherche heraus, dass die Ausgabe auf die ersten 1000 Einträge limitiert ist.

Die Einträge des Atom-Feeds [37] enthalten

- der Vulnerability-Note zugeordnete Kategorien,

- Name und Email-Adresse des Autors,
- den Titel,
- einen Link zur ausführlichen HTML-Version der Vulnerability-Note,
- eine Identifikationsnummer,
- das Datum der Veröffentlichung,
- das Datum der letzten Modifikation,
- einen kurzen Überblick,
- eine detaillierte Beschreibung der Sicherheitsschwachstelle,
- eine Beschreibung der möglichen Auswirkungen,
- Informationen über verwundbare bzw. ausgebesserte Softwareversionen und
- den CVSS-Score.

Damit enthält der Atom-Feed jene Informationen, die auch auf der Website www.kb.cert.org bereitgestellt werden.

Es bestünde die Möglichkeit das gesamte Archiv zu extrahieren, indem aus den auf www.kb.cert.org verfügbaren Listen Links zu sämtlichen Vulnerability-Notes extrahiert werden. Links an sich enthalten aber noch keine verwertbaren Informationen. Für jeden Eintrag der Datenbank müsste also eine Anfrage an die Website gestellt werden, wodurch der Datentransfer ineffizienter und der Server einer höheren Belastung ausgesetzt würde. Zudem enthält die National Vulnerability Database Vulnerability-Notes des US-CERT. [38] Sie deckt demnach die historischen Einträge der Vulnerability Notes Database des US-CERT ab.
Auf Basis der genannten Aspekte erscheint es sinnvoll, die Beschränkung der Einträge des Atom-Feeds zugunsten einer effizienteren und simpleren Verarbeitung in Kauf zu nehmen.

Beinahe alle wesentlichen Informationen sind aufgrund der Strukturierung des Inhalts einer Vulnerability-Note maschinenlesbar. Die Ausnahme bilden die Angaben zur Verwundbarkeit bestimmter Softwareversionen. Ähnlich wie bei Debian (siehe Abschnitt 2.2.1) und Ubuntu (siehe Abschnitt 2.2.2) handelt es sich um Text in englischer Sprache. [39] Das Problem hierbei ist, dass die Formulierungen der Sätze häufiger wechseln und teilweise die Bedeutung der Versionsangaben verschieben bzw. ins Gegenteil verkehren. Bei der Implementierung der Auswertung und der damit einhergehenden Analyse der 1000 neuesten Vulnerability-Notes wurden drei verschiedene Bedeutungen einer Versionsangabe ermittelt:

- Der Fehler wurde in dieser Softwareversion behoben.
- Diese Softwareversion ist verwundbar.
- Es existiert (derzeit) keine Lösung für das zugrunde liegende Problem.

Von den bisher genannten Datenquellen (Debian, Ubuntu, NIST) ist das US-CERT aufgrund der nicht maschinenlesbaren Versionsangaben diejenige, die am wenigsten zuverlässig ausgewertet werden kann.

Die Details der Auswertung werden in Abschnitt 10.9.4 behandelt.

2.2.5 Open Source Vulnerability Database

> *„Q. How is OSVDB different than CVE?*
> *A: Common Vulnerabilities and Exposures (CVE) simply provides a standardized name for vulnerabilities, much like a dictionary. OSVDB is database that provides a wealth of information about each vulnerability. Where appropriate, entries in the OSVDB reference their respective CVE names.“ [40]*

Innerhalb der OSVDB werden Security-Advisories als *Vulnerabilities* bezeichnet. Der RSS-Feed, der neue Vulnerabilities auflistet, wurde während des Zeitraums der Recherche deaktiviert. [41]
Die OSVDB bietet eine Rest-API, die es ermöglicht Anfragen direkt an die aktuelle Version der Datenbank zu stellen. Obwohl bei dieser Variante dem Server, der die Rest-API betreibt, zumindest die Namen der eigenen Pakete bekannt gegeben werden, ist sie aus technischer Sicht eleganter, da keine lokale Kopie der Datenbank erforderlich ist. Die API erfordert allerdings eine Registrierung und ist auf hundert Anfragen pro Tag beschränkt. [42]

Der Download von Kopien der Datenbank erfordert ein aktives Benutzerkonto auf osvdb.org. [43]
Zu Beginn der Recherchen (März 2012) waren die Datenbank-Dumps veraltet. Am 9. Juli 2012 wurde nur noch eine Fehlermeldung angezeigt. In einem Blog-Eintrag vom 31. März 2012 beschreibt Brian Martin (einer der Projektleiter [44]) die Gründe für eine Umstrukturierung[45], die das Fortbestehen der OSVDB sichern soll.

Doch selbst wenn die Datenbank-Dumps aktuell wären, erschwert die Lizenz der OSVDB die Auswertung, da die Lizenz nicht weitergegeben werden kann, sondern jede/r Nutzer/in sich selbst um das Zustandekommen des entsprechenden Rechtsgeschäfts kümmern muss:

> *„This License is non-transferable. This means that it applies to you and your company, not the people you distribute the product to in any fashion (e.g., customers or users). Customers or users are subject to the original OSVDB License and Copyright and must retain permission or their own license agreement for further redistribution“ [43]*

Aus dem obigen Auszug der OSVDB-Lizenz geht hervor, dass sich jede/r die/der die Daten der OSVDB ohne Umweg über die Website benutzen möchte, sich auf osvdb.org registrieren muss. Daraus resultiert ein erhöhter Installationsaufwand für Software, die die OSDVB auswertet.

Anstelle der Rest-API könnte auch das angebotene HTML-Suchformular genutzt werden, das käme aus rechtlicher Sicht jedoch wahrscheinlich der Enumeration sämtlicher Einträge der Datenbank anhand ihrer URLs gleich. Obwohl die Daten wieder aus den HTML-Dateien, die die Vulnerability-Informationen beinhalten, extrahiert werden könnten, ist ein solches Vorgehen aus rechtlicher Sicht sehr fragwürdig. Es stünde den Lizenzbestimmungen sowie den implementierten Mechanismen zur Zugriffskontrolle entgegen. Weder in den Lizenzbestimmungen der Datenbank an sich [43] noch in den Nutzungsbedingungen der Website selbst [46] findet sich ein Passus der das oben beschriebene Vorgehen rechtfertigen würde.

Die HTML-Version einer Vulnerability beinhaltet [47]

- den Titel,
- eine Identifikationsnummer,
- das Datum der Bekanntmachung der Sicherheitsschwachstelle,
- das Datum an dem der Hersteller eine Lösung veröffentlicht hat,
- eine Beschreibung,
- eine standardisierte Klassifikation der Vulnerability,
- die Lösung,
- betroffene Produkte inkl. Versionsnummern,
- Referenzen zu anderen Security-Advisory-Datenbanken,
- den CVSS-Base-Score und
- Kommentare.

Zum Zeitpunkt der Fertigstellung dieser Diplomarbeit wurde aufgrund der komplexen Anforderungen der Lizenz aber auch wegen der ungewissen Zukunft der OSVDB von einer Nutzung derselben abgesehen.

2.2.6 Red Hat

> *„Red Hat grants you a personal, non-assignable license to use Red Hat Content for your own internal use **while you are a Red Hat Customer** (as defined in Section 2 above). Distributing any portion of Red Hat Content to a third party, using any Red Hat Content for the benefit of a third party or using Red Hat Content in connection with software other than Red Hat Software under an active Red Hat subscription are all prohibited. [...]“ [48]*

Aus den von Red Hat veröffentlichten Nutzungsbedingungen geht hervor, dass die Nutzung der Inhalte nur für Kunden von Red Hat erlaubt ist. Damit werden die Security-Advisory-Feeds von Red Hat als nicht frei auswertbar betrachtet. Ein/e Spezialist/-in für anglo-amerikanisches Recht wäre hier (evt. aufgrund des *Fair-Use*-Prinzips [49]) vielleicht anderer Meinung, jedoch stand eine solche Person während des Verfassens dieser Diplomarbeit nicht zur Verfügung.

Die Security-Advisories von Red Hat weisen Referenzen zu jenen der National Vulnerability Database auf, welche (annähernd) die gleichen Informationen enthalten. [50] [51] Damit deckt die NVD (zumindest teilweise) Produkte von Red Hat ab.

2.2.7 CentOS

Während der Recherche konnten kein Dokument gefunden werden, welches sich explizit mit den Nutzungsbedingungen der Inhalte des CentOS-Projekts beschäftigt. Nur in der Fußnote der Website sind Informationen zum Copyright vermerkt:

> *„’Linux’ is a registered trademark of Linus Torvalds. | All other trademarks are property of their respective owners. | All other content is Copyright @ 2004-2009 by the CentOS Project or ’each individual contributor (forums, comments, etc.) unless otherwise assigned’.[...]“ [52]*

Dies lässt gemäß dem *Fair-Use*-Prinzip [49] die nicht-kommerzielle, interne Nutzung zu.

Das CentOS-Projekt stellt lediglich eine Mailing-Liste [53] zur Verfügung, deren monatliche Zusammenfassungen heruntergeladen werden können. Die Einträge bzw. Emails dieser Liste bestehen im Wesentlichen aus einer einfach zu verarbeitenden Liste von Paketnamen, welche die jeweilige Versionsnummer bereits enthalten,

sowie einer Referenz zum entsprechenden Security-Advisory von Red Hat. [54] Es wird keine Beschreibung der Sicherheitsschwachstelle angegeben.
D.h. obwohl sich verwundbare und die Sicherheitsschwachstelle behebende Paketversionen leicht aus dem Inhalt ableiten lassen, kann – sofern Inhalte von Red Hat ausgespart bleiben – nie ein Bedrohungsgrad abgeleitet werden.

2.3 Fazit der einführenden Überlegungen

Die Informationen, die aus Paketverwaltungssystemen extrahiert werden können, sind im Kontext dieser Diplomarbeit die zuverlässigsten, da sie maschineller Verarbeitung entstammen. Auf Basis dieser Daten lässt sich die automatisierte Erstellung und Wartung eines Software-Inventars (siehe Kapitel 9) relativ einfach implementieren.

Mit dieser Datenbasis können nun die Security-Advisories, deren Nutzung erlaubt ist, nach relevanten Einträgen durchsucht werden. Salopp formuliert „weiß“ die Komponente, die die diversen Websites durchsucht, nun wonach sie „fragen“ muss. Dies reduziert die zu verarbeitende Datenmenge und somit den notwendigen Datenverkehr sowie die Rechenlast auf das notwendige Minimum.

Die verschiedenen Datenquellen weisen in ihrer Strukturierung durchaus Gemeinsamkeiten auf. Diese lassen es zwar sinnvoll erscheinen Teile der Auswertungsprogramme in wiederverwendbarer Weise zu implementieren, jedoch ist es nötig eine auswertende Komponente pro Datenquelle zu schaffen. In den folgenden Kapiteln wird dieser Gedanke detailliert ausgeführt.

Aufgabenstellung

Die Forschungsfrage (siehe Abschnitt 1.3) zielt auf die Korrelation der Daten von Paketverwaltungssoftware und Security-Advisories ab. Die Datenverarbeitung lässt sich grob mit Hilfe der folgenden drei Stufen beschreiben:

1. **Extraktion** relevanter Daten aus den in Abschnitt 2 genannten Datenquellen in einem normalisierten Format.
2. Speicherung dieser Daten an zentraler Stelle und **Korrelation** derselben, um die Bedrohungslevel zu ermitteln.
3. **Darstellung** der Ergebnisse der Korrelation, sodass sie möglichst schnell und intuitiv erfassbar sind und leicht auf weiterführende Informationen zugegriffen werden kann.

Abgesehen von der reinen Erfüllung der Forschungsfrage soll die im Rahmen dieser Diplomarbeit geschriebene Software als Freie-Software-Projekt eine dezentrale Weiterentwicklung ermöglichen. Daraus, und aus der Anforderung den Code wartbar und vor allem lesbar zu halten, ergibt sich die Sinnhaftigkeit eines modularen Aufbaus der im nächsten Abschnitt (4) beschrieben wird.

Architektur und Design

Abbildung 4.1 zeigt grob die an der Korrelation beteiligten Komponenten und Akteure. Als Name für die im Zuge dieser Diplomarbeit entwickelten Software wurde Wizzan gewählt. (*Wizzan* ist die althochdeutsche Form des Wortes *Wissen*. [55, S. 136f]) Entsprechend enthalten die Namen aller Komponenten der Software das Präfix *Wizzan*. Dies ermöglicht auch eine explizite Bezeichnung der Komponenten von Wizzan als solche und beugt Missverständnissen bei der Verwendung üblicher Begriffe wie *Agent*, *Crawler* oder *Server* in einem allgemeineren Sinn vor.

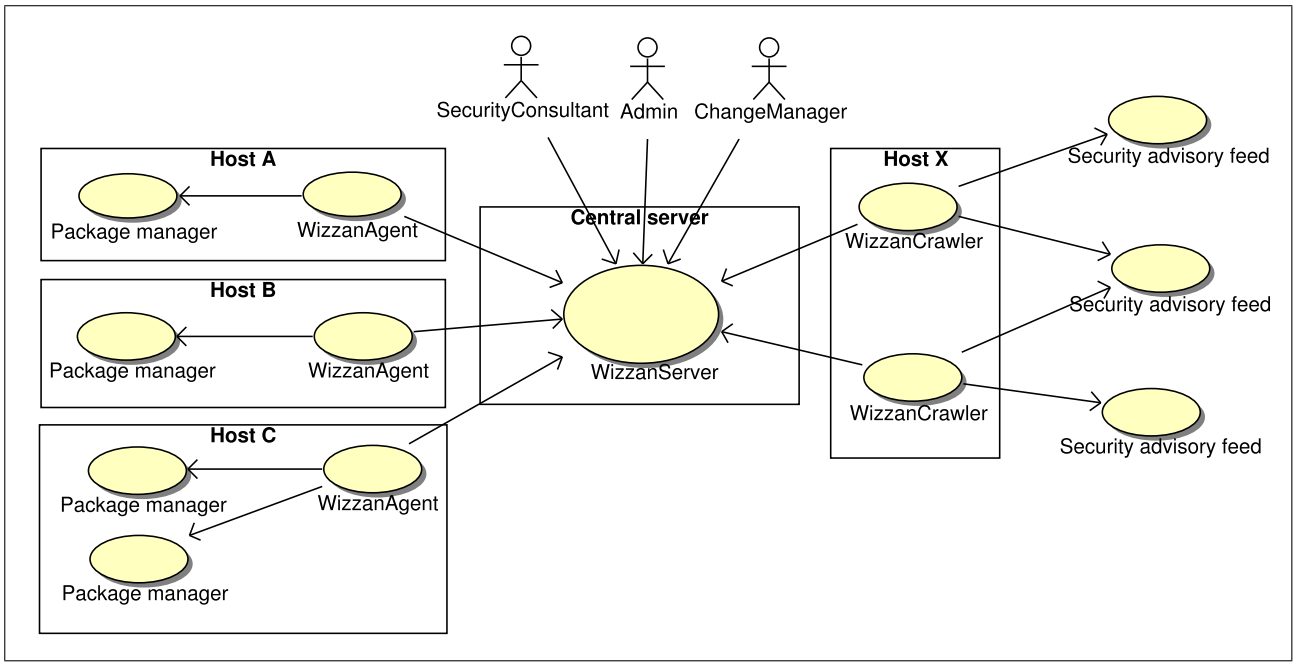

Abbildung 4.1: Gesamtüberblick

Ein **WizzanAgent** wird auf einem Host installiert, dessen Bedrohungsgrad durch Sicherheitsschwachstel-

len ermittelt werden soll. Er liest die Daten aus den am Host verfügbaren Paketmanagern aus, normalisiert sie und leitet diese in Form von (siehe Abschnitt 7.2) *installierten Paketen* und *verfügbaren Updates* an den WizzanServer weiter.

Ein **WizzanCrawler** lädt Security-Advisiories von den diversen Websites herunter und wertet deren Inhalte aus. Die normalisierten Ergebnisse dieser Auswertung werden als *Advisories* an den WizzanServer gesandt. Auf welchem Rechner ein WizzanCrawler installiert ist, spielt eine untergeordnete Rolle. Wichtig ist nur, dass Zugriff aufs Internet besteht.

WizzanAgent und WizzanCrawler verwenden sogenannte ***Plugins***, die sich jeweils der Verarbeitung einer einzelnen Datenquelle (Paketmanager bzw. Website) widmen. Soll eine neue Datenquelle erschlossen werden, so muss lediglich ein neues Plugin implementiert werden, der Großteil des Quelltextes bleibt unangetastet. Dies ermöglicht Entwicklern/-innen die rasche Programmierung eigener Plugins. Die beiden Komponenten sind in Python geschrieben, welches sich hervorragend für die schnelle Entwicklung von Prototypen eignet und eine rasche Anpassung an veränderte Gegebenheiten ermöglicht. [56, S.6]

Der **WizzanServer** ist der Dreh- und Angelpunkt von Wizzan. Unter Verwendung offener Standards soll die Kommunikation mit mehreren Benutzer/-innen bzw. Maschinen zeitgleich erfolgen können. Er ist als Webapplikation konzeptioniert, da ein Webserver den nötigen stabilen Unterbau (TCP-Verbindungen, HTTP, HTTPS, etc.) bietet und somit die Entwicklungsarbeit auf das nötige Minimum beschränkt bleibt. Als Code-Basis dient das PHP-Framework Kohana, welches sich besonders gut für die rasche und saubere Entwicklung von Webapplikationen eignet. [57] Nähere Details hierzu sind in Kapitel 8 zu finden.

Der WizzanServer verwaltet die Informationen über

- Hosts,
- verwendetes kryptographisches Material,
- installierte und verfügbare Pakete,
- relevante Security-Advisories sowie
- die Beziehungen dieser Entitäten zueinander.

Die Aufbereitung und Darstellung dieser Beziehungen in intuitiv erfassbarer Weise stellt den wesentlichen Nutzen für Anwender von Wizzan dar. Werden in diese Verknüpfungen die Angaben zur Kritikalität von Sicherheitsschwachstellen, die in Security-Advisories zu finden sind, einbezogen, so lässt sich daraus für Hosts und Pakete der jeweilige Bedrohungsgrad ableiten.

4.1 Bedrohungsmodell (Threat Model) und Sicherheitsfunktionen

Die innerhalb von Wizzan zusammenlaufenden, detaillierten Informationen über die von Wizzan überwachte IT-Infrastruktur sind ein lohnendes Ziel für potentielle Angriffe und somit besonders schützenswert. Um diesen zu begegnen, wurden im Vorfeld der Implementierungen mögliche Bedrohungen und Gegenmaßnahmen nach Microsoft STRIDE [58] ermittelt.

STRIDE ist eine einfache und übersichtliche Methode um Bedrohungen (*Threats*) und Gegenmaßnahmen (*Mitigations*) einander gegenüberzustellen. [58] Dabei ist das übergeordnete Ziel herauszufinden, welche möglichen Auswirkungen von Sicherheitsschwachstellen mit welchem Aufwand verhindert bzw. reduziert werden können und daraus eine Priorisierung der Umsetzungsschritte abzuleiten.
Ein Beispiel: Der richtige Einsatz erprobter Kryptographie-Bibliotheken ist üblicherweise relativ einfach zu bewerkstelligen und verhindert gleichzeitig die Offenlegung und Manipulation von Daten sowie das Vorspiegeln einer falschen Identität.

STRIDE ist ein Akronym der folgenden Begriffe, die die verschiedenen Bedrohungskategorien widerspiegeln. [58]

- ***Spoofing identity*** – Das Vorspiegeln einer falschen Identität.
- ***Tampering with data*** – Böswillige Manipulation von Daten
- ***Repudiation*** – Abstreitbarkeit von Handlungen
- ***Information disclosure*** – Offenlegung von Daten gegenüber Personen, denen sie eigentlich verborgen bleiben sollten.
- ***Denial of Service*** – Verweigerung des Zugriffs für berechtigte Benutzer. (Z.B. indem sämtlicher Hauptspeicher belegt oder der Prozessor voll ausgelastet wird.)
- ***Elevation of privilege*** – Ein/e unprivilegierte/r Benutzer/-in verschafft sich (unerlaubter Weise) höhere Zugriffsrechte am System.

Abbildung 4.2 stellt das Bedrohungsmodell dar, das aufzeigt, welche Bedrohungen die verschiedenen Komponenten zutreffen und welche Gegenmaßnahmen daraus abzuleiten sind. Auf eine Priorisierung der Maßnahmen wird verzichtet. Wizzan stellt eine Neuentwicklung dar, d.h. alle Sicherheitsfunktionen werden (im Vergleich zum „Nachrüsten“ bereits vorhandener Software) mit nur geringem Mehraufwand implementiert.

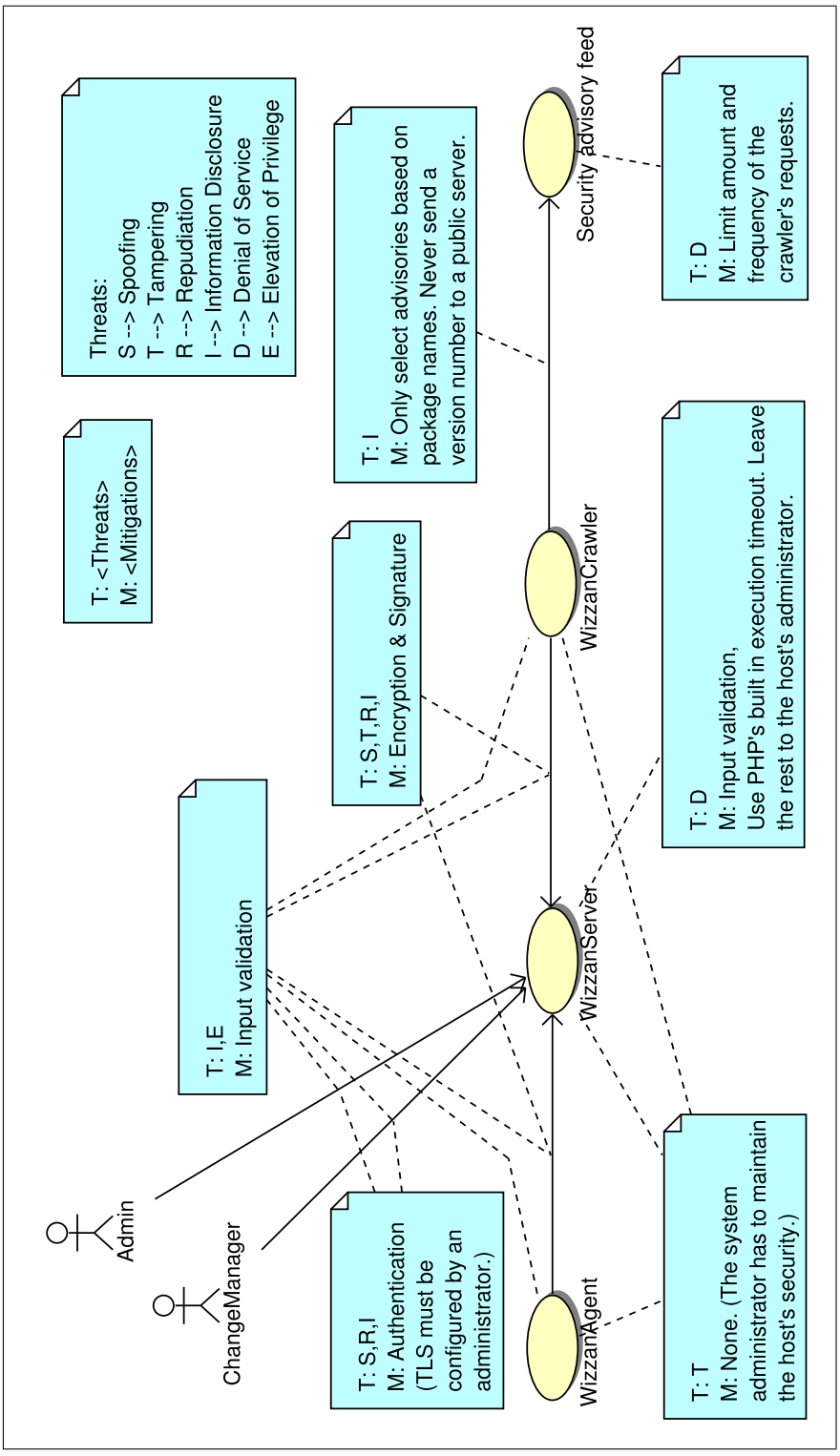

Abbildung 4.2: Threat model

4.1.1 WizzanAgent

Der **WizzanAgent** ist im Wesentlichen den selben Bedrohungen ausgesetzt wie der Host, auf dem er installiert ist. Tampering ist also dann auszuschließen, wenn der Host nicht kompromittiert ist. Außer Rückmeldungen über Erfolg oder Misserfolg seiner Anfrage an den WizzanServer erhält, erwartet und verarbeitet der WizzanAgent keine Daten vom Server. Der WizzanAgent wird periodisch vom Betriebssystem gestartet und beendet sich nach Abschluss seiner Tätigkeit selbst. D.h. er ist zu keinem Zeitpunkt via Netzwerk erreichbar. Eine Kompromittierung auf Basis der Kommunikation mit dem WizzanServer ist somit nahezu ausgeschlossen.

Der WizzanAgent sollte nur mit den unbedingt erforderlichen Rechten am System ausgeführt werden. Auf vielen unixoiden Systemen sind hierfür der Benutzer nobody und die Gruppe **nogroup** vorgesehen. Um im Fall einer von Wizzan unabhängigen Kompromittierung des Hosts das kryptographische Material zu schützen, empfiehlt sich jedoch die Verwendung eines eigenen Benutzers, der exklusiven Zugriff auf den privaten Schlüssel hat.

4.1.2 WizzanCrawler

Auf den **WizzanCrawler** treffen weitgehend dieselben Bedrohungen zu wie auf den WizzanAgent. Dies gilt insbesondere für die Verwendung eines eigenen nicht privilegierten Benutzers zum Schutz des kryptographischen Materials. Der wesentliche Unterschied ist aber, dass die Eingabedaten (die Website, die den Security-Advisory-Feed bereitstellt) theoretisch beliebig manipuliert sein könnten, um Sicherheitsschwachstellen im WizzanCrawler auszunutzen. Die Abwehrmaßnahme besteht darin Daten von öffentlich zugänglichen Quellen niemals als ausführbaren Code zu interpretieren. Schlagen notwendige Umwandlungen in andere Datentypen fehl, werden die entsprechenden Security-Advisories übersprungen. Damit sollte sich die Angriffsfläche auf jene der verwendeten Programmiersprache reduzieren.
Dennoch sollte der WizzanCrawler nie auf dem selben Host wie der WizzanServer betrieben werden, damit die Kompromittierung eines WizzanCrawlers möglichst geringe Auswirkungen auf die Sicherheit bzw. Unversehrtheit des WizzanServers hat.

Desweiteren dürfen aus den Anfragen an öffentlich verfügbare Websites keine Rückschlüsse auf eingesetzte Softwareversionen gezogen werden können.
Existierte die zwingende Vorgabe gar keine Rückschlüsse auf die Gegebenheiten der eigenen Infrastruktur zu erlauben, so müsste der WizzanCrawler den relevanten Teil der Website in seiner Gesamtheit herunterladen und auf Dauer synchron halten. In manchen Fällen trifft diese Verhaltensweise aufgrund äußerer Gegebenheiten auch zu. Die Abschnitte 10.9.3 und 10.9.4 beschreiben Beispiele hierfür.

Das beschriebene Vorgehen ist in manchen Fällen aus technischer Sicht weit komplexer, als auf Basis eines vorhandenen Index nur die relevanten Security-Advisories herunterzuladen. Deshalb nutzt der WizzanCrawler, sofern es sinnvoll ist, auch diese Möglichkeit (siehe Abschnitte 10.9.1 und 10.9.2). Um erkennen zu können, welche Security-Advisories relevant sind, fordert der WizzanCrawler vom WizzanServer *Listen von Paketennamen* an (siehe Abschnitt 10.3) und vergleicht diese mit den verfügbaren Daten des Archivs. Diese vereinfachte Darstellung wird in Abschnitt 10.4 konkretisiert. Aus den Anfragen an öffentliche Rechner sind in dieser Situation demnach bestenfalls die Paket*namen* ersichtlich.

Der einzige Weg, die beschriebene Zuordnung eines Profils der IT-Infrastruktur zur eigenen IP-Adresse bzw. zum eigenen Unternehmen zuverlässig zu verhindern, ist die richtige Nutzung eines Anonymisierungsnetzwerks.

4.1.3 Security-Advisory-Feeds

Zusätzlich kann der WizzanCrawler bzw. sein Verhalten eine Bedrohung für **Security-Advisory-Feeds** bedeuten. Wenn ein einzelner WizzanCrawler am Limit der Leistungsfähigkeit seines Hosts eine Website nach Security-Advisories durchsucht, kann dies für den Server bereits eine Lastspitze bedeuten. Tritt nun der Fall der Verbreitung von Wizzan ein, könnten mehrere WizzanCrawler-Installationen einen Effekt bewirken, der sich auf Seiten des Servers wie ein *Distributed Denial of Service* ausnimmt. Um dieses Szenario zu verhindern, implementiert der WizzanCrawler konfigurierbare Zeitintervalle zwischen den Anfragen an den Server, der die Security-Advisories bereitstellt.

4.1.4 WizzanServer

Der **WizzanServer** ist aufgrund seiner zentralen Funktion und seiner ständigen Erreichbarkeit am stärksten durch mögliche Sicherheitsschwachstellen gefährdet. Wie auch für WizzanAgent und WizzanCrawler ist die Kompromittierung des Hosts, auf dem der WizzanServer installiert ist, aus Sicherheitssicht gleichbedeutend mit dessen Kompromittierung. Zusätzlich müssen am Server Maßnahmen gegen eine mögliche Überlastung getroffen werden, die zu einer Unterbrechung des Service führen könnte. Der WizzanServer stützt sich hier hauptsächlich auf das *Execution-Timeout* der zugrundeliegenden Sprache PHP. Der/Die zuständige Administrator/-in ist hier in der Pflicht eine ausreichend gesicherte Plattform bereitzustellen.

Die Validierung sämtlicher Eingabedaten ist die wichtigste Sicherheitsmaßnahme, die in den WizzanServer implementiert ist. Der Abschnitt 8 geht hier auf die wichtigsten Details ein.

Um die Vertraulichkeit, Integrität und Authentizität jener Nachrichten zu gewährleisten, die zwischen WizzanAgent und WizzanServer bzw. WizzanCrawler und WizzanServer ausgetauscht werden, kommt jeweils auf beiden Seiten GnuPG (GNU Privacy Guard) zum Einsatz.
Kapitel 5 beschäftigt sich mit der sicheren Kommunikation zwischen den genannten Komponenten.

Auch für die Kommunikation zwischen Benutzern bzw. deren Browsern und dem WizzanServer sind Vertraulichkeit, Integrität und Authentizität zu gewährleisten.
Die Authentizität der Benutzer/-innen wird durch Eingabe von Benutzer(innen)name und Passwort sichergestellt. Es werden klarerweise nur die *HMACs* der Passworte, *nicht* die Passworte selbst in der Datenbank hinterlegt. Rollenbasierte Zugriffskontrolle ermöglicht eine granulare Einstellung der Zugriffsrechte. Für den Anfang reichen die Rollen *Benutzer/-in* und *Administrator/-in*. Benutzer/-innen dürfen die von Wizzan generierten Daten einsehen und in begrenztem Maße modifizieren. Administratoren/-innen haben dieselben Rechte wie Benutzer/-innen und darüber hinaus das Recht die Konfiguration der Webapplikation (sofern es das WebUI erlaubt) zu verändern.
Vertraulichkeit und Integrität bietet der Einsatz von TLS. Hier obliegt es wieder dem/der zuständigen Administrator/-in für eine korrekte Implementierung (Integration eines validen Zertifikats in den Webserver, etc.) zu sorgen.

Kommunikation zwischen den Komponenten

Dieses Kapitel beschreibt die Hintergründe sowie die Implementierung der Kommunikation zwischen den Wizzan-Komponenten (WizzanServer, WizzanAgent und WizzanCrawler).

WizzanAgent und WizzanCrawler verwenden dieselbe Software-Bibliothek für das Versenden von Nachrichten. Sie werden deshalb um der erleichterten Lesbarkeit willen zusammenfassend als WizzanClients bezeichnet.

Kommunikation erfolgt immer zwischen einem WizzanClient und dem WizzanServer, niemals zwischen zwei WizzanClients direkt. (Dies wäre auch mangels offenem TCP-Port auf Seiten der WizzanClients nicht möglich.)

Da der WizzanServer als Webapplikation konzeptioniert ist (siehe Abschnitt 4 und 8), wird für die Übertragungen der Nachrichten HTTP oder wahlweise HTTPS eingesetzt. Es erfolgt im Fall einer *zusätzlichen* Absicherung der Kommunikation durch HTTPS allerdings derzeit keine Überprüfung des Zertifikats durch den WizzanClient, da die Authentifikation nicht auf den Verbindungen, sondern auf den Nachrichten selbst basiert. Diese Herangehensweise begünstigt den unkomplizierten Einsatz eines HTTP-Proxy, sofern der/die Administrator/-in einen solchen vorsehen wollte.

Die Verschlüsselung der Nachrichten erfolgt durch die Webapplikation auf Basis von GnuPG bzw. OpenPGP, nicht durch den Webserver auf Basis von TLS.
GnuPG bietet hybride Kryptographie auf Basis des OpenPGP-Standards. [59] Das bedeutet: Der Schutz der zwischen einem WizzanClient und dem WizzanServer ausgetauschten Nachrichten orientiert sich an Email-Verschlüsselung.

5.1 Verteilung und Verifikation des Schlüsselmaterials

Die nachfolgenden Betrachtungen orientieren sich maßgeblich am Einsatz von Wizzan in Netzwerken mit mehr als 20 zu überwachenden Rechnern. Der Anteil manueller Arbeiten ist somit möglichst gering zu halten. Es wird angenommen, jedoch nicht vorausgesetzt, dass Konfigurationsverwaltungswerkzeuge zur zentralisierten Administration von Software auf den zu überwachenden Rechnern eingesetzt werden.

5.2 Deployment-Phase

Der Begriff *Deployment-Phase* fasst an dieser Stelle alle Schritte im Zuge der Installation von Wizzan-Komponenten zusammen, die nicht durch diese Komponenten selbst durchgeführt werden können. Diese Abgrenzung wurde gewählt, da die Wizzan-Komponenten selbstständig Schlüsselmaterial austauschen.

Nach der Installation des **WizzanServers** generiert der/die Administrator/-in mit GnuPG manuell ein Schlüsselpaar bestehend aus öffentlichem und privatem Schlüssel.
Der öffentliche Schlüssel wird in der Konfiguration aller **WizzanClients** bei deren Installation hinterlegt und ist somit aus Sicht des WizzanClients verifiziert.

5.3 Operativer Einsatz

Ein WizzanClient übermittelt als eindeutiges Erkennungsmerkmal den *Fingerprint* seines öffentlichen Schlüssels. Anhand dieses Fingerprints prüft der WizzanServer die Signatur der Nachricht und versucht diese im Erfolgsfall zu entschlüsseln. Nur Nachrichten, bei denen Verifikation und Entschlüsselung erfolgreich waren, werden weiterverarbeitet.
Jeder WizzanClient übermittelt Basis-Informationen (Hostname, *Fully Qualified Domain Name*) über das System, auf dem er installiert ist.

Bleibt noch das Problem der **Verifikation des Fingerprints** des WizzanClients durch den/die Administrator/-in des WizzanServer. [60, S.10]
Erst wenn der Fingerprint und somit der WizzanClient als authentisch gilt, können die von ihm übermittelten Informationen als vertrauenswürdig angesehen werden.

Nachdem ein **WizzanCrawler** potentiell sämtliche Security-Advisory-Feeds erreichen kann, reicht eine bzw. reichen einige wenige Installationen aus. Für diese kann die *zwingend erforderliche*, manuelle Verifikati-

on des Fingerprints problemlos durchgeführt werden. Ein weiteres Argument spricht für eine Verifikation: Der WizzanCrawler ist in der Lage eine Liste aller Paketnamen vom WizzanServer anzufordern (siehe Abschnitt 10.3). Diese Daten werden vom WizzanServer nur herausgegeben, wenn die Authentizität des Kommunikationspartners gegeben ist.

Bei einem Massen-Deployment von **WizzanAgents** ist die manuelle Prüfung der Fingerprints schlicht unrealistisch. Aus diesem Grund wird die Verifikation des Fingerprints eines WizzanAgents vom WizzanServer nicht berücksichtigt. Dies bedeutet allerdings, dass ein potentieller Angreifer jederzeit neue Hosts sowie mit diesen verknüpfte Software-Versionen oder falsche Informationen und Paketversionen zu einen bestehenden Host in Wizzan integrieren könnte.

Um diesem Umstand zu begegnen, werden abhängig vom übermittelten FQDN Maßnahmen zur Falsifikation der Authentizität des Fingerprints bzw. des WizzanAgents ergriffen:

- Ist der **FQDN bekannt**, so muss der entsprechende Host-Eintrag der Datenbank bereits mit dem Fingerprint des für die Übertragung verwendeten Schlüssels verknüpft sein. Ansonsten kann von einem Angriff (oder einer Neuinstallation des WizzanAgents inkl. der Löschung der Konfigurationsdateien) ausgegangen werden.
- Ist der **FQDN *un*bekannt**, so liegen keinerlei Daten vor, anhand derer die Authentizität des Fingerprints falsifizierbar wäre. Spoofing der Informationen eines überwachten Rechners ist nur dann möglich, wenn der FQDN noch nicht in der Datenbank existiert. Aus diesem Grund ist der/die Administrator/-in des WizzanServer in der Lage eine Deadline zu setzen, bis zu der sich WizzanAgents selbstständig in die Wizzan-Datenbank eintragen können. Die Begrenzung durch eine Deadline wurde gewählt, weil der/die Administrator/-in so nicht darauf vergessen kann die automatische Eintragung der WizzanAgents wieder zu deaktivieren.

Natürlich werden alle Verstöße gegen die beschriebenen Regeln in einer Log-Datei festgehalten. Das Erkennen von Angriffen ist somit möglich.

5.4 Nachrichtenformat

In diesem Abschnitt wird die *Struktur* der Nachrichten, die von den WizzanClients an den WizzanServer gesandt werden, sowie dessen Response behandelt. Die nachfolgenden Kapitel gehen detaillierter auf den Inhalt ein.

Die Grundidee ist die zu übertragenden Daten mit Hilfe von JSON [61, S.1f] als Zeichenkette darzustellen, diese zu verschlüsseln, zu signieren und dann die Übertragung zu starten. Die Antwort wird vom WizzanServer in der gleichen Weise kodiert.

JSON wurde gegenüber XML-RPC der Vorzug gegeben, da es in der Anwendung simpler und schneller ist als XML. [61, S.6] Desweiteren kommen *Remoteprozeduraufrufe* (in dem Umfang wie XML-RPC sie bietet) innerhalb von Wizzan nur sehr beschränkt zum Einsatz. (Abschnitt 10.3 beschreibt die Abfrage von Paketlisten durch den WizzanCrawler, die im weitesten Sinne einen RPC darstellt.)

Der WizzanServer erwartet folgende Parameter in einem POST-Request eines WizzanClients:

- **`fingerprint`** – Ein WizzanClient muss zwingend den Fingerprint seines öffentlichen Schlüssels angeben. Dies ermöglicht dem WizzanServer eine wesentlich schnellere Verarbeitung der empfangenen Daten. Klarerweise wird mit Hilfe von GnuPG überprüft, ob das empfangene, kryptographische Material zum angegebenen Fingerprint passt.
- **`ciphertext`** – Die Nutzdaten werden zuerst mit JSON kodiert und anschließend mit dem öffentlichen Schlüssel des WizzanServers verschlüsselt.
- **`signature`** – Der *Ciphertext* wird separat mit dem privaten Schlüssel des WizzanClients signiert. Die Signatur wird separat in diesem Parameter transportiert.
- **`public_key`** – Ein WizzanClient sendet seinen öffentlichen Schlüssel an den WizzanServer, sofern ihn dieser dazu auffordert.

Bis auf die allgemeine Fehlermeldung und die Aufforderung den öffentlichen Schlüssel zu senden verschlüsselt und signiert der WizzanServer seine Antworten ebenfalls. Die folgenden stehen ihm zu Verfügung:

- **`ok`** – Der erfolgreiche Empfang der Daten wird bestätigt.
- **`send_your_public_key`** – Fordert den WizzanClient auf seinen öffentlichen Schlüssel zu senden, damit der WizzanServer ihn in seine Datenbank aufnehmen kann. (Absatz 5.3 geht auf die Verifikation des Schlüsselmaterials ein.)
- **`fail`** – Diese unspezifische Fehlermeldung zeigt zwar den Fehlschlag der Anfrage an, jedoch erlaubt sie keine Rückschlüsse auf den Grund, um potentielle Angreifer nicht mit zusätzlichen Informationen zu versorgen. Der/Die zuständige Administrator/-in kann den genauen Grund des Fehlschlags aus den Log-Dateien des WizzanServer bzw. des jeweiligen WizzanClients ersehen.

- **Eine Liste neu hinzugefügter Pakete** kann vom WizzanClient angefordert werden, um auf dieser Basis die relevanten Security-Advisories auswählen zu können.

Zum Erstellen verschlüsselter Antworten bedient sich der WizzanServer der selben Vorgehensweise wie der WizzanClient. (Kodieren mit JSON – Verschlüsseln – Anhängen einer Signatur)
Da dem WizzanServer die separaten Parameter eines POST-Request nicht zur Verfügung stehen, kodiert er analog zum WizzanClient die Parameter `ciphertext` und `signature` mit Hilfe von JSON, bevor er den Response sendet. Für die übrigen beiden Parameter (`fingerprint` und `public_key`) besteht keine Verwendung.

Die Nutzdaten selbst können folgende in JSON kodierte Parameter enthalten:

- Parameter, die von WizzanAgent und WizzanCrawler verwendet werden:
  - **`role`** – Mit diesem gibt ein WizzanClient bekannt, ob er `agent` oder `crawler` ist. Diese Angabe ist zum Zwecke der effektiven Input-Validierung zwingend erforderlich.
  - **`sysinfo`** – Wie bereits in Abschnitt 5.3 erwähnt übermitteln alle WizzanClients ein Minimum an Information über ihren Host. Beim WizzanAgent werden diese Basisinformationen erweitert, sodass eine umfassende Beschreibung des Betriebssystems entsteht. Dies ist in Abschnitt 7 beschrieben.
- Parameter, die ausschließlich vom WizzanAgent verwendet werden:
  - **`installed_packages`** – Die Liste *installierter Pakete* des Hosts.
  - **`updatable_packages`** – Die Liste von *Updates* die für den Hosts verfügbar sind.
- Parameter, die ausschließlich vom WizzanCrawler verwendet werden:
  - **`advisories`** – Die Liste von Security-Advisories, die vom WizzanCrawler als relevant eingestuft wurden.

Agent & Crawler: Separated at birth

Wie das vorige Kapitel 5 bereits anklingen lässt, verwenden WizzanAgent und WizzanCrawler für die Kommunikation mit dem WizzanServer dieselben Funktionen. Darüber hinaus ähnelt sich die Struktur ihres Codes sehr stark, da der WizzanCrawler aus der Code-Basis des WizzanAgents entwickelt wurde.
Funktionen, die gleichermaßen von WizzanAgent und WizzanCrawler verwendet werden, sind in der **WizzanBaseLib** zusammengefasst.

Diese Funktionen können in die Bereiche **Konfiguration**, **Logging**, **Kryptographie** (GnuPG; siehe Kapitel 5), **RFC3339-konforme Zeitangaben**, Auslesen von **Host-Informationen** und dynamisches **Laden von Plugins** eingeteilt werden.

Zusätzlich bringt die WizzanBaseLib eine Funktion zum **Testen der Verbindung** zum WizzanServer mit. Dieser Test der Erreichbarkeit ist nötig, da die WizzanClients ihre Ergebnisse nicht zwischenspeichern. Durch den Wegfall der Zwischenspeicherung entfällt einerseits die sonst nötige Bereitstellung und Verwaltung von Festplattenspeicher, andererseits sind flüchtige Daten für einen potentiellen Angreifer ungleich schwieriger auszuwerten. Es hat demnach wenig Sinn Datenquellen auszulesen, wenn im Vorfeld unklar ist, ob diese Daten im Anschluss auch an den WizzanServer gesandt werden können, der diese schlussendlich dauerhaft speichert.

Für die **Anbindung an GnuPG** wird das Python-Modul GnuPGInterface verwendet. Es wird von allen großen Linux-Distributionen unterstützt [62]. GnuPGInterface verwendet File-Deskriptoren, um mit GnuPG zu kommunizieren. [63] GnuPG stellt selbst die Integrität des *Schlüsselbunds* sicher. Die Leistungsfähigkeit dieser Konstruktion sollte leicht für WizzanClients ausreichen.

Das **Plugin-System** besteht im Wesentlichen aus der Konfigurationsdatei des jeweiligen WizzanClients, welche die zu ladenden Plugins benennt und konfiguriert, einem Unterverzeichnis, das diese Plugins enthält und dem entsprechenden Code der WizzanBaseLib.
Ein Plugin besteht aus einer Datei (einem sogenannten *Python-Modul*), die den Code enthält, der zur Auswertung einer bestimmten Datenquelle erforderlich ist. Die Python-Module und auch die in ihnen enthaltenen

Klassen, die den Code bzw. die Funktionalität kapseln, sind nach der Datenquelle benannt, die sie auswerten. Auf die verschiedenen Plugins wird in den Kapiteln 7 und 10 detailliert eingegangen.

Als administrative Gemeinsamkeit von WizzanAgent und WizzanCrawler ist zu nennen, dass beide periodisch vom Betriebssystem zur Ausführung gebracht werden müssen. Der Vorteil dieser Herangehensweise liegt darin, dass die WizzanClients nicht als Dienste bzw. *Daemons* konzipiert werden müssen, wodurch die Komplexität der Umsetzung auf das nötige Minimum beschränkt wird. Es ist dennoch wichtig darauf zu achten, dass das Intervall groß genug ist, dass der WizzanClient seine Aufgaben vollständig erledigen kann. Ohnehin sollte es reichen die WizzanClients alle sechs bis zwölf Stunden auszuführen.
Sinnvollerweise werden diese periodischen Starts von einem sogenannten ***Cronjob*** bewerkstelligt, der bei der Installation mit eingerichtet wird. Paketverwaltungssysteme unterstützen üblicherweise die automatische Einrichtung von Cronjobs im Zuge der automatisierten Konfiguration am Ende des Installationsvorgangs. Dies ist bei der Paketierung der Software der WizzanClients zu beachten und ggf. zu nutzen, um die anwendenden Administratoren/-innen zu entlasten.

Das Minimum an **Informationen über den Host** eines WizzanClients, das der WizzanServer für die Erledigung seiner Aufgaben benötigt, sind der *Hostname* und der *Fully Qualified Domain Name*. Auf die Ermittlung dieser Daten wird im nächsten Kapitel (7) detailliert eingegangen.

Auswertung von Hosts und Paketmanagern

Der WizzanAgent extrahiert Daten aus den Paketmanagern, die auf seinem Host installiert sind, ebenso wie Informationen über den Host selbst. Anschließend sendet er die normalisierten Daten in der in Abschnitt 5.4 beschriebenen Weise an den WizzanServer.

Die Abbildungen 7.1 und 7.2 illustrieren die Inhalte der nachfolgenden beiden Abschnitte 7.1 und 7.2.

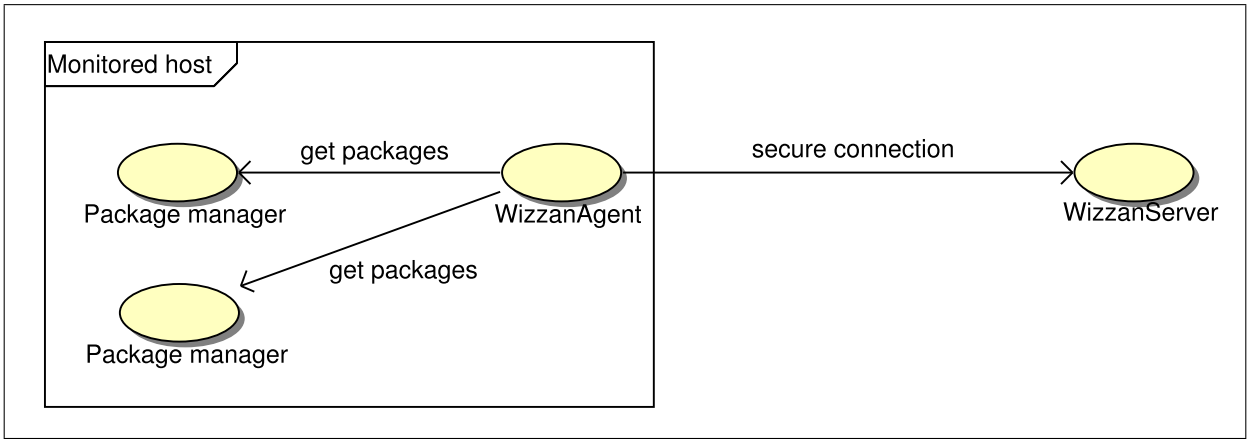

Abbildung 7.1: Anwendungsfall-Diagramm des WizzanAgents

Es folgt ein Überblick über die Arbeitsweise des WizzanAgents anhand der Abbildung 7.2.

Im Vorfeld der Extraktion von Paketdaten wird die Konfigurationsdatei gelesen (1) und ein Objekt instantiiert, welches bei seiner Erstellung die System-Informationen ausliest (2). Anschließend werden die Plugins geladen (3) und der Verbindungstest (4) durchgeführt. (Zu den Schritten 1 bis 4 bietet Kapitel 6 detaillierte Informationen.) In Schritt 5 wird jedes Plugin angewiesen Daten über installierte Pakete (6) und verfügbare Updates (7) aus dem jeweiligen Paketmanager zu extrahieren. Sobald die Extraktion abgeschlossen ist, wird eine Nachricht mit allen erhobenen Daten an den WizzanServer gesendet (8 und 9), welcher diese dann in sein

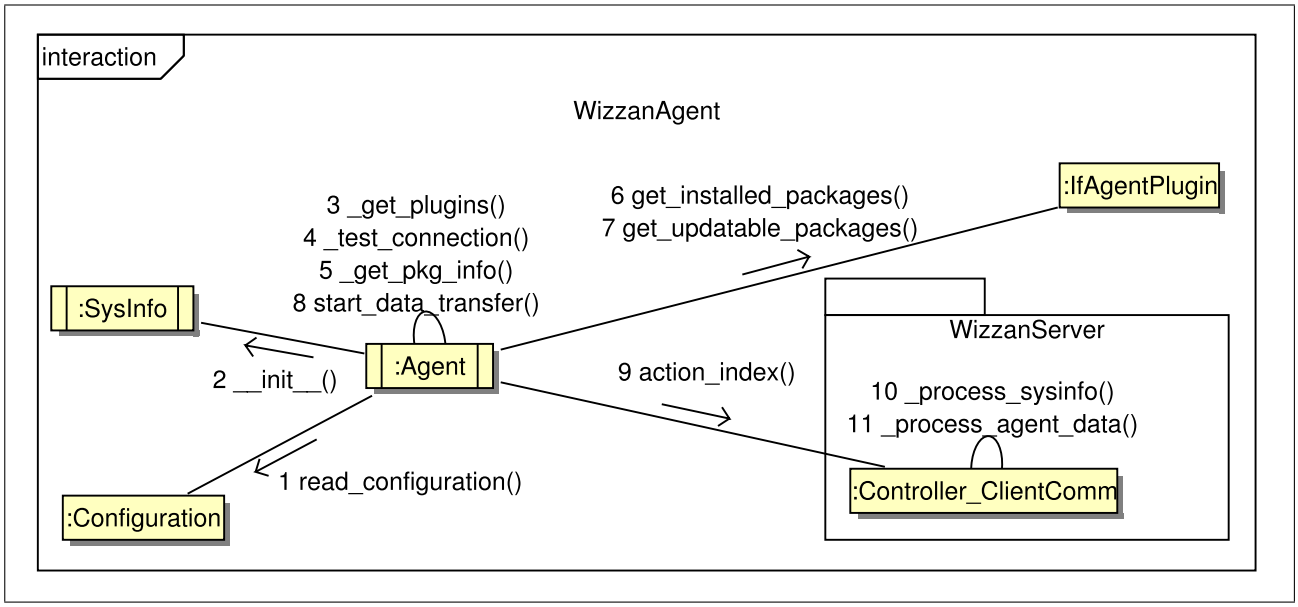

Abbildung 7.2: Kommunikationsdiagramm des WizzanAgents

Inventar (siehe Kapitel 9) einspeist (10 und 11).

Die nachfolgenden Abschnitte beschreiben die wesentlichen Aspekte dieses Vorgangs in ihren Details.

7.1 Systeminformationen

Die in Abschnitt 5.3 angesprochenen Basis-Informationen werden erweitert, sodass ein detailliertes Bild des Hosts entsteht, auf dem der WizzanAgent ausgeführt wird. Bis auf den FQDN werden alle Daten mit Hilfe des Python-Moduls platform ausgelesen. Dieses Modul zeichnet sich durch seine Plattformunabhängigkeit [64] aus, indem es generell auf jedem System funktioniert, das Python in der Version 2.6 (oder höher) unterstützt. Für die Ermittlung des FQDN wird das Python-Modul socket eingesetzt, welches auf der *Socket*-Schnittstelle von BSD aufsetzt, und somit auf der Mehrheit der Betriebssysteme verfügbar ist. [65]

- **`hostname`** – Der Name des Hosts, der den WizzanClient ausführt.
- **`fqdn`** – Der *Fully Qualified Domain Name* des Hosts. Da dieser vom WizzanServer zur eindeutigen Identifizierung von Hosts verwendet wird, verhindert die WizzanBaseLib bereits im Vorfeld, dass Werte übergeben werden, die `localhost` oder `localdomain` enthalten.

- **`architecture`** – Die Prozessorarchitektur. Beispiele sind: `i368` und `amd64`
- **`kernel`** – Die Bezeichnung des Kernels bzw. des umgebenden Systems. (Beispielsweise könnte dieser Parameter auch den Wert `Java` annehmen.)
- **`kernelrelease`** – Die Version des Kernels bzw. des umgebenden Systems. (z.B. `2.2.0` oder `NT`)
- **`python_version`** – Die Version des Python-Interpreters, der zum Ausführen des WizzanAgents verwendet wird.
- **`distid`** – Der Name der *nix-Distribution an sich. (z.B. `Ubuntu`, `CentOS` oder `Debian`)
- **`distcodename`** – Der Codename eines bestimmten *Release* einer *nix-Distribution wird oft alternativ zur Versionsnummer des Release verwendet. (Beispielwerte sind: `squeeze` oder `precise`)
- **`distrelease`** – Die Release-Nummer der *nix-Distribution. (z.B. `12.04`)

Die letzten drei Parameter sind nur auf unixoiden Plattformen verfügbar. Bei einer (möglichen) Portierung des WizzanAgents auf Windows würde die Funktion, die im Hintergrund verwendet wird, einfach leere Werte zurückliefern.

7.2 Paketinformationen

Das Auslesen der Daten verschiedener Paketmanager ist die zentrale Aufgabe des WizzanAgents. Dabei ist die Grundidee das Maximum an verfügbaren Informationen zu extrahieren, ohne einen manuellen Eingriff durch den/die Administrator/-in – z.B. eine Anpassung der Konfiguration – erforderlich zu machen.

Das Plugin-System kommt diesem Anliegen sehr entgegen. Jedes Plugin prüft selbstständig, ob die Voraussetzungen für eine erfolgreiche Auswertung überhaupt gegeben sind. In der Praxis reicht es zu prüfen, ob ein bestimmter Paketmanager installiert ist. Ist dies nicht der Fall, beendet sich das Plugin selbst, ansonsten führt es die Auswertung durch.

Indem der WizzanAgent nun alle verfügbaren Plugins nacheinander aufruft, erzielt er eine maximale Informationsausbeute. Verfügt ein System über mehrere vom WizzanAgent unterstützte Paketverwaltungssysteme, so können alle ohne gegenseitige Einschränkungen ausgewertet werden.
Gleichzeitig sinkt der Administrationsaufwand, da nicht anwendbare Plugins automatisch übersprungen werden. Der/Die Administrator/-in kann somit für die gesamte IT-Infrastruktur eine Konfiguration verwenden, in

der alle verfügbaren Plugins aktiviert sind, ohne Gefahr zu laufen, dass der WizzanAgent nicht mehr korrekt funktioniert.

Aus den Daten eines Paketmanagers können grundsätzlich zwei Kategorien von Paketlisten generiert werden: **installierte Pakete** und **Pakete, die als Updates zur Verfügung stehen**.
Es wäre auch denkbar nur eine Liste zu generieren und die Pakete als *installiert* bzw. *verfügbar* zu markieren. Nachdem Python jedoch eine objektorientierte Sprache ist, die über das Mittel der Vererbung verfügt, existiert hier eine probatere Lösung um die Wirklichkeit von Paketmanagern abzubilden.

Der WizzanServer geht – wie in Kapitel 9 beschrieben – davon aus, dass der WizzanAgent in jeder Nachricht, die Paketinformationen enthält, **vollständige Paketlisten** überträgt. Das versetzt ihn nämlich in die Lage erkennen zu können, welche Pakete obsolet sind und aus dem Inventar gelöscht werden können.

Auf Seiten des WizzanAgents ist ein Paket recht simpel strukturiert, erst der WizzanServer fügt im Zuge der Korrelation weitere Merkmale und Verknüpfungen hinzu. (Siehe Kapitel 9)

- **`name`** – Der Name des Pakets, so wie er vom Paketmanager angegeben wird. (z.B. `libsmbclient-dev`)
- **`version`** – Die Version des Pakets, so wie vom Paketmanager angegeben. An dieser Stelle ist bereits erkennbar, dass die Versionierungschemata von Distributor und Software-Entwicklern/-innen nicht exakt übereinstimmen.
  Beispielsweise deutet Zusatz `-2ubuntu2.2` der Versionsangabe `2:3.6.3-2ubuntu2.2` auf eine fortlaufende Anpassung durch die Entwickler(innen)gemeinden von Debian und Ubuntu hin. [66]
- **`description`** – Eine Beschreibung des Pakets wird nur hinzugefügt, sofern der jeweilige Paketmanager eine solche liefert.
- **`rfc3339_timestamp`** – Es wird eine menschenlesbare, eindeutiges UTC-Zeitangabe verwendet, um den Erstellungszeitpunkt eines Pakets zu markieren.
- **`source_plugin`** – Der Name des Plugins, das das Paket ausgelesen hat.

Aus Sicherheitsgründen sollte der WizzanAgent nur mit geringen Rechten am System ausgeführt werden. Teilweise können geringe Berechtigungen aber auch ein Problem darstellen. So kann beispielsweise auf einem auf **Debian** basierenden System nur der Benutzer **root** den Zwischenspeicher des Paketmanagers **apt** aktualisieren. Dies ist aber zwingend erforderlich, um Informationen über aktuell für den Host verfügbare Updates zu erhalten. Eine mögliche Abhilfe bietet hier ein Cronjob, der – unabhängig vom WizzanAgent, aber zeitlich ei-

nige Minuten vor dessen Start (siehe Kapitel 6) – als **root** den Befehl zum Aktualisieren des Zwischenspeichers ausführt.

Um Missverständnissen vorzubeugen sei noch erwähnt, dass ein Plugin nicht notwendigerweise die Extraktion von *installierten Paketen* **und** *verfügbaren Updates* implementieren muss. Eine Spezialisierung auf eine der beiden Kategorien bei gleichzeitigem Ignorieren der anderen ist je nach verwendetem Paketmanager sinnvoll bis unumgänglich.

7.3 Debian Package Manager

Listing 7.1 stellt die ersten Zeilen der Ausgabe von dpkg dar. Wie aus den ersten beiden Zeilen (gewünschter und aktueller Status) ersichtlich ist und in Abschnitt 2.1.1 bereits beschrieben wurde, kümmert sich dpkg rein um die *Installation* von Paketen. Das bedeutet zwar, dass auch Zustände wie „Nur noch Konfigurationsdateien vorhanden“ oder „Installation nicht abgeschlossen“ ausgelesen werden könnten, doch interessieren diese im Kontext dieser Diplomarbeit nicht. D.h. auf Basis von dpkg lässt sich nur die Liste installierter Pakete befüllen.

```
% dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig
  -pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name              Version          Description
+++-==============-==============-=====================================
    =====
ii  accountsservic 0.6.15-2ubuntu query and manipulate user account
   informatio
ii  acl             2.2.51-5ubuntu Access control list utilities
ii  acpi            1.6-1          displays information on ACPI devices
ii  acpi-support    0.140          scripts for handling many ACPI events
ii  acpid           1:2.0.10-1ubun Advanced Configuration and Power
   Interface e
ii  adduser         3.113ubuntu2   add and remove users and groups
```

Listing 7.1: Ausgabe von dpkg

Das Plugin verarbeitet die Ausgabe von dpkg zeilenweise und spaltet dabei jede Zeile in Statusfeld, Name, Versionsnummer und Beschreibung auf. Ist einer der beiden Status „installiert“ bzw. „zu installieren“, so wird ein Objekt vom `InstalledPackage` erzeugt, dessen Membervariablen anschließen die genannten Spaltenwerten der betreffenden Zeile zugewiesen werden. Diese Objekte werden vom Plugin zu einer Liste zusammengefasst, welche anschließend an die Liste der installierten Pakete des WizzanClient angefügt wird.

7.4 Advanced Packaging Tool

Wie gegen Ende des Abschnitts 7.2 bereits erwähnt, sind die Informationen des apt-Plugins nur so aktuell wie der *Apt-Cache*. Eine mögliche Abhilfe ist, den Apt-Cache per Cronjob aktuell zu halten. Dazu führt dieser das folgende Kommando aus, welches in Listing 7.2 dargestellt ist. Das Kommando testet zuerst, ob das benötigte Programm apt-get überhaupt verfügbar ist. Im Erfolgsfall wird apt-get verwendet, um den Apt-Cache zu aktualisieren. Die Standard-Ausgabe (nicht eventuelle Fehlermeldungen!) wird dabei verworfen.

```
test -x /usr/bin/apt-get && /usr/bin/apt-get update > /dev/null
```

Listing 7.2: Kommando zum Aktualisieren des Apt-Cache

apt-get bietet die Möglichkeit ein Upgrade des Systems nur zu simulieren und die dabei aktualisierten Pakete inklusive ihrer Versionsnummern anzuzeigen. Das Ergebnis eines solchen Trockenlaufs zeigt Listing 7.3. Die Zusammenfassung im oberen Teil der Ausgabe ist für die Zwecke des Plugins uninteressant.

```
% apt-get -s -u upgrade 2>/dev/null
NOTE: This is only a simulation!
      apt-get needs root privileges for real execution.
      Keep also in mind that locking is deactivated,
      so don't depend on the relevance to the real current situation!
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages have been kept back:
  libatk1-ruby1.8 libcairo-ruby1.8 libgdk-pixbuf2-ruby1.8
  libglib2-ruby1.8 libgtk2-ruby1.8 libpango1-ruby1.8
```

```
The following packages will be upgraded:
  icedtea-6-jre-cacao icedtea-6-jre-jamvm openjdk-6-jdk openjdk-6-jre
  openjdk-6-jre-headless openjdk-6-jre-lib
6 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
Inst icedtea-6-jre-cacao [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0
   .12.04.1 Ubuntu:12.04/precise-updates [amd64]) []
Inst openjdk-6-jre-lib [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0
   .12.04.1 Ubuntu:12.04/precise-updates [all]) []
Inst icedtea-6-jre-jamvm [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0
   .12.04.1 Ubuntu:12.04/precise-updates [amd64]) []
Inst openjdk-6-jre-headless [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0
   .12.04.1 Ubuntu:12.04/precise-updates [amd64])
Inst openjdk-6-jre [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0.12.04.1
   Ubuntu:12.04/precise-updates [amd64])
Inst openjdk-6-jdk [6b24-1.11.1-4ubuntu3] (6b24-1.11.3-1ubuntu0.12.04.1
   Ubuntu:12.04/precise-updates [amd64])
Conf openjdk-6-jre-headless (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/
   precise-updates [amd64])
Conf openjdk-6-jre-lib (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/precise
   -updates [all])
Conf icedtea-6-jre-cacao (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/
   precise-updates [amd64])
Conf icedtea-6-jre-jamvm (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/
   precise-updates [amd64])
Conf openjdk-6-jre (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/precise-
   updates [amd64])
Conf openjdk-6-jdk (6b24-1.11.3-1ubuntu0.12.04.1 Ubuntu:12.04/precise-
   updates [amd64])
```

Listing 7.3: Ausgabe des Simulation einer Systemaktualisierung mit apt-get upgrade

Lediglich Zeilen, die mit `Inst` beginnen, sind relevant, denn sie enthalten den Paketnamen und in *runden* Klammern die neuen Paketversionen. Die Paketbeschreibung wird mit Hilfe des Kommandos apt-cache show separat ermittelt. Analog zu dpkg (Abschnitt 7.3) werden Objekte vom Typ `UpdatablePackage` instanziiert, mit den entsprechenden Werten versehen und zu einer Liste verfügbarer Updates Zusammengefasst, die

schlussendlich dem WizzanAgent übergeben wird.

Dem geschulten Auge wird nicht entgehen, dass in diesem Beispiel die eigentliche Versionsnummer in der Form `Zahl.Zahl.Zahl` von einem Präfix und einem Suffix umgeben ist. Auf die Problematik, dass das Präfix die Wertigkeit der gesamten Versionsangabe verändert, sobald diese für lexikalische (bzw. wo möglich numerische) Vergleiche herangezogen wird, geht Abschnitt 11.3.1 detailliert ein.

WizzanServer

Wie Abbildung 4.1 zeigt, ist der WizzanServer das zentrale Element von Wizzan, in welchem die Rohdaten zusammenlaufen. Er bereitet diese auf und leitet aus ihnen mittels Korrelation (siehe Kapitel 11) Bedrohungslevel für die überwachten Hosts bzw. deren Pakete ab.

Zudem muss der WizzanServer rasch und flexibel an zukünftige Anforderungen angepasst werden können. Neuen Entwicklern/-innen soll ein schneller Einstieg möglich sein.

Aus den genannten Überlegungen lassen sich folgende Anforderungen an den WizzanServer bzw. an das ihm zugrunde liegende Framework ableiten, welche im Zuge der Implementierung schließlich auch erfüllt worden sind.

- Der WizzanServer empfängt und speichert die Rohdaten der WizzanClients und ermittelt aus ihnen die Bedrohungslevel der überwachten Hosts.
- Die intuitive Benutzeroberfläche gestattet die Bedienung ohne vorherige Schulungen.
- Alle Informationen sind so aufbereitet, dass sich aus den Verknüpfungen in der Benutzeroberfläche gleichsam die Zusammenhänge innerhalb der Datenbank und somit die Ergebnisse der Korrelation erschließen.
- Wie in Abschnitt 4.1.4 bereits angeschnitten ist es für Anwender/-innen zwingend erforderlich sich am WizzanServer anzumelden (Authentifikation). Damit die entsprechenden Zugriffsregelungen schnell und schlüssig implementiert werden können, muss das Framework eine fertige Implementierung zur Autorisierung von Benutzern bieten.
- Validierung und Normalisierung von Eingabedaten, sowie die Maskierung von Output tragen wesentlich zur Steigerung der Sicherheit von Wizzan bei. (vgl. Abschnitt 4.1)

- Wizzan kann leicht in andere Sprachen übersetzt werden, da die Benutzeroberfläche ihre Texte aus eigens vorgesehenen Sprachdateien bezieht.
- Der WizzanServer ist in der weit verbreiteten Sprache PHP geschrieben. Somit ist sichergestellt, dass genügend Programmierer vorhanden sind, die in der Lage sind, (für das Projekt, für deren eigenes Unternehmen, etc.) Anpassungen vorzunehmen. Außerdem ist eine solche Sprache bereits auf die meisten Betriebssysteme portiert worden, der/die Administrator/-in muss oft nur noch die entsprechenden Pakete installieren.
- Eine steile Lernkurve und ausführliche Dokumentation ermöglichen Entwicklern/-innen die zügige Entwicklung neuer Features.
- Die zusätzliche Integration von Wizzan unabhängiger Module ins Framework ist und bleibt möglich. Durch diese Wiederverwendung von Code lässt sich der Funktionsumfang von Wizzan leichter erweitern.
- Um die Verbreitung von Wizzan zu fördern, wird die Software unter eine von der FSF (Free Software Foundation) anerkannte Freie-Software-Lizenz gestellt und ihr Quellcode öffentlich zugänglich gemacht.

8.1 Framework

Als Grundlage für die Entwicklung des WizzanServer dient das PHP5-Framework **Kohana** in der Version 3.2. Während der Recherche zeigte sich, dass Kohana die oben genannten Anforderungen am vollständigsten abdeckt.

Kohana steht unter der modifizierten BSD-Lizenz [67], welche die freie Nutzung des Quelltextes in großzügiger Weise erlaubt, und von der Free Software Foundation als zur GNU General Public License kompatibel [68] angesehen wird.

Wie viele andere Frameworks auch verfolgt Kohana das *HMVC*-Pattern (*Hierarchical Model View Controller*). [69] Das *MVC-Pattern* beschreibt ein Architekturmodell zur Strukturierung von Code in die drei Kategorien *Model* (Speicherung von Daten), *View* (Darstellung von Daten) und *Controller* (die eigentliche „Geschäfts“-Logik). Diese Herangehensweise stellt sicher, das der Quelltext stets sauber strukturiert und lesbar bleibt, wodurch die Wartbarkeit auch bei wechselnden bzw. mehreren am Code arbeitenden Entwicklern/-innen gewährleistet bleibt. Das „Hierarchical“ in *HMVC* beschreibt im Kontext von Kohana dessen Fähigkeit

ausgehend von einem Controller interne Anfragen an weitere Controller zu senden, die ihrerseits wieder eine MVC-Triade aufrufen.

Kohana beinhaltet ein *ORM*-Modul (*Objekt Relational Mapping*) [70], welches das *Active-Record-Pattern* umsetzt und es ermöglicht PHP-Objekte und deren Beziehungen zueinander ohne Mehraufwand für den Programmierer in einer relationalen Datenbank zu speichern.
Dieser Umstand erleichtert die Implementierung des WizzanServer erheblich, denn die Korrelation besteht im weitesten Sinn darin aus den Rohdaten Objekte zu generieren und diese zueinander in Beziehung zu setzen.

Des weiteren ist in Kohana Input-Validierung direkt in ORM integriert. [70] Für jedes Datenmodell (z.B. `Model_Host` oder `Model_Advisory` werden Filter und Validierungsregeln definiert, um mögliche Angriffe über die Eingabedaten abzuwehren. Filter werden wie auch Validierungsregeln für jede Membervariable eines Datenmodells definiert. Filter modifizieren die Eingabedaten [71], sodass diese gefahrlos in der Datenbank gespeichert werden können. Die Validierungsregeln prüfen unmittelbar vor der Speicherung, ob die (ggf. gefilterten) Daten eines Objekts den Anforderungen des Datenmodells genügen. [72] Im Erfolgsfall wird das Objekt gespeichert, im Fehlerfall wird eine *Exception* geworfen.

ORM maskiert intern alle Zeichen, die zu einer *SQL-Injections* führen könnten. [73] Sofern anstelle von ORM (etwa aus Gründen der Performance) Datenbank-Anfragen direkt an den Datenbank-Server gestellt werden, kommen *Prepared Statements* zum Einsatz, welche SQL-Injections praktisch unmöglich machen.
Nur wenn die Validierung durch das Datenmodell nicht möglich oder sinnvoll ist, werden Filterung und/oder Validierung in einem Controller durchgeführt.

Um *XSS* (*Cross-Site Scripting*) zu verhindern, werden HTML-Tags generell aus allen übermittelten Texten entfernt. Zudem werden potentiell missbräuchlich verwendbare Zeichen in der Ausgabe (d.h. in der generierten HTML-Seite) maskiert, sodass sie vom Browser nur als Text wahrgenommen werden.

8.2 Datenbanken

Der WizzanServer nutzt zwei voneinander unabhängige Datenbanken. Für Hilfszwecke, die sich rein auf die Webapplikation an sich beziehen (z.B. die Benutzerverwaltung), wird die Standard-Datenbank von Kohana verwendet.
Die eigentlichen Nutzdaten von Wizzan sowie die Ergebnisse der Korrelation werden in einer separaten Datenbank-Instanz gespeichert, die fortan *Wizzan-Datenbank* oder kurz ***WizzanDB*** genannt wird.

Diese Trennung hat folgende Vorteile:

- Die Wizzan-Datenbank kann anderen Applikationen zugänglich gemacht werden, ohne dass (wie in einer kombinierten Datenbank) zwingend Berechtigungen festgelegt werden müssen. Dies bedeutet auch, dass das einfachere Datenbank-Schema mit höherer Wahrscheinlichkeit kompatibel zu anderen SQL-Datenbank-Systemen ist.
- Das Framework bzw. dessen Module und somit das Datenbank-Schema der Standard-Datenbank-Instanz können unabhängig von der Wizzan-Datenbank-Instanz aktualisiert werden.

Es sei an dieser Stelle noch erwähnt, dass der WizzanServer für den Zugriff auf die WizzanDB zwei Datenbank-User einsetzt, je nachdem ob schreibender Zugriff (*CRUD*: *Create Read Update Delete*) für eine Datenbank-Anfrage erforderlich ist oder lesender Zugriff ausreicht. Dadurch wird für die überwiegende Mehrheit der lesenden Zugriffe auf die Datenbank das Risiko einer Kompromittierung minimiert.

8.3 Schema der Wizzan-Datenbank

Abbildung 8.1 stellt das Datenbank-Schema der WizzanDB dar. Dieses wird nun in seinen wesentlichen Punkten erläutert, um dem/der Leser/-in das Verständnis der nachfolgenden Kapitel zu erleichtern.

Zuerst wird auf jene Tabellen eingegangen, die eine eher untergeordnete Rolle spielen, um den Weg für die Kernthemen von Wizzan zu ebnen.
Die beiden *Views* newestpackages und newestpackages_unsorted_duplicates dienen lediglich zur Optimierung der Performance von Abfragen der Liste neu hinzugefügter Pakete. Auf die Funktion dieser Liste geht Abschnitt 10.3 ein.
Die Tabelle advisory_queue stellt lediglich einen Pufferspeicher für die umfangreichen Daten der Wizzan-Crawlers zur Verfügung. Abschnitt 11.1 beschreibt die Abarbeitung dieser Warteschlange.

Die Tabelle **hosts** enthält pro von Wizzan überwachtem Host einen Eintrag. Die Bedeutung der Tabellenspalten geht aus Abschnitt 7.1 hervor.
Jedem Host sind ein oder mehrere öffentliche Schlüssel von WizzanClients (d.h. Einträge in der Tabelle **public_keys** zugeordnet. (siehe Kapitel 5) Für diese Relation existieren in der WizzanDB keine *Foreign-Key-Constraints*. Dies ist auf den Umstand zurückzuführen, dass ein öffentlicher Schlüssel dem WizzanServer zum Zweck der Kommunikation bekannt sein muss, *bevor* der Host-Eintrag angelegt werden kann. Um nun dieses Henne-Ei-Problem zu lösen, sorgt an dieser Stelle der WizzanServer auf Applikationsebene für die referentielle Integrität.

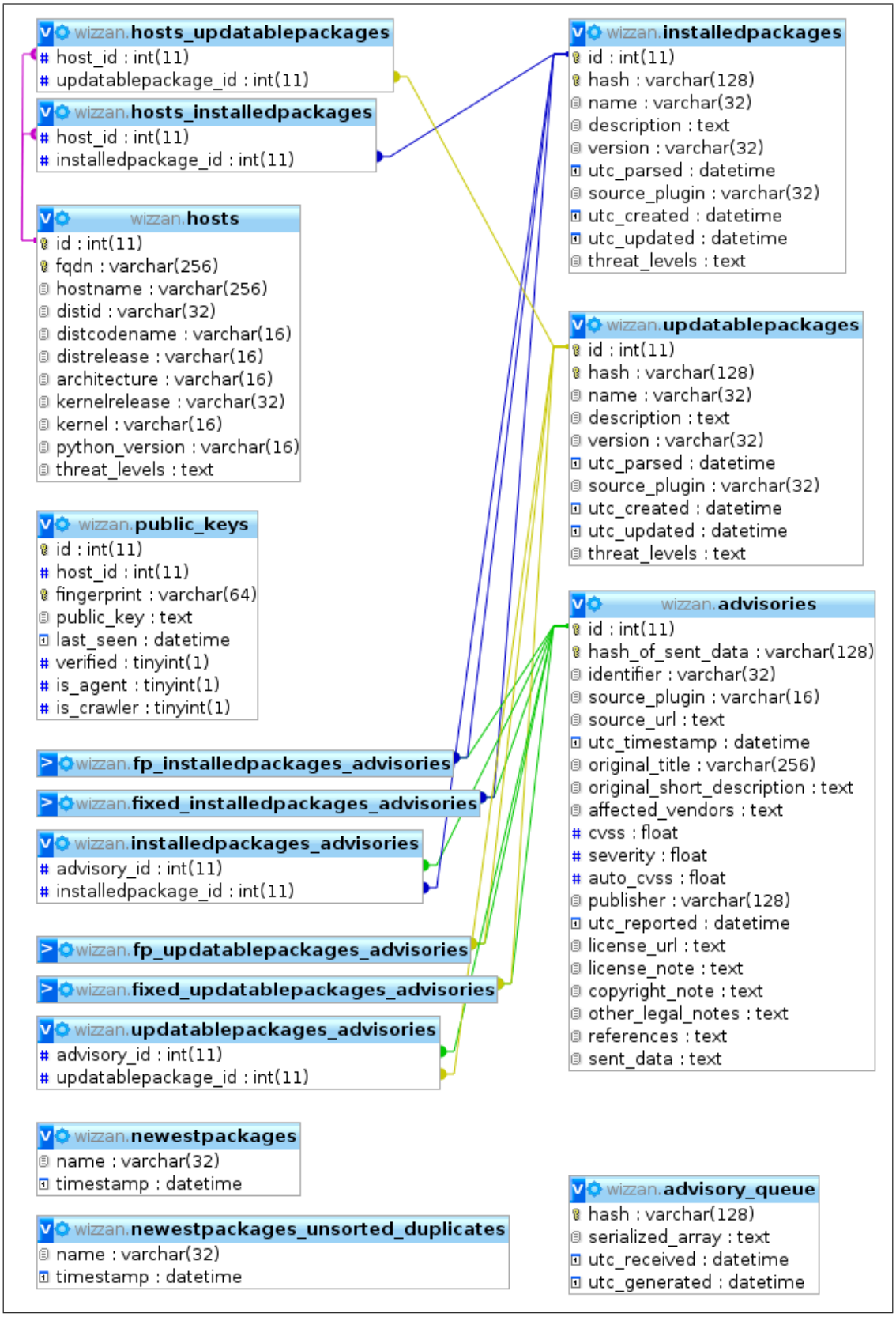

Abbildung 8.1: Schema der Wizzan-Datenbank

Für jede der beiden Kategorien von Paketen (*installiert*/*verfügbar*) steht eine eigene Tabelle (**installedpackages** bzw. **updatablepackages**) zur Verfügung. Dieses Vorgehen ist nötig, da das ORM von Kohana für die Speicherung von Instanzen von Datenmodellen (Objekte) jeweils eine eigene Tabelle verwendet.
Hosts und Pakete stehen zueinander in einer n:m-Beziehung. Die Abbildung dieser Beziehung erfolgt durch die *Pivot*-Tabellen **hosts_installedpackages** bzw. **hosts_updatablepackages**. Auf den Inhalt dieser Paket- und Pivot-Tabellen wird in Abschnitt 7.2 und Kapitel 9 eingegangen.

Die Tabelle **advisories** enthält die aus der Advisory-Warteschlange entnommenen, normalisierten und für relevant befundenen Security-Advisories. Verwundbare Pakete und Advisories stehen zueinander wiederum in einer n:m-Beziehungen, welche in den Tabellen **installedpackages_advisories** und **updatablepackages_advisories** abgebildet wird. Diese Beziehungen können durch den/die Benutzer/-in als falsch-positiv gekennzeichnet werden. Die geschieht im Hintergrund über einen Eintrag in einer der Tabellen **fp_installedpackages_advisories** bzw. **fp_updatablepackages_advisories**. Analog dazu existieren auch Beziehungen zwischen Advisories und Paketen, die das im Advisory beschriebene Problem beheben. Diese Beziehungen werden in den Tabellen **fixed_installedpackages_advisories** und **fixed_updatablepackages_advisories** abgebildet. Der Inhalt dieser Tabellen wird in Kapitel 11 erläutert.

Aus dieser Konstellation ist erkennbar, dass der Bedrohungsgrad von Hosts und Paketen nur indirekt bestimmt werden kann. Um dennoch einen performanten Seitenaufbau bei der Darstellung von Host- oder Paketlisten zu gewährleisten, wird am Ende des Korrelationsvorgangs das Threat-Level jedes verknüpften Advisories in ein *assoziatives Array* eingetragen. Dieses wird per Filter-Regel (siehe Abschnitt 8.1) serialisiert und im Feld threat_levels des entsprechenden Datensatzes gespeichert. Die Kapitel 11 und 12 gehen detailliert auf den Inhalt der Datenfelder der Advisory-Tabelle sowie auf das Feld threat_levels ein.

Wie in Abbildung 8.1 dargestellt, werden Foreign Key Constraints eingesetzt, um die *referentielle Integrität* der Datenbank sicherzustellen. Die Einträge der Pivot-Tabellen werden gelöscht, sobald der entsprechende *Primärschlüssel* in den Tabellen hosts, installedpackages, updatablepackages bzw. advisories gelöscht wird.

Normalerweise würden im Datenbank-Schema Trigger definiert werden, die bei der Löschung eines Hosts alle Pakete entfernen würden, die ansonsten verwaist (d.h. ohne Bezug zu einem Host) wären. Gleiches würde für Pakete und verwaiste Advisories gelten.
Nun hat MySQL InnoDB, welches für die Entwicklung des WizzanServers als Datenbank-Server verwendet wird, einen Bug [74] [75], der bewirkt, dass Trigger nicht reagieren, wenn die Aktualisierung bzw. Löschung eines Datensatzes durch ein Foreign-Key-Constraint ausgelöst wurde. Es ist fraglich ob MySQL InnoDB hier

in einer Weise *ACID*-kompatibel (*Atomicity, Consistency, Isolation, Durability*) ist, die von einer modernen Datenbank-Implementierung erwartet wird. [76] [74]

Als Workaround dient ein simpler *Garbage-Collector*, der Pakete und Advisories löscht, deren Primärschlüssel nicht in einer der Pivot-Tabellen vorkommen. Somit ist sichergestellt, dass Wizzan **niemals obsolete bzw. verwaiste Einträge speichert**.

Wo notwendig werden *Transaktionen* verwendet, um die Atomarität von komplexeren Datenbank-Operationen zu gewährleisten.

Automatisierte Erstellung des Paket-Inventars

Dieses Kapitel beschreibt, wie die vom WizzanAgent gelieferten Paketlisten verarbeitet werden und wie sie ins Inventar integriert werden. Im weiteren Sinn umfasst das Inventar alle Hosts, deren öffentliche Schlüssel, alle Pakete, Updates und Advisories. Damit kann das Inventar als Teilmenge der WizzanDB betrachtet werden, da diese wie in Abschnitt 8.3 beschrieben zusätzliche Tabellen enthält.
Für dieses Kapitel erscheint jedoch die Einführung des spezifischeren Begriffs ***Paket-Inventar*** sinnvoll. Das Paket-Inventar wird definiert als die Menge aller installierten Softwarepakete und verfügbaren Updates, die von den überwachten Hosts extrahiert werden konnten.

9.1 Verarbeitung empfangener Paketlisten

Dieser Abschnitt führt durch den Ablauf der Erzeugung von Paket-Objekten vom Empfang der Daten des WizzanAgents bis zur Speicherung der validierten Objekte. Der hier beschriebene Ablauf erfolgt in gleicher Weise zuerst für installierte Softwarepakete und anschließend für als Update verfügbare Pakete. Beide Entitäten werden im Folgenden um der besseren Lesbarkeit willen als „Paket“ bezeichnet.

Zu Beginn der Verarbeitung der Daten eines einzelnen Pakets wird aus den Werten von `name`, `version` und `source_plugin` (vgl. Abschnitt 7.2) ein HMAC berechnet, der als eindeutiges Identifikationsmerkmal dient, welches aus den Merkmalen des Paketes berechnet werden kann. Beim Speichern eines Paket-Objekts wird dieser Hash automatisch gesetzt und mit in die Datenbank geschrieben. Dieser Umstand ermöglicht es nun, ein Paket anhand dieses Hashes zu laden. Um diese Datenbankanfrage möglichst performant zu halten ist die entsprechende Tabellenspalte in der Datenbank als *BTREE-Index* definiert.

Schlägt der Ladevorgang fehl, bedeutet dies, dass der WizzanAgent ein bis dato unbekanntes Paket gesendet hat. Die empfangenen Werte werden in diesem Fall in ein neu erstelltes Paket-Objekt geschrieben, welches anschließend in der Datenbank hinterlegt wird.

Unabhängig davon, ob ein Paket-Objekt nun neu erstellt wurde, oder schon bekannt war, setzt der WizzanServer die entsprechenden Relationen zum sendenden Host.

War die Verarbeitung des jeweiligen Pakets inklusive des Setzens der Relationen erfolgreich, so wird in einer Liste vermerkt, welches Paket an den WizzanServer gesendet wurde. Diese Liste spielt im nächsten Abschnitt 9.3 beim Löschen obsoleter Pakete eine wesentliche Rolle.

9.2 Entfernen alter Pakete

Wie bereits in Abschnitt 7.2 erwähnt erwartet der WizzanServer *von jedem Plugin* des WizzanAgents – sofern dieses überhaupt Daten extrahiert hat – *vollständige* Paketlisten. Die Differenzmenge der in der Datenbank gespeicherten Pakete und der empfangenen Pakete ist die Menge der Pakete, deren Relationen zum Host des WizzanAgents gelöscht werden können.
Tritt nun der Fall ein, dass ein Paket von keinem Host mehr referenziert wird, so löscht es der Garbage-Collector aus der Datenbank. (siehe Abschnitt 8.3) Indem der beschriebene Löschvorgang auf jene Paketlisten bezogen wird, die von einem bestimmten Plugin generiert wurden, ist der WizzanAgent nicht gezwungen alle Plugins hintereinander auszuführen, bzw. führt der Fehlschlag eines Plugins nicht zur Löschung aller Pakete, die es zu einem früheren Zeitpunkt übermittelt hat.

9.3 Paket-Objekt

Der Inhalt von Paket-Objekten entspricht genau dem der vom WizzanAgent übermittelten Paketdaten, wie sie in Abschnitt 7.2 beschriebenen werden.

Zusätzlich wird ein Paket-Objekt mit drei UTC-Zeitstempeln versehen. `utc_parsed` gibt den Zeitpunkt der Erstellung der Paketdaten an, `utc_created` und `utc_updated` markieren den Zeitpunkt der Erstellung bzw. der letzten Modifikation des Paket-Objekts.

Die Funktion des Feldes `threat_levels` in Zusammenhang mit der grafischen Aufbereitung von Bedrohungsgraden wird in Abschnitt 8.3 sowie in den Kapiteln 11 und 12 erläutert.

9.4 Fazit zum Inventar

Aus technischer Sicht ist die Erstellung des Paket-Inventars beinahe trivial. Dennoch sind es die Informationen über Hosts und Pakete, welche im Inventar hinterlegt sind, die eine belastbare, präzise und eindeutige Datenbasis bilden, auf deren Grundlage die Auswertung von Security-Advisories überhaupt erst stattfinden kann.

Anders ausgedrückt: Wie sollen schlüssige Aussagen über Bedrohungsgrade getroffen werden, wenn unklar ist, welche Software in der eigenen IT-Infrastruktur eingesetzt wird?

Auswertung von Security-Advisory-Feeds

Der WizzanCrawler ist – wie in Kapitel 4 beschrieben – ein *Web-Crawler*, der Security-Advisories aus Websites extrahiert. Wie in Abbildung 10.1 dargestellt, kümmert sich dabei jedes Plugin um eine spezifische Datenquelle bzw. Website. Das Gesamtergebnis dieser Auswertungen wird anschließend dem WizzanServer übermittelt.

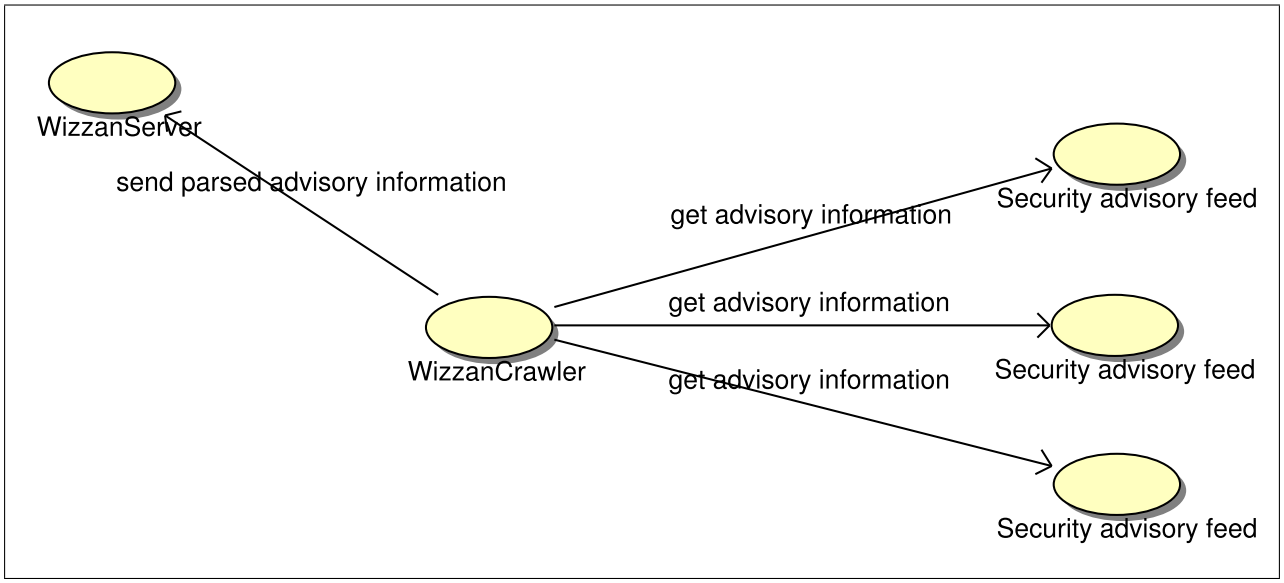

Abbildung 10.1: Anwendungsfall-Diagramm des WizzanCrawlers

Abbildung 10.2 beschreibt den Programmlauf des WizzanCrawlers. Auf die Hintergründe der Schritte 1, 2, 3 und 6 wird in Kapitel 6 eingegangen. Mit den Details des Datentransfers und des Nachrichtenformats (3, 4, 5, 9, 10 ,18 ,19) beschäftigt sich Kapitel 5.

Nach dem Start des WizzanCrawlers wird die Konfiguration geladen (1) und es erfolgt der Test (2) der

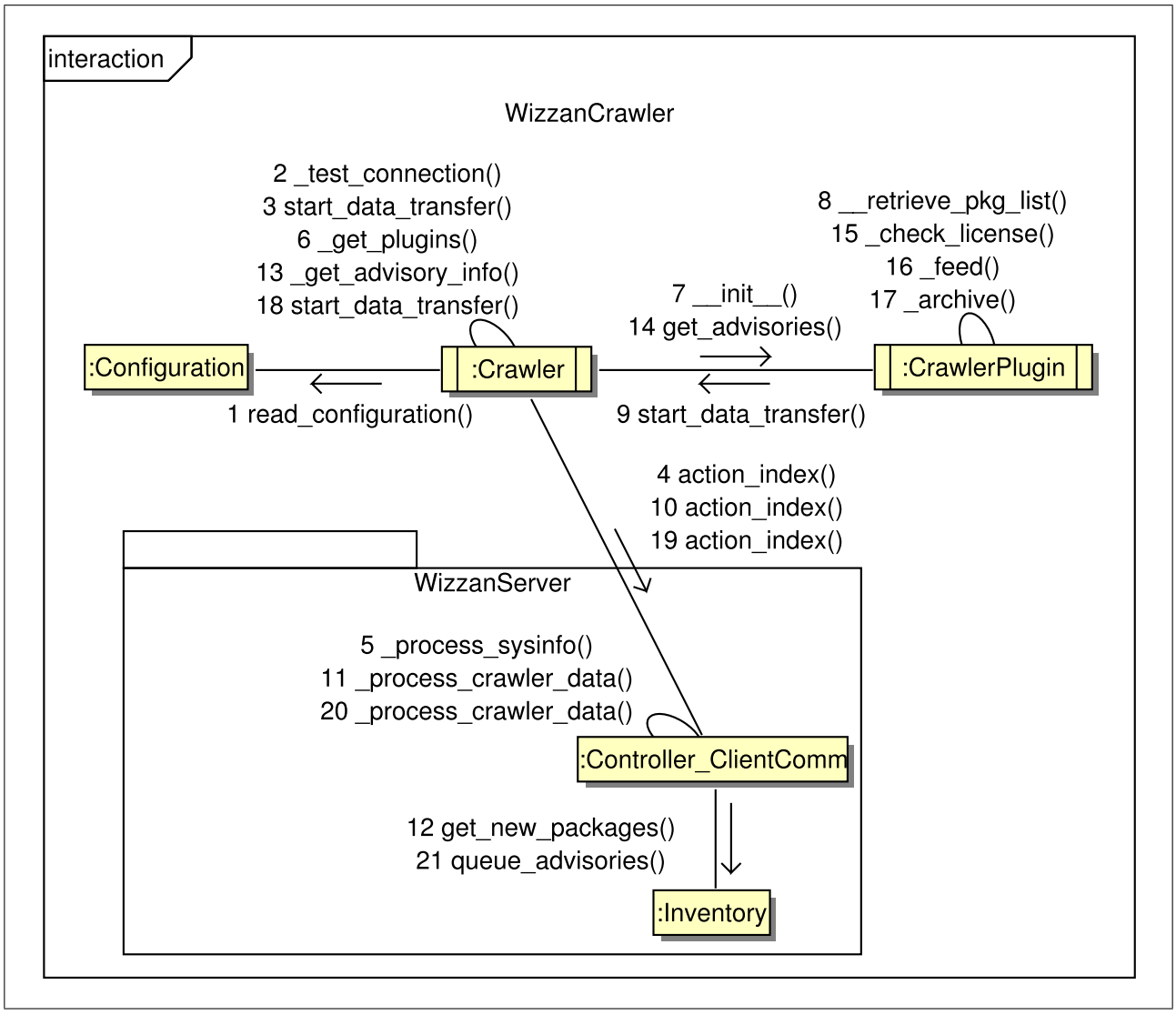

Abbildung 10.2: Kommunikationsdiagramm des WizzanCrawlers

Verbindung zum WizzanServer. Bevor die Auswertung der Security-Advisory-Feeds beginnt, werden grundlegende Informationen über den eigenen Host (Hostname, FQDN) ermittelt und an den WizzanServer gesandt (3, 4). Dieser legt ggf. ein neues Host-Objekt an, oder aktualisiert ein bereits vorhandenes. (5)

Sofern diese erste Kommunikation mit dem WizzanServer erfolgreich verläuft, werden die in der Konfiguration aktivierten Plugins geladen (6). Die Plugins fordern bei ihrer Initialisierung (7) die in Abschnitt 10.3 beschriebenen Paketlisten an (8,9). Jedes Plugin formuliert dabei eine eigene Anfrage (10) sodass der WizzanServer (11, 12) nur jene Daten sendet, die benötigt werden.

Sind alle Plugins geladen, werden sie der Reihe nach angewiesen Security-Advisories aus den entsprechenden Datenquellen auszulesen (14). Um den/die Administrator/-in und somit dessen/deren Unternehmen bei der Einhaltung der Lizenz- und Nutzungsbedingungen zu unterstützen, überprüft jedes Plugin zu Beginn, ob die wesentlichen Passagen oder die Angabe des Herausgebers geändert wurden (15). Sollte dies der Fall sein, wird eine Warnmeldung ins Log geschrieben. Diese Hilfestellung entbindet den/die Verantwortliche/n *nicht* von seiner/ihrer Pflicht, die Rechtmäßigkeit der Nutzung der Inhalte Dritter regelmäßig zu überprüfen.

Im nächsten Schritt (16) wird der *Syndication-Feed* des jeweiligen Herstellers, Distributors bzw. Sicherheitsdienstleisters ausgewertet. Der Begriff ***Syndication-Feed*** wird im Kontext dieser Diplomarbeit auf alle Datenformate angewandt, die eine *begrenzte Anzahl der aktuellsten Security-Advisories* beinhalten und dazu gedacht sind von einem großen Publikum gelesen zu werden. Üblicherweise kommt auf Seiten des Herausgebers zumindest eines der beiden *Syndication*-Formate *RSS*- und *Atom* oder eine vergleichbare XML-Datei zum Einsatz. Diese beiden Formate haben sich als Quasi-Industriestandard etabliert, und stellen einen einfachen Weg dar die Mehrfachverwendung von Medieninhalten (engl. *Syndication*) zu ermöglichen. [77]

Sind alle Security-Advisories aus dem Syndication-Feed extrahiert, beginnt das Plugin mit der Auswertung des *Archivs* (17). Als ***Archiv*** wird im Kontext dieser Diplomarbeit eine *möglichst vollständige Zusammenstellung aller jemals publizierten Security-Advisories* eines bestimmten Herausgebers verstanden.

Die begriffliche Trennung in *Feed* und *Archiv* ist nötig, um einerseits auf den verschiedenen Inhalt (aktuelle/historische Daten) und andererseits auf die (meist) wesentlichen Unterschiede der Aufbereitung hinweisen zu können. Wo dieser explizite Hinweis nicht vonnöten ist, wird zugunsten der besseren Lesbarkeit weiterhin der in der Einführung (siehe Abschnitt 2.2) eingebrachte Begriff des ***Security-Advisory-Feeds*** vereinheitlichend für Feeds und Archive verwendet.

Sind alle Datenquellen ausgewertet, startet der WizzanCrawler seinen letzten Datentransfer (18) in dem der dem WizzanServer alle extrahierten Security-Advisories in einem normalisierten Format übermittelt (19, 20). Dieser speichert die empfangenen Advisories in einer Warteschlange (21) und führt anschließend die in Kapitel

11 beschriebene Korrelation durch.

10.1 Technische Aspekte

Die Korrelation und damit die Bewertung der einzelnen Security-Advisories findet per Design (siehe Kapitel 4) am und durch den WizzanServer statt. Wären durch rechtliche Aspekte, Rechenleistung und Datentransfer keine Beschränkungen gesetzt, so wäre die einfachste und gleichzeitig robusteste Lösung *sämtliche* Security-Advisories lediglich zu normalisieren und direkt an den WizzanServer zu senden. Dieses Vorgehen wird im Folgenden als ***Take-All-Ansatz*** bezeichnet.
Erzwingen die genannten Beschränkungen nun die Implementierung eines Auswahlverfahrens, so darf dieses (Detail-)Entscheidungen der Korrelation nicht vorwegnehmen. **Security-Advisories, die vom WizzanCrawler *fälschlicherweise* als irrelevant eingestuft werden, sind für alle weiteren Verarbeitungsschritte verloren.**

Im Fall der **Syndication-Feeds** ist der Take-All-Ansatz aufgrund deren geringer Datenmenge leicht umsetzbar. Deshalb wird er für die Implementierung der Auswertung von Syndication-Feeds als verbindlicher Grundsatz festgelegt. Eine Ausnahme von dieser Regel bilden Datenformate bzw. Inhalte, deren Auswertung nur auf Basis vordefinierter Wertemuster (z.B. regulären Ausdrücken) stattfinden kann. Abschnitt 10.9.4 behandelt ein konkretes Beispiel hierzu.

Bei der Auswertung von **Archiven** – vor allem wenn diese Security-Advisories für verschiedenste Hersteller/Distributoren bzw. Produkte enthalten – ist eine Vorauswahl zwingend erforderlich. Die Grundidee ist nur jene Archiv-Einträge auszuwählen, für die auch ein entsprechendes Paket in der Datenbank existiert. Im Endeffekt wird aktiv nach Paketen gesucht. Um mehrfache Suchen nach den selben Paketen möglichst zu verhindern bzw. zu reduzieren, muss erkannt werden, welche Pakete bereits abgehandelt wurden, bzw. welche neu hinzugekommen sind. Abschnitt 10.3 beschäftigt sich mit der Erstellung entsprechender Paketlisten.
Die Herausforderung ist nun, kein Security-Advisory fälschlicherweise zu verwerfen. Abschnitt 10.4 ist diesem Aspekt gewidmet.

Aus Sicht eines Paket-Objekts, das im Inventar angelegt wird, muss bezogen auf ein einzelnes WizzanCrawler-Plugin Folgendes gewährleistet sein, um keine sicherheitsrelevanten Informationen zu verpassen:

- Das Plugin durchsucht das jeweilige Archiv aktiv nach dem Paket, solange dieses als „neu hinzugefügt“ betrachtet wird, und extrahiert alle relevanten Security-Advisories.
- Sobald das Paket nicht mehr als „neu hinzugefügt“ gilt, greift der Take-All-Ansatz des Syndication-Feeds, welcher die Weiterleitung aller zukünftigen Security-Advisories gewährleistet.

Dabei muss der/die Administratorin (durch entsprechende Konfiguration und funktionierende Hosts) sicherstellen, dass das „Abtastintervall“ des Plugins stets so gering ist, dass alle Einträge des Syndication-Feeds sicher erfasst werden.

Teilweise ist es sinnvoll und erlaubt das komplette Archiv bzw. ein Äquivalent zu dessen Index herunterzuladen. Beispiele hierfür sind in den Abschnitten 10.9.2, 10.9.3 und 10.9.4 beschrieben.
Das Erstellen einer solchen lokalen Kopie erfordert auch deren laufende Aktualisierung, die im einfachsten Fall durch ein erneutes Herunterladen erfolgt. Dadurch wird sichergestellt, dass Syndication-Feed und lokale Kopie einander zeitlich stets überlappen und keine „Lücken“ entstehen, welche dazu führen würden, dass Security-Advisories verpasst werden. Ein guter Richtwert für das zeitliche Intervall zwischen zwei Aktualisierungen ist die Hälfte des Alters des ältesten Eintrags des Syndication-Feeds.

10.2 Rechtliche Aspekte

Nur jene Security-Advisory-Feeds, deren Herausgeber einer Weiterverarbeitung, wie Wizzan sie durchführt, zustimmen, dürfen als Datenquellen für WizzanCrawler-Plugins verwendet werden. Abschnitt 2.2 geht näher auf diese Thematik ein und zitiert die relevanten Auszüge der Nutzungsbedingungen bzw. Lizenzen der verwendeten Security-Advisory-Feeds.

Die ***nutzungsrechtliche Arbeitsgrundlage*** für die Implementierung von WizzanCrawler-Plugins und auch den entsprechenden Teilen des WizzanServer wird als „kleinster gemeinsamer Nenner“ aller in Abschnitt 2.2 betrachteten Nutzungsbedingungen verstanden. Sie versucht das Minimum an nutzbaren Rechten und das Maximum an erfüllbaren Pflichten in sich zu vereinen, um die Nutzungsbedingungen aller verwendeten Security-Advisory-Feeds zu erfüllen und gleichzeitig die „juristische Angriffsfläche“ zu minimieren.
Die folgende Betrachtung dient rein der Diskussion der rechtlichen Rahmenbedingungen und erhebt keinerlei Anspruch auf Rechtssicherheit oder -gültigkeit. Die professionelle Meinung eines Anwalts wurde nicht eingeholt.

- Die **Rechte** beschränken sich auf die *nicht-kommerzielle*, *interne* Nutzung zum Zwecke des *Lernens* und der *Fortbildung*. Nach anglo-amerikanischem Recht fiele diese Art der Nutzung wahrscheinlich unter die als *Fair Use* bezeichnete Limitierung des Copyright, welche zudem keine explizite Zustimmung des Copyright-Inhabers benötigt [49]. Da für die in dieser Diplomarbeit vorgestellten Security-Advisory-Feeds nur britisches bzw. US-amerikanisches Recht zutrifft, entfällt die Betrachtung anderer Rechtssysteme an dieser Stelle.

- Die **Pflichten** umfassen das *Setzen von Copyright- und Lizenzhinweisen* sowie das Beibehalten oder zusätzliche Vermerken *weiterer Hinweise*. Im Format des Advisorys, welches in Abschnitt 10.6 detailliert behandelt wird, existieren entsprechende Datenfelder für genau diese Informationen.
  WizzanCrawler-Plugins müssen die folgenden Felder mit den entsprechenden Daten füllen: *Herausgeber*, *Copyright-Vermerk*, *Lizenz-Vermerk*, *URL des vollständigen Lizenztextes* und *weitere rechtliche Vermerke*. Dieses Vorgehen führt zwar zur Speicherung redundanter Daten in der WizzanDB, jedoch erlangt der/die Entwickler/-in eines Plugins die benötigte Flexibilität um nicht-trivialen rechtlichen Anforderungen rasch gerecht zu werden.

Dies scheint die aus rechtlicher Sicht unproblematischste Nutzungsvariante zu sein. Wie bereits in Abschnitt 2.2 zum Ausdruck gebracht, muss vor Entwicklung eines WizzanCrawler-Plugins die rechtliche Situation in Bezug auf die zu verwendende Datenquelle eingehend geklärt werden.

Soll Wizzan in einer Weise eingesetzt werden, die den Nutzungsbedingungen bzw. Lizenzen eines oder mehrerer Herausgeber zuwider läuft, so kann sich der/die Verantwortliche damit behelfen die entsprechenden WizzanCrawler-Plugins zu deaktivieren.

Dem Szenario einer zukünftigen Copyright- oder Urheberrechtsverletzung wird mit dem Grundsatz der Datensparsamkeit begegnet. Plugins *lesen* zwar den kompletten Syndication-Feed, das komplette Archiv bzw. den Archiv-Index, niemals jedoch werden die darin enthaltenen Einträge vollständig auf den WizzanServer übertragen. Archive werden nur heruntergeladen, wenn dies erlaubt und klar die beste Herangehensweise ist. Stattdessen besteht ein Advisory-Datensatz – wie aus Abschnitt 10.6 ersichtlich – praktisch nur aus Meta-Daten und Referenzen.

Als Ausblick auf die rechtlichen Aspekte der Darstellung der extrahierten Informationen in Kapitel 12 sei erwähnt, dass die grafische Aufbereitung eines Advisory-Objekts durch den WizzanServer mehr eine Ansammlung von Links und Meta-Informationen als ein informierendes Security-Advisory ist.

10.3 Anfordern von Paketlisten

Anders als die Daten, die aus Paketmanagern extrahiert werden, sind nicht alle verfügbaren Security-Advisories automatisch auch relevant. (siehe Abschnitt 10.1) Wenn bestimmte Software sich nicht im Einsatz befindet, kann aus ihren Sicherheitsschwachstellen auch kein Bedrohungspotential erwachsen.
Vor diesem Hintergrund ist es nötig den WizzanCrawler-Plugins im Vorfeld der Verarbeitung von Archiven, die eine hohe Anzahl an Security-Advisories beinhalten, die Liste neu hinzugefügter Paketnamen aus dem

Software-Inventar zu übermitteln. Damit kann ein Crawler entscheiden, welche Advisories als relevant einzustufen und somit zu verarbeiten sind.

In der WizzanDB existiert zu diesem Zweck die *View* newestpackages. Sie listet für jedes Paket-Objekt den *Paketnamen* und den *Zeitpunkt der letzten Modifikation* auf. Die Sortierung erfolgt absteigend nach diesen Zeitstempeln.
Indem das jeweilige Plugin bei der Abfrage dieser Liste den Zeitstempel des zuletzt verarbeiteten Pakets angibt, wird die genannte Liste aller Paketnamen auf die seit dem neu hinzugekommenen Einträge reduziert. Das Ergebnis einer solchen Abfrage wird im Folgenden als *Liste neuer Paketnamen* bezeichnet. Sie wird für jedes Plugin separat ermittelt und existiert nur solange im Hauptspeicher des Hosts, bis der WizzanCrawler seine Ausführung beendet hat.

Während der Implementierung haben sich folgende Umsetzungsvarianten als gangbar erwiesen:

- Die Abfrage der ***vollständigen Liste aller Paketnamen*** (ohne Duplikate) ist erforderlich, wenn Paketnamen nur aufgrund dieser Liste überhaupt aus dem Text von Security-Advisories extrahiert werden können. Ein Beispiel für diesen Fall wird in Abschnitt 10.9.4 behandelt.
- Liegt ein Archiv als lokale Kopie vor, auf die beliebig viele Zugriffe durchgeführt werden können, ohne die Services anderer unnötig zu belasten, so wird die ***Liste neuer Paketnamen*** verwendet. Nachdem alle relevanten Security-Advisories extrahiert wurden, vermerkt das Plugin die vollständige Abarbeitung der Liste, indem der Zeitstempel des zuletzt verarbeiteten Pakets in einer Datei gespeichert wird.
- Es ist auch eine Variante implementiert, die das ***Durchlaufen der Liste neuer Paketnamen Eintrag für Eintrag*** erlaubt. Dabei wird jeder Eintrag separat bestätigt.

Der WizzanServer kann in einer Sekunde mehrere Paket-Objekte erstellen bzw. aktualisieren. Deshalb ist es nötig zusätzlich zum oben genannten Zeitstempel eine Liste der verarbeiteten Pakete zu speichern, welcher jener Sekunde bzw. jenem Zeitstempel zugeordnet sind. Anstelle des Paketnamens wird hier jedoch ein vom WizzanServer aus Paketname und Zeitstempel berechneter HMAC verwendet, damit ein potentieller Angreifer aus den Dateien des WizzanAgent keine Dateinamen herauslesen kann.

In diesem Zusammenhang sei nochmals darauf verwiesen, dass der in Abschnitt 4.1.1 beschriebene Schutz des privaten Schlüssels des WizzanCrawlers vor unbefugtem Zugriff gleichbedeutend mit dem Schutz der beschriebenen Paketlisten ist. Der Besitz des privaten Schlüssels autorisiert die Abfrage der Listen. (siehe Abschnitt 5.3)

10.4 Matching

Der Begriff *Matching* bezeichnet im Kontext dieser Diplomarbeit die Bestimmung der Relevanz – d.h. die Auswahl – von Security-Advisories auf Grundlage der in Abschnitt 10.3 beschriebenen Paketlisten. Dieser Abgleich ist immer dann erforderlich, wenn der in Abschnitt 10.1 beschriebene *Take-All*-Ansatz nicht angewendet werden kann. Das bedeutet auch, dass Matching nur in Ausnahmefällen bei Syndication-Feeds zum Einsatz kommt.

Matching stellt bezogen auf das Gesamtergebnis von Wizzan eine Vorauswahl von Security-Advisories dar. Dementsprechend ist das oberste Ziel die Anzahl der zu verarbeitenden Security-Advisories den tatsächlich relevanten anzunähern, ohne fälschlicherweise relevante Security-Advisories zu verwerfen. (*False-Negatives*) Die Lösung besteht darin, die Bedingungen für einen „Treffer“ vage genug zu halten, um jeden tatsächlich relevanten Archiv-Eintrag sicher zu erfassen. Die Kehrseite dieser Medaille sind irrelevante Advisory-Daten, die fälschlicherweise an den WizzanServer gesandt werden. (*False-Positives*) Dieser verfügt jedoch über eine wesentlich bessere Datenbasis (Inventar) als der WizzanCrawler und kann somit dessen Aussagen bezüglich der Relevanz von Security-Advisories falsifizieren. (siehe Kapitel 11)

Im Folgenden werden die Ansätze diskutiert, die während der Implementierung auf ihre Tauglichkeit überprüft wurden.

- Teilweise verleitet der Aufbau der Security-Advisory-Feeds zu der Annahme, dass die in den Einträgen genannten Paketnamen direkt auf die Einträge der Liste neuer Paketnamen (siehe Abschnitt 10.3) umsetzen lassen. Die Debian Security Advisories (10.9.1) sind hierfür ein Beispiel. Auch die Mailingliste der Ubuntu Security Notices (10.9.2) wurde im ersten Ansatz in eine Liste von Paketnamen umgewandelt. Diese Methode ist ironischerweise gerade aufgrund ihrer Exaktheit ungeeignet, da sie schlicht zu viele False-Negatives produziert.

- Wie sieht nun die Alternative aus, wenn – wie im Fall der Debian Security Advisories – Listen von Verweisen auf Security-Advisories, die im Verweistext den Paketnamen enthalten, den besten Ansatzpunkt darstellen?
  Reguläre Ausdrücke, die aus den in Abschnitt 10.3 beschriebenen Paketlisten erstellt werden, bieten hier eine robuste, treffsichere Lösung. Die einzelnen Paketnamen werden verkürzt, indem beginnend mit dem ersten nicht alphanumerischen Zeichen bis zum Wortende alles verworfen wird. Anschließend werden die Ergebnisse aneinander gereiht, sodass beispielsweise folgender regulärer Ausdruck entsteht: `(libpng|php5|mysql|xterm)` Der besseren Lesbarkeit halber wird ein von der Liste neuer Pakete

abgeleitete reguläre Ausdruck mit *LNPRA* abgekürzt.
Sofern ein Paketname aus dem Verweistext extrahiert werden kann, wird er ebenfalls nach der oben beschriebenen Methode verkürzt. Das Akronym lautet in diesem Fall *VTRA* für „aus dem Verweistext abgeleiteter regulärer Ausdruck“.
Nun erfolgen pro Verweis zwei Vergleiche, wobei ein positives Ergebnis das entsprechende Security-Advisory als relevant einstuft. Zuerst wird überprüft, ob die Anwendung des LNPRA auf den Verweistext einen Treffer liefert. Sollte dies nicht der Fall sein, wird der VTRA der Reihe nach auf die Einträge der Liste neuer Paketnamen angewandt. Ist das Ergebnis ebenfalls negativ, wird der entsprechende Verweis verworfen. Das beschriebene Verfahren wird im Folgenden als ***zweifaches Matching*** bezeichnet. Anstelle des LNPRA können hier auch einzelne Paketnamen, die nach obigem Schema verkürzt werden, zum Einsatz kommen. Dies ist vor allem dann interessant, wenn anstelle der Verweise einzelne Pakete als Grundlage für den Wiederanlauf verwendet werden. (siehe Abschnitt 10.5)

- Kann das Archiv eines Security-Advisory-Feeds als ganzes heruntergeladen werden, so empfiehlt es sich den Text der einzelnen Einträge nach Paketnamen zu durchsuchen. Die Verwendung eines LNPRA ergibt sich nun aus Performance-Überlegungen, da er den wiederholten Durchlauf einer Paketliste überflüssig macht.
  Das Archiv wird Eintrag für Eintrag durchlaufen, dabei wird überprüft, ob die Anwendung des LNPRA auf den komplettem Text (inkl. Metadaten, Tags, etc.) des Security-Advisories ein positives Ergebnis liefert. Aus dem Ergebnis ist auch ersichtlich, welche Zeichenkombination bzw. Stelle „getroffen“ wurde. Dieses Ergebnis kann durchaus in eine Liste betroffener Pakete integriert werden, jedoch ist es ratsam (weil zuverlässiger) den Paketnamen (zusammen mit einigen anderen wichtigen Meta-Daten) gesondert aus dem Security-Advisory zu extrahieren, sofern dessen Strukturierung es zulässt. Das beschriebene Verfahren stellt eine ***Volltextsuche*** dar und wird fortan auch als solche bezeichnet.

Die soeben beschriebenen Fälle können als Stereotype betrachtet werden. Die tatsächliche Implementierung orientiert sich zwar an ihnen, jedoch soll jedes Plugin ein Maximum an Informationen extrahieren, wodurch es oft zu Abweichungen von diesem vorgezeichneten Weg kommt.

10.5 Limitierungen und Wiederaufnahme

Um die Server von Distributoren, Herstellern bzw. Sicherheitsdienstleistern keiner unnötigen Belastung auszusetzen, sind in alle WizzanCrawler-Plugins konfigurierbare Limitierungen der Anzahl und Frequenz von Anfragen an diese Server integriert.

Limitierungen der Anzahl dürfen überschritten werden, wenn ansonsten nicht alle Security-Advisories extrahiert werden würden, die ein bestimmtes Paket betreffen. Die Frequenz wird über das Intervall zwischen den Anfragen eingestellt.

Zur Gewährleistung der lückenlosen Verarbeitung von Security-Advisory-Feeds ist es erforderlich die Verarbeitung an den Stellen wiederaufzunehmen, an denen sie aufgrund einer Limitierung unterbrochen wurde. Eine Ausnahme stellen hier die Syndication-Feeds dar. Sie sind ausreichend kurz, um ohne Unterbrechung verarbeitet zu werden. Dennoch steht es dem/der Administrator/-in frei, die Anzahl der verarbeiteten Einträge zu begrenzen, um redundante Abfragen zu reduzieren.

Im Falle der Verarbeitung von Archiven kann die Wiederaufnahme entweder auf Basis der einzelnen Paket-Einträge (vgl. dazu Abschnitt 10.3) oder auf Basis der temporären Speicherung der Identifikatoren bereits verarbeiteter Security-Advisories erfolgen. Die schlussendliche Umsetzung hängt maßgeblich von den Gegebenheiten des jeweiligen Security-Advisory-Feeds ab.

10.6 Normalisierung von Security-Advisories

Im Kontext dieser Diplomarbeit wird der Begriff der ***Advisory-Daten*** auf jene Daten angewandt, die ein WizzanCrawler-Plugin aus einem einzelnen Security-Advisory extrahiert. Aus diesen Daten kann der WizzanServer später ein *Advisory-Objekt* generieren.

Wie in den Überlegungen zu den rechtlichen Aspekten in Abschnitt 10.2 bereits erwähnt, enthalten Advisory-Daten neben Titel und Kurzbeschreibung nur Metadaten, die entweder rechtlich erforderlich sind oder die Grundlage für die in Abschnitt 11 beschriebene Korrelation durch den WizzanServer bilden. Damit diese überhaupt stattfinden kann, müssen die Advisory-Daten in einem *normalisierten* (d.h. vereinheitlichten) Format vorliegen. Aufgabe der Plugins ist es nun die Einträge der Security-Advisory-Feeds in einer Weise neu zu strukturieren, die weder zum Verlust wesentlicher Informationen noch zu Bedeutungsänderungen oder der Generierung falscher Informationen führt.

Die folgende Auflistung behandelt für die Korrelation irrelevante Felder zuerst, um den Weg für eine detaillierte Betrachtung der Informationsaufbereitung zu ebnen.

- **`identifier`** – Hier wird der Bezeichner eingesetzt, der innerhalb des Security-Advisory-Feeds verwendet wird.
- **`original_title`** – Der Titel, wie er im Security-Advisory angegeben ist.

- **original_short_description** – Von allen verfügbaren, beschreibenden Texten wird aufgrund der in Abschnitt 10.2 angeführten rechtlichen Überlegungen lediglich die Kurzbeschreibung auf den WizzanServer übertragen.

- **reported_date** – Es ist nicht immer klar, ob die in einem Security-Advisory enthaltene Zeitangabe sich auf dessen erste Veröffentlichung, dessen letztes Update oder das Bekanntwerden der Sicherheitsschwachstelle selbst bezieht. Da dieses Datum bei der Korrelation so gut wie keine Rolle spielt, wird im Fall mehrerer Datumsangaben innerhalb eines Security-Advisorys einfach diejenige verwendet, die am ehesten dessen erster Veröffentlichung entspricht.

- **rfc3339_timestamp** – Der Zeitstempel der Erstellung der Advisory-Daten wird gemäß RFC 3339 formatiert und stellt eine menschenlesbare, eindeutige Zeitangabe dar. Dieser Wert wird automatisch gesetzt.

- **source_plugin** – Der Name des WizzanCrawler-Plugins, welches die Advisory-Daten generiert hat.

- **source_url** – Dieses Feld enthält die URL zum Original-Eintrag des Security-Advisory-Feeds, sofern eine entsprechende Web-Repräsentanz existiert.

- **references** – Dieses Feld enthält eine Liste von Links (genauer: URL und Verweistext), die innerhalb des Security-Advisorys als Referenz zu weiterführenden Informationen deklariert sind.

- Auf die Felder **publisher**, **copyright_note**, **license_note**, **license_url** und **other_legal_notes** wird in Abschnitt 10.2 detailliert eingegangen. Sie dienen dem geforderten Hinweis auf die Quellen bzw. die Nutzungsbedingungen eines Advisory-Objekts.`cvss`

- Als Werte für **cvss** und **severity** sind Fließkommazahlen zu verwenden, die auf einer Skala von 0 bis 10 den Schweregrad einer Sicherheitsschwachstelle angeben. Sofern das Security-Advisory den CVSS-Score enthält, kann dieser einfach ins Feld `cvss` übernommen werden. Wird allerdings „nur“ eine Schätzung der Kritikalität angegeben, so ist diese nach einer einer Konvertierung in eine Fließkommazahl innerhalb des offenen Intervalls von 0 bis 10 ins Feld `severity` einzutragen.
  Sofern das Plugin die automatische Berechnung eines CVSS-Score auf Basis des Textes des Security-Advisories unterstützt, kann dieses Ergebnis im Feld **auto_cvss** vermerkt werden.
  Welcher der drei Werte nun in die Ermittlung des Bedrohungsgrades einfließt, wird in Abschnitt 11.5 erläutert.

- **affected_vendors** – Dieses Feld enthält eine Liste der Namen betroffener Hersteller bzw. Distributoren. Die Extraktion dieser Liste ist ausdrücklich erwünscht, jedoch nicht immer durchführbar. Sie

könnte zukünftig bei der Korrelation eingesetzt werden, um als zusätzliches Kriterium für eine höhere Zuverlässigkeit der getroffenen Aussagen zu sorgen.

- Den Feldern **`fixed_package_versions`** und **`vulnerable_package_versions`** wird aufgrund der Länge ihrer Beschreibung der nächste Abschnitt (10.7) gewidmet.

10.7 Extraktion von Versionsinformationen

Die Korrelation fußt im Kern auf den Versionsangaben, die die WizzanCrawler-Plugins aus den Security-Advisories beziehen. Idealerweise lässt sich aus den Security-Advisories ableiten, *welche Versionen welcher Pakete* (spezifiziert durch ihre Paketnamen) betroffen (verwundbar) bzw. nicht betroffen (nicht verwundbar) sind.

Das Aufbereiten von Versionsinformationen geschieht ausschließlich durch die Plugins des WizzanCrawlers. Nur sie verfügen über den vollen Kontext der Originaldaten des einzelnen Security-Advisories. Der WizzanServer ist aufgrund der Reduziertheit der Advisory-Daten schlicht nicht qualifiziert, Annahmen über Versionsangaben zu treffen.
Daraus entsteht für die Plugins die Anforderung die Versionsangaben so ausführlich wie möglich zu gestalten. Auch müssen ggf. zusätzliche Versionsinformationen eingefügt werden, die aus den ursprünglichen Versionsangaben logisch abzuleiten sind.

Das Extrahieren von Versionsangaben aus normalem Text stellt dabei eine besondere Herausforderung dar. Oft bleibt nichts anderes übrig als die im Security-Advisory enthaltene Beschreibung einer Sicherheitsschwachstelle nach für Versionsangaben typischen Formulierungen zu durchsuchen. Dankenswerterweise werden oftmals die immer gleichen Wortkombinationen wie z.B. *„has been fixed in“*, *„is fixed in“*, *„has been adressed in version“*, etc. verwendet. Mit einer gewissen Zuverlässigkeit lässt sich aus diesem Kontext ableiten, ob es sich bei einer bestimmten Versionsangabe nun beispielsweise um eine einzelne verwundbare Version, um eine Grenze eines Bereichs verwundbarer Versionen oder um eine Version, in der das Problem behoben wurde, handelt.
Schlussendlich muss jedoch festgestellt werden welches wort-artige, von Leer- und/oder Satzzeichen umgebene Gebilde nun die eigentliche Versionsnummer ist. Auf Grundlage der im Satz erkannten Schlüsselworte werden nun mögliche Kandidaten ausgewählt und einer Funktion übergeben, die jenen zurückliefert, der die meisten Satzzeichen, Sonderzeichen und Ziffern enthält, und somit am wahrscheinlichsten eine Versionsangabe ist.

Der wesentliche Nachteil an diesem Ansatz ist, dass er nur für bereits bekannte Security-Advisories wirk-

lich zuverlässig funktioniert. Zukünftige Formulierungen können nicht vorausgesehen werden. Auf einen Versuch mit selbstlernenden Filtern, wie sie beispielsweise zu Abwehr von SPAM eingesetzt werden, wurde aus Zeitgründen verzichtet.

10.8 Aufbereiten von Versionsinformationen

Listing 10.1 zeigt Versionsinformationen, die aus dem Debian Security Advisory DSA-2475-1 openssl extrahiert wurden. Dieses dient als Fallbeispiel um das Verständnis der nachfolgenden Ausführungen zu erleichtern.

Die Versionsinformationen werden in Form eines **dreidimensionalen Arrays** dargestellt, wobei die ersten beiden Dimensionen assoziativ sind. Der Index der ersten Dimension wird aus den Namen der jeweiligen *Distributionen* gebildet. Kann die Distribution nicht ermittelt werden, kann auch das Zeichen $\star$ als Wildcard eingesetzt werden. Für die zweite Dimension werden die *Paketnamen* als Index verwendet. Der Index der dritten Dimension ist numerisch. Diese letzte Ebene der Verschachtelung ist eine simple Aneinanderreihung von Versionsangaben.

```
[fixed_package_versions] => Array (
    [squeeze] => Array (
        [openssl] => Array (
            [0] => 0.9.8o-4squeeze13 ))
    [wheezy] => Array (
        [openssl] => Array (
            [0] => 1.0.1c-1 ))
    [sid] => Array (
        [openssl] => Array (
            [0] => 1.0.1c-1 ))
  )
[vulnerable_package_versions] => Array (
    [squeeze] => Array (
        [openssl] => Array (
            [0] => <0.9.8o-4squeeze13 ))
    [wheezy] => Array (
```

```
        [openssl] => Array (
            [0] => <1.0.1c-1 ))
    [sid] => Array (
        [openssl] => Array (
            [0] => <1.0.1c-1 ))
)
```

Listing 10.1: Extrahierte Versionsinformationen

Viele Security-Advisory-Feeds listen – wie in Abschnitt 2.2 beschrieben – ausschließlich jene Paketversionen auf, in denen das zugrunde liegende Problem behoben wurde. In diesem Fall kennzeichnet die Angabe der Versionsnummer im Array `fixed_package_versions` die entsprechende Paketversion als „problembehebend“. Dieser Umstand könnte – wie aus dem nächsten Absatz hervorgeht – auch anders ausgedrückt werden. Jedoch kommt dieses Array ohne logische Verknüpfungen aus, wodurch der Spielraum für Missinterpretationen verringert wird.

Ziel des WizzanCrawler ist es aber *verwundbare* Paketversionen zu identifiziert. Unabhängig davon, ob verwundbare Versionen angegeben werden, oder nicht, muss aus jeder problembehebenden Version der Bereich der verwundbaren Versionen abgeleitet werden. Während der Implementierung wurde kein Security-Advisory gefunden, auf das folgender Schluss nicht zutrifft:
Wenn das Problem in Version x.y.z behoben wurde, so müssen alle Versionen kleiner als x.y.z verwundbar sein.
Ein entsprechender Eintrag wird im Array `vulnerable_package_versions` vermerkt. Dieses bietet die folgenden Möglichkeiten verwundbare Paketversionen zu spezifizieren:

- `=VERSION` bzw. `VERSION` – Exakt diese Version des Pakets ist verwundbar.
- `!VERSION` – Exakt diese Version ist *nicht* verwundbar.
- `<VERSION` – Alle jüngeren Versionen sind verwundbar.
- `<=VERSION` – Alle jüngeren Versionen inkl. der angegebenen sind verwundbar.
- `>VERSION` – Alle älteren Versionen sind verwundbar.
- `>=VERSION` – Alle älteren Versionen inkl. der angegebenen sind verwundbar.

Werden keine Angaben zu verwundbaren Versionen gemacht, so kommt am WizzanServer das in in Abschnitt 11.4 beschriebene Fallback zum Einsatz.

Die Verknüpfung der Versionsangaben findet am WizzanServer im Zuge der Korrelation von Paket-Objekten und Advisory-Daten statt und wird in Abschnitt 11.3.3 beschrieben.

10.9 Implementierungsbeispiele

Dieser Abschnitt beschreibt die Implementierung der WizzanCrawler-Plugins, um einen Eindruck von der tatsächlichen Umsetzung der obigen Ausführungen zu vermitteln. Abschnitt 2.2 behandelt hierzu die theoretischen Grundlagen.

Alle Plugins sind von einer Basisklasse (`CrawlerPlugin`) abgeleitet, welche gemeinsame Funktionalitäten bündelt. Dazu zählen beispielsweise der Zugriff auf die Konfiguration, Logging, das Anlegen und Verwalten von Dateien, die Verwaltung von Zeitstempeln, das Abrufen von Paketlisten (siehe Abschnitt 10.3), der Download von Dateien und das Auslesen von Versionsnummern aus Text.

WizzanCrawler-Plugins *müssen* die Funktion zum Überprüfen der Nutzungs- bzw. Lizenzbedingungen `_check_license()` und die Funktion zum Auslesen des Syndication-Feeds `_feed()` implementieren. Ein Plugin könnte zwar auf die Funktion zum Verarbeiten von Archiveinträgen `_archive()` verzichten, jedoch entgingen Wizzan dadurch evt. wichtige Informationen.

Im Folgenden werden die Besonderheiten der einzelnen Plugins als Kontrast zu den bisher beschriebenen Gemeinsamkeiten behandelt.

10.9.1 Debian Security Advisories

Der vom Debian-Projekt bereitgestellte Syndication-Feed listet ähnlich wie das Web-Archiv an sich lediglich Links zu den eigentlichen Security-Advisories auf. In beiden Fällen wird somit die Webseite eines zu verarbeitenden Security-Advisories heruntergeladen und auf dieselbe Weise (d.h. durch dieselbe Funktion) verarbeitet.

Der wesentliche Unterschied liegt darin, dass `_feed()` schlicht *jeden* Link verwendet, während `_archive()` zuerst bestimmen muss, welche Security-Advisories Relevanz besitzen.

Dazu werden in einem ersten Schritt alle der nach Jahren geordneten Archiv-Seiten bis zurück zum Jahr 1999 aufgerufen. (Der/Die Administrator/-in kann hier wahlweise auch ein jüngeres Jahr einstellen.) Aus den Webseiten werden die Links zu den Security-Advisories extrahiert und zu einer Gesamtliste vereint. Dabei bilden die aus dem Verweistext extrahierten Paketnamen den assoziativen Index dieser Liste, deren Elemente wiederum eine Liste von URLs enthalten, die dem entsprechenden Paketnamen zugeordnet sind.

Der Wiederanlauf dieses Moduls setzt auf der in Abschnitt 10.3 beschriebenen Liste neuer Pakete auf. Es wird ein Paket nach dem anderen aus der Liste herausgelöst und als „verarbeitet“ vermerkt.

Dieser Paketname wird nun mit Hilfe von *zweifachem Matching* (siehe Abschnitt 10.4) mit der Gesamtliste aller Security-Advisories abgeglichen. Die aus diesem Vergleich hervorgehenden Links führen nunmehr zu als relevant eingestuften Security-Advisories.

Die Strukturierung der Debian Security Advisories ermöglicht es beim sequentiellen Durchlaufen des Textes anhand der Überschriften die interessanten Datenblöcke zu finden und zu extrahieren. Der Satzbau variiert von allen im Zuge dieser Diplomarbeit analysierten Security-Advisory-Feeds am wenigsten, wodurch sich die Security-Advisories beinahe als maschinenlesbar bezeichnen ließen.

Eine Besonderheit der Debian Security Advisories ist zudem die vom Datum abhängige Lizenzierung. Das Plugin setzt abhängig vom *Reported Date* entweder die MIT-Lizenz oder die Open Publication License Version 1.0 ein. Für alle Security-Advisories, die nach dem 25. Jänner 2012 berichtet wurden, erfolgt die Weiterverarbeitung der Inhalte unter der MIT-Lizenz.

10.9.2 Ubuntu security notices

Die Verarbeitung des Syndication-Feed erfolgt auf dieselbe Art wie bei den Debian Security Advisories. Jeder Eintrag enthält einen Link zum eigentlichen Security-Advisory, welches in der Folge verarbeitet wird.

Es existiert zwar ein Web-Archiv, jedoch ist dieses nicht in Form von Listen abrufbar, sondern müsste „durchgeblättert“ werden. Was jedoch viel schwerer wiegt, ist die Tatsache, dass die Verweistexte keine Paketnamen bzw. nur Abwandlungen davon enthalten. Somit kann auf dieser Ebene keine Ermittlung der Relevanz stattfinden. Das komplette Web-Archiv müsste heruntergeladen und aktuell gehalten werden, wäre da nicht wie in Abschnitt 2.2.2 beschrieben die Mailingliste ubuntu-security-announce [30], die als vollständiges Archiv im mbox-Format zum Download angeboten wird. Die Mailingliste enthält sämtliche Security-Advisories in jener Form, in der sie zum Zeitpunkt ihrer Publikation vorlagen. Aus diesem Grund wird das Archiv lediglich als Index verwendet, da die Funktion, welche die Security-Advisories auswertet so nur das einheitliche(re) Format der Website berücksichtigen muss.

Ähnlich wie bei den Debian Security Advisories wird auch hier das Archiv verwendet um daraus Verweise zu relevanten Security-Advisories zu extrahieren. Wie schon erwähnt wird über die Relevanz eines Eintrags nicht auf Basis des Titels sondern auf Grundlage des gesamten Textes entscheiden. Im konkreten Fall bedeutet dies, dass der Quelltext jedes in der mbox gespeicherten Emails mit Hilfe einer ***Volltextsuche*** (siehe Abschnitt 10.4 mit der Liste neuer Pakete abgeglichen wird.

Diese Methode bedeutet außer dem gelegentlichen Download der mbox keine zusätzliche Belastung für den bereitstellenden Server.

Aus diesem Abgleich geht eine duplikatfreie Liste von URLs hervor, die zu als relevant eingestuften Security-Advisories führen. Erst wenn diese Liste vollständig abgearbeitet wurde, vermerkt das Plugin jene Pakete, die in den regulären Ausdruck der Volltextsuche miteinbezogen wurden, als „abgearbeitet“ bzw. „nicht (länger) neu“.
Um einen Wiederanlauf nach der zuletzt aufgerufenen URL zu ermöglichen, wird die Liste der abgearbeiteten URLs bis zur Bestätigung der vollständigen Abarbeitung der Liste neuer Pakete gespeichert. Dadurch können die entsprechenden URLs einfach übersprungen werden, sodass es zu keiner doppelten Verarbeitung von Security-Advisories kommt.

Die Verarbeitung der einzelnen Security-Advisories geschieht wie im Fall der Debian Security Advisories durch Analyse des (eigentlich für Menschen geschriebenen) Textes. Jedes Security-Advisory enthält eine Sektion mit Instruktionen zum Aktualisieren des Systems. [20] In dieser werden exakte Paketnamen nebst deren problembehebenden Versionen in Form von Links angeführt. Diese Listen sind nach Distributionsversion unterteilt.
Durch die beschriebene Strukturierung ist das Auslesen der Paketversionen äußerst zuverlässig zu bewerkstelligen. Sofern in Zukunft keine größeren Formatänderungen stattfinden, können Ubuntu security notices als (nahezu) maschinenlesbar angesehen werden.

10.9.3 NIST / National Vulnerability Database

Das NIST verwendet zur Bereitstellung von Syndication-Feed und Archiv das selbe XML-Schema, welches vollständige Einträge der *NVD* enthält. Nach dem Herunterladen der XML-Dateien können diese ohne die Server des NIST zu belasten verarbeitet werden. Die Funktionen `_feed()` und `_archivei()` greifen schlussendlich beide auf dieselbe Funktion zum verarbeiten der jeweiligen XML-Datei zurück. Der wesentliche Unterschied besteht darin, dass `_archivei()` Routinen beinhaltet, um die Archive aktuell zu halten. Die Archive vergangener Jahre werden nur einmalig heruntergeladen, wohingegen das Archiv des aktuellen Jahres ca. alle drei Wochen aktualisiert wird.

Um zu verhindern, dass zwischen Syndication-Feed und dem Archiv des aktuellen Jahres eine „Lücke“ entsteht, wird dieses Archiv erneut heruntergeladen, wenn erkannt wird, dass der Syndication-Feed mehr als 24 Stunden nicht verarbeitet wurde.

Die (doch recht großen) XML-Dateien werden aus Performance-Gründen nicht direkt einem XML-Parser

zugeführt. Stattdessen werden sie mit Hilfe eines regulären Ausdrucks in ihre einzelnen Einträge aufgespalten. Um nun zu erkennen, ob ein solcher Eintrag relevant ist, wird eine **Volltextsuche** (siehe Abschnitt 10.4) eingesetzt. Diese bezieht sich jedoch nicht auf den vollständigen Eintrag, sondern auf jene Bereiche, welche die Paketinformationen enthalten.

Als relevant erkannte Einträge werden in normalisierte Advisory-Daten (siehe Abschnitt 10.6) überführt. Dies ist Dank der maschinenlesbar aufbereiteten Daten der NVD sehr leicht und zuverlässig durchzuführen. Eine Besonderheit der NVD-Einträge ist, dass im Gegensatz zu den Debian Security Advisories und den Ubuntu security notices verwundbare Versionen explizit angegeben werden. Diese Listen können mitunter mehr als 2000 Einträge umfassen. Die in Kapitel 11 behandelte Korrelation muss diesem Umstand Rechnung tragen, um Performance-Engpässe zu vermeiden.

Trotz der Tatsache, dass die Archive relativ statisch sind, müssen für einen fehlerfreien Wiederanlauf *sämtliche* Identifikatoren (konkret: *CVE-IDs*) verarbeiteter Security-Advisories temporär gespeichert werden. Bei der erneuten Ausführung nach einer Unterbrechung werden die entsprechenden Einträge übersprungen.
Würde nur die CVE-ID des zuletzt verarbeiteten Eintrags vermerkt, könnten bei einer möglichen Veränderung der Liste neuer Pakete Security-Advisories mit einer niedrigeren CVE-ID als der vermerkten fälschlicherweise übersprungen werden.

Die Limitierung der verarbeiteten Einträge pro Ausführung geschieht hier nicht, um die Server des NIST zu schützen, sondern um den Host des WizzanCrawlers bzw. den WizzanServer nicht zu überlasten. Der Umfang der NVD stellt einen nicht zu unterschätzenden Faktor dar.

Die Bestätigung der Abarbeitung der Liste neuer Pakete erfolgt wie bei den Ubuntu security notices für die gesamte Liste, sofern während der aktuellen Ausführung keine Unterbrechung aufgrund einer Limitierung erfolgte.

10.9.4 US-CERT

Wie in Abschnitt 2.2.5 beschrieben fallen Syndication-Feed und „Archiv“ im Fall des US-CERT zusammen. Derselbe Atom-Feed wird schlicht mit unterschiedlichen Parametern aufgerufen. Das bedeutet auch, dass – ähnlich wie beim NIST – im Hintergrund dieselbe Funktion die Verarbeitung der einzelnen Einträge durchführt.

Der Atom-Feed ist auf maximal 1000 Einträge begrenzt. Dies ist gleichzeitig auch die theoretisch maximale Anzahl als relevant eingestufter Security-Advisories. Aufgrund dieser Begrenzung wird auf die Implementierung eines Wiederanlaufprocederes verzichtet.
Die Limitierung der Anzahl extrahierter Advisories ist dennoch möglich. Der Atom-Feed beginnt mit den ak-

tuellsten Security-Advisories, somit gingen im Falle einer Limitierung nur ältere Einträge verloren.
Die Einträge an sich stellen vollständige Security-Advisories dar. Der Aufruf von Webseiten ist nicht erforderlich.

Die Ermittlung relevanter Security-Advisories erfolgt analog zu den Ubuntu security notices durch eine **Volltextsuche** (siehe Abschnitt 10.4). Die Besonderheit liegt hier in der Ermittlung der Paketnamen. Bei den bisher vorgestellten Security-Advisory-Feeds konnten die Paketnamen stets sauber identifiziert und extrahiert werden. Aufgrund der wechselnden Formulierungen ist dies beim US-CERT nicht der Fall. Stattdessen wird der in Abschnitt 10.4 beschriebene LNPRA auf den Titel jedes Eintrags angewandt. Als Ergebnis liefert dieser die übereinstimmenden Textstellen zurück, die in diesem Fall wort-ähnlich sind. Diese werden (mangels besserer bzw. robusterer) Alternativen als Paketnamen angegeben.

Das US-CERT legt den Fokus seiner Security-Advisories klar auf die Information und Koordination von Fachkräften. Die daraus resultierenden, häufig wechselnden Formulierungen können dennoch mit Hilfe regulärer Ausdrücke in normalisierte Advisory-Daten umgewandelt werden. Ob das Plugin in der Lage ist auch zukünftige Security-Advisories zuverlässig zu verarbeiten bleibt fraglich.

Korrelation von Paketen und Advisories

Dieses Kapitel behandelt die Korrelation der Daten des Software-Inventars mit den Informationen aus den Security-Advisory-Feeds. Es beschreibt die Verarbeitung von Advisory-Daten von deren Empfang bis zum Setzen der Beziehungen zwischen den einzelnen Objekten. Im Folgenden ist der Ablauf der Verarbeitung eines einzelnen Advisories in Form eines groben Überblicks beschrieben. Die nachfolgenden Kapitel gehen detailliert auf die wesentlichen Punkte ein.

1. Die Daten eines einzelnen Advisories werden aus der Warteschlange geladen.
2. Unerwünschte führende und nachgesetzte Zeichen werden aus den Versionsnummern entfernt.
3. Es werden jene Versionen extrahiert, die Teil einer Versionsangabe sind, welche die Grenze eines Intervalls bildet.
4. Aus diesen werden *Branches* (siehe Abschnitt 11.3.2) abgeleitet, welche es ermöglichen parallel entwickelte bzw. gewartete Versionen zu berücksichtigen.
5. Unter Berücksichtigung der Branches werden logische Verknüpfungen formuliert, welche Bereiche verwundbarer Versionen definieren. (siehe Abschnitt 11.3.3)
6. Die Versionsinformationen sind nach den Namen der entsprechenden Pakete gruppiert. Für jeden dieser in den Advisory-Daten spezifizierten Paketnamen werden aus der Datenbank alle passenden Paket-Objekte geladen. (siehe Abschnitt 11.2)
7. Für jedes Paket wird bestimmt, ob es durch die im Security-Advisory beschriebene/n Sicherheitsschwachstelle/n verwundbar ist, oder ob es diese behebt. (siehe Abschnitt 11.3)
8. Wenn dies der Fall ist, wird aus den Advisory-Daten ein Advisory-Objekt generiert, welches samt der entsprechenden Relationen in der Datenbank gespeichert wird.

Die Korrelation von Advisories und Paketen erfolgt also in zwei Schritten. Zuerst werden aus der Datenbank Paket-Objekte geladen, welche zum im Advisory spezifizierten Paketnamen passen. Im zweiten Schritt erfolgt für jedes Paket-Objekt der Vergleich der Versionsangaben.

11.1 Advisory-Queue und Advisory-Dispatcher

Wie bereits in Abschnitt 8.2 beschrieben, werden die vom WizzanCrawler empfangenen Advisory-Daten in eine Warteschlange gespeichert, anstatt sie direkt zu verarbeiten. Dies geschieht, um die Lastspitze am WizzanServer zu reduzieren, die beim Empfang der Advisory-Daten entstehen kann. (Der WizzanCrawler ist in der Lage einige hundert bis tausend Einträge zu übertragen.)

Um ein Advisory zu diesem Zeitpunkt eindeutig identifizieren zu können, wird (unter Ausschluss der Zeitstempel) aus seinen serialisierten Daten ein HMAC generiert. Dieser, die serialisierten Daten selbst und die Zeitpunkte der Generierung der Advisory-Daten bzw. der Erstellung des Warteschlangen-Eintrags werden in die Warteschlange aufgenommen.

Der HMAC wird später in den für relevant befundenen Advisory-Objekten weiterverwendet, sodass der WizzanServer sicherstellen kann, dass kein Advisory doppelt verarbeitet wird. Die Datenbank-Felder, in denen die HMACs gespeichert werden, sind dazu als eindeutiger Index definiert. Dieser bedeutet im Vergleich zu nicht indizierten Daten eine Effizienzsteigerung, da im Hintergrund ein balancierter Baum (*B-Tree*) für die Suchläufe verwendet wird.

Die Aufgabe des ***Advisory-Dispatchers*** ist es nun die in der Warteschlange befindlichen Advisory-Daten nacheinander der Verarbeitung zuzuführen. Weil der WizzanServer in PHP implementiert ist, wird dabei darauf geachtet, dass das *Maximum Execution Timeout* nicht überschritten wird.

Angestoßen wird der Advisory-Dispatcher durch einen bei der Installation des WizzanServer eingerichteten Cronjob, der schlicht eine bestimmte URL des WizzanServer aufruft und dadurch den entsprechenden Controller zur Ausführung bringt.

Die in den nachfolgenden Abschnitten beschriebenen, weiteren Verarbeitungsschritte sind zum Zweck des leichteren Verständnisses nicht anhand der Reihenfolge ihrer Abarbeitung sondern anhand ihrer Grundkonzepte beschrieben. Die genaue Abfolge erschließt sich aus der Einleitung von Kapitel 11.

11.2 Korrelation der Paketnamen

Analog zu den in Abschnitt 10.4 beschrieben Matching-Varianten des WizzanCrawler (Exakte Übereinstimmung, Gegenseitiger Abgleich anhand regulärer Ausdrücke und Volltextsuche) benötigt auch der WizzanServer Methoden um trotz ungenauer Angabe von Paketnamen relevante Paket-Objekte aus der Datenbank zu laden. Dieser erste Schritt im Prozess der Korrelation muss der Bestimmung verwundbarer Paketversionen anhand empfangener Versionsangaben vorausgehen, da er die dazu benötigten Paket-Objekte liefert.

Diese Korrelation von Paketnamen wird aus Performance-Gründen durch ein SQL-Statement – d.h. rein in der Datenbank – implementiert. Dies bedeutet gegenüber dem WizzanCrawler, der einen gegenseitigen Abgleich anhand der Varianten zweier Paketnamen durchführen kann, dass mit SQL nur noch ein Paketname variiert werden kann. Dies ist dadurch begründet, dass der Wert eines Datenbankfelds während einer Abfrage nicht gleichzeitig modifiziert und für einen Vergleich herangezogen werden kann.

11.2.1 Die Datenbankabfrage

Der WizzanServer erzeugt auf Basis eines gegebenen Paketnamens eine Liste von Paket-Objekten, die auf diesen Namen passen. Dazu stehen ihm mehrere miteinander kombinierbare Varianten zur Verfügung. Bei jeder dieser Varianten wird der im Hintergrund verwendeten Datenbank-Abfrage eine weitere Bedingung hinzugefügt, indem diese mittels logischem Oder mit dem SQL-Befehl verknüpft wird. D.h. die Erfüllung einer Bedingung reicht, damit ein Paket-Objekt geladen wird.

Im Folgenden werden die implementierten Bedingungen bzw. Varianten in Prosa und anhand kurzer Pseudocode-Ausschnitte erklärt. Worte mit vorangestelltem Dollar-Zeichen sind als Platzhalter für die tatsächlich verwendeten Werte zu verstehen.
Der SQL-Befehl beginnt mit `SELECT * FROM $Pakettabelle`, die Befehle werden mit `OR` verknüpft.

- **`WHERE name = '$empfangener-paketname'`** – Die Überprüfung der Paketnamen auf exakte Übereinstimmung ist in der Praxis von eher geringer Bedeutung, weil keiner der in dieser Diplomarbeit betrachteten Security-Advisory-Feeds zuverlässig exakte Paketnamen angibt.
- **`WHERE name LIKE '$empfangener%$paketname%'`** – Bei dieser Variante treten an die Stelle nicht alphanumerischer Zeichen im empfangenen Paketnamen Bereiche (markiert durch `%`), die beliebige Zeichen enthalten können. Damit müssen die Paketnamen nicht mehr exakt übereinstimmen, sondern der Name eines Paket-Objekts muss aus den gleichen „Bestandteilen“ bestehen wie der Paketname des

Advisories.

- **`WHERE name LIKE '$empfangener%'`** – Um die „Trefferchance“ weiter zu erhöhen, kann die nötige Übereinstimmung auf den ersten Bestandteil *am Beginn* des Namens beschränkt werden. Während der Testläufe hat sich diese Herangehensweise als die brauchbarste erwiesen.
- **`WHERE name LIKE '%$empfangener%$paketname%'`** – Es ist auch möglich die nötige Übereinstimmung auf das Vorkommen der Bestandteile *an beliebiger Stelle* des Paketnamens zu reduzieren.
- **`WHERE name LIKE '%$empfangener%'`** – Dies kann ebenso auf den ersten Bestandteil des Namens eingeschränkt werden.

Die beiden letztgenannten Varianten produzieren relativ viele falsch-positive Ergebnisse. Dementsprechend zurückhaltend sollten sie aktiviert werden, wenn der/die Administrator/-in in der Konfiguration des WizzanServers den WizzanCrawler-Plugins die jeweils passenden Varianten der Paket-Korrelation zuweist.

11.2.2 Aliases

Bestimmte Fälle können durch die oben vorgestellten Korrelationsvarianten nicht abgedeckt werden. Ein Beispiel hierfür ist der Adobe Flash Player. Dieser wird bei Ubuntu und Debian als flashplugin-installer [78] bzw. flashplugin-nonfree [79] bezeichnet, während NIST die Herstellerbezeichnung verwendet. [80]

Anstatt noch ungenauere Korrelationsvarianten zu implementieren, wird dem/der Administrator/-in die Möglichkeit gegeben Aliases anhand von Äquivalenzklassen zu definieren. Um beim oben angeführten Beispiel zu bleiben, würden sich hier alle drei Bezeichnungen in der selben Äquivalenzklasse befinden. Anstatt nur einer Datenbankanfrage (z.B. `WHERE name LIKE 'adobe%'`) werden zwei weitere eingesetzt, die jeweils einen der äquivalenten Paketnamen verwenden.

11.3 Korrelation der Paketversionen

Der zweite Schritt der Korrelation besteht nun darin aus den bei der Korrelation der Paketnamen geladenen Paketen jene auszuwählen, deren Versionsnummer den Versionsangaben im Security-Advisory entspricht. Diese Übereinstimmung kann zwei Ausprägungen annehmen:

1. Ein Paket ist durch die im Security-Advisory beschriebene Sicherheitsschwachstelle *verwundbar*.

2. Die im Security-Advisory beschriebene Sicherheitsschwachstelle wird durch ein Paket *behoben*. Dabei ist die exakte Übereinstimmung der Versionsnummern ausschlaggebend, da ansonsten jedes Paket, das nicht verwundbar ist, gleichzeitig als das Problem behebend angesehen werden müsste.

Aus den in den Advisory-Daten enthaltenen Versionsangaben leitet der WizzanServer logisch verknüpfte Bedingungen ab, anhand derer er entscheidet, ob einer der zwei oben genannten Fälle eintritt, oder ob das Advisory irrelevant ist.

Die Hersteller bzw. Distributoren verwenden teilweise Ergänzungen zu den von den eigentlichen Software-Entwicklern/-innen benutzten Versionsnummern. Hier ist es notwendig die Versionsnummern entsprechend aufzubereiten, um die weitere Verarbeitung nicht aufgrund der unterschiedlichen Schemata ad absurdum zu führen.

Die nachfolgenden Unterabschnitte beschreiben die Aufbereitung der Versionsangaben sowie die Entscheidungsfindung des WizzanServers.

11.3.1 Ignorieren von Prä- und Postfix

Eine Versionsnummer, die um Prä- und Postfix ergänzt wurde ist beispielsweise `1:1.4.1-1.sarge7`. Die eigentliche Versionsnummer ist fett hervorgehoben.
Diese Zusätze zu Versionsnummern werden von Distributoren verwendet, um Ergänzungen am Quelltext zu kennzeichnen, die keine wesentlichen Änderungen an der Funktionalität der Software bedeuten. [66] Üblicherweise dienen solche Ergänzungen der besseren Anpassung an die Systemumgebung der jeweiligen Distribution.

Gesetzt den Fall, ein Advisory bezeichnete alle Versionen unterhalb von 1.4.9 als verwundbar, kann der Vergleich `'1:1.4.1-1.sarge7' < '1.4.9'` nicht sinnvoll durchgeführt werden. Die eigentliche Versionsnummer muss demnach vor dem Vergleich der Versionsnummern freigestellt werden. Dies gilt für die Versionsnummern der im Inventar befindlichen Paket-Objekte ebenso wie für Versionsangaben von Advisory-Daten. Implementiert wird dieser Vorgang mit Hilfe regulärer Ausdrücke.

In obigem Beispiel wird zur Vereinfachung ignoriert, dass ein rein lexikalischer Vergleich für die meisten Versionsschemata unzureichend ist. Abschnitt 11.3.4 ist dem richtigen Vergleichen von Versionsnummern gewidmet.

Prä- und Postfixe werden zwar während der Korrelation weitgehend *ignoriert*, jedoch bleiben sie in den Datensätzen von Paket- und Advisory-Objekten *erhalten*. Dadurch kann bei späteren Änderungen am Code des WizzanServer immer noch auf die Originaldaten zurückgegriffen werden. Zudem wird den Benutzern/-innen

auf diese Weise der volle Informationsumfang zuteil.

11.3.2 Branches

Als *Branches* werden in der Softwareentwicklung parallel geführte Entwicklungszweige der selben Software bezeichnet. [81, S.5] Wo sich die Quelltexte dieser Zweige nicht unterscheiden, enthalten sie potentiell dieselben Sicherheitsschwachstellen. Wird nun eine solche behoben, ist im entsprechenden Security-Advisory üblicherweise eine Versionsangabe pro Branch enthalten.

Um das Konzept der Branches verständlich aufzubereiten, sei ein Fallbeispiel angeführt. Das Debian Security Advisory DSA-2492-1 php5 – buffer overflow [82] gibt an, dass die Sicherheitsschwachstelle in den beiden Versionen `5.3.3-7+squeeze13` und `5.4.4 rc1-1` behoben ist. Dementsprechend generiert das WizzanCrawler-Plugin für die Angabe verwundbarer Versionen die Einträge **<**`5.3.3-7+squeeze13` und **<**`5.4.4 rc1-1`.

Werden diese Bedingungen mit der Information kombiniert, dass die oben genannten Versionen *nicht* verwundbar sind, ließe sich folgende Reduktion ableiten: `<5.4.4 rc1-1 AND NOT 5.3.3-7+squeeze13`. Aufgrund der Tatsache, dass PHP in mehreren Zweigen entwickelt wird, ist diese Folgerung jedoch nicht korrekt. [83] Versionsnummern zwischen `5.3.3-7+squeeze13` und `5.4` dürfen somit *nicht* als verwundbar eingestuft werden.

Damit der WizzanServer diese Ausnahme treffen kann, müssen zuerst mögliche Branches aus den Versionsnummern abgeleitet werden. Die Ermittlung von Branches ist jedoch nur für Versionsangaben erforderlich, welche die Grenze eines Intervalls definieren. Exakte Versionsangaben (im Sinne von Gleichheit bzw. Ungleichheit) werden in diesem Zusammenhang ignoriert bzw. im Programmlauf übersprungen.
Dies bewahrt den WizzanServer vor massiven Performanceproblemen, wenn in einem Security-Advisory mehrere hundert bis tausend verwundbare Versionsnummern angeführt sind.

Nachdem mögliche Ergänzungen der Versionsnummern (siehe Abschnitt 11.3.1) entfernt wurden, werden die Branches durch paarweises Vergleichen der Versionsnummern ermittelt. Dabei entstehen aus jedem Paar potentiell zwei Branches. Die entstehende Liste enthält keine Duplikate, somit reduziert sich die Anzahl der Branches auf das nötige Mindestmaß.

Die Ermittlung zweier Branches ist an sich relativ simpel:

1. Zuerst wird jener Teil am Beginn der Versionsnummern bestimmt, der bei beiden übereinstimmt. Um beim oben angeführten Beispiel zu bleiben ist dies „`5.`“.

2. An diesen Teil wird jeweils das nächste Zeichen der entsprechenden Versionsnummer angefügt. Sofern es sich dabei um eine Ziffer handelt, werden so lange alle folgenden Ziffern angefügt, bis das nächste Zeichen keine Ziffer mehr ist, oder das Ende der Versionsnummer erreicht wurde.

Die aus obigem Beispiel resultierenden Branches sind `5.3` und `5.4`. Falls keine Branch ermittelt werden kann, wird eine mit einem Leerstring bezeichnete Branch angelegt, um die Struktur der im Folgenden beschriebenen Arrays nicht zu verändern.

Um nun die oben beschriebene Ausnahme zu treffen, werden die Versionsangaben der Advisory-Daten um die nötigen Intervallgrenzen erweitert, sodass folgende Bedingungen entstehen: `<5.4.4 rc1-1 AND >=5.4 AND <5.3.3-7+squeeze13` Die niedrigste Branch wird hier nicht miteinbezogen, da die Bedingung `>=5.3` eine unnötige Begrenzung des Bereichs verwundbarer Versionen darstellen würde. Im Fall einer einzelnen „leeren“ Branch wäre diese gleichzeitig die „niedrigste“ Branch.

11.3.3 Definition verwundbarer Versionen

Der vorige Abschnitt geht bereits auf die logische Verknüpfungen der einzelnen Versionsangaben ein. Diese sind jedoch nur ein Teilaspekt bei der Definition verwundbarer Versionen, welche im Folgenden anhand eines Fallbeispiels erläutert werden.

Listing 11.1 zeigt den relevanten Ausschnitt der Advisory-Daten, wie er vom entsprechenden WizzanCrawler-Plugin aus dem Debian Security Advisory DSA-1266-1 gnupg – several vulnerabilities generiert wurde.

Das Advisory nennt für jede der zum Zeitpunkt der Veröffentlichung aktuellen Distributionsversionen (stch, sarge und sid) den Paketnamen und jene Versionen, in denen die zugrundeliegende Sicherheitsschwachstelle behoben wurde. [84] Das WizzanCrawler-Plugin ergänzt diese Daten um den Schluss, dass alle niedrigeren Versionen verwundbar sein müssen.

```
[fixed_package_versions] => Array (
    [etch] => Array (
        [gnupg] => Array (
            [0] => 1.4.6-2 ))
    [sarge] => Array (
        [gnupg] => Array (
            [0] => 1.4.1-1.sarge7 ))
```

```
    [sid] => Array (
        [gnupg] => Array (
            [0] => 1.4.6-2 ))
)
[vulnerable_package_versions] => Array (
    [etch] => Array (
        [gnupg] => Array (
            [0] => <1.4.6-2 ))
    [sarge] => Array (
        [gnupg] => Array (
            [0] => <1.4.1-1.sarge7 ))
    [sid] => Array (
        [gnupg] => Array (
            [0] => <1.4.6-2 ))
)
```

Listing 11.1: Angabe verwundbarer und problembehebender Versionen in den Advisory-Daten

Aus diesen Daten werden – wie in Abschnitt 11.3.2 beschrieben – die Branches ermittelt. Das Ergebnis ist in Listing 11.2 dargestellt. Die Distributionsversion wird ignoriert, da sie nicht durchgehend von allen Security-Advisory-Feeds angegeben wird. Dennoch wird sie weiterhin als Bestandteil der Advisory-Daten von den WizzanCrawler-Plugins geliefert, da sie zu einem späteren Zeitpunkt Verwendung als zusätzliches Plausibilitätskriterium bei der Korrelation von Paketnamen dienen könnte.

```
Array (
    [gnupg] => Array (
        [0] => 1.4.1
        [1] => 1.4.6
)   )
```

Listing 11.2: Aus den Advisory-Daten extrahierte Branches

Die Definition der verwundbaren Versionen entsteht nun aus der Kombination der Liste der Branches mit den Versionsangaben in den Advisory-Daten. Dabei werden die Branches absteigend sortiert, beginnend mit der spezifischsten Angabe bzw. jener mit der höchsten Versionsnummer. Somit können die Original-Versionen

anhand eines einfachen lexikalischen Vergleichs der ersten passenden Branch zugeordnet werden. (*First Match*) Listing 11.3 zeigt das Ergebnis dieses Vorgangs.

```
Array (
    [gnupg] => Array (
        [1.4.6] => Array (
            [!] => Array (
                [0] => 1.4.6-2 )
            [<] => Array (
                [0] => 1.4.6-2 )
            [>=] => Array (
                [0] => 1.4.6 )
        )
        [1.4.1] => Array (
            [!] => Array (
                [0] => 1.4.1-1.sarge7 )
            [<] => Array (
                [0] => 1.4.1-1.sarge7 )
)   )   )
```

Listing 11.3: Definition verwundbarer Versionen

Dieses Array ist der Kern der Korrelation der Paketversionen. Sofern bei der Korrelation der Paketnamen (siehe Abschnitt 11.2) zu dem angegebenen Namen (hier: `gnupg`) Paket-Objekte geladen wurden, wird versucht jedes dieser Pakete anhand seiner Versionsnummer einer Branch zuzuordnen. Einer mit Leerstring bezeichneten Branch werden alle Paket-Objekte zugeordnet, sofern sie nicht vorher mit einer spezifischeren Branch assoziiert werden.

Fällt ein Paket-Objekt in eine Branch, so werden die Versionsangaben und die Paketversion mit Hilfe der entsprechenden logischen Operatoren und Verknüpfungen miteinander verglichen. Negationen (`!`, `!=`) und Operatoren, die sich auf Bereiche beziehen, (`<`, `<=`, `>`, `>=`) werden mit Hilfe eines logischen **Und** verknüpft. Exakte Versionsangaben (`=`) werden mit den übrigen durch ein logisches **Oder** verknüpft.

Werden die Mengen aller möglichen Versionsangaben als Gerade begriffen, so ist es mit den implementierten logischen Verknüpfungen nicht möglich, voneinander unabhängige Streckenabschnitte innerhalb der selben Branch zu beschreiben. Während der Implementierung zeigte sich jedoch, dass diese vereinfachende

Herangehensweise vollkommen ausreicht, weil eine Sicherheitsschwachstelle im Regelfall an einem bestimmten Punkt der Gerade behoben wird und danach nicht wieder „reaktiviert“ wird. Ferner wird angenommen, dass für voneinander unabhängige Sicherheitsschwachstellen voneinander unabhängige Security-Advisories publiziert werden.

11.3.4 Vergleichen von Versionsnummern

Die Schwierigkeit beim Vergleich von Versionsnummern liegt darin, dass kein einheitliches, verbindliches Versionierungsschema existiert. [85] Jedoch ist im Linux- und UNIX-Umfeld üblicherweise ein Schema anzutreffen, welches sich aus Hauptnummer (*major number*), Nebennummer (*minor number*) und Revisionsnummer (*revision number*) getrennt durch Punkte zusammensetzt.

Von diesem Schema geht auch die in PHP implementierte Funktion **`version_compare()`** aus. [86] Sie ist ausreichend robust implementiert, um auch Abwandlungen dieses Versionierungsschemas korrekt vergleichen zu können.

> *Zuerst ersetzt die Funktion _, - und + durch einen Punkt . in den Versionsangaben und setzt vor und nach jeder Kette aus nicht-numerischen Zeichen Punkte ein, sodass beispielsweise '4.3.2RC1' zu '4.3.2.RC.1' wird. Dann wird diese Zeichenkette an den Punkten aufgespalten wie wenn man explode('.', $ver) benutzen würde. Anschließend werden von links nach rechts die Teile verglichen. Wenn ein Teil spezielle Zeichen enthält, werden diese nach der folgenden Reihenfolge behandelt:* ***jede Zeichenkette, die nicht in dieser Liste vorkommt < dev < alpha = a < beta = b < RC = rc < # < pl = p****. Auf diese Weise können nicht nur Versionen verschiedener Tiefe wie '4.1' und '4.1.2' sondern auch alle anderen Versionen verglichen werden, die sich auf bestimmte Entwicklungsstadien [...] beziehen. [86]*

Der Funktion werden als Parameter die beiden zu vergleichenden Versionsnummern und der Vergleichsoperator (siehe Listing 11.3 bzw. Abschnitt 10.8) übergeben. Es wird ein boolescher Wert zurückgeliefert, der angibt, ob die so formulierte Ungleichung wahr oder falsch ist.

11.4 Fallback für die Korrelation von Paketversionen

Enthalten Advisory-Daten keinerlei Versionsnummern sondern nur Paketnamen, basiert die Entscheidung, ob ein Paket-Objekt verwundbar ist oder nicht auf den Zeitstempeln der beiden Entitäten.

Liegt der Zeitpunkt der Veröffentlichung des Security-Advisories nach dem der Erstellung des Paket-Objekts, so wird dieses als verwundbar betrachtet.

11.5 Ermittlung des Bedrohungsgrades

Wie in Abschnitt 10.6 beschrieben können die Advisory-Objekte potentiell drei unterschiedliche Angaben zum Bedrohungsgrad enthalten:

- den *CVSS-Score* (`cvss`), wie er im Security-Advisory angegeben wird,
- die *Herausgeber-spezifische Bewertung* der Kritikalität der Sicherheitsschwachstelle (`severity`),
- sowie den *automatisch generierten CVSS-Score* (`auto_cvss`).

Bei der späteren Ermittlung des Bedrohungsgrades am WizzanServer wird dem CVSS-Score der Vorzug gegenüber dem angepassten Schätzwert gegeben. Dieser wiederum wird gegenüber einem automatisch berechneten CVSS-Score bevorzugt. Welcher Wert schlussendlich verwendet wird, hängt demnach vom (bei der Implementierung angenommenen) Informationsgehalt der Originaldaten sowie von der angenommenen Zuverlässigkeit der drei Werte ab.

Über die Beziehungen zwischen Advisory-Objekten und Paket-Objekten sowie zwischen Paket-Objekten und Host-Objekten lassen sich aus den drei oben genannten Angaben die Bedrohungsgrade von Hosts und Paketen ableiten.

Um diese Bedrohungsgrade für alle Objekte in Listendarstellungen anzeigen zu können ohne sämtliche assoziierten Objekte laden zu müssen, besitzen die Host- und Paket-Objekte das Feld `threat_levels`. (siehe Abschnitt 8.3) Dieses enthält ein serialisiertes Array bestehend aus den ID-Nummern aller das Objekt betreffenden Advisories sowie deren Bedrohungsgraden. Der höchste dieser Bedrohungsgrade wird als der Bedrohungsgrad des jeweiligen Host- bzw. Paket-Objekts gewertet. Der Inhalt des Feldes wird beim Setzen bzw. Löschen von Beziehungen mitgeführt.

11.6 Aussortieren falsch-positiver Relationen

Die Benutzeroberfläche gibt Anwendern/-innen die Möglichkeit Beziehungen zwischen Paket- und Advisory-Objekten als falsch-positiv zu kennzeichnen und dies auch wieder rückgängig zu machen. Diese Kennzeichnung geschieht im Hintergrund dadurch, dass die Beziehung aus der jeweiligen, standardmäßig verwendeten Pivot-Tabelle installedpackages_advisories bzw. updatablepackages_advisories gelöscht und in der entspre-

chenden Tabelle fp_installedpackages_advisories bzw. fp_updatablepackages_advisories neu angelegt wird. (Das Datenbank-Schema wird in Abschnitt 8.3 behandelt.)

Durch dieses Vorgehen bleiben die Advisory-Objekte sicher erhalten. Somit müssen die Relationen nicht nach jedem Lauf des WizzanCrawler erneut entfernt werden.

11.7 Erkennen problembehebender Paket-Objekte

Um die Benutzer/-innen beim Aussortieren falsch-positiver Relationen zu unterstützen, werden im Zuge der Korrelation einem Paket-Objekt jene Advisories zugeordnet, die das Paket-Objekt explizit als das grundlegende Problem behebend kennzeichnen. In diesem Zusammenhang könnten Fehlentscheidungen dazu führen, dass verwundbare Pakete nicht als solche erkannt werden. Deshalb muss beim Paketnamen *und* bei der Paketversion *exakte Übereinstimmung* herrschen, bevor diese Beziehung gesetzt wird.

Das Setzen dieser Relationen geschieht über die Tabellen fixed_installedpackages_advisories bzw. fixed_updatablepackages_advisories.

Darstellung der Ergebnisse der Korrelation

Sämtliche Host-, Paket- und Advisory-Objekte können in Listen und einzeln dargestellt werden. In Listendarstellung sind die Objekte auf ihre wesentlichen Merkmale reduziert. Entlang der Beziehungen der Objekte untereinander kann der/die Benutzer/-in schnell und unkompliziert zwischen Objekten bzw. Objektlisten navigieren. Dadurch sind diese Zusammenhänge für ihn/sie intuitiv erfassbar.

Jedes Objekt erhält eine Hintergrundfarbe, die auf Grundlage von Bedrohungsgrad und Zuverlässigkeit ermittelt wird.
Der **Bedrohungsgrad**, dessen Werte von null bis zehn reichen wird auf einer Farbskala, die von Grün über Gelb bis Rot reicht, aufgetragen. Ist er Bedrohungsgrad nicht spezifiziert, wird ein Grauton verwendet, um zu kennzeichnen dass für das betreffende Objekt Advisories vorliegen und es (aller Wahrscheinlichkeit nach) verwundbar ist.
Die **Zuverlässigkeit** ist durch den/die Administrator/-in konfigurierbar. Ihr Wert wird in Prozent angegeben und anhand der Konfiguration aus dem Feld `source_plugin` des Advisory-Objekts ermittelt. (siehe Abschitt 10.6) Dazu wird aus dem Feld `threat_levels` der höchste Bedrohungsgrad und dessen Advisory-ID ermittelt. Mit deren Hilfe wird das zugeordnete Advisory-Objekt geladen und darüber der Wert des Feldes `source_plugin` festgestellt. Über die Zuverlässigkeit wird die Buntheit (d.h. die Entfernung des Farbwertes von Weiß) gesteuert, wobei ein höherer Wert sattere Farben bedeutet.

Als Ausblick auf die rechtlichen Aspekte der Darstellung der extrahierten Informationen in Abschnitt 12.1 sei erwähnt, dass die grafische Aufbereitung eines Advisory-Objekts durch den WizzanServer mehr eine Ansammlung von Links und Meta-Informationen als ein informierendes Security-Advisory ist.

Abgesehen von den rechtlichen Aspekten dient diese Form der Darstellung einem viel einfacheren Zweck: Aufgrund der Vielfalt der Formate, in denen Security-Advisory-Feeds bereitgestellt werden, ist die Erfassung des beschreibenden Inhalts eines Security-Advisories nicht trivial. Anstatt Auslassungen und ggf. Bedeutungsänderungen zu riskieren, wird lediglich per Link auf das Security-Advisory verwiesen. So gesehen zeigt Wizzan

den Benutzern/-innen keine Advisories an, sondern *verweist* sie nur darauf.

Für Advisories und Paket-Objekte wird zusätzlich noch ausgewiesen, welche Pakete das Problem beheben, bzw. welche Advisories Sicherheitsschwachstellen beschreiben, die durch das Paket behoben werden. Dies soll den/die Benutzer/-in beim Aussortieren von falsch-positiven Relationen unterstützen. Dies erfolgt wiederum, indem der/die Benutzer/-in auf die entsprechenden Links in der Benutzeroberfläche klickt.

12.1 Lizenzrechtliche Aspekte

Die folgenden Ausführungen knüpfen an die Betrachtungen des Abschnitt 10.2 an, in welchem unter anderem die nutzungsrechtliche Arbeitsgrundlage der Weiterverarbeitung von Security-Advisories definiert wird. Ziel ist es die Daten in einer Weise aufzubereiten, die weder zu Missverständnissen noch zu falschen Annahmen durch die Anwender/-innen von Wizzan führt.

Die Darstellung der Daten von Advisory-Objekten und damit den Abwandlungen von Security-Advisories erfolgt analog zu deren Darstellung in Feed-Readern. Im Wesentlichen sind dies Titel, Kurzbeschreibung, und Kritikalität. Dazu kommen rechtliche Hinweise, Verweise auf das originale Security-Advisory, weiterführende Informationen und Lizenztexte. Links auf externe Seiten sind als solche grafisch gekennzeichnet. Zudem ist bei der Darstellung von Advisory-Objekten ein Satz eingefügt, welcher klarstellt, dass sich alle angezeigten, rechtlichen Vermerke auf den Inhalt der Originaldaten, d.h. der Security-Advisory-Feeds beziehen.

Wizzan ist für die firmeninterne Verwendung konzipiert. Der/Die Betreiber/-in der Software kann durch diese Konstellation zwar Security-Advisory-Feeds mit relativ restriktiven Nutzungsbedingungen (siehe Abschnitt 2.2.2) verarbeiten lassen, jedoch ist er/sie nicht unbeteiligte/r Konsument/-in, sondern Akteur/-in. Somit muss er/sie seinen/ihren aus den Nutzungsbedingungen entstehenden Pflichten eigenverantwortlich nachkommen, da er sämtliche Informationen direkt von den Security-Advisory-Feeds der Hersteller bzw. Distributoren und Sicherheitsdienstleister bezieht. Wizzan ist in diesem Zusammenhang einem Feed-Reader bzw. einem Webbrowser ähnlich.

Die Implikationen, die öffentlicher oder quasi-öffentlicher Zugang zum WizzanServer mit sich bringen könnte, werden in dieser Diplomarbeit nicht erörtert.

Manche Herausgeber von Security-Advisory-Feeds verlangen die Angabe zusätzlicher Informationen, die nicht in wirklich als Copyright- oder Lizenzvermerk einzustufen sind. (siehe auch Abschnitt 2.2.5) Zu diesem Zweck existiert das Feld `other_legal_notes`, welches Raum für derartige Einträge bietet.

Lizenzierung von Wizzan

Sämtlicher Quelltext, der im Rahmen dieser Diplomarbeit erstellt wurde, ist unter der GNU General Public License Version 2 or later (GNU GPLv2+) lizenziert. Dies trifft für den Code der WizzanClients (**WizzanAgent**, **WizzanBaseLib** und **WizzanCrawler** sowie für das Kohana-Modul wizzan zu, welches den **WizzanServer** implementiert.

Wizzan basiert jedoch auf dem Quelltext von Entwicklern, die ihre Arbeit ihrerseits unter einer anderen Lizenz zu Verfügung stellen. Diese Lizenzen sind laut der Free Software Foundation kompatibel zur GNU GPLv2+. [68] (Sofern im Folgenden keine Referenzen angegeben sind, werden die Lizenzen im jeweiligen Quelltext angegeben.)

- Das Python-Modul **rfc3339** steht unter der ISC License.
- Das **Kohana-Framework** verwendet die Modified (3-point) BSD License. [67]
- Das Kohana-Modul **useradmin**, welches die Benutzerverwaltung implementiert, steht unter der Simplified BSD License.
- Das Kohana-Modul **pagination**, das eine Abhängigkeit von useradmin darstellt, steht unter der Modified BSD License.
- Das Kohana-Modul **tableau**, das den Aufbau von Tabellen implementiert, verwendet die MIT License.
- Das JavaScript-Framework **jQuery** steht unter der MIT License.
- Das jQuery-Plugin **DataTables** verwendet die Modified (3-point) BSD License.

Bekannte Probleme, Weiterentwicklung

Die Möglichkeiten der Auswertung von Security-Advisories sind noch nicht voll ausgereizt. Es fehlt vor allem noch an Erfahrungswerten, die nur durch eine längere Einsatzdauer von Wizzan zustande kommen können.
Aufgrund des modularen Aufbaus von Wizzan können jederzeit weitere Datenquellen „angezapft“ werden. Eine breite Anwendbarkeit über verschiedenste Distributionen bzw. Produkte hinweg kann jedoch nur durch entsprechende finanzielle oder tätige Hilfe erreicht werden.

Bei der Auswertung von Security-Advisories sollte eine automatische Berechnung des CVSS-Score auf Basis der im Text enthaltenen Schlüsselwörter implementiert werden, um dem oftmaligen Fehlen der Angaben zur Kritikalität entgegenzuwirken.
Ebenso sollte erhoben werden wieviel Potential der Einsatz selbstlernender Filter (wie sie z.B. in SPAM-Filtern verwendet werden) zur Auswertung nicht maschinenlesbarer Security-Advisories in sich birgt.
Desweiteren könnten die Advisory-Daten mit Informationen vervollständigt werden, welche über die im Security-Advisory angegebenen Referenzen ermittelbar sind.

Bei der Korrelation sollte die *Distributions*version miteinbezogen werden, um Security-Advisory-Feeds abzudecken, die prinzipiell keine Paketversionen sondern ausschließlich Distributionsversionen transportieren. Advisory-Objekte würden sich also nicht nur auf Paket-Objekte sondern auch direkt auf die Host-Objekte beziehen.
Jedoch ist zu bedenken, dass (zumindest auf dpkg-basierten Systemen) die Möglichkeit besteht andere Paketversionen als die der Distributionsversion zugeordneten zu installieren. Dies kann die Ermittlung von Bedrohungsgraden ad absurdum führen.

Auch könnte die Liste betroffener Hersteller als zusätzliches Plausibilitätskriterium Eingang in die Korrelation finden, um für eine höhere Zuverlässigkeit der getroffenen Aussagen zu sorgen.

Aus rein technischer Sicht ist zu nennen, dass für die Leistungsfähigkeit der Implementierung der Schnitt-

stelle zu GnuPG keine belastbaren Daten (im Sinne eines *Benchmarks*) vorhanden sind. Sollte es an dieser Stelle zu Performance-Engpässen kommen, könnte die Verschlüsselung auch auf Basis einer anderen Kryptographie-Suite wie beispielsweise OpenSSL oder GnuTLS implementiert werden.

Es ist denkbar andere Varianten der Benutzer-Authentifizierung wie z.B. SSL-Zertifikate, OpenID oder Kerberos durch entsprechende Schnittstellen in der Benutzerverwaltung zu unterstützen.

Die zwingende Verifikation des Fingerprints des WizzanAgents sollte per Konfigurationseinstellung aktiviert werden können. Gleichzeitig müsste Konfigurationsverwaltungssoftware die Fingerprints von den WizzanAgents einsammeln und sie dem WizzanServer zur Verfügung stellen. (Z.B.: Datenbank-Einträge oder Datei mit einem Fingerprint pro Zeile)

Klarerweise kann Konfigurationsmanagementsoftware verwendet werden, um die Komponenten von Wizzan und insbesondere die WizzanClients auf den Hosts zu installieren und zu verwalten. Für die Zukunft kann jedoch auch darüber nachgedacht werden, ob Software-Aktualisierungen über die Benutzeroberfläche von Wizzan angestoßen oder autorisiert werden sollen.

Eine für den Einsatz in Unternehmen wichtige Hilfsfunktion – das Audit-Log – sollte sämtliche Meldungen des WizzanServers sowie jene der WizzanClients enthalten. Dazu ist eine Erweiterung des Nachrichtenformats um Log-Meldungen erforderlich. Alternativ könnten auch alle Komponenten den Logging-Dienst des jeweiligen Host-Systems benutzen. Jedoch ist dieser eine benutzerfreundliche Aufbereitung der Wizzan betreffenden Log-Einträge innerhalb der Benutzeroberfläche des WizzanServers wahrscheinlich vorzuziehen.
In diesem Zusammenhang sei darauf verwiesen, dass die WizzanCrawler die mögliche Änderung der Lizenz bzw. der Nutzungsbedingungen eines Security-Advisory-Feeds anzeigen. Diese Log-Einträge sollten besonders hervorgehoben sein, um nicht übersehen zu werden.

Sobald das Audit-Log einsatzfähig ist, kann Benutzern/-innen ermöglicht werden Objekte des Inventars zu bearbeiten bzw. neu anzulegen.

Fazit

Im Ergebnis zeigt sich, dass die automatische Erstellung eines Software-Inventars aus den Daten von Paket-Managern relativ einfach zu implementieren ist. Indem die Plugins des WizzanAgents die verschiedenen Paket-Manager abdecken, kann Wizzan praktisch an jedes (Betriebs-)System angepasst werden, aus dem an zentraler Stelle Paketversionen ausgelesen werden können. Die Ausgaben sind aufgrund ihrer gleichbleibenden Strukturierung maschinenlesbar. Dies führt zu einer sehr hohen Präzision der Angaben im Software-Inventar, sofern diese Host- und Paket-Objekte betrifft. In diesem Bereich ist die Qualität der erhobenen Daten nur von der Ausgabe des jeweiligen Paketmanagers abhängig.

Die im Inventar verzeichneten Paketnamen bilden die Basis der gezielten Auswahl relevanter Einträge aus den Security-Advisory-Feeds verschiedener Hersteller, Distributoren und Sicherheitsdienstleister. Die unterschiedliche Strukturierung der Inhalte erschwert hier die Extraktion aller notwendigen Informationen, jedoch kann dies durch entsprechenden Mehraufwand bei der Entwicklung ausgeglichen werden.
Das tatsächlich begrenzende Element stellt aber vielfach nicht die Machbarkeit an sich dar. Viel mehr sind es die rechtlichen Rahmenbedingungen, die eine weitgehend von rechtlichen Risiken freie Nutzung nur bedingt zulassen.

Aus der Korrelation von Advisory-Daten und Paket-Objekten entstehen über die Relationen zwischen den Objekten die Bedrohungsgrade von Software-Paketen und Hosts. Falsch-positive Beziehungen können von den Benutzern/-innen als solche gekennzeichnet werden.
Die ermittelten Bedrohungsgrade werden schließlich entlang einer Farbskala dargestellt, die eine intuitive Erfassung der Bedrohungslage der von Wizzan überwachten IT-Infrastruktur ermöglicht.

Die vorliegende Diplomarbeit kommt zu dem Schluss, dass mit Hilfe der Daten aus Paket-Managern und jenen aus Security-Advisory-Feeds (entsprechende Nutzungsbedingungen vorausgesetzt) die aktuelle Bedrohungslage automatisiert ermittelt werden kann. Um die Zuverlässigkeit der durch die Korrelation getroffenen Aussagen auf ein höheres Niveau zu heben, ist jedoch weitere Entwicklungsarbeit nötig.

Literaturverzeichnis

[1] A. Meyer, “Configuration Management in the Security World,” August 2007. [Online]. URL: http://www.sans.edu/research/security-laboratory/article/meyer-config-manage [ausgehoben am 28.07.2012]

[2] Secunia ApS, “Secunia Corporate Software Inspector,” 2012. [Online]. URL: http://secunia.com/?action=fetch&filename=Secunia_CSI_Technical_Productsheet.pdf [ausgehoben am 28.07.2012]

[3] Microsoft, “Microsoft Security Intelligence Report Volume 11,” 2011. [Online]. URL: http://download.microsoft.com/download/0/3/3/0331766E-3FC4-44E5-B1CA-2BDEB58211B8/Microsoft_Security_Intelligence_Report_volume_11_English.pdf [ausgehoben am 07.07.2012]

[4] M. Scheck, Cisco Systems Inc., “Example of CVSS based Patching Policy,” 2012. [Online]. URL: http://www.first.org/_assets/cvss/cvss-based-patch-policy.pdf [ausgehoben am 07.07.2012]

[5] I. Murdock, “How package management changed everything,” Juli 2007. [Online]. URL: http://ianmurdock.com/solaris/how-package-management-changed-everything/ [ausgehoben am 06.07.2012]

[6] The Debian Project, “The Debian GNU/Linux FAQ: Chapter 7 - Basics of the Debian package management system,” August 2011. [Online]. URL: http://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html [ausgehoben am 06.07.2012]

[7] F. Ronneburg, “Debian-Anwenderhandbuch: 4.37 dpkg,” 2012. [Online]. URL: http://debiananwenderhandbuch.de/dpkg.html [ausgehoben am 06.07.2012]

[8] The Debian Project, “APT HOWTO (Obsolete Documentation): Kapitel 2 - Einführung,” April 2003. [Online]. URL: http://www.debian.org/doc/manuals/apt-howto/ch2.de.html [ausgehoben am 06.07.2012]

[9] F. Ronneburg, “Debian-Anwenderhandbuch: 4.15 apt-get,” 2012. [Online]. URL: http://debiananwenderhandbuch.de/apt-get.html [ausgehoben am 06.07.2012]

[10] The FreeBSD German Documentation Project, “Das FreeBSD-Handbuch: 5.5. Benutzen der Ports-Sammlung,” 2012. [Online]. URL: http://www.freebsd.org/doc/de/books/handbook/ports-using.html [ausgehoben am 07.07.2012]

[11] S. Vermeulen u.a., “Gentoo Linux x86 Handbook: 3. Portage Features,” Juni 2012. [Online]. URL: http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=3 [ausgehoben am 07.07.2012]

[12] Arch Linux Wiki Community, “Arch Build System,” 2012. [Online]. URL: https://wiki.archlinux.de/title/Arch_Build_System [ausgehoben am 07.07.2012]

[13] heise online, “Neues Rating-System für Sicherheitsfehler vorgestellt,” September 2005. [Online]. URL: http://www.heise.de/newsticker/meldung/Neues-Rating-System-fuer-Sicherheitsfehler-vorgestellt-130997.html [ausgehoben am 07.07.2012]

[14] C. Kühnast, “Aus dem Alltag eines Sysadmin: Fail2ban,” Oktober 2007. [Online]. URL: http://www.linux-magazin.de/Heft-Abo/Ausgaben/2007/10/Unbestechlicher-Tuersteher [ausgehoben am 07.07.2012]

[15] K. Patzwaldt, “Suchmaschinen und die Datei robots.txt,” Juni 2008. [Online]. URL: http://www.at-web.de/grundlagen/robots-txt.htm [ausgehoben am 07.07.2012]

[16] Secunia ApS, “Terms & Conditions,” 2012. [Online]. URL: https://secunia.com/company/terms_conditions/ [ausgehoben am 07.07.2012]

[17] VUPEN Security, “VUPEN.COM Disclaimer and Copyright,” 2012. [Online]. URL: http://www.vupen.com/english/copyright.php [ausgehoben am 07.07.2012]

[18] Software in the Public Interest, Inc., “Lizenz der Debian-Webseiten,” Februar 2012. [Online]. URL: http://www.debian.org/license [ausgehoben am 07.07.2012]

[19] ——, “DSA-2507-1 openjdk-6 – several vulnerabilities,” Juli 2012. [Online]. URL: http://www.debian.org/security/2012/dsa-2507 [ausgehoben am 07.07.2012]

[20] Canonical Ltd., “Ubuntu Security Notice USN-1498-1,” Juli 2012. [Online]. URL: http://www.ubuntu.com/usn/usn-1498-1/ [ausgehoben am 07.07.2012]

[21] S. Sobola, “Wer haftet, wenn es schief geht?” Oktober 2004. [Online]. URL: http://www.manager-magazin.de/unternehmen/it/0,2828,322293-2,00.html [ausgehoben am 07.07.2012]

[22] National Institute for Standards and Technology, “Vulnerability Summary for CVE-2012-1942,” Juni 2012. [Online]. URL: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-1942 [ausgehoben am 07.07.2012]

[23] ——, “NVD Data Feed and Product Integration,” 2012. [Online]. URL: http://nvd.nist.gov/download.cfm [ausgehoben am 07.07.2012]

[24] US-CERT, “US-CERT Website Policies and Terms of Use,” 2012. [Online]. URL: https://www.us-cert.gov/privacy/ [ausgehoben am 07.07.2012]

[25] Software in the Public Interest, Inc., “Sicherheits-Informationen,” 2012. [Online]. URL: http://www.debian.org/security/ [ausgehoben am 07.07.2012]

[26] ——, “Sicherheitsankündigungen 2012,” 2012. [Online]. URL: http://www.debian.org/security/2012/ [ausgehoben am 07.07.2012]

[27] ——, “DSA-2502-1 python-crypto – programming error,” Juni 2012. [Online]. URL: http://www.debian.org/security/2012/dsa-2502 [ausgehoben am 07.07.2012]

[28] Canonical Ltd., “ubuntu.com: Legal,” 2012. [Online]. URL: http://www.ubuntu.com/legal [ausgehoben am 08.07.2012]

[29] ——, “Ubuntu security notices,” 2012. [Online]. URL: http://www.ubuntu.com/usn [ausgehoben am 08.07.2012]

[30] ——, “The ubuntu-security-announce Archives,” 2012. [Online]. URL: https://lists.ubuntu.com/archives/ubuntu-security-announce/ [ausgehoben am 08.07.2012]

[31] ——, “USN-1476-1: Linux kernel (OMAP4) vulnerabilities,” 2012. [Online]. URL: http://www.ubuntu.com/usn/usn-1476-1/ [ausgehoben am 08.07.2012]

[32] ——, “CVE-2011-4131 in Ubuntu,” Juni 2012. [Online]. URL: http://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-4131.html [ausgehoben am 08.07.2012]

[33] National Institute for Standards and Technology, “nvdcve-2.0-modified.xml,” Juli 2012. [Online]. URL: http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-modified.xml [ausgehoben am 08.07.2012]

[34] US-CERT, “US-CERT Website Policies and Terms of Use,” 2012. [Online]. URL: https://www.us-cert.gov/legal.html [ausgehoben am 09.07.2012]

[35] ——, “Vulnerability Notes Database,” 2012. [Online]. URL: http://www.kb.cert.org/vuls [ausgehoben am 09.07.2012]

[36] ——, “Vulnerability Notes Database View Help,” 2012. [Online]. URL: http://www.kb.cert.org/vuls/html/viewhelp [ausgehoben am 09.07.2012]

[37] ——, “Vulnerability-Feed des US-CERT,” Juli 2012. [Online]. URL: http://www.kb.cert.org/vulfeed [ausgehoben am 09.07.2012]

[38] NIST, “National Vulnerability Database Version 2.2,” Juli 2012. [Online]. URL: http://nvd.nist.gov/ [ausgehoben am 09.07.2012]

[39] US-CERT, “ISC BIND 9 zero length rdata named vulnerability,” Juni 2012. [Online]. URL: http://www.kb.cert.org/vuls/id/381699 [ausgehoben am 09.07.2012]

[40] OSVDB, “FAQ,” 2012. [Online]. URL: http://osvdb.org/faq [ausgehoben am 09.07.2012]

[41] ——, “Startseite der OSVDB,” 2012. [Online]. URL: http://osvdb.org/ [ausgehoben am 09.07.2012]

[42] ——, “About the OSVDB API,” 2012. [Online]. URL: http://osvdb.org/api/about [ausgehoben am 09.07.2012]

[43] ——, “Open Source Vulnerability Database (OSVDB) License,” 2012. [Online]. URL: http://www.osvdb.org/osvdb_license [ausgehoben am 09.07.2012]

[44] ——, “OSVDB: About,” 2012. [Online]. URL: http://osvdb.org/about [ausgehoben am 09.07.2012]

[45] B. Martin, “We’re Still Here - Update on OSVDB Project: Data and Exports,” März 2012. [Online]. URL: http://blog.osvdb.org/2012/03/30/were-still-here-update-on-osvdb-project-data-and-exports [ausgehoben am 09.07.2012]

[46] OSVDB, “Terms and Conditions of Use,” 2012. [Online]. URL: http://osvdb.org/tos [ausgehoben am 09.07.2012]

[47] ——, “70734 : Apache CouchDB Request / Cookie Handling Unspecified XSS,” 2012. [Online]. URL: http://osvdb.org/show/osvdb/70734 [ausgehoben am 09.07.2012]

[48] Red Hat, Inc., “Red Hat Terms of Use,” 2012. [Online]. URL: https://access.redhat.com/site/help/terms_conditions.html [ausgehoben am 29.07.2012]

[49] Electronic Frontier Foundation, “Fair Use Frequently Asked Questions,” 2012. [Online]. URL: http://www.teachingcopyright.org/handout/fair-use-faq [ausgehoben am 16.07.2012]

[50] Red Hat, Inc., “CVE-2012-0022,” 2012. [Online]. URL: https://access.redhat.com/security/cve/CVE-2012-0022 [ausgehoben am 29.07.2012]

[51] National Institute for Standards and Technology, "Vulnerability Summary for CVE-2012-0022," 2012. [Online]. URL: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-0022 [ausgehoben am 29.07.2012]

[52] The CentOS Project, "The Community ENTerprise Operating System," 2012. [Online]. URL: https://www.centos.org/ [ausgehoben am 29.07.2012]

[53] ——, "The CentOS-announce Archives," 2012. [Online]. URL: http://lists.centos.org/pipermail/centos-announce/ [ausgehoben am 29.07.2012]

[54] Johnny Hughes, "[CentOS-announce] CESA-2012:1061 Moderate CentOS 5 kernel Update," Juli 2012. [Online]. URL: http://lists.centos.org/pipermail/centos-announce/2012-July/018707.html [ausgehoben am 29.07.2012]

[55] T. Birkmann, *Präteritopräsentia: morphologische Entwicklungen einer Sonderklasse in den altgermanischen Sprachen*. Tübingen: Max Niemeyer Verlag, 1987.

[56] M. F. Sanner, "Python: A programming language for software integration and development," The Scripps Research Institute, 10550 North Torrey Pines Road, La Jolla, CA-92037, Tech. Rep., November 2007.

[57] Kohana Team, "What is Kohana?" 2012. [Online]. URL: http://kohanaframework.org/3.2/guide/kohana [ausgehoben am 11.07.2012]

[58] Laurie Williams, "Risk-based Security Testing: Prioritizing Security Testing with Threat Modeling," Oktober 2007. [Online]. URL: http://resist.isti.cnr.it/free_slides/security/williams/RiskBasedSecurityTesting.pdf [ausgehoben am 10.07.2012]

[59] W. Koch, *Using the GNU Privacy Guard*, Free Software Foundation, 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA, März 2012. [Online]. URL: http://www.gnupg.org/documentation/manuals/gnupg.pdf [ausgehoben am 11.07.2012]

[60] M. Ashley, *The GNU Privacy Handbook*, Free Software Foundation, 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA, 1999. [Online]. URL: http://www.gnupg.org/gph/en/manual.pdf [ausgehoben am 11.07.2012]

[61] N. Nurseitov u.a., "Comparison of JSON and XML Data Interchange Formats: A Case Study," Department of Computer Science Montana State University – Bozeman, Bozeman, Montana, 59715, USA, Tech. Rep., Juni 2009.

[62] Linux Packages Search, “Search Results for gnupginterface,” Juli 2012. [Online]. URL: http://pkgs.org/search/?keyword=gnupginterface [ausgehoben am 12.07.2012]

[63] Frank J. Tobin, “GnuPGInterface.py,” 2001. [Online]. URL: http://py-gnupg.cvs.sourceforge.net/viewvc/py-gnupg/py-gnupg/GnuPGInterface.py?view=markup [ausgehoben am 12.07.2012]

[64] Python Software Foundation, “Python v2.7.3 documentation: 15.15. platform — Access to underlying platform’s identifying data,” Juli 2012. [Online]. URL: http://docs.python.org/library/platform.html [ausgehoben am 12.07.2012]

[65] ——, “Python v2.7.3 documentation: 17.2. socket — Low-level networking interface,” Juli 2012. [Online]. URL: http://docs.python.org/library/socket.html [ausgehoben am 12.07.2012]

[66] Marius Ducea, “Ubuntu package version naming explanation,” Juni 2006. [Online]. URL: http://www.ducea.com/2006/06/17/ubuntu-package-version-naming-explanation/ [ausgehoben am 13.07.2012]

[67] Kohana Team, “Kohana License Agreement,” 2012. [Online]. URL: http://kohanaframework.org/license [ausgehoben am 13.07.2012]

[68] Free Software Foundation, Inc., “Various Licenses and Comments about Them,” 2012. [Online]. URL: https://www.gnu.org/licenses/license-list.en.html [ausgehoben am 13.07.2012]

[69] Kohana Team, “Kohana 3.2 Documentation: Requests,” 2012. [Online]. URL: http://kohanaframework.org/3.2/guide/kohana/requests [ausgehoben am 13.07.2012]

[70] ——, “Kohana 3.2 Documentation: Object Relational Mapping,” 2012. [Online]. URL: http://kohanaframework.org/3.2/guide/orm/ [ausgehoben am 13.07.2012]

[71] ——, “Kohana 3.2 Documentation: ORM: Filters,” 2012. [Online]. URL: http://kohanaframework.org/3.2/guide/orm/filters [ausgehoben am 13.07.2012]

[72] ——, “Kohana 3.2 Documentation: ORM: Validation,” 2012. [Online]. URL: http://kohanaframework.org/3.2/guide/orm/validation [ausgehoben am 13.07.2012]

[73] Kohana Community, “Kohana Community Support: Kohana Security,” Dezember 2010. [Online]. URL: http://forum.kohanaframework.org/discussion/7675/kohana-security/p1 [ausgehoben am 13.07.2012]

[74] MySQL Bugtracker, “Triggers not executed following foreign key updates/deletes,” Juni 2005. [Online]. URL: http://bugs.mysql.com/bug.php?id=11472 [ausgehoben am 14.07.2012]

[75] ——, “MySQL/InnoDB does not execute trigger ON DELETE for cascade deleting rows,” September 2005. [Online]. URL: http://bugs.mysql.com/bug.php?id=13102 [ausgehoben am 14.07.2012]

[76] ——, “InnoDB NOT ACID COMPLIANT ???” August 2011. [Online]. URL: http://bugs.mysql.com/bug.php?id=62209 [ausgehoben am 14.07.2012]

[77] S. Ruby u.a., “Rss20AndAtom10Compared,” September 2008. [Online]. URL: http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared [ausgehoben am 15.07.2012]

[78] Canonical Ltd., “Informationen über Paket flashplugin-installer in precise,” 2012. [Online]. URL: http://packages.ubuntu.com/precise/flashplugin-installer [ausgehoben am 25.07.2012]

[79] Software in the Public Interest, Inc., “Informationen über Paket flashplugin-nonfree in squeeze,” 2012. [Online]. URL: http://packages.debian.org/squeeze/flashplugin-nonfree [ausgehoben am 25.07.2012]

[80] National Institute for Standards and Technology, “Vulnerability Summary for CVE-2012-0779,” Juli 2012. [Online]. URL: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-0779 [ausgehoben am 25.07.2012]

[81] B. Appleton u. a., “Streamed Lines: Branching Patterns for Parallel Software Development,” 1998. [Online]. URL: http://www.hillside.net/plop/plop98/final_submissions/P37.pdf [ausgehoben am 25.07.2012]

[82] Software in the Public Interest, Inc., “DSA-2492-1 php5 – buffer overflow,” Juli 2012. [Online]. URL: http://www.debian.org/security/2012/dsa-2492 [ausgehoben am 26.07.2012]

[83] The PHP Group, “PHP: Downloads,” 2012. [Online]. URL: http://www.php.net/downloads.php [ausgehoben am 25.07.2012]

[84] Software in the Public Interest, Inc., “DSA-1266-1 gnupg – several vulnerabilities,” März 2007. [Online]. URL: http://www.debian.org/security/2007/dsa-1266.en.html [ausgehoben am 26.07.2012]

[85] Jeff Atwood, “What’s In a Version Number, Anyway?” Februar 2007. [Online]. URL: http://www.codinghorror.com/blog/2007/02/whats-in-a-version-number-anyway.html [ausgehoben am 26.07.2012]

[86] The PHP Group, “PHP: version_compare – Manual,” Juli 2012. [Online]. URL: http://php.net/manual/de/function.version-compare.php [ausgehoben am 26.07.2012]

Abbildungsverzeichnis

2.1 Detections of exploits targeting CVE-2011-0611, April-July, 2011 4
2.2 Detections of exploits targeting CVE-2011-2110, June-August, 2011 4
2.3 CVSS Vulnerability Assessment Results 5

4.1 Gesamtüberblick 20
4.2 Threat model 23

7.1 Anwendungsfall-Diagramm des WizzanAgents 34
7.2 Kommunikationsdiagramm des WizzanAgents 35

8.1 Schema der Wizzan-Datenbank 46

10.1 Anwendungsfall-Diagramm des WizzanCrawlers 52
10.2 Kommunikationsdiagramm des WizzanCrawlers 53