INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Netzbasierte Erkennung von Systemen und Diensten zur Verbesserung der IT-Sicherheit

Andreas Bernhard

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Netzbasierte Erkennung von Systemen und Diensten zur Verbesserung der IT-Sicherheit

Andreas Bernhard

Aufgabensteller:	PD Dr. Wolfgang Hommel
Betreuer:	Felix von Eye Stefan Metzger
Abgabetermin:	18. März 2014

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

München, den 18. März 2014

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Die Kenntnis über die eigene Computer-Infrastruktur zählt zu einem der wichtigsten Sicherheitsziele für Administratoren in mittleren bis großen Computernetzen. Ein sehr wichtiges Asset sind dabei die Systeme und Dienste. Viele Sicherheitsmaßnahmen hängen direkt oder indirekt davon ab, welche Hostsysteme mit unterschiedlichen Diensten im Netz aktiv sind. Eine aktive Suche nach Zielsystemen und darauf installierten Diensten kommt allerdings oft nicht in Frage, da diese je nach Ausprägung zu Beeinträchtigungen im Betrieb führen oder unbeabsichtigte Sicherheitsereignisse auslösen können.

In dieser Arbeit wird eine Kategorisierung erarbeitet, bei der die verschiedenen Eigenschaften eines Systems analysiert und bewertet werden. Auf Basis dieser Ergebnisse wird ein Prototyp entwickelt, der mithilfe von passiven Verfahren eine Erkennung von Zielsystemen und darauf laufenden Diensten ermöglicht. Dieser Prototyp wird dabei in eine Weboberfläche integriert, auf der Analysen gestartet und ausgewertet werden können.

Es wurde festgestellt, dass es allein durch den Einsatz von passiven Erkennungsmethoden wie der Netflow-Technik möglich ist, im Netz kommunizierende Hostsysteme und deren jeweiligen Dienste zuverlässig zu identifizieren. Zusätzlich ist es bei einigen Hosts möglich, das installierte Betriebssystem zu erkennen.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 1
- 1.2. Aufgabenstellung und Ziele . . . 2
- 1.3. Struktur der Arbeit . . . 2

2. Vorgehen **5**
- 2.1. Kategorisierung . . . 5
  - 2.1.1. Physische Eigenschaften . . . 5
  - 2.1.2. Nutzergruppen . . . 6
  - 2.1.3. Betriebssystem . . . 6
  - 2.1.4. Dienste . . . 7
  - 2.1.5. Schwachstellen . . . 7
  - 2.1.6. Fazit . . . 8
- 2.2. Aktive Erkennungsmethoden . . . 9
  - 2.2.1. Portscanner . . . 9
  - 2.2.2. Vulnerability Scanner . . . 11
- 2.3. Passive Erkennungsmethoden . . . 11
  - 2.3.1. Netflow . . . 12
  - 2.3.2. PRADS . . . 17
  - 2.3.3. Deep Packet Inspection . . . 18

3. Anforderungsanalyse **21**
- 3.1. Anwendungsszenario . . . 21
- 3.2. Funktionale Anforderungen . . . 22
  - 3.2.1. Weboberfläche . . . 22
  - 3.2.2. Datenbank . . . 22
  - 3.2.3. Aufgabenplaner . . . 23
  - 3.2.4. Wahl des Scan-Bereichs . . . 23
  - 3.2.5. Passive Erkennung von Systemen und Diensten . . . 23
  - 3.2.6. Erfassung von Daten . . . 24
  - 3.2.7. Aktive Erkennung zur Kontrolle . . . 24
  - 3.2.8. Regelmäßiges Scanning . . . 24
  - 3.2.9. Mandantenfähigkeit . . . 25
  - 3.2.10. Benachrichtigung per E-Mail . . . 25
  - 3.2.11. IPv6-Unterstützung . . . 25
  - 3.2.12. Kompatibilitätsprüfung der Internetprotokolltypen . . . 25
- 3.3. Nichtfunktionale Anforderungen . . . 26
  - 3.3.1. Zuverlässigkeit . . . 26
  - 3.3.2. Aufbau, Handhabung und Benutzbarkeit . . . 26

3.3.3. Leistung und Effizienz . . . 26
3.3.4. Betrieb und Umgebungsbedingungen . . . 26
3.3.5. Portierbarkeit und Übertragbarkeit . . . 27
3.3.6. Sicherheitsanforderungen . . . 27
3.3.7. Korrektheit . . . 27
3.3.8. Skalierbarkeit . . . 28
3.4. Liste der Anforderungen . . . 28
3.5. Themenverwandte Arbeiten . . . 28

4. Entwurf **31**
4.1. Funktionale Anforderungen . . . 31
4.1.1. Weboberfläche . . . 31
4.1.2. Datenbank . . . 35
4.1.3. Aufgabenplaner . . . 36
4.1.4. Wahl des Scan-Bereichs . . . 39
4.1.5. Passive Erkennung von Systemen und Diensten . . . 39
4.1.6. Erfassung von Daten . . . 45
4.1.7. Aktive Erkennung zur Kontrolle . . . 46
4.1.8. Regelmäßiges Scanning . . . 48
4.1.9. Mandantenfähigkeit . . . 48
4.1.10. Benachrichtigung per E-Mail . . . 50
4.1.11. IPv6-Unterstützung . . . 50
4.1.12. Kompatibilitätsprüfung der Internetprotokolltypen . . . 51
4.2. Nichtfunktionale Anforderungen . . . 51
4.2.1. Zuverlässigkeit . . . 51
4.2.2. Aufbau und Handhabung . . . 52
4.2.3. Leistung und Effizienz . . . 52
4.2.4. Betrieb und Umgebung . . . 52
4.2.5. Portierbarkeit . . . 53
4.2.6. Sicherheitsanforderungen . . . 53
4.2.7. Korrektheit . . . 54
4.2.8. Skalierbarkeit . . . 54

5. Implementierung **55**
5.1. Funktionale Anforderungen . . . 55
5.1.1. Weboberfläche . . . 55
5.1.2. Datenbank . . . 59
5.1.3. Aufgabenplaner . . . 60
5.1.4. Wahl des Scan-Bereichs . . . 67
5.1.5. Passive Erkennung von Systemen und Diensten . . . 68
5.1.6. Erfassung von Daten . . . 72
5.1.7. Regelmäßiges Scanning . . . 75
5.1.8. Mandantenfähigkeit . . . 76
5.1.9. IPv6-Unterstützung . . . 78
5.2. Nichtfunktionale Anforderungen . . . 81
5.2.1. Zuverlässigkeit . . . 81
5.2.2. Aufbau und Handhabung . . . 81

5.2.3. Betrieb und Umgebung . . . 81
5.2.4. Portierbarkeit . . . 82
5.2.5. Sicherheitsanforderungen . . . 82
5.2.6. Korrektheit . . . 83

6. Test und Evaluation **85**
6.1. Beschreibung der Testumgebung . . . 85
6.2. Auswertung . . . 86
6.2.1. IPv6-Hosts . . . 89

7. Zusammenfassung und Ausblick **91**
7.1. Diskussion . . . 91
7.2. Ausblick . . . 92

Anhang **93**
A. Einrichtung der LAMP-Umgebung . . . 93
A.1. Installation . . . 93
A.2. Konfiguration . . . 94
B. Einrichtung der Erkennungsmethoden . . . 95
B.1. Netflow . . . 95
C. Anlegen der Datenbankstruktur . . . 96
D. Quelltexte . . . 98
D.1. Standard-Konfiguration der Anwendung . . . 98
D.2. Initialisierung der Erkennungsmethode(n) . . . 100
D.3. Netflow-Analyse (IPv4 / IPv6 kompatibel) . . . 102
D.4. HostMapper . . . 109

Abkürzungsverzeichnis **111**

Abbildungsverzeichnis **115**

Quellcodeverzeichnis **117**

Literaturverzeichnis **119**

1. Einleitung

Das erste Kapitel gibt den Hintergrund der Arbeit, eine Zusammenfassung der Aufgabenstellung und der Ziele der Arbeit wieder. Außerdem wird die Struktur der Arbeit vorgestellt.

1.1. Motivation

Das MWN (Münchner Wissenschaftsnetz) verbindet die Universitäten, Hochschulen und weitere Gebäude, die in diesem Zusammenhang genutzt werden, in und um München. Als Netzbetreiber ist das Leibniz-Rechenzentrum (LRZ) für die Sicherheit und Integrität des Netzes sowie für den Schutz deren Nutzer verantwortlich. Deshalb wurden an den Schnittstellen zwischen internem und externem Netz (siehe Abbildung 1.1, beispielsweise am Übergang des MWN zum X-WiN[1]) Sensoren eingerichtet, die den Netzverkehr überwachen. Dadurch ist es möglich, Missbrauchs- und Schadensfälle frühzeitig zu erkennen.

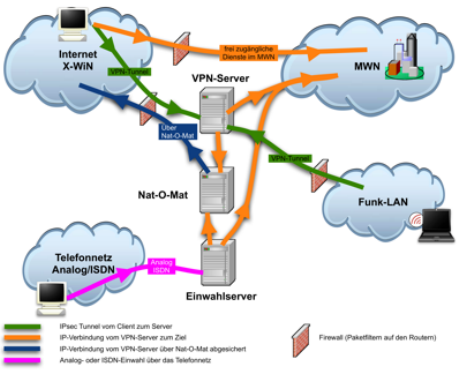


Abbildung 1.1.: Anbindung des MWN an das X-WiN, Quelle: [Lei13]

Aufgrund der großen Zahl der Nutzer (im Juli 2013 waren es nach [Lei13] 103.000 Geräte) ist das Datenaufkommen an den Übergangsstellen hoch und für die Auswahl und den Betrieb der Sensoren nicht leicht zu bewerkstelligen, da die klassischen Analyse- und Auswertungsverfahren nicht einfach beliebig auf eine bestimmte Größe skalierbar sind und somit einen unverhältnismäßigen Aufwand verursachen.

Dabei ist das LRZ lediglich bis zum Netzanschluss des Benutzers verantwortlich (Datendose, WLAN (Wireless Local Area Network)-Router) und nicht für die dahinter liegenden Systeme und Dienste. Diese werden von den jeweiligen Hochschulen und Institutionen sowie deren Mitarbeiter und Studenten verwaltet. Als Systeme kommen sowohl Arbeitsplatzrechner, mobile Geräte aber auch Server und Peripherie-Geräte wie beispielsweise Drucker in Frage.

Für das LRZ wäre es essentiell zu wissen, welche Systeme außerhalb ihres Zugriffsbereichs betrieben werden — auch wenn kein physischer oder logischer Zugriff darauf möglich ist.

[1] Nationales Backbone des Deutschen Forschungsnetzes, welches mehr als 500 deutsche Hochschulen und Forschungseinrichtungen miteinander verbindet.

Dabei geht es darum, die für die Zielsysteme relevanten Angriffe möglichst genau ermitteln zu können und die dort betriebenen Systeme so besser schützen zu können. Auch können die vorhandenen, stets begrenzten Ressourcen, auf diese Weise effizienter eingesetzt werden.

1.2. Aufgabenstellung und Ziele

In dieser Arbeit soll in einem ersten Schritt eine Kategorisierung erstellt werden. Eine Kategorie kann gekennzeichnet sein durch ein bestimmtes Betriebssystem (z.B. Windows-System oder Android), der darauf laufenden Dienste (z.B. Mail-Daemon) oder durch den Zweck des Systems (z.B. Webserver). Ein besonderes Augenmerk soll auf eine zusätzliche Schwachstellenkategorie gelegt werden. Dabei kann ein System einer bestimmten Kategorie zugewiesen werden, wenn eine Schwachstelle, z.B. eine bestimmte Betriebssystemversion oder ein offener Port, der darauf schließen lässt, gegeben ist.

Grundlegend für eine sinnvolle Kategorisierung ist, wie genau Systeme in einem nächsten Schritt in die erstellten Kategorien eingeordnet werden können. Als Hilfestellung für die Erkennung der verschiedenen Systeme und deren Zweck können sowohl aktive wie auch passive Erkennungsmethoden dienen. Zu den aktiven zählen das Portscanning und das Vulnerability-Scanning. Vor allem der Einsatz letztgenannter Methode kann aber Auswirkungen auf die Verfügbarkeit des gescannten Systems haben. Mit den passiven Erkennungsmethoden wie der Netflow-Analyse und der Deep-Packet-Inspection gäbe es allerdings eine Alternative, Informationen über ein System zu erhalten, ohne direkt oder indirekt darauf zuzugreifen.

Die verschiedenen Methoden werden im Rahmen dieser Arbeit zunächst vorgestellt und es wird in einer ausführlichen Analyse auf deren Vor- und Nachteile eingegangen. Auch werden die dabei aufkommenden Herausforderungen dargestellt, einschließlich verschiedener Lösungsansätze.

Anhand einer konkreten Implementierung sollen die zuvor gewonnenen Kenntnisse auf ein Beispiel kleinerer Größenordnung angewandt werden. Eine vollständige Analyse und Zuweisung aller denkbaren Kategorien wäre im Rahmen dieser Arbeit nicht möglich. Deshalb soll eine begrenzte Auswahl der zur Verfügung stehenden Verfahren zum Einsatz kommen, durch die die Systeme eines exemplarischen Testnetzes kategorisiert werden können.

Die verschiedenen, teilweise sehr zeitaufwändigen Verfahren sollen über eine Weboberfläche angesteuert werden können. Der Vorteil der Weboberfläche besteht darin, dass mehrere Erkennungsmethoden parallel eingetragen werden können und der Benutzer stets den aktuellen Status der laufenden bzw. noch nicht gestarteten Aufgaben kennt. Die Aufgaben werden dabei zeitgesteuert im Hintergrund ausgeführt. Beispielsweise kann so ein Portscan auf mehrere Ziele gleichzeitig initiiert werden, welche dann nach und nach durchgeführt werden. Nach Vorliegen der Ergebnisse als Rohdaten sollen diese mittels Filterregeln analysiert und gespeichert werden. Anschließend können die ausgewerteten Daten und die daraus erhaltene Kategorisierung über eine Weboberfläche grafisch angezeigt und als Download abgerufen werden.

1.3. Struktur der Arbeit

In Kapitel 2 wird das Vorgehen beschrieben, wie eine sinnvolle Kategorisierung gefunden werden kann. Anschließend wird untersucht, welche Daten man von den verschiedenen Methoden (z.B. Portscanning) erhält und welche Schlüsse aus den Ergebnissen der verschiedenen

Methoden gezogen werden können. In Kapitel 3 werden die Anforderungen an den Prototyp definiert, welcher in Kapitel 4 entworfen wird. In Kapitel 5 wird die Implementierung vorgestellt, welche im nachfolgenden Kapitel 6 in einer Testumgebung installiert und anschließend evaluiert wird. Im Schlusskapitel 7 wird eine Zusammenfassung der Ergebnisse gegeben und wie diese zu bewerten sind.

2. Vorgehen

In diesem Kapitel wird zunächst eine sinnvolle Kategorisierung erarbeitet und es wird anschließend auf die verschiedenen Methoden eingegangen, anhand deren man auf eine Kategorie folgern kann.

Für potentielle Angreifer ist es von Interesse herauszufinden, welches Betriebssystem und welche Dienste auf dem Zielsystem installiert sind. So können noch weitgehend unbekannte, aber auch bereits veröffentlichte Sicherheitslücken ausgenutzt werden. Selbst wenn die Sicherheitslücke vom Herausgeber des Betriebssystems oder des Dienstes unlängst behoben und ein Patch bereitgestellt wurde, bedeutet dies noch lange nicht, dass dieser auch auf allen über das Internet erreichbaren Systeme eingespielt wird. Die Sicherheitslücke kann also - obwohl bekannt und offiziell behoben - nach wie vor ein großes Bedrohungspotential haben.

2.1. Kategorisierung

Ein System kann grundsätzlich durch verschiedene Eigenschaften klassifiziert werden. So sind vom Hersteller über das OS (Operating System) bis zu den installierten Diensten viele Kategorien möglich. In dieser Arbeit wird auf einige dieser möglichen Kategorien eingegangen und am Ende dieses Abschnitts werden wir uns auf eine Kategorisierung festlegen.

2.1.1. Physische Eigenschaften

Ein Rechnersystem kann aufgrund von physischen Eigenschaften kategorisiert werden. So haben die meisten Systeme eine Hauptplatine mit einem Chipsatz, auf dem ein bestimmtes BIOS (Basic Input Output System) installiert ist. Hierdurch ergeben sich Sicherheitslücken, die besonders bei einem direkten - also physischen - Zugriff auf das Rechnersystem von großer Bedeutung sind. So könnte ein Angreifer durch das Ersetzen des vorinstallierten BIOS Zugang zu einem passwortgeschützten Rechner erlangen.

Auf der Hauptplatine sind die zentralen Recheneinheiten wie eine oder mehrere CPU (Central Processing Unit) bzw. GPU (Graphics Processing Unit) angebracht. Diese können jeweils weiter unterteilt und zugeordnet werden nach Eigenschaften wie:

- Hersteller
- Anzahl der Kerne
- Baujahr
- Architektur (x86, x64)

Eine Unterstützung bei der Erkennung von beispielsweise einem Storage-Server oder einem NAS (Network Attached Storage) bietet die verfügbare Gesamtzahl der installierten Festplatten und der jeweiligen Speicherkapazität. Bei einer Gesamtkapazität von 64 TB (Terabyte) liegt im Jahr 2013 der begründete Verdacht nahe, dass es sich um eine Art

von NAS handeln könnte, also einen Dateiserver, der im Netzwerk Speicherkapazität zur Verfügung stellt.

Denkbar ist auch eine weitere Unterteilung der Systeme nach Farbe oder Größe, jedoch spielen diese Eigenschaften keine Rolle im Bezug auf die Sicherheit im Netz. Aber auch die physischen Eigenschaften wie der Chipsatz oder die eingesetzten Recheneinheiten sind bei einer Betrachtung auf Netzebene nicht von großer Bedeutung. So erfolgt ein Zugriff auf das System nahezu ausnahmslos über das darauf installierte Betriebssystem. Einzige Ausnahmen sind dabei lokale Zugriffsmöglichkeiten wie beispielsweise Wake on LAN (Local Area Network), welche aber nicht über das Internet erreichbar sein sollten. Die Absicherung solcher physischen Zugriffsarten obliegt dem Systemadministrator oder dem lokalen Netzadministrator.

2.1.2. Nutzergruppen

In Wissenschaftsnetzen wie dem MWN sind viele verschiedene Personengruppen angebunden. So gibt es Nutzer, die selbst Dienste wie einen Webserver anbieten, aber auch eine große Zahl von Personen, die im Netz lediglich als Client unterwegs sind. Tendenziell sind so vielleicht wissenschaftliche Mitarbeiter an einer informationstechnischen Fakultät mehr als Anbieter von IT (Informationstechnik)-Diensten tätig als beispielsweise Mitarbeiter einer nicht naturwissenschaftlichen Fakultät.

Eine weitere Unterscheidungsmöglichkeit ergibt sich aus der Zugehörigkeit zu einer Universität bzw. Hochschule. Die Einteilung in Gruppen erfolgt dann nach dem Vorbild der Matrixorganisation. Jeder Person wird dabei ein institutioneller Bereich und ein Funktionsbereich zugewiesen. Ein Professor an der LMU (Ludwig-Maximilians-Universität München) würde dann beispielsweise in die Kategorien LMU und Professor eingeordnet.

An manchen im MWN betriebenen Geräten ist zudem ein spezifischer Verwendungszweck fest vorgegeben. So kann an einem Bibliotheksrechner meist nur eine Bestandsrecherche durchgeführt werden. Es ergibt sich daraus ein geringeres Gefährdungsrisiko und das System müsste lediglich vor allgemeinen Angriffen geschützt werden bzw. gegen spezifische Angriffe auf die Bibliothekssoftware, aber nicht vor Angriffen auf installierte Dienste wie Web- oder Mailserver.

2.1.3. Betriebssystem

Von großer Bedeutung sind die jeweiligen Patchlevel bzw. Versionsnummern des Betriebssystems aber auch bei den installierten Diensten. Hieraus ergeben sich meist erst die potentiellen Schwachstellen. So sind das Gros der bekannten und kritischen Schwachstellen in der jeweils neuesten Programmversion vom Hersteller behoben. In der Realität werden diese neuen Programmversionen aber jeweils nicht weltweit auf allen betroffenen Systemen automatisch installiert und somit gibt es trotz verfügbarem Sicherheitsupdate nach wie vor viele Systeme mit veralteten Installationen. Selbstverständlich ist ein rundum aktuelles System keineswegs eine Garantie dafür, dass keine Schwachstellen enthalten sind. Jede Änderung im Programmcode - und sei es nur die Behebung einer anderen Schwachstelle - kann wiederum eine neue Schwachstelle erzeugen.

Sind das Betriebssystem und die installierten Dienste einmal korrekt erfasst, kann in einem nächsten Schritt auf den eigentlichen Einsatzzweck des Systems geschlossen und mögliche Schwachstellen können benannt werden.

Beispielsweise kann ein erkanntes Windows 2000 System den Windows-Systemen zugeordnet werden und ein Debian-System den Linux-Systemen. Sicherheitstechnisch relevant ist aber wie erwähnt nicht nur, ob es beispielsweise ein Windows oder Linux-System ist, sondern insbesondere welche Version dieses bestimmte System hat. So kann eine Kategorisierung anhand folgender Tabelle erfolgen:

Erkannter Name	Betriebssystemkern	Version
Windows 2000 Professional	Windows NT	5.0 Build 2195.6717 (SP 4)
Ubuntu 12.04	Linux Kernel 3.5.0-23.35	Ubuntu 12.04.3 LTS
OS X 10.9	Darwin 13.0.0	10.9.0-13B42
Android 4.4 KitKat	Linux Kernel 3.4.0-gadb2201	4.4.2-KOT49H

Abbildung 2.1.: Kategorisierung anhand der Betriebssysteme

Das Betriebssystem ist aber nicht das einzige Einfallstor für einen geplanten Angriff. Von großem Interesse können installierte Programme und deren Dienste sein. Auf Server-Systemen wäre dies beispielsweise ein installierter Webserver oder auf Windows-Systemen eine aktivierte Netzwerkfreigabe. Jede installierte Anwendung stellt somit zugleich ein mögliches Sicherheitsrisiko dar. Je mehr Anwendungen also auf dem Zielsystem installiert sind, die in irgendeiner Art über das Netzwerk kommunizieren, desto größer können die Erfolgschancen für den Angreifer sein, dass ein Zugriff auf das System gelingt.

2.1.4. Dienste

Systeme erfüllen heute verschiedenste Aufgaben, sind aber trotz einer Vielzahl von Anwendungen häufig relativ einfach einem eindeutigen Einsatzzweck zuordenbar. So wird eine internetfähige Spielkonsole selten als Webserver genutzt und ein Webserver im Umkehrschluss nicht als Spielkonsole. Es gibt aber auch Fälle, in denen Systeme im Mischbetrieb genutzt werden. So kann ein Server zugleich als Web- sowie Datenbankserver konfiguriert sein. Aus diesem Grund ist es möglich, dass ein System in mehrere Kategorien eingeordnet werden muss.

Erkannter Name	Einsatzzweck	Dienste
Windows 2000 Professional	Workstation	Dateifreigabe
Ubuntu 12.04	Server	Webserver, Datenbankserver
OS X 10.9	Privatrechner	-
Android 4.4 KitKat	Smartphone	DLNA-Server

Abbildung 2.2.: Kategorisierung anhand des Einsatzzweckes und Diensten

2.1.5. Schwachstellen

Zusätzlich zu den bereits aufgezählten Kategorien ist eine Art Schwachstellenkategorie denkbar, welche Systeme aufgrund des Vorhandenseins einer oder mehrerer Eigenschaften einer bestimmten Schwachstelle zuweist. So kann das Vorkommen einer bestimmten Kombination von verschiedenen Diensten oder auch nur die einfache Installation einer bestimmten Softwareversion eine Gefahr darstellen. Wäre beispielsweise eine bestimmte Schwachstelle

vorhanden und das System somit sofort und sogar von Amateuren kompromittierbar, könnten die Netzadministratoren reagieren und den entsprechenden Host auf das Risiko hinweisen - ohne sich physisch auf dem Host einzumischen. Auch können bestimmte Schwachstellen einen eindeutigen Hinweis auf ein Betriebssystem liefern, da sie nur bei diesem spezifischen OS auftreten.

Erkannter Name	Schwachstelle
Windows 2000 Professional	MS05-043
Ubuntu 12.04	MySQL 5.1.x - CVE-2012-2122
OS X 10.4.8	MOAB-09-01-2007 Finder
Android 4.4 KitKat	-

Abbildung 2.3.: Kategorisierung anhand einer Schwachstelle

2.1.6. Fazit

Nachdem wir eine Auswahl an möglichen Kategorien betrachtet haben, legen wir uns auf die folgende Kategorisierung fest, in die die jeweiligen Systeme später eingeordnet werden sollen. Weder die physischen Eigenschaften eines Systems (2.1.1) noch die Nutzergruppen (2.1.2) wurden darin aufgenommen, da diese bei einer netzbasierten Analyse wie bereits ausgeführt nur wenig Aufschluss über einen Host geben. Stattdessen sind das Betriebssystem (2.1.3) und die installierten Dienste (2.1.4) von großer Bedeutung. Ist das Betriebssystem bekannt, kann man es besser schützen und von den installierten Diensten kann auf den Einsatzzweck geschlossen werden. Zusätzlich sind die Schwachstellen (2.1.5) von Bedeutung, da durch sie direkt auf ein Betriebssystem oder auf installierte Dienste geschlossen werden kann. Ziel der

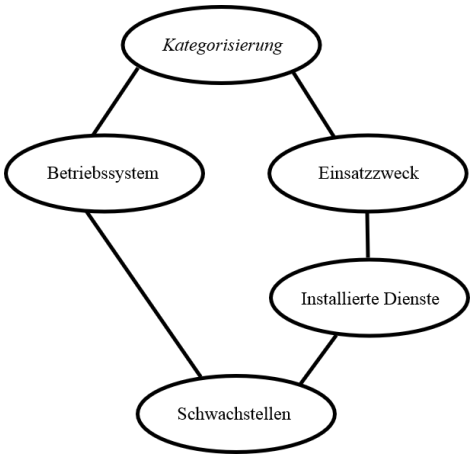


Abbildung 2.4.: Kategorisierung der unterschiedlichen Systeme

kombinierten Nutzung von den im Folgenden vorgestellten Methoden soll sein, dass man die Vorteile der unterschiedlichen Herangehensweisen miteinander verknüpft. So könnte man in einem ersten Schritt mittels einer Netflow-Analyse ein System kategorisieren. Im nächsten Schritt kann zusätzlich eine DPI (Deep Packet Inspection) durchgeführt werden, um die vorherige Kategorisierung zu überprüfen.

2.2. Aktive Erkennungsmethoden

Es stehen verschiedene Methoden zur Verfügung, anhand derer auf die Kategorien eines spezifischen Zielsystems geschlossen werden kann. Diese werden im Folgenden vorgestellt und auf deren Eignung zur Erkennung der verschiedenen Kategorien untersucht.

Speziell - aber nicht nur - bei der Erkennung von Betriebssystemen unterscheidet man beim sogenannten OS-Fingerprinting (auch bekannt als TCP (Transmission Control Protocol)/IP (Internet Protocol)-Stack-Fingerprinting) aktive und passive Erkennungsmethoden. Dabei macht man sich die Eigenschaft zunutze, dass jedes Betriebssystem seine eigene TCP/IP-Protokollstapel-Implementierung hat. Durch die Untersuchung der Header von Netzwerkpaketen können diese ausgelesen und wichtige Hinweise auf das Betriebssystem gefunden werden.

Bei aktiven Verfahren erfolgt also ein Zugriff auf das zu testende System in irgendeiner Art und Weise, während bei passiven Verfahren die Verbindungen und die übertragenen Datenpakete auf Netzwerkebene analysiert werden. Der genaue Ablauf wird bei der jeweiligen Erkennungsmethode beschrieben.

2.2.1. Portscanner

Mithilfe eines Portscanners ist es möglich, zu überprüfen welche Ports (TCP oder UDP (User Datagram Protocol)) auf einem Zielsystem geöffnet sind. Dies kann Hinweise auf die verwendeten Dienste auf dem Zielsystem geben, beispielsweise läuft ein Webserver standardmäßig über den TCP-Port 80. Eine Garantie, dass es sich dabei um einen Webserver handelt, hat man dadurch aber noch nicht. So können Dienste auf beliebigen Portnummern betrieben werden.

```
nmap -sV -T4 -F www.lrz.de

Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-09 23:52 CET
Nmap scan report for www.lrz.de (129.187.254.92)
Host is up (0.00057s latency).
rDNS record for 129.187.254.92: www.lrz-muenchen.de
Not shown: 96 filtered ports
PORT     STATE SERVICE    VERSION
80/tcp   open  http       Apache httpd 2
443/tcp  open  ssl/http   Apache httpd 2
8080/tcp open  tcpwrapped
8443/tcp open  tcpwrapped
```

Abbildung 2.5.: Nmap-Scan inklusive Dienst- und Versionserkennung

Die Software Nmap (Abkürzung für Network Mapper) zählt zu den Portscannern und bietet darüber hinaus noch viele weitere Möglichkeiten zur genauen Analyse, welche Dienste und welches Betriebssystem auf dem Host läuft. So können über Kommandozeilen-Optionen zusätzliche Funktionen aktiviert werden, wie beispielsweise die Erkennung von Betriebssystem (Kommandozeilen-Flag $-O$) und installierten Diensten inklusive deren Versionsnummer (Kommandozeilen-Flag $-sV$). Die Option $-F$ im Beispiel bewirkt, dass lediglich die Ports

geprüft werden, die einen Eintrag in */etc/services* auf dem eigenen System besitzen.

Nmap sendet nach [Nma13] verschiedene Testpakete aus einer eigenen Datenbank an den Zielhost und erkennt im Erfolgsfall anhand der Antworten das Betriebssystem, den Dienst und die genaue Versionsnummer. Wenn Nmap also einen offenen TCP-Port 80 feststellt und zudem einen Apache mit Versionsnummer ausgibt, so handelt es sich um einen Apache Webserver. Selbstverständlich handelt es sich bei allen Ergebnissen stets nur um mehr oder minder gute Hinweise.

Eine absolute Sicherheit, dass es sich um ein bestimmtes System oder um eine bestimmte Dienstgruppe handelt, bekommt man dabei allerdings nicht. Das allein schon aus dem Grund, weil der Besitzer sein Rechnersystem im MWN frei konfigurieren kann. So ist es nicht ausgeschlossen, dass sich hinter dem Port 80 - Beispiel ein Filesharing-Client verbirgt, der sich zur Tarnung als Webserver ausgibt.

Auch lässt sich mittels Nmap eine Erkennung von Betriebssystemen durchführen, dabei werden nach [Fyo98] verschiedene Tests durchgeführt, zwei werden im Folgenden kurz vorgestellt.

FIN Probe Bei der FIN Probe wird ein FIN Paket - also ein Bit zum Verbindungsabbau - an einen offenen Port gesandt und auf eine Antwort gewartet. Das darauf korrekte Verhalten eines OS wäre, nicht zu antworten. Viele Implementierungen sind allerdings fehlerhaft und geben eine Antwort wie beispielsweise ein RESET zurück.

BOGUS Flag Test Es wird beim Verbindungsaufbau ein undefiniertes TCP-Flag im Header eines SYN-Pakets eingefügt. Dabei verhalten sich bestimmte, ältere Linux-Systeme in der Art, dass sie dieses undefinierte Flag wiederum in ihrer Antwort mitsenden. Andere Betriebssysteme setzen dagegen die Verbindung zurück, falls sie so ein verfälschtes Paket erhalten. So erhält man weitere Hinweise zur Unterscheidung der verschiedenen OS.

Nmap ist bei Angreifern und auch bei Administratoren aufgrund seiner Zuverlässigkeit sehr beliebt und ist unter anderem auch in manchen Vulnerability Scannern integriert (z.B. Nessus).

Da ein Portscan stets eine bzw. mehrere Verbindung(en) zum Host aufbaut, zählt man diese Methode folglich zu den aktiven Erkennungsmethoden.

Das am LRZ eingesetzte Open-Source-Werkzeug Dr. Portscan ([vEMH13]) unterstützt bei der Automatisierung von Portscans, sodass die zeitaufwändige Erfassung der Hosts erleichtert wird.

Je nach Konfiguration der Hosts wäre es möglich, die System- und Dienstekategorisierung vollständig anhand der Portscanner-Resultate durchzuführen. In Kombination mit einem guten Vulnerability Scanner ist dies eine zuverlässige Lösung für die Kategorisierung der Systeme. Allerdings zählen beide Verfahren zu den aktiven Erkennungsmethoden, was zu mehreren Problemen führen kann. So könnte ein regelmäßiger Portscan auf vielen Systemen das Netz auslasten oder man verursacht durch die aktive Suche nach Schwachstellen selbst Schäden auf den zu testenden Systemen, obwohl man diese eigentlich schützen wollte. Außerdem könnte man jeweils einen Alarm auf den getesteten Systemen auslösen, wenn beispielsweise ein IDS (Intrusion Dedection System) installiert ist.

Interessant für diese Arbeit sind deshalb besonders die passiven Erkennungsmethoden, wovon wir eine - die Netflow-Technik - später noch kennenlernen werden.

2.2.2. Vulnerability Scanner

Eine weitere interessante Methode sind Programme, die Systeme gezielt auf Schwachstellen und bekannten Sicherheitslücken prüfen - sogenannte Vulnerability Scanner. Diese können eine sinnvolle Ergänzung zu der erweiterten Portscan-Funktionalität von Nmap sein. Immer wenn eine Kategorisierung bei einem System nicht oder nur teilweise erfolgen konnte, kann durch einen Vulnerability Scanner eine zusätzlich Erkennungsmethode angewandt werden. Da diese im Vergleich zu einem Portscan sehr aufwändig ist, sollte diese nur sekundär eingesetzt werden. Primär sollte die Erkennung der Kategorien eines Systems über den weniger aufwändigen Portscan erfolgen.

Aber auch ohne den Einsatz eines Schwachstellenscanners können sich bereits nach dem Einsatz der zuvor vorgestellten Methode und deren erzeugten Kategorisierung automatisch Schwachstellen ergeben. Das liegt daran, dass die Kombination aus Betriebssystem und der installierten Dienste (inklusive deren jeweiliger Version) eindeutig auf eine Schwachstellen-Kategorie hinweisen kann. Beispielsweise kann jedes beliebige Betriebssystem in Kombination mit einem MySQL-Dienst, der eine kleinere Versionsnummer als 5.1.x hat, automatisch der Schwachstelle CVE-2012-2122 zugewiesen werden - ohne eine zusätzliche Prüfung mit einem Vulnerability Scanner.

Zum Teil überschneidet sich diese Erkennung allerdings mit derer von gängigen Vulnerability Scanner. Diese führen im Falle von Metasploit beispielsweise zusätzlich noch Exploits aus, um zu testen, ob das Zielsystem verwundbar gegenüber einer bekannten Schwachstelle ist. Die Ergebnisse solcher Scanner können also ebenfalls zur verbesserten Kategorisierung von Schwachstellen genutzt werden.

Dabei ist grundsätzlich zu unterscheiden zwischen intrusiven (aktiven) und nicht-intrusiven (passiven) Scanner. Beim intrusiven Scanning nach einer Schwachstelle versucht die Software tatsächlich eine Schwachstelle auszunutzen. Im Unterschied dazu wird bei der nicht-intrusiven Anwendung je nach Einsatzzweck nur gelesen. So kann beim lokalen Einsatz nach bestimmten Dateien im Dateisystem oder nach bestimmten Werten in der Windows-Registrierungsdatenbank gesucht werden. Ohne direkten Zugriff auf das System - also ohne Anmeldung - bedeutet die nicht-intrusive Anwendung, dass das Netzwerk ähnlich wie bei der Netflow-Technik oder bei der DPI auf mögliche Hinweise auf Schwachstellen hin untersucht wird.

Bekannte Werkzeuge für das aktive und intrusive Scannen nach Schwachstellen sind Nessus[1] und Metasploit[2]. Im Bereich der nicht-intrusiven also passiven Scanner ist die kommerzielle Lösung PVS (Passive Vulnerability Scanner) von Tenable Network Security bekannt.

2.3. Passive Erkennungsmethoden

Anders als bei den aktiven Verfahren, bei dem ein Zielsystem aktiv gescannt wird, wird bei passiven Erkennungsmethoden mit dem aufgezeichneten Datenverkehr gearbeitet, ohne dass die zu untersuchenden Hosts den Vorgang feststellen können.

[1] Hersteller ist Tenable Network Security, offizielle Website: `http://www.tenable.com/products/nessus`
[2] Open-Source-Projekt von Rapid7, offizielle Website: `http://www.metasploit.com/`

2.3.1. Netflow

Die Netflow-Technik arbeitet mit IP-Datenströmen. Es handelt sich also um eine passive Erkennungsmethode und der beobachtete Verkehr wird dadurch nicht beeinflusst. Die Aufzeichnung und Auswertung des Datenstroms übernimmt ein Netflow-Collector, der vom Router exportierte UDP-Datagramme erhält (siehe Abbildung 2.6).

Eine freie Software, mit der der Netzwerkverkehr mitgeschnitten und analysiert werden kann, ist nfdump. nfdump kann die Daten selbst erheben oder aber bereits vorhandene Logdateien einlesen und auswerten. Die Ausgabe erfolgt in beiden Fällen auf einer grafischen Oberfläche über einen eigenen Webserver oder optional in eine Textdatei. Weitere bekannte Implementierungen zum Aufzeichnen des Netzverkehrs sind die freien Programme ntop[3], Tcpdump[4] und Wireshark[5].

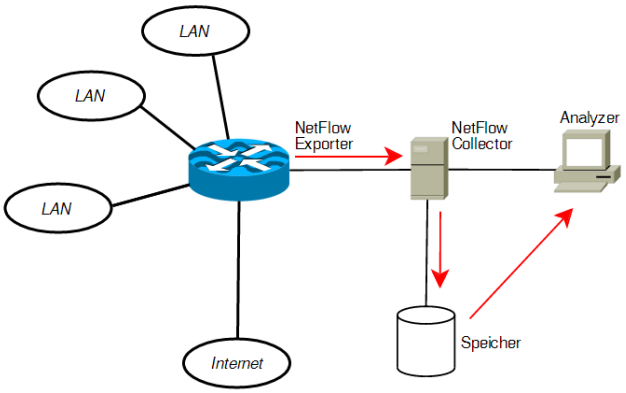


Abbildung 2.6.: Architektur der Netflow-Technik, angelehnt an [hel07]

Bei der Netflow-Technik kommen in der Regel zwei verschiedene Arten von Analysen vor:

1. Die Top-N-Analyse: es werden während eines frei definierbaren Zeitraums diejenigen Elemente gesucht, die den meisten Verkehr erzeugen.

2. Die Zeitanalyse: der gesamte Verkehr über eine Komponente wird für einen definierbaren Zeitraum aufgezeichnet.

Die für diese Arbeit aufschlussreichere Variante ist die Zeitanalyse, da sonst nur verkehrsreiche Dienste entdeckt werden könnten. In einem Flow-Eintrag sind neben den Metadaten wie Versionsnummer oder Zeitstempel, folgende für die Kategorisierung relevante Daten gespeichert:

- Quell- und Ziel-IP-Adressen
- Quell- und Ziel-Ports
- TCP-Flags
- Protokoll-Typ

[3] Abkürzung für network top, offizielle Website: `http://www.ntop.org/`
[4] Open-Source-Projekt, offizielle Website: `http://tcpdump.org`
[5] freies Programm zur Analyse von Netzwerk-Kommunikationsverbindungen, offizielle Website: `http://www.wireshark.org`

Portbasierte Diensterkennung

Anhand eines kleinen Beispiels kann die Vorgehensweise bei der Netflow-Analyse verdeutlicht werden: Aufgrund der Häufigkeit von Anfragen an einen bestimmten Port kann es sich um eine bestimmte Dienstgruppe handeln. Bauen also beispielsweise nur externe IP-Adressen eine Verbindung zu einem System über den Port 80 auf, könnte es sich dabei um einen Webserver handeln. Dieser Vermutung lässt sich mit den folgenden Netflow-Daten leicht nachvollziehen.

```
Date flow start  Dur.  Proto.  Src IP Addr:Port          Dst IP Addr:Port
23:07:11.628     0.250 TCP     129.187.254.92:80    ->   127.191.42.12:49892
23:07:11.628     0.250 TCP     129.187.254.92:80    ->   127.191.42.12:49894
23:07:11.628     0.250 TCP     127.191.42.12:49895  ->   129.187.254.92:80
23:07:11.628     0.250 TCP     129.187.254.92:80    ->   127.191.42.12:49893
23:07:12.750     0.250 TCP     129.187.254.92:80    ->   134.176.3.91:47308
23:07:14.128     3.250 TCP     129.187.254.92:80    ->   36.149.38.21:52306
23:07:19.378     3.000 TCP     129.187.254.92:80    ->   36.149.38.21:57581
23:07:22.878     3.250 TCP     129.187.254.92:80    ->   36.149.38.21:57317
```

Abbildung 2.7.: Netflow-Daten von www.lrz.de

So könnte der Host mit der IP-Adresse 129.187.254.92 ein Webserver, der Anfragen von verschiedenen Besuchern auf Port 80 beantwortet, sein. Jedoch stützt sich diese Aussage bis jetzt nur auf einen erkannten Port und die Art der Verbindungen, konkreter gesagt, um die große Zahl an ein- und ausgehenden Verbindungen über den gleichen Port. Die Daten wurden in diesem Beispiel aus Gründen der Übersichtlichkeit gekürzt.

Aufgrund der festgestellten Charakteristika gehen wir im Folgenden davon aus, dass es sich um einen Webserver handelt und wollen nun feststellen, um welchen Webserver-Dienst es sich handeln könnte. Es könnte beispielsweise ein Apache HTTP (Hypertext Transfer Protocol) Server oder ein IIS (Microsoft Internet Information Services) installiert sein. Daraus kann man in einem nächsten Schritt sogar ein Rückschluss auf das Betriebssystem ziehen. So kann bei einem entdeckten IIS aufgrund der Bindung an ein Microsoft-eigenes OS auf eine Windows-Version geschlossen werden. Ein typischer, mittels der Netflow-Technik aufgezeichneter Eintrag enthält wie oben erwähnt weitere Metadaten, auf deren Betrachtung wir bisher verzichtet haben. Für die Erkennung von Diensten könnten diese allerdings wichtige Hinweise liefern. So werden pro Flow-Eintrag zusätzlich noch die Anzahl der Pakete und die Größe in Bytes aufgezeichnet. So könnte sich ein einzelner Dienst anhand eines bestimmten Merkmals qualifizieren, beispielsweise stets ein Paket mit einer bestimmten Byte-Größe beim Verbindungsaufbau oder Verbindungsabbau.

Des Weiteren lassen sich in der aktualisierten Version Netflow v9 von Cisco noch viele weitere zusätzliche Daten erfassen, was eine Auswertung vereinfachen kann. Für diese Arbeit wird auf diese Funktionalität allerdings verzichtet, da in der Praxis Version 5 die am häufigsten verwendete Version ist.

Weitere Möglichkeiten zur Diensterkennung

Die Diensterkennung auf Basis der Ports hat eine entscheidende Schwäche. So kann prinzipiell jeder Dienst auf jedem Port laufen. Ein Beispiel wäre ein FTP (File Transfer Protocol)-Dienst, welcher auf Port 22 statt wie standardmäßig vorgesehen auf Port 21 lauscht. In

einem Flow-Eintrag gibt es neben der Quell- und Zieladresse standardmäßig auch drei weitere Spalten:

- `Duration`: Dauer der Übertragung in Millisekunden
- `Packets`: Anzahl der Pakete
- `Bytes`: Größe in Bytes der übermittelten Pakete

Das Ziel ist also, für Dienste bestimmte Eigenschaften zu finden, anhand derer man diese identifizieren kann. Da solche Regelmäßigkeiten für das menschliche Auge nur sehr schwer zu finden sind, wird in einem kleinen Prototypen eine Netflow-Datei nach bestimmten Regeln ausgewertet und zwar sollen Duplikate gefunden werden, für die eines der folgenden Kriterien gilt:

- gleiche Anzahl der Pakete
- gleiche Übertragungsdauer (Spalte `Duration`)
- identische Größe in `Bytes` der übermittelten Pakete

Ausdrücklich nicht gleich sein muss der jeweilige Port, da es sich um einen Dienst handeln könnte, der verschiedene Ports nutzt. Beispielsweise der FTP-Dienst, der auf einem System auf Port 21, auf einem anderen wiederum auf Port 22 lauscht.

Quellcode 2.1: Auswertung von Netflow-Daten hinsichtlich Dauer der Übertragung

```
1  $handle = fopen('ipv6_lastday.txt',"r");
2  $count=0;
3  while (($line = fgets($handle)) !== false) {
4          $count++;
5          if($count==1) { continue; }
6          $line = str_replace('->','',$line);
7          $line = str_replace(chr(1),' ',$line);
8          $line = preg_replace('/\s\s+/',' ', $line);
9          if($line[0]!=2) {
10                 continue;
11         }
12         list($date, $time, $duration, $protocol, $source, $target, $packets,
               $bytes) = explode(" ", $line);
13         list($sourceIP, $sourcePort) = explode('.',$source);
14     list($targetIP, $targetPort) = explode('.',$target);
15         $result['Duration: '.$duration]=array(
16                 'count'=>@$result['Duration: '.@$duration]['count']+1,
17                 'sourcePort'=>@$result['Duration: '.@$duration]['sourcePort']."
                       , ".$sourcePort,
18                 'targetPort'=>@$result['Duration: '.@$duration]['targetPort']."
                       , ".$targetPort,
19         );
20 }
21 foreach($result as $index => $values) {
22         if($values['count']>5) {
23                 echo $index." / ";
24                 foreach($values as $name => $value) {
25                         echo $name.": ".$value." - ";
26                 }
27                 echo "<br />";
28         }
29 }
```

In einem kurzen PHP-Skript (Quellcode 2.1 stellt die Auswertung der Daten hinsichtlich der `Duration` dar) werden verschieden große Netflow-Dateien hinsichtlich dieser Kriterien ausgewertet. Das Ergebnis erhalten wir in 2.2:

Quellcode 2.2: Ergebnis der Auswertung von Netflow-Daten hinsichtlich der Übertragungsdauer

```
1 Duration: 19.000 / count: 6 - sourcePort: 49163, 53045, 49188, 49190, 49193,
      49191 - targetPort: 41994, 60653, 49978, 22771, 57971, 60204 -
2 Duration: 40.500 / count: 18 - sourcePort: 80, 443, 443, 443, 443, 443, 443,
      443, 443, 443, 443, 443, 443, 443, 50752, 50753, 443, 443 - targetPort:
      42873, 59455, 59457, 59462, 59463, 59464, 59465, 59470, 61532, 59474,
      59475, 59476, 59477, 59484, 80, 80, 59488, 59489 -
3 Duration: 15.250 / count: 8 - sourcePort: 56299, 43244, 37447, 35305, 33264,
      443, 41786, 49453 - targetPort: 50036, 56131, 26346, 21765, 17706, 51535,
      35609, 56332 -
4 Duration: 36.750 / count: 6 - sourcePort: 50802, 80, 50800, 443, 443, 443 -
      targetPort: 80, 50800, 80, 59403, 59404, 59405 -
5 Duration: 19.750 / count: 6 - sourcePort: 60718, 54112, 54114, 60720, 60723,
      54118 - targetPort: 14136, 61222, 11497, 23461, 16261, 60193 -
6 Duration: 40.250 / count: 7 - sourcePort: 443, 443, 443, 443, 443, 443, 61865 -
       targetPort: 59454, 59456, 59471, 59482, 59483, 59485, 443 -
7 Duration: 45.000 / count: 6 - sourcePort: 30274, 56650, 42778, 443, 443, 16874
      - targetPort: 20836, 80, 443, 42778, 51133, 30560 -
```

Das Problem bei dieser Auswertung ist augenscheinlich, dass die Daten nicht eindeutig sind. So ist es zwar so, dass einige Pakete von einer Quelle mit Port 443 gesandt werden, jedoch ist diese Zahl verglichen mit der Gesamtzahl an aufgezeichneten Pakete gering. Des Weiteren hängt die Übertragungsdauer nicht nur vom verwendeten Dienst ab, sondern auch von weiteren Faktoren wie der aktuellen Netzauslastung. Eine bessere Eigenschaft könnte deshalb die Paketgröße sein, die unabhängiger von äußeren Faktoren ist. Wir passen deshalb unser Skript in 2.1 hinsichtlich der Variablen so an, dass die Paketgröße ausgewertet wird.

Quellcode 2.3: Ergebnis der Auswertung von Netflow-Daten hinsichtlich der übertragenen Bytes

```
1 Bytes: 1280 / count: 21 - sourcePort: , 80, 443, 443, 443, 443, 443, 443, 443,
      443, 443, 443, 443, 443, 443, 443, 443, 443, 443, 443 -
      targetPort: , 42873, 59455, 59454, 59456, 59457, 59462, 59463, 59464,
      59465, 59470, 59471, 59474, 59475, 59476, 59477, 59482, 59483, 59484,
      59485, 59488, 59489 -
2 Bytes: 1031 / count: 6 - sourcePort: , 179, 179, 179, 179, 6352, 32604 -
      targetPort: , 14941, 37790, 40022, 27703, 179, 179 -
3 Bytes: 60 / count: 7 - sourcePort: , 52221, 54534, 443, 60749, 62972, 49384,
      49387 - targetPort: , 443, 443, 62442, 443, 443, 443, 443 -
4 Bytes: 96 / count: 9 - sourcePort: , 123, 123, 123, 123, 123, 123, 12347,
      29396, 25069 - targetPort: , 123, 123, 123, 123, 123, 123, 53, 53, 53 -
5 Bytes: 156 / count: 21 - sourcePort: , 20969, 51413, 51413, 51413, 51413,
      31143, 51413, 41900, 51413, 51413, 53342, 51413, 47905, 51413, 47905,
      51413, 53, 47905, 53, 59665, 29491 - targetPort: , 36183, 42753, 51863,
      45713, 59912, 26596, 23074, 55940, 51034, 63915, 13955, 14970, 14236,
      24548, 59865, 51413, 23184, 36849, 63416, 49041, 27834 -
6 Bytes: 161 / count: 13 - sourcePort: , 49256, 443, 53, 64098, 53, 51230, 53,
      53, 53, 59004, 53, 443, 53 - targetPort: , 443, 49256, 47939, 443, 45299,
      443, 12347, 33160, 21411, 443, 29396, 64849, 63738 -
7 Bytes: 93 / count: 9 - sourcePort: , 16043, 29469, 5866, 44158, 19776, 2527,
      17992, 31046, 58725 - targetPort: , 53, 53, 53, 53, 53, 53, 53, 53, 53 -
8 Bytes: 155 / count: 7 - sourcePort: , 53, 53, 53, 53, 53, 53, 53 - targetPort:
      , 63152, 31826, 17557, 37909, 6548, 6129, 49927 -
9 Bytes: 68 / count: 8 - sourcePort: , 52742, 52258, 53208, 56736, 57569, 56698,
      56699, 57574 - targetPort: , 59461, 33253, 29967, 42467, 61557, 56281,
      59385, 63094 -
```

Das Ergebnis ist in 2.3 dargestellt. Man sieht, dass die übertragenen Bytes auf einen bestimmten Dienst hinweisen können. Wir haben beispielsweise einige Zeilen mit der Größe von 93 Bytes, die an einen DNS (Domain Name System)-Port gerichtet sind. Oder aber viele Flow-Einträge, welche überwiegend den Quellport 443 und die gleiche übertragene `Bytes`-Zahl von 1280 haben. Jedoch sieht man, dass diese nicht einheitlich sind. So gibt es offensichtlich auch DNS-Anfragen mit der Größe 96 Bytes oder Server, die auf dem Port 443 mit einer anderen Paketgröße antworten.

Wir wollen als letztes nun noch die Anzahl der übermittelten Pakete betrachten und ob man daraus bestimmte Dienste identifizieren kann. Die Variablen wurden nochmals angepasst und das Ergebnis ist in 2.4 dargestellt.

Quellcode 2.4: Ergebnis der Auswertung von Netflow-Daten hinsichtlich der Anzahl der Pakete

```
1 Packets: 9 / count: 22 - sourcePort: , 35419, 49163, 52245, 51401, 60718,
      54112, 53045, 51409, 54114, 60720, 443, 60723, 49188, 51329, 443, 49190,
      49193, 49191, 59476, 54118, 53914, 443 - targetPort: , 443, 41994, 80,
      47251, 14136, 61222, 60653, 38918, 11497, 23461, 53587, 16261, 49978, 443,
      51535, 22771, 57971, 60204, 443, 60193, 443, 51749 -
2 Packets: 7 / count: 8 - sourcePort: , 37394, 50802, 50801, 5222, 35132, 51871,
      60864, 443 - targetPort: , 179, 80, 80, 54084, 179, 80, 443, 60864 -
3 Packets: 16 / count: 22 - sourcePort: , 80, 443, 443, 443, 443, 443, 443, 443,
      443, 443, 443, 443, 443, 443, 443, 443, 443, 443, 443, 443, 443, 63834 -
      targetPort: , 42873, 59455, 59454, 59456, 59457, 59462, 59463, 59464,
      59465, 59470, 59471, 59474, 59475, 59476, 59477, 59482, 59483, 59484,
      59485, 59488, 59489, 80 -
4 Packets: 11 / count: 7 - sourcePort: , 50799, 443, 38548, 51898, 49965, 50781,
      443 - targetPort: , 80, 42778, 443, 80, 443, 80, 49406 -
5 Packets: 12 / count: 12 - sourcePort: , 80, 80, 16874, 16874, 44578, 42778,
      50737, 16874, 50760, 80, 80, 443 - targetPort: , 50800, 50814, 27973,
      46700, 23427, 443, 443, 46928, 80, 50760, 51872, 49965 -
```

Auch hier ergibt sich leider wieder kein einheitliches Bild, auch wenn bestimmte Zahlen, wie beispielsweise 16 auf einen aktiven Webserver mit SSL (Secure Sockets Layer) auf Port 443, hindeuten.

Abschließend ist zu sagen, dass es durchaus möglich ist, einen Dienst anhand solcher Eigenschaften zu erkennen. Jedoch gibt es dabei mehrere Herausforderungen. Zum einen benötigt man eine sehr große Datenbasis, anhand derer man die Dienste klassifizieren kann. Zum anderen ist die Erkennung ungenau, weil theoretisch jeder andere Dienst schnell auf die gleiche Paketgröße oder Verbindungsdauer kommen könnte. Für letzteren Punkt wiederum gibt es mehrere Lösungsansätze: Man könnte eine komplexe Heuristik entwerfen, die verschiedene vorliegende bzw. nicht vorliegende Eigenschaften voraussetzt und wenn ein bestimmter Schwellenwert überschritten ist, dann auf einen Dienst schließen lässt. Oder aber man nutzt die Technik lediglich dafür, um bestehende Vermutungen, die beispielsweise durch einen aktiven Port entstanden sind, durch zusätzliche qualifizierenden Merkmale wie der Anzahl der Pakete oder Dauer der Übertragung, zu bestätigen.

Betriebssystemerkennung

Anhand der bereits erwähnten TCP-Flags ist es zudem möglich, neben den Diensten auch das Betriebssystem latent zu erkennen. Jedes Betriebssystem verfügt über eine eigene TCP/IP-Protokollstapel-Implementierung, die sich jeweils zwischen den verschiedenen Betriebssystemen unterscheiden.

Insgesamt variieren nach [Wik13] folgende Felder in den verschiedenen Implementierungen und können daher für die OS-Erkennung herangezogen werden.

- Initial Time to Live (8 Bit)
- Receive Window Size (16 Bit)
- Maximum Segment Size (16 Bit)
- Don't fragment flag (1 Bit)
- sackOK option (1 Bit)
- nop option (1 Bit)
- Window Scale Option (8 Bit)
- Initial packet size (16 Bit)

Nach [Fyo98] gibt es einige Methoden und Techniken, wie man die verschiedenen Betriebssysteme voneinander unterscheiden kann - ein kleiner Auszug wird davon im Folgenden vorgestellt. Grundsätzlich ist hier allerdings darauf hinzuweisen, dass sich diese Daten in der Konfiguration vieler OS anpassen lassen und so eine Verfälschung der Erkennung möglich ist.

TCP-Sequenznummern Nach [Fyo98] werden bei dieser Methode Muster bei den zu Beginn der Kommunikation verwendeten TCP-Sequenznummern gesucht. So werden diese je nach verwendetem Betriebssystem beispielsweise zufällig oder nach einem zeitbasierten Modell pro Zeiteinheit erhöht.

Don't Fragment Bit Viele Betriebssysteme setzen ein Bit in bestimmten Paketen. Da dies nicht alle OS und das auch nicht bei jedem Paket machen, ist damit ein weiteres Unterscheidungsmerkmal gefunden.

2.3.2. PRADS

PRADS (Passive Real-time Asset Detection System) ist ein Programm, das passiv den Netzwerkverkehr mithört. Dabei sammelt es Informationen über Hosts und deren Dienste, die im Netzwerk sichtbar sind. Es läuft eigenständig und kann entweder in Echtzeit lauschen oder bereits aufgezeichneten Netzwerkverkehr einlesen, vorausgesetzt die Daten liegen im pcap-Format vor.

PRADS nutzt dabei einige der bereits vorgestellten Techniken zur Erkennung der Systeme und Dienste, unter anderem:

- OS-Fingerprinting, sowohl SYN als auch SYN+ACK (IP/TCP)
- TCP-Diensterkennung durch Fingerprinting
- TCP-Erkennung von Hosts (SYN und SYN+ACK)
- UDP-Erkennung von Hosts
- UDP OS Fingerprinting

PRADS lässt sich leicht installieren[6] und die Erfassung der Systeme kann über die Konsole

[6] Installation nach offizieller Anleitung `https://github.com/gamelinux/prads/blob/master/doc/INSTALL`

mit dem folgenden Befehl gestartet werden.

Quellcode 2.5: Lauschen mit PRADS auf Netzwerkadapter

```
prads -i eth0 -l prads.log
```

Das Programm lauscht dann auf dem angegebenen Netzwerkadapter `eth0` und schreibt alle Ergebnisse in die angegebene Log-Datei `prads.log`. Eine Beispielausgabe der `.log`-Datei sieht wie folgt aus:

Quellcode 2.6: PRADS Aufzeichnung in .log-Datei

```
10.43.2.181,0,54354,6,SYN,[65535:64:1:64:M1460,N,W2,N,N,T,S,E,E:P:MacOS:iPhone
    OS 3.1.3 (UC):link:ethernet/modem:uptime:1574hrs],0,1300882012
10.43.2.181,0,0,0,ARP (Apple),C8:BC:C8:48:65:CA,0,1300882017
```

Diese Ergebnisse können dann weiter ausgewertet werden, sodass man am Ende eine Liste von Assets mit detaillierten Informationen wie beispielsweise offene Ports oder eingesetzte Protokolle erhält. Zusätzlich wird die Speicherung dieser Ergebnisse in einer Datenbank angeboten.

Alles in allem ist PRADS ein sehr nützliches Werkzeug, um eine Übersicht von aktiven Hosts in einem Netzwerk von einem Rechner aus zu bekommen. Im Unterschied zur Netflow-Technik, bei der ein Router alle Anfragen, die über diesen Router laufen, an einen Netflow Collector sendet, sind Werkzeuge wie PRADS Netzwerksniffer. Bei dieser Art von Netzwerkanalyse gibt es zwei verschiedene Betriebsmodi:

Non-Promiscuous Mode Es wird nur der ankommende und abgehende Datenverkehr des eigenen Systems aufgezeichnet, also nur die an das eigene System adressierten Datenframes.

Promiscuous Mode Es wird der gesamte Datenverkehr aufgezeichnet, auch die an andere Systeme adressierten Datenframes. PRADS zählt zu dieser Kategorie.

Dabei ist es von der Netzstruktur abhängig, welche Pakete empfangen werden können und welche nicht. Bereits der Einbau eines Switches kann eine Aufzeichnung fremder Datenpakete erheblich erschweren, da nicht an das System gerichtete Datenframes dann nicht mehr empfangen werden können. Aus diesem Grund wird in dieser Arbeit auf die Netflow-Technik gesetzt, da dort alle Verbindungsdaten auf Router-Ebene kontrolliert an einen Empfänger gesandt werden. Daraus folgt, dass wenn ein Host im Netz aktiv ist, dieser stets ermittelt werden kann.

2.3.3. Deep Packet Inspection

Eine weitere Möglichkeit, Hinweise für eine mögliche Kategorisierung zu finden, ist der Einsatz einer DPI. Diese Erkennungsmethode zählt ebenfalls zu den passiven Verfahren, ist aber aufwändiger als eine Netflow-Analyse. Sie sollte aus diesem Grund lediglich als Ergänzung angewandt werden, also immer dann wenn eine vorherige Erkennung anhand der Netflow-Technik nicht zum gewünschten Erfolg geführt hat.

Bei der DPI wird sowohl der Datenteil als auch der Header eines Pakets auf bestimmte Merkmale untersucht. Die Technologie wird allerdings auch zum Abhören und Sammeln von Informationen sowie zur Zensur im Internet eingesetzt, bei der Daten in Echtzeit manipuliert werden (DPM (Deep Packet Modification)). Der Einblick und deren Durchleuchtung ist

nach [Bed09] rechtlich problematisch, da aber keine Paketdaten verändert werden und Daten lediglich zu Sicherheitszwecken erfasst werden, ist der Einsatz der Methode prinzipiell möglich.

Wie bereits bei der Netflow-Technik werden auch bei der DPI einzelne Verbindungsdaten untersucht. Das hat den Nachteil, dass bereits bei kleineren Netzen schnell sehr große Datenmengen anfallen. Im Gegensatz zu einem Flow-Eintrag, der lediglich die Verbindungsdaten erfasst, wird bei der DPI das gesamte Datenpaket analysiert und wiederum in einem nächsten Schritt auf die Kategorien geschlossen. Ein weiterer Vorteil ist, dass die DPI eine passive Erkennungsmethode ist, sodass der Host diese nicht feststellen kann.

Von den Machern von ntop gibt es auch das plattformübergreifende Tool nDPI, das auf dem inzwischen nicht mehr weiter entwickelten OpenDPI basiert. Stand heute kann nDPI neben unverschleierten Protokollen wie HTTP nach [Nto13] auch verschleierte Protokolle wie HTTPS (Hypertext Transfer Protocol Secure) oder das Skype Protokoll erkennen. Ein weiteres quelloffenes Werkzeug zur Netzwerkanalyse, das ebenfalls eine DPI durchführt, ist ngrep. Es unterstützt allerdings nur Klartext-Protokolle wie HTTP, SMTP (Simple Mail Transfer Protocol) oder FTP und wird seit 2006 nicht mehr aktiv fortgeführt.

Netzbasierte IDS arbeiten vergleichbar zu den DPI-Tools, da diese auch anhand der Paketdaten erkennen müssen, ob es sich um einen Angriff handelt oder nicht. Dabei wird versucht, alle Pakete in einem Netzwerk zunächst aufzuzeichnen und anschließend auszuwerten. Bekannte netzbasierte IDS-Systeme wie Snort oder Suricata versuchen zusätzlich Angriffsmuster über eine Musterdatenbank (siehe Abbildung 2.8) oder über heuristische Methoden für bisher unbekannte Angriffe zu erkennen.

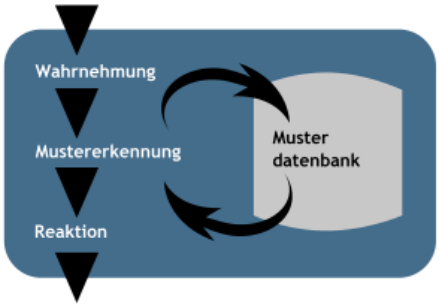


Abbildung 2.8.: Angriffserkennung durch Musterdatenbank

Verdächtige Aktivitäten sollen als Alarmereignis an eine für die Sicherheit zuständige Person oder Organisation gemeldet werden. Im Vergleich dazu sind IPS (Intrusion Prevention System) zusätzlich in der Lage, die betroffenen Datenpakete zu verwerfen, die Verbindung abzubrechen oder die übertragenen Daten abzuändern. Das bereits erwähnte Snort ist eine Kombination aus NIDS (Netzwerk-Basierte IDS) und NIPS (Network Intrusion Prevention System) und kann zusätzlich ähnlich wie tcpdump als Netzwerkanalyse-Werkzeug dienen.

3. Anforderungsanalyse

Im vorhergehenden Kapitel 2.1.6 wurde eine Kategorisierung erarbeitet und eine größere Auswahl der zur Verfügung stehenden Erkennungsverfahren vorgestellt. In diesem Kapitel sollen nun die Anforderungen an ein System, das diese Kategorisierung vornimmt, ermittelt und gewichtet werden. Anhand der Beschreibung eines Beispielszenarios, welches anschließend verallgemeinert wird, können die funktionalen und nichtfunktionalen Anforderungen abgeleitet werden. Da das für diese Arbeit erstellte Beispielsystem kleinerer Größenordnung nicht alle Anforderungen abbilden kann, werden die spezifischen Anforderungen in Pflicht- und Wunschkriterien eingeteilt.

3.1. Anwendungsszenario

An das MWN sind viele verschiedene Organisationen angeschlossen. So gibt es im Wintersemester 2013/14 allein an der Ludwig-Maximilians-Universität über 50.000 Studenten, an der Technischen Universität München nochmals 35.000 und viele weitere an den Hochschulen in und um München. Zusätzlich sind die Institutionen und deren Mitarbeiter selbst im Netz aktiv und betreiben eigene Dienste wie Webserver, FTP-Server oder Mailserver. So verändert sich die IT-Landschaft tagtäglich durch Neuinstallationen oder Veränderungen an bestehenden Rechnersystemen.

Aufgrund der großen Menge an Geräten im vom LRZ betriebenen Netz, gibt es für jede Unterorganisation und deren Nutzer eigene Netzverantwortliche - insgesamt mehr als 700 nach [Lei13]. Das können beispielsweise speziell geschulte Hochschulmitarbeiter einer Fakultät an einer der angeschlossenen Universitäten sein. Netzverantwortliche, eine mögliche Bezeichnung wäre auch Netzadministratoren, sind für den reibungslosen Betrieb und die Sicherheit in ihrem Netzbereich zuständig. In ihrem lokalen Subnetz sind sie jeweils Ansprechpartner für Security- und Abusefälle. Der im MWN eingeführte Mischbetrieb von IPv4 (Internet Protocol Version 4) und IPv6 (Internet Protocol Version 6) erhöht die Komplexität des Netzes zusätzlich.

Da das LRZ keinen administrativen Zugriff auf die von Endbenutzern im Netz betriebenen Geräte besitzt, können existierende Sicherheitslücken nicht durch die manuelle Nachinstallation von Updates geschlossen werden. Sicherheitsmaßnahmen können von den Netzverantwortlichen allerdings als organisatorische oder technische Maßnahmen implementiert werden, wie Abbildung 3.1 zeigt.

Bei unzureichender Absicherung über die Firewall sind die Netzteilnehmer, beispielsweise der Lehrstuhlmitarbeiter mit selbst betriebenem Webserver, nicht optimal geschützt. Aus diesem Grund werden die Firewalls meist so konfiguriert, als müssten theoretisch alle Dienste im nachgeschalteten Netz geschützt werden. So gibt es häufig Regeln für den Schutz von Mail-, Web- oder FTP-Server, obwohl nur Bürorechner ohne Serverdienste betrieben werden. Wenn die Netzverantwortlichen aber wissen würden, welche Systeme sich hinter den einzelnen IP-Adressen verbergen und welches Betriebssystem installiert ist bzw. welche Dienste angeboten werden, könnte die Firewall oder ein IDS, das als Ergänzung zu einer Firewall nach Angriffen

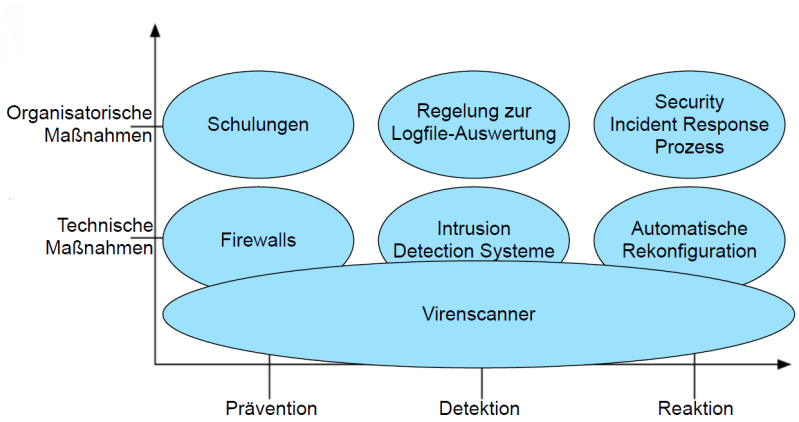


Abbildung 3.1.: Kategorisierung von Sicherheitsmaßnahmen, Quelle: [HR13]

sucht, viel spezifischer konfiguriert werden. Dabei gibt es zwei Möglichkeiten: Einerseits können grundsätzlich alle Dienste erlaubt werden mit Ausnahme von einer Liste von blockierten (Blacklist). Andererseits kann auch eine weitaus restriktivere Variante gewählt werden, wobei lediglich Dienste über Ports kommunizieren dürfen, die ausdrücklich zugelassen wurden (Whitelist). Bei beiden Varianten ist es von großer Bedeutung, dass man weiß, welche Dienste eigentlich in einem Netz aktiv sind und ob man diese gewähren lassen möchte oder nicht. Anhand einer detaillierten Auflistung aller Assets können die Firewall- oder IDS-Regeln bei beiden Varianten kontinuierlich überprüft und aktualisiert werden.

3.2. Funktionale Anforderungen

Eines der Hauptziele dieser Arbeit ist die Entwicklung eines Prototypen, der Hostsysteme in einem Netz aufgrund spezifischer Eigenschaften den im ersten Schritt erstellten Kategorien zuordnet. Dieser Prototyp wird durch die in diesem Abschnitt beschriebenen funktionalen Anforderungen genauer beschrieben.

3.2.1. Weboberfläche

Der Benutzer soll die Funktionalität des Systems über eine Weboberfläche nutzen können. Diese bietet den Vorteil einer zeitgemäßen und intuitiven Steuerung der Anwendung und ist einfach erweiterbar für mehrere Benutzer, die zur gleichen Zeit mit dem Programm arbeiten sollen. Zusätzlich können die Ergebnisse nach der Analyse grafisch besser dargestellt werden.

3.2.2. Datenbank

Für die Speicherung der über die Weboberfläche getätigten Aufträge und Einstellungen, soll ein DBMS (Database Management System) zum Einsatz kommen. Ebenfalls sollen die Analysedaten und Ergebnisse nach der Auswertung in einer Datenbank gespeichert werden. So ist zum einen wiederum ein Mehrbenutzerbetrieb problemlos möglich, zum anderen können auch große Datenmengen effizient verarbeitet und verknüpft werden. Zusätzlich ist bei geeigneter Wahl des DBMS das ACID-Prinzip erfüllt, folgende Eigenschaften sind also erfüllt:

- Abgeschlossenheit
- Konsistenzerhaltung
- Isolation
- Dauerhaftigkeit

Die Anbindung in der Weboberfläche und das DBMS selbst sollen hierbei austauschbar sein.

3.2.3. Aufgabenplaner

Aufgaben wie beispielsweise eine Auswertung von Daten, welche in Anforderung 3.2.6 erfasst werden, sollen nicht direkt ausgeführt werden, sondern über einen Aufgabenplaner verwaltet werden.

Dabei wird jede auszuführende Aufgabe als Job in eine Datenbank gespeichert und von einem Hintergrundprozess ausgeführt. Es lässt sich über die Konfiguration einstellen, wie viel Hintergrundprozesse gleichzeitig gestartet werden sollen. Es sollen bis zu vier Hintergrundprozesse gleichzeitig möglich sein. Der aktuelle Zustand des Jobs wird ebenfalls in einer Datenbank gespeichert.

Laufende Aufgaben können abgebrochen werden, dabei werden die Ergebnisse verworfen. Noch nicht gestartete Aufgaben können gelöscht werden.

3.2.4. Wahl des Scan-Bereichs

Der Benutzer soll wählen können, welche Bereiche im Netzwerk gescannt werden sollen. So kann er direkt eine oder mehrere Zieladressen angeben. Zusätzlich soll es möglich sein, eine oder mehrere Subnetze zu scannen, hierfür soll als Notation CIDR (Classless Inter-Domain Routing) verwendet werden. Es sollen also folgende Arten von Eingaben möglich sein:

- einzelne IP-Adresse: 10.0.0.1
- mehrere IP-Adressen: 10.0.0.1, 10.0.0.2
- IP-Bereiche: 10.0.0.1/24
- Kombination davon: 10.0.1.0,10.0.0.1/24

Die Wahl des Scan-Bereichs soll sowohl für die aktive (3.2.7) als auch für die passive Erkennung von Systemen und Diensten (3.2.5) gelten. Bei einer Realisierung der Anforderung 3.2.11, der IPv6-Unterstützung, soll es standardmäßig nicht möglich sein, IPv6-Bereiche bei einer aktiven Erkennungsmethode anzugeben, da der Adressbereich in IPv6 schnell sehr groß sein kann.

3.2.5. Passive Erkennung von Systemen und Diensten

Da die Analyse nicht nur in Testnetzen, sondern in einem nächsten Schritt auch in Live-Netzen erfolgen soll, liegt der Schwerpunkt dieser Arbeit auf der passiven Erkennung von Systemen und Diensten. Bei einer aktiven Erkennung mit beispielsweise einem Schwachstellenscanner könnte andernfalls eine Störung der Hostsysteme, aber auch eine Beeinflussung der Netzqualität nicht ausgeschlossen werden. Deshalb sollen die beiden bereits vorgestellten passiven Erkennungsmethoden zum Einsatz kommen, die Netflow-Technik und die DPI.

In einem ersten Schritt wird das weniger aufwändige Verfahren der Netflow-Technik ausgeführt, wobei wie im ersten Teil der Arbeit erläutert, lediglich die Verbindungsdaten ohne die Inhalte der Pakete analysiert werden. Ist eine klare Zuordnung mithilfe dieser Erkennungsmethode nicht möglich, kann der Benutzer zusätzlich auswählen, dass in einem weiteren Schritt eine DPI vorgenommen wird.

Bei beiden Erkennungsmethoden müssen die auszuwertenden Daten bereits vorliegen. Ziel dieser Anforderung ist explizit nicht die Beschaffung besagter Daten, diese werden in Anforderung 3.2.6 erfasst.

3.2.6. Erfassung von Daten

Die für die Analyse notwendigen Rohdaten müssen insbesondere für die Anforderung 3.2.5 bereitgestellt werden.

Hierfür soll es drei verschiedene Varianten geben:

Dateisystem Die Daten sollen über das Dateisystem bereitgestellt werden können. Es kann ein Pfad angegeben werden, auf dem diese Dateien liegen. Es kann der Pfad zu einer bestimmten Datei angegeben werden, aber auch ein Ordner, in dem automatisch alle Dateien ausgewertet werden.

Dateiupload Der Benutzer soll zusätzlich die Möglichkeit haben, eine oder mehrere Dateien über einen Dateiupload zu übermitteln. Dies geschieht über die in Anforderung 3.2.1 definierte Weboberfläche.

Eigenständige Aufzeichnung Das Programm soll zusätzlich selbständig in der Lage sein, auf Kommando eine Aufzeichnung in einem bestimmten Zeitraum durchzuführen und diese aufgezeichneten Rohdaten für eine spätere Auswertung bereitzustellen.

3.2.7. Aktive Erkennung zur Kontrolle

Zur Überprüfung der Ergebnisse soll es möglich sein, optional einen Portscan über das vorgestellte Werkzeug Nmap als aktive Erkennungsmethode starten zu können. Dabei ist der Aufruf bereits mit den notwendigen Parametern vorkonfiguriert und die Ergebnisse werden dem anfragenden Benutzer direkt dargestellt.

Zusätzlich soll es dem Benutzer möglich sein, zusätzliche Parameter für den Aufruf von Nmap übergeben zu können.

3.2.8. Regelmäßiges Scanning

Benutzer führen regelmäßig anstehende Aufgaben nicht oder nicht lückenlos aus, sodass es zusätzlich möglich sein soll, Scan-Vorgänge wiederkehrend ausführen zu lassen. Eine Aufgabe kann folglich auch, wenn gewünscht, als sich automatisch wiederholende Aufgabe angelegt werden. Dabei kann aus einem Intervall gewählt werden, z.B. täglich, wöchentlich oder monatlich, und zusätzlich ein Startdatum gesetzt werden. Ist kein Startdatum gesetzt, wird das aktuelle Datum als Startdatum angenommen.

Außerdem kann ein Enddatum gesetzt werden, bis zu welchem Zeitpunkt die Aufgabe wiederholt wird. Ohne Enddatum wird die Aufgabe unendlich lange wiederholt.

Alle bekannten Optionen für einen einmaligen Scan in dieser Maske werden für alle zukünftigen Wiederholungen des Scans wiederverwendet.

Eine spätere Anpassung regelmäßiger Scans soll möglich sein.

3.2.9. Mandantenfähigkeit

Aufgrund der hohen Zahl an Netzverantwortlichen soll das System mandantenfähig sein. Es sollen dabei mehrere verschiedene Zugänge angelegt werden mit unterschiedlicher Berechtigung. Administratoren können Benutzer und andere Administratoren verwalten. Alle von einem Benutzer über die Weboberfläche gestarteten Vorgänge sollen aufgezeichnet werden und auf einer separaten Seite angezeigt werden können. Zusätzlich soll es möglich sein, den Scan-Bereich eines Nutzers einzuschränken. Somit erhalten Benutzer lediglich Ergebnisse von den Netzbereichen, für die sie eine Berechtigung besitzen.

Da das System Zugang zu sehr sensiblen Daten gewährt und je nach Ausbaustufe und Konfiguration der gesamte Datenverkehr im Netz aufgezeichnet werden kann, soll ein Zugangsschutz nur mittels einfachem Passwort nicht zulässig sein. Stattdessen soll der Zugriff auf das System durch zwei verschiedene Ansätze abgesichert werden. Zum einen sollen lediglich ausgewählte Clients überhaupt erst auf das System zugreifen dürfen. In dieser sogenannten Whitelist werden dann die zugriffsberechtigten Systeme hinterlegt, also beispielsweise ihre jeweilige Host-Adresse als IPv4 oder IPv6. Zum anderen sollte die Authentifizierung über ein Zertifikat-basiertes Verfahren durchgeführt werden.

3.2.10. Benachrichtigung per E-Mail

Sowohl für einmalige Scans als auch für das regelmäßige Scanning soll eine Benachrichtigung per E-Mail möglich sein, sobald die Aufgabe ausgeführt und die Analyse somit abgeschlossen ist. Der Benutzer kann konfigurieren, ob er die Ergebnisse direkt in der E-Mail erhalten möchte oder aus Gründen des Datenschutzes lediglich eine Benachrichtigung über die vorliegenden Ergebnisse gesandt werden. Im Fall der Übermittlung in der E-Mail, kann zusätzlich ausgewählt werden, ob diese als Plain-Text eingefügt oder angehängt werden sollen.

3.2.11. IPv6-Unterstützung

Neben dem klassischen IP-Protokoll IPv4 soll zusätzlich IPv6 unterstützt werden. Dies ist eine sehr wichtige Anforderung, da es am MWN bereits seit längerem eingesetzt wird. So sollen nicht nur direkte IPv6-Zieladressen angegeben werden können, sondern auch Subnetze. Der Benutzer muss hierbei nicht angeben, um welche Adressart (IPv4 oder IPv6) es sich handelt, sondern das soll automatisch festgestellt werden. Eine Kombination bei der Eingabe der Adressen soll möglich sein, es können also gemischt IPv4 und IPv6-Adressen eingegeben werden.

3.2.12. Kompatibilitätsprüfung der Internetprotokolltypen

In größeren Netzen wie dem MWN werden heutzutage die beiden Internetprotokolltypen IPv4 und IPv6 gleichzeitig eingesetzt. Dabei kann es allerdings zu Konfigurationsfehlern kommen, sodass ein System nur auf der IPv4 Adresse auf einen bestimmten Port hört, nicht aber auf der IPv6 Adresse auf dem selben Port. Dies ist beispielsweise bei einer fehlerhaften Virtual Host Konfiguration beim Apache Webserver möglich, wenn als Hostadresse nur die IPv4

Adresse eingetragen wird. Neben dem Punkt der Nichtverfügbarkeit, kann dies aber auch zum Vergessen von offenen Ports führen. Dies ist genau dann der Fall, wenn man lediglich eines der beiden Protokolle auf offenen Ports prüft und das jeweils andere vergisst.

Die Anforderung der Kompatibilitätsprüfung beschreibt, dass offene Ports gesucht werden, die lediglich über IPv4 erreichbar sind und nicht über IPv6 oder genau andersherum. Die betroffenen Systeme sollen dann samt der unterschiedlich konfigurierten Ports ausgegeben werden, sodass eine genauere, manuelle Nachprüfung möglich ist. Eine solche Konfiguration kann aber in manchen Fällen explizit auch so gewünscht bzw. technisch aufgrund mangelnder IPv6-Unterstützung notwendig sein.

3.3. Nichtfunktionale Anforderungen

Nach der Beschreibung der funktionalen Anforderungen werden im Folgenden die nichtfunktionalen Anforderungen erarbeitet.

3.3.1. Zuverlässigkeit

Das System soll auch während es in Betrieb ist, stets bedienbar bleiben. So müssen Vorgänge abgebrochen werden können und es soll ein Status über den aktuellen Zustand angezeigt werden. Durch die strikte Trennung der Steuerungs- und Ausführungsprozesse soll ein Fehlerfall im Ausführungsprozess (beispielsweise eine Netflow-Auswertung) keine Auswirkungen auf die Steuerung (also das Starten und Beenden von Aufgaben) haben.

3.3.2. Aufbau, Handhabung und Benutzbarkeit

Die Weboberfläche des beschriebenen Systems soll in allen aktuellen Browsern ohne Einschränkungen benutzbar sein und die einzige Zugriffsmöglichkeit auf das beschriebene System sein. Eine besondere Optimierung für mobile Geräte erfolgt nicht, da ein Zugriff für ungesicherte Geräte wie Smartphones und Tablet-Computer generell nicht möglich sein sollte (siehe hierzu Sicherheitsanforderungen 3.3.6).

Das System soll zudem intuitiv bedienbar sein. Das bedeutet, dass die grundlegenden Funktionen des beschriebenen Systems für den versierten Nutzer durch die Darstellung im Programm selbsterklärend sind.

3.3.3. Leistung und Effizienz

Das System hat als Priorität, On-Demand Rückmeldungen auf Anfragen zu liefern. Zeitintensive Aufgaben sollen in einen Aufgabenplaner (3.2.3) übernommen werden und durch einen Prozess-Scheduler im Hintergrund ausgeführt werden.

Die Auswertungen selbst sind in der Regel nicht zeitkritisch, da die Ergebnisse für präventive Maßnahmen, beispielsweise als Grundlage für Regeln in einer Firewall-Konfiguration, genutzt werden.

3.3.4. Betrieb und Umgebungsbedingungen

Serverseitig soll ein Linux-System zum Einsatz kommen, welches die Möglichkeit bietet, eine Interpretersprache wie PHP, Perl oder Ruby auszuführen. Für die Speicherung der Daten wird ein DBMS vorausgesetzt.

Die Wahl eines Linux-Systems als Betriebssystem-Architektur ist in der engen Verzahnung des beschriebenen Systems begründet. So sind einige obligatorisch notwendige Dienste wie Nmap oder flowd ursprünglich für Linux-Systeme entwickelt worden und häufig erst später oder gar nicht für andere Plattformen verfügbar gemacht worden. Zusätzlich ist die Verfügbarkeit eines Cron-Daemon für die zeitbasierte Ausführung von Prozessen notwendig. Für die optionale Benachrichtigung per E-Mail ist ein beliebiger aber korrekt konfigurierter Mailserver vonnöten.

Auf Seiten des Clients ist für die Benutzung der Weboberfläche ein moderner Webbrowser notwendig. JavaScript und Cookies müssen unterstützt werden und jeweils aktiviert sein.

3.3.5. Portierbarkeit und Übertragbarkeit

Das beschriebene System soll leicht von einem auf ein anderes Rechnersystem gleicher Architektur übertragen werden können. Prinzipiell ist ein plattformübergreifender Einsatz denkbar, aufgrund des primären Einsatzortes am LRZ wird aber der Schwerpunkt auf Linux-Systeme gesetzt.

Implementierungen von Erkennungsmethoden wie der Netflow-Technik sollen ausgetauscht und erweitert werden können. Die Anbindung zur Datenbank soll mittels einer objektrelationalen Abbildung abstrahiert werden, sodass die Datenbank ausgetauscht werden kann. Dabei sollen die gängigen Datenbankmanagementsysteme wie MySQL, PostgreSQL oder SQLite unterstützt werden.

3.3.6. Sicherheitsanforderungen

Sicherheit ist die für das beschriebene System die wichtigste nichtfunktionale Anforderung. Die Einschränkung des Zugriffs auf das physische oder virtuelle Rechnersystem obliegt dem jeweiligen Administrator. Aufgrund der erfassten Daten sollte die Absicherung des Rechnersystems allerdings oberste Priorität haben. Ein Angreifer könnte andernfalls im Worst-Case den gesamten Datenverkehr eines Netzes abhören und aufzeichnen.

Ein Zugriff auf die Weboberfläche des installierten Systems soll in einer Produktivumgebung nur geschützt möglich sein. Zusätzlich soll der gesamte Datenverkehr zwischen Benutzer und Weboberfläche SSL-verschlüsselt sein. Ein Zugriff über mobile Geräte sollte aus Sicherheitsgründen nicht möglich sein, die Absicherung obliegt allerdings dem Administrator, da er die zugriffsberechtigten Systeme verwaltet.

3.3.7. Korrektheit

Mittels geeigneter manueller Tests soll das beschriebene System auf eine korrekte Funktionsweise hin überprüft werden können. Ein formaler Nachweis über die Korrektheit kann aufgrund der Komplexität und der Nicht-Eindeutigkeit von Ergebnissen nicht geführt werden. So kann nie mit absoluter Sicherheit festgestellt werden, dass es sich bei einem System um beispielsweise ein Windows-System handelt - schließlich könnte es sich ja auch um ein anderes System handeln, das sich lediglich als solches ausgibt. Die Ergebnisse einer Erkennungsmethode sollen allerdings durch den Einsatz eines zweiten, tiefer gehenden Verfahrens verifizierbar sein. So können die Ergebnisse einer aktiven Erkennung über einen Portscan durch einen erweiterten Scan-Vorgang eines Vulnerability Scanners überprüft werden.

3.3.8. Skalierbarkeit

Das System soll keine Einschränkungen aufgrund einer größeren Eingabemenge von auszuwertenden Systemen besitzen. Die Verarbeitungsgeschwindigkeit soll allein von der zur Verfügung stehenden Hardware-Umgebung abhängen.

Prinzipiell ist aufgrund der Funktionalität des Aufgabenplaners auch ein Betrieb als verteiltes System möglich. So können die einzelnen, voneinander unabhängigen Aufgaben von verschiedenen Installationen parallel abgearbeitet werden.

3.4. Liste der Anforderungen

An dieser Stelle werden die beschriebenen funktionalen und nichtfunktionalen Anforderungen in tabellarischer Form dargestellt. Diese werden jeweils in der Spalte Priorität wie folgt gewichtet:

- 0: Anforderung wünschenswert
- +: Anforderung ist wichtig
- ++: Anforderung ist unbedingt notwendig, eine Nutzung andernfalls nicht möglich

Anforderung	Kurzbezeichnung	Priorität	Entwurf	Implementierung
3.2.1	Weboberfläche	++	4.1.1	5.1.1
3.2.2	Datenbank	++	4.1.2	5.1.2
3.2.3	Aufgabenplaner	++	4.1.3	5.1.3
3.2.4	Wahl des Scan-Bereichs	+	4.1.4	5.1.4
3.2.5	Passive Erkennung	++	4.1.5	5.1.5
3.2.6	Datenerfassung	++	4.1.6	5.1.6
3.2.7	Aktive Erkennung	0	4.1.7	-
3.2.8	Regelmäßiges Scanning	+	4.1.8	5.1.7
3.2.9	Mandantenfähigkeit	++	4.1.9	5.1.8
3.2.10	Benachrichtigungen	0	4.1.10	-
3.2.11	IPv6-Unterstützung	+	4.1.11	5.1.9
3.2.12	Kompatibilitätsprüfung	0	4.1.12	-

Abbildung 3.2.: Funktionale Anforderungen mit Referenzen

3.5. Themenverwandte Arbeiten

Die bislang umrissene Konzeption der vorliegenden Arbeit wird im Folgenden von solchen Arbeiten abgegrenzt, zu denen sie auf den ersten Blick Ähnlichkeiten aufweist. So behandelt die Publikation von [Kle12] eine ähnliche Thematik, sieht jedoch weder eine Mandantenfähigkeit noch eine IPv6-Unterstützung vor. Zudem erfolgt die Darstellung der Ergebnisse nicht über eine Weboberfläche und der Scan-Bereich lässt sich nur eingeschränkt begrenzen. Die Programme PRADS (vorgestellt in Abschnitt 2.3.2) und p0f[1] unterstützen zwar IPv6 und

[1] Urheber ist Michal Zalewski, offizielle Website: `http://lcamtuf.coredump.cx/p0f3/`

Anforderung	Kurzbezeichnung	Priorität	Entwurf	Implementierung
3.3.1	Zuverlässigkeit	+	4.2.1	5.2.1
3.3.2	Aufbau und Handhabung	+	4.2.2	5.2.2
3.3.3	Leistung und Effizienz	0	4.2.3	-
3.3.4	Betrieb und Umgebung	+	4.2.4	5.2.3
3.3.5	Portierbarkeit	+	4.2.5	5.2.4
3.3.6	Sicherheitsanforderungen	++	4.2.6	5.2.5
3.3.7	Korrektheit	++	4.2.7	5.2.6
3.3.8	Skalierbarkeit	0	4.2.8	-

Abbildung 3.3.: Nichtfunktionale Anforderungen mit Referenzen

haben starke Filtermöglichkeiten. Sie werden allerdings als lokale Instanzen betrieben und können eine Aufzeichnung des gesamten Netzverkehrs, anders als bei der Netflow-Technik, nicht garantieren. Deswegen und nicht zuletzt aufgrund des Anwendungsszenarios sind die beiden genannten Werkzeuge für die gestellten Anforderungen ungeeignet.

4. Entwurf

Im Kapitel zuvor wurden die Anforderungen an das zu erstellende System beschrieben. Dieses Kapitel beschäftigt sich mit dem Entwurf der Implementierung, dementsprechend sollen die funktionalen (Tabelle 3.2) und nicht-funktionalen Anforderungen (Tabelle 3.3) erfüllt werden. Dabei werden mögliche Umsetzungen vorgestellt und diskutiert. Anschließend erfolgt eine Bewertung nach Aufwand und Nutzen. Als Ergebnis werden dann die Anforderungen festgelegt, welche im nachfolgenden Kapitel 5 implementiert werden sollen.

4.1. Funktionale Anforderungen

Die funktionalen Anforderungen treffen Aussagen über die zu erfüllenden Eigenschaften des beschriebenen Systems. Im Folgenden wird für jede funktionale Anforderung ein Entwurf erarbeitet.

4.1.1. Weboberfläche

Das beschriebene System soll über eine Weboberfläche bedienbar sein, welche auf dem klassischen Client-Server-Modell aufbaut (siehe Abbildung 4.1). Sie kann technisch unterschiedlich konzipiert werden. So ist eine rein statische Darstellung möglich, wobei jeder Aufruf die gesamte Website neu lädt. Seit einigen Jahren setzt sich allerdings unter dem bekannten Schlagwort Web 2.0 die dynamische Weboberfläche mehr und mehr durch. Damit einhergehend hat die Verbreitung von dynamischen Websites stark zugenommen, sodass alle bekannten Internetdienste auf dieses Konzept setzen. Im Folgenden werden die einzelnen Komponenten vorgestellt, welche für eine solche Oberfläche benötigt werden und anschließend ein Entwurf, wie eine derartige Oberfläche aussehen kann, konzipiert.

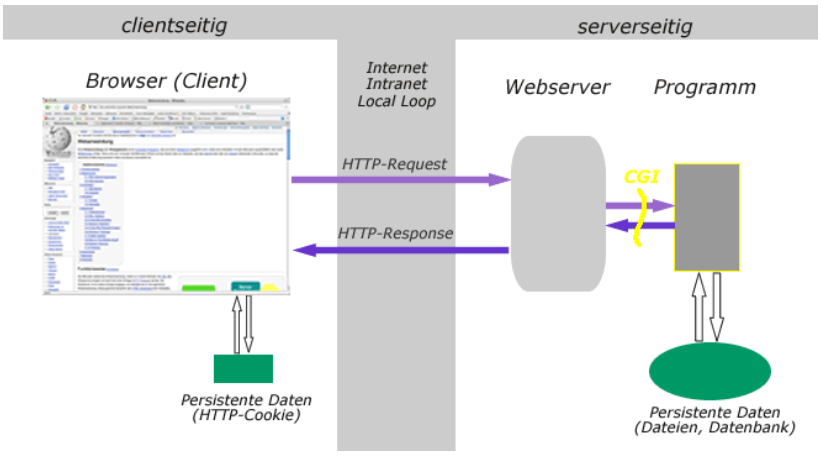


Abbildung 4.1.: Weboberfläche im Client-Server-Modell, Quelle: [Fra06]

Browser An erster Stelle steht der Internetbrowser des Benutzers als Client. Dieser stellt eine erste Anfrage an den Server und erhält das Grundgerüst der Website vom Server zugestellt. Der Browser verarbeitet heutzutage nicht nur das vom Server gesandte HTML (Hypertext Markup Language), sondern kann auch eine vollumfängliche Plattform für eine RIA (Rich Internet Application) anbieten, die eine vollwertige Desktop-Anwendung im Browser als Ziel hat. Hierbei kamen zunächst Plugin-basierte Technologien wie Java, Flash oder Silverlight zum Einsatz. In den letzten Jahren haben sich aber aufgrund der Plattformunabhängigkeit Lösungen durchgesetzt, die auf HTML in Kombination mit JavaScript basieren, beispielsweise Google Web Toolkit, jQuery oder Ext JS.

Webserver Der Server liefert die angefragten Daten an den Webbrowser. Zudem informiert er den Webbrowser mittels Push-Notifications auch bei Ereignissen, die den angemeldeten Benutzer in diesem Webbrowser betreffen könnten. Er identifiziert bei der zustandslosen Verbindung den Webbrowser jeweils über eine eigene Session-ID bzw. eigenen HTTP-Cookie. Die verwendete Skriptsprache ist hierbei prinzipiell austauschbar, es bietet sich jedoch an, eine am LRZ gängige Sprache wie Perl oder PHP einzusetzen.

Datenbank Die Datenbank stellt dem Server auf Anfrage Informationen zur Verfügung und dient gleichzeitig als persistenter Speicher. Die Datenbank wird in 4.1.2 genauer abgehandelt.

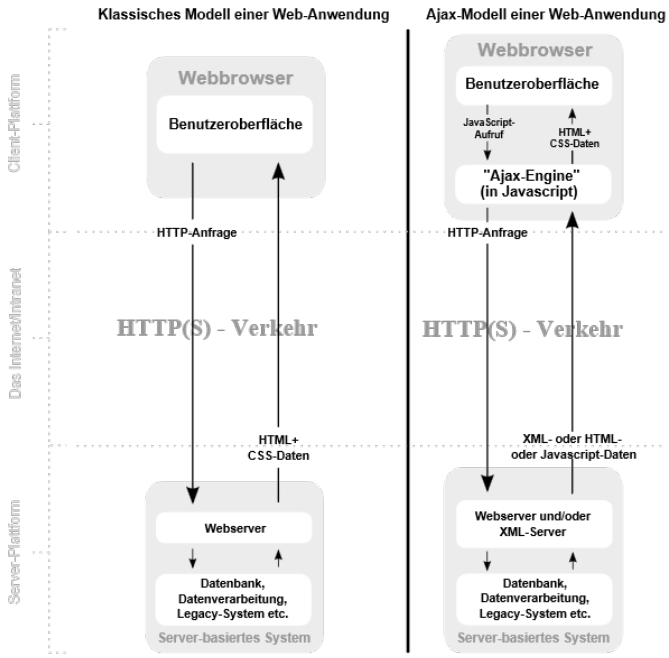


Abbildung 4.2.: Vergleich einer herkömmlichen Web-Anwendung mit Ajax Web-Anwendung, Quelle: [Dan07]

An den Webbrowser wird in einem ersten Schritt ein vollständiges Gerüst einer Website gesandt (Abbildung 4.3), welche die gesamte Funktionalität für die Anwendung mitliefert. Bei jedem nachfolgenden Aufruf wird mittels der AJAX (Asynchronous JavaScript and XML)-Technologie lediglich die angeforderte Programmkomponente angefragt (siehe Abbildung 4.2)

und nachgeladen. Diese Technologie erlaubt eine asynchrone Datenübertragung zwischen Browser und Webserver. Während eine HTML-Seite angezeigt wird, können HTTP-Anfragen und im Anschluss Seitenänderungen durchgeführt werden, ohne dass die Seite dabei selbst neu geladen wird. Praktisch realisiert werden kann diese Technologie mit einer JavaScript-Bibliothek. Wir werden die nach [Gel12] meistverwendete Bibliothek namens jQuery wählen, welche auf zwei Dritteln aller meistbesuchtesten Websites eingesetzt wird. Im nachfolgenden Kapitel 5 wird genauer auf die Implementierung eingegangen.

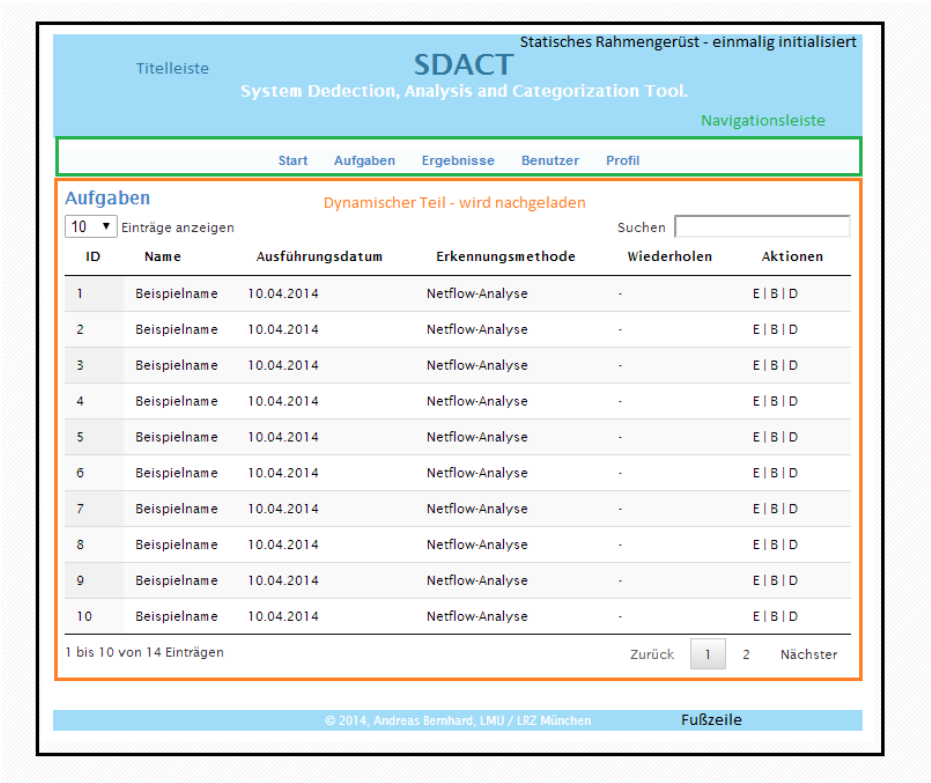


Abbildung 4.3.: Entwurf der Weboberfläche

Ein Ziel der Weboberfläche war deren einfache Bedienung, deshalb ist der Entwurf sehr schlicht gehalten und sieht lediglich drei Bereiche vor. Der erste Bereich ist das `Statische Rahmengerüst` mit der Titelleiste und der Beschreibung zum Tool. Zu diesem Rahmengerüst, das einmalig beim ersten Programmaufruf an den Browser gesandt wird, gehören außerdem noch der Bereich der `Navigationsleiste`, hier mit den verschiedenen Menüpunkten bereits angedeutet und die Fußzeile. Der eigentliche Seiteninhalt selbst wird in der Abbildung 4.3 als `Dynamischer Teil` gekennzeichnet ist. Dieser Teil wird bei den jeweiligen Aufrufen durch die jeweils geladene Komponente ersetzt.

Die Navigation umfasst für den Administrator fünf Punkte. Der einfache Benutzer hat dabei allerdings keinen Zugriff auf die Benutzerverwaltung. Die Berechtigungen verschiedener Benutzer werden im Abschnitt 4.1.9 genauer erläutert.

Start Über die Startmaske (Abbildung 4.4) gelangt ein Benutzer in das Programm. Von dort aus kann er die verschiedenen Aktionen durchführen.

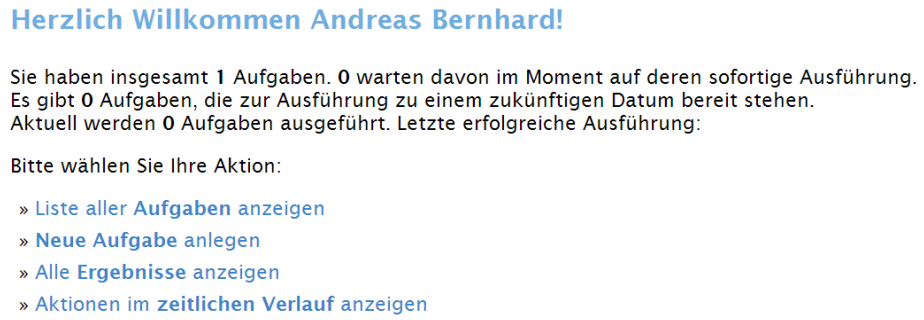


Abbildung 4.4.: Entwurf einer Startmaske für das System

Aufgaben Es können alle Aufgaben angezeigt werden, wofür der aktuelle Zugang eine Berechtigung besitzt. Ein Administrator sieht alle Aufgaben aller Mandanten. Über jeden Zugang können neue Aufgaben angelegt und anschließend verwaltet werden, auch ein Löschen ist möglich. Die Darstellung der Übersicht aller Aufgaben erfolgt in Tabellenform, wie in Abbildung 4.5 ersichtlich.

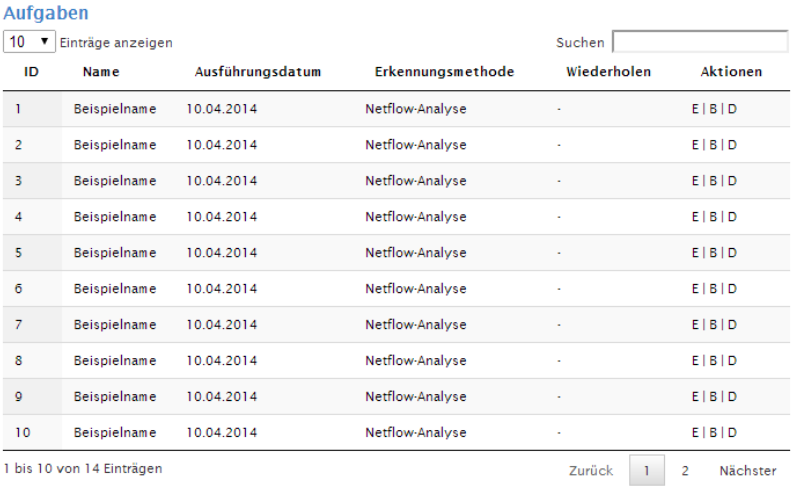


Abbildung 4.5.: Entwurf der Tabelle zur Verwaltung von Aufgaben

Ergebnisse Neben dem Aufruf der Ergebnisse zu einer konkreten Aufgabe, können auch alle bisher ausgeführten Aufgaben in eine Ergebnisübersicht zusammengefasst werden. So können alle jemals analysierten Systeme und der darauf laufenden Dienste betrachtet werden.

Benutzer Benutzer können von Administratoren in der Benutzerverwaltung angelegt, bearbeitet und gelöscht werden. Ausgeführt wird diese Funktionalität in 4.1.9.

Profil Im Profil kann der angemeldete Benutzer seine E-Mailadresse anpassen.

4.1.2. Datenbank

Die Datenbank dient dem zu erstellenden Softwaresystem als persistenter Datenspeicher. Dabei findet die Datenverarbeitung auf einer höheren Ebene statt als bei der Verwendung einfacher Dateien und es macht sie unabhängig von der verwendeten Software. Da die verschiedenen Tabellen in der geplanten Datenbasis miteinander in Beziehung stehen, ist eine klassische relationale Datenbank angedacht. Zudem können die in 3.2.2 geforderten ACID-Eigenschaften mittels Transaktionen ohne großen Mehraufwand realisiert werden.

Es sind für diese Anwendung verschiedene DBMS denkbar. So kann eine Datenbank mit einer einfachen SQLite-Bibliothek ohne zusätzlichen Serverdienst direkt in das Programm eingebunden werden. Der Nachteil hier ist, dass ein gleichzeitiger Zugriff verschiedener Programmteile, beispielsweise des zeitgesteuerten Aufgabenplaners und eines Benutzers auf der Weboberfläche nicht möglich ist, da bei einem Schreibzugriff jeweils der gesamte Datenbestand in der Datenbank gesperrt wird. Größere DBMS wie PostgreSQL oder MySQL oder der freien Implementierung davon, MariaDB, lösen dies durch zeilenbasierte Lese- und Schreibsperren.

Im ERM (Entity-Relationship-Modell), welches in Abbildung 4.6 dargestellt ist, wird davon ausgegangen, dass alle im Entwurf vorgestellten Anforderungen realisiert werden.

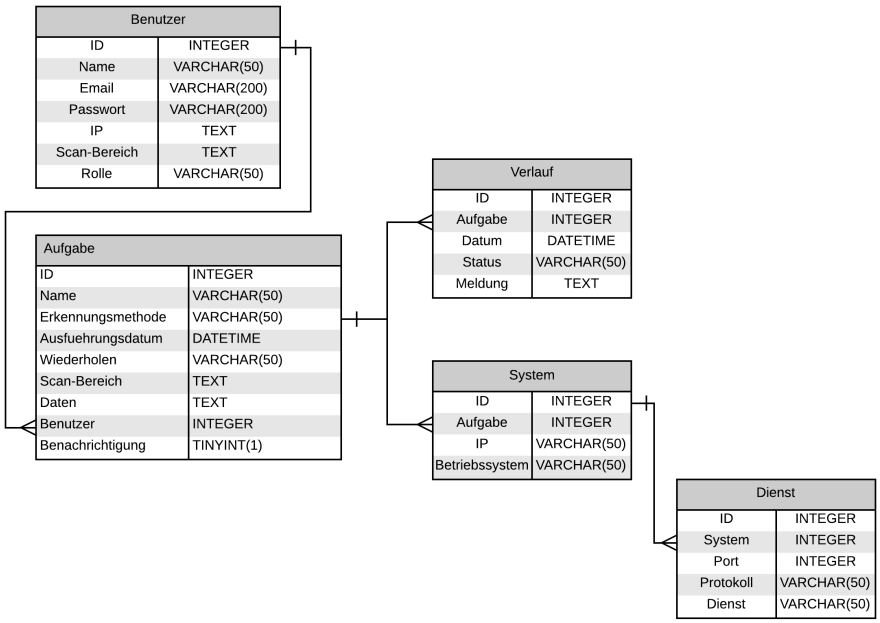


Abbildung 4.6.: Entwurf eines ER-Modell der Anwendungsobjekte

Die Relation `Benutzer` beinhaltet alle im System angelegten Benutzer, welche eindeutig über eine `ID` gekennzeichnet sind. Jeder Benutzer kann beliebig viele Aufgaben (Relation `Aufgabe`) anlegen, die einem einzigen Benutzer eindeutig zugeordnet sind. Ein Benutzer hat einen Namen (`Name`) und eine `E-Mail`, welche gleichzeitig zur Identifizierung dient. Zur Anmeldung muss der Benutzer zusätzlich ein Passwort eingeben, welches als Hash in der Spalte `Passwort` gespeichert wird. Der Zugang ist dabei nicht von jedem beliebigen System aus möglich, sondern nur von Systemen, deren IPv4- bzw. IPv6-Adresse im Feld `IP` hinterlegt

ist. Es gibt zusätzlich verschiedene Benutzertypen, welcher jeweils pro Benutzer im Feld `Rolle` gespeichert wird. Der für einen Mandanten erlaubte Scan-Bereich wird für jeden Benutzer separat in der Spalte `Scan-Bereich` definiert.

Jede Aufgabe ist wiederum eindeutig über eine `ID` definiert und hat verschiedene Attribute, welche im Aufgabenplaner in 4.1.3 ausführlich erläutert sind. Das Attribut `Benachrichtigung` bei einer Aufgabe kann den Wert 1 oder 0 annehmen, je nach dem ob der Benutzer nach erfolgreicher Ausführung der Aufgabe eine E-Mail-Benachrichtigung erhalten soll oder nicht.

Eine Aufgabe kann während ihrer Lebensdauer zwischen unterschiedlichen Status wechseln. Diese Übergänge werden jeweils in der Relation `Verlauf` erfasst. Jeder Eintrag im Verlauf ist durch eine eindeutige `ID` gekennzeichnet und einer Aufgabe zugewiesen. Zusätzlich wird das aktuelle Datum mit Uhrzeit erfasst und der neue Status, welcher durch eine detaillierte Meldung - beispielsweise im Fehlerfall - ergänzt werden kann. Auch der Pfad zu den Eingabedaten einer Aufgabe wird jeweils gespeichert.

Nachdem eine Aufgabe ausgeführt wurde, gibt es im Regelfall Ergebnisse. Diese Ergebnisse bestehen aus einer Liste von erkannten Systemen, welche pro System einen Eintrag in der Relation `System` erhalten. Dabei werden die Ergebnisse - also die Systeme - pro Aufgabe separat gespeichert, damit man Systeme auch über den Zeitverlauf beobachten und so Änderungen feststellen kann. Deshalb fungiert die `IP` nicht als Primärschlüssel, sondern nur als Attribut, da es mehrere verschiedene Einträge zu einem System geben kann. Jedes `System` ist also einer `Aufgabe` zugeordnet und als weiteres Attribut wird das erkannte Betriebssystem gespeichert.

Zu jedem System können beliebig viele Dienste (Relation `Dienst`) gespeichert werden. Diese sind eindeutig einem System zugewiesen und haben als Attribute den Port, das eingesetzte Protokoll und den erkannten Dienstnamen. Wie das `System` wird auch der `Dienst` über eine eindeutige `ID` identifiziert.

4.1.3. Aufgabenplaner

Der Aufgabenplaner beinhaltet zwei zu realisierende Bereiche. Zum einen müssen Aufgaben über die Weboberfläche angezeigt und verwaltet werden können und zum anderen zeitgesteuert abgearbeitet werden.

Aufgabenplanung

Über die Weboberfläche kann die Aufgabenplanung über den Navigationspunkt Aufgabenplanung erreicht werden. Auf der Übersicht können alle erstellten Aufgaben mit ihrem Namen, Beschreibung und aktuellem Status angezeigt werden. Dabei gibt es für jede Aufgabe folgende Funktionalität:

- Bearbeiten: Erlaubt das Ändern von Eigenschaften der Aufgabe. Nur zulässig, wenn Aufgabe aktuell nicht ausgeführt wird.
- Löschen: Erlaubt das Löschen einer Aufgabe. Es erscheint eine Sicherheitsnachfrage, falls Aufgabe aktuell ausgeführt und Aufgabe erzwungenermaßen gelöscht werden soll (beispielsweise im Fehlerfall).
- Duplizieren: Aufgabe wird mit gleichen Eigenschaften kopiert, aber der Status wird auf `Neu` gesetzt. Zusätzlich erhält die duplizierte Aufgabe eine neue eindeutige ID und der

Name der duplizierten Aufgabe wird in der Form erweitert, dass `Aufgabe` zu `Aufgabe (dupliziert)` wird. Die Namensänderung erfolgt zur besseren Unterscheidung der neuen von der bisherigen Aufgabe.

- Ergebnisse: Können lediglich dann angezeigt werden, wenn Aufgabe erfolgreich ausgeführt wurde.

Zusätzlich gibt es auf der Übersichtsmaske eine Verknüpfung zum Anlegen einer neuen Aufgabe. Jede Aufgabe gehört einem Benutzer und setzt sich aus verschiedenen Eigenschaften zusammen, welche im Folgenden vorgestellt werden.

ID Eine eindeutige Kennzeichnung der Aufgabe. Wird als Referenz bei der Verwaltung und in der Datenbank genutzt und kann vom Benutzer nicht verändert werden.

Name Eine Bezeichnung für die Aufgabe, die der Benutzer frei festlegen kann.

Erkennungsmethode Wählt das auszuführende Verfahren aus. Die Auswahl ergibt sich aus den später vorgestellten und implementierten Erkennungsmethoden.

Ausführungsdatum Wann die Aufgabe ausgeführt werden soll. Der Wert wird automatisch auf den aktuellen Zeitpunkt gesetzt, kann jedoch beliebig geändert werden.

Eingabe Die zu verarbeitenden Daten in Form eines Ordnerpfades, welche in der Anforderung Datenerfassung 4.1.6 genauer beschrieben werden.

Status Der aktuelle Status. Standardmäßig für neue und duplizierte Aufgaben wird der Status `Neu` gesetzt. Folgende Status sind vorgesehen:

- `Neu`: Für alle angelegten Aufgaben, die noch nicht ausgeführt werden sollen.
- `Warten auf Ausführung`: Aufgabe ist vom Benutzer freigegeben zur Ausführung und wird zum nächsten Zeitpunkt von einer Instanz des ausführenden Aufgabenplaners (siehe 4.1.3) bearbeitet.
- `Ausführung`: Aufgabe wird aktuell ausgeführt und kann somit nicht bearbeitet werden. In diesem Zustand werden die übergebenen Daten analysiert und ausgewertet.
- `Erledigt`: Aufgabe wurde erfolgreich ausgeführt und Ergebnisse können angezeigt werden.
- `Fehler`: Ausführung der Aufgabe war nicht erfolgreich. Dabei wird zusätzlich eine Fehlermeldung erfasst.

Verlauf Die verschiedenen Daten, wann die Aufgabe angelegt, zur Bearbeitung freigegeben und erledigt bzw. nicht erfolgreich abgearbeitet werden konnte, werden für einen nachvollziehbaren zeitlichen Verlauf automatisch gespeichert und angezeigt. Diese können vom Benutzer nicht verändert werden.

Erledigung der Aufgaben

Das System hat die anstehenden Aufgaben zeitgesteuert abzuarbeiten. Hierfür stehen je nach zu Grunde liegendem Betriebssystem verschiedene Hilfsmittel zur Verfügung. Auf Windows-Systemen gibt es die Aufgabenplanung (taskschd.msc), bei der über eine GUI (Graphical User Interface) verschiedene wiederkehrende Aufgaben, beispielsweise ein Aufruf eines bestimmten Programms, angelegt werden können. Da wir uns aber aufgrund verschiedener Gründe, welche in den Abschnitten zur Portierbarkeit in 3.3.5 und 4.2.5 erläutert wurden und werden, auf eine Linux-Umgebung festgelegt haben, wird genauer auf den dort verfügbaren Cron-Daemon eingegangen.

Der Cron-Daemon dient auf Betriebssystemen, die auf Unix basieren, beispielsweise Linux, BSD oder Mac OS X, zur zeitbasierten und wiederkehrenden Ausführung von Prozessen. Ein auszuführender Prozess wird dabei in einem sogenannten Crontab gespeichert, dabei stellt jedes Betriebssystem seine eigenen Bordmittel zur Verfügung, wie diese verwaltet werden können. Jede Zeile in einer Crontab-Datei folgt aber einem festen Muster. Die ersten fünf Spalten definieren gemeinsam den Zeitpunkt, wann der Prozess ausgeführt werden soll und die sechste Spalte definiert den Pfad zum auszuführenden Prozess.

Quellcode 4.1: Exemplarische Crontab-Datei auf Linux-System

```
1  #-------------------------------------------------------------
2  # Shell variable for cron
3  SHELL=/bin/bash
4  # PATH variable for cron
5  PATH=/usr/local/bin:/usr/local/sbin:/sbin:/usr/sbin:/bin:/usr/bin
6  # M S T M W   Befehl
7  #-------------------------------------------------------------
8  * * * * *   /home/user/script_minutely.sh
9  */10 *    * * *   /usr/bin/script_ten_minute.sh
10 30 3 * * 0,4 /home/user/script_nightly.sh
11 #-------------------------------------------------------------
```

Dabei stehen die ersten fünf Spalten für:

1. die Minute (0-59)
2. die Stunde (0-23)
3. den Tag (1-31)
4. den Monat (1-12)
5. den Wochentag (0-6)

In der ersten Zeile des Beispiels wird das Skript `/home/user/script_minutely.sh` jede Minute ausgeführt. Der Stern steht also für einen beliebigen Wert und generiert keine Einschränkung. In der zweiten Zeile wird das Skript alle zehn Minuten ausgeführt, dies resultiert aus `*/10`, also führe das Skript aus in jeder Minute, die durch 10 teilbar ist - inklusive der 0. Das letzte Beispiel wird jeden Sonntag und Donnerstag um 3:30 Uhr nachts ausgeführt.

Der eigentliche Prozess wird also über den Cron-Daemon aufgerufen und führt dann die definierte Aufgabe, beispielsweise die Netflow-Auswertung (beschrieben in Anforderung 3.2.5) einer hochgeladenen Datei (beschrieben in Anforderung 3.2.6), durch. Ein Filterkriterium, das bei der Ausführung einer Aufgabe durch den Aufgabenplaner mit übermittelt werden soll,

kann die Wahl des Scan-Bereichs sein (beschrieben in Anforderung 3.2.4). Direkt nachdem der Aufgabenplaner die Ausführung einer Aufgabe übernommen hat, markiert er den Status von `Warten auf Ausführung` auf `Ausführung`. Eine Bearbeitung durch eine mögliche andere Instanz des Aufgabenplaners ist dadurch nicht möglich. Nachdem eine Aufgabe erfolgreich abgeschlossen wurde, wird der Status auf `Erledigt` gesetzt. Im Fehlerfall kann der Status `Fehler` gesetzt werden und optional eine Fehlermeldung hinterlegt werden.

4.1.4. Wahl des Scan-Bereichs

Bei der Eingabe des Scan-Bereichs kann der Benutzer auswählen, welche Hostsysteme analysiert werden sollen. Diese Eingabe wird für jede Aufgabe (siehe 4.1.3) separat festgelegt.

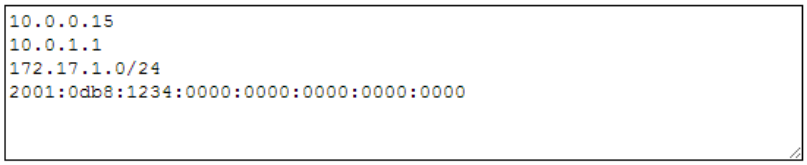


Abbildung 4.7.: Exemplarisches Eingabefeld für Scan-Bereich

Je Zeile des Eingabefeldes (Abbildung 4.7) kann dabei eine der folgenden Angaben gemacht werden:

- einzelne IPv4 Adresse
- einzelne IPv6 Adresse
- IPv4-Adressbereich in CIDR-Notation (Aktiv- und Passiverkennung)
- IPv6-Adressbereich in CIDR-Notation (nur Passiverkennung)

Eine Eingabe eines IPv6-Bereichs ist bei einer aktiven Erkennungsmethode allerdings wie in der Anforderung bereits erwähnt explizit ausgeschlossen. Andernfalls könnte bei einer unvorsichtigen Angabe des Scan-Bereiches eine Flut von Anfragen generiert werden, da der IPv6-Adressbereich sehr groß ist.

Falls der Benutzer keine Eingabe tätigt, werden alle Hostsysteme analysiert, was genauer in Abschnitt 4.1.5 beschrieben wird.

4.1.5. Passive Erkennung von Systemen und Diensten

Der Aufgabenplaner (siehe 4.1.3) stellt die zu analysierenden Systeme in Form einer Liste von IP-Adressen bereit. Ist die Liste der zu analysierenden Systeme leer oder nicht gegeben, werden alle aufgezeichneten Hosts analysiert. Er liefert auch die Information, welche Erkennungsmethode - also beispielsweise die Netflow-Analyse - ausgeführt werden soll. Zusätzlich liegen zum Startzeitpunkt der Passiverkennung bereits die auszuwertenden Daten vor, das kann zum Beispiel eine Datei mit Flow-Einträgen sein. Diese Daten werden beim Anlegen der Aufgabe in 4.1.6 erfasst.

Netflow-Technik

Für die Netflow-Analyse stehen verschiedene Tools zur Verfügung. Das bekannteste und zugleich noch frei verfügbare Tool heißt Nfdump und besteht aus einem Set von mehreren Werkzeugen. Das Zusammenspiel der wichtigsten Werkzeuge wird in Abbildung 4.8 dargestellt.

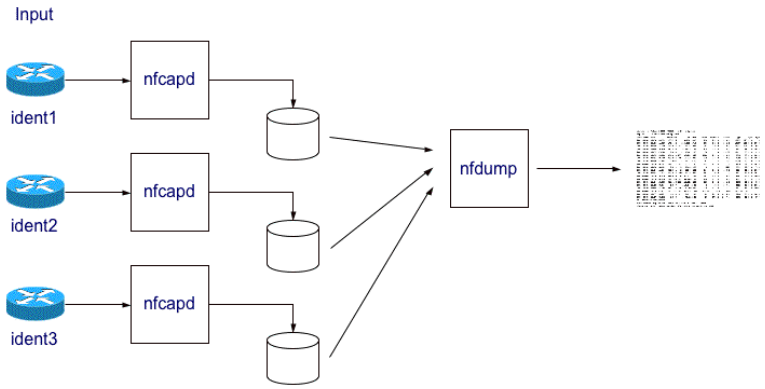


Abbildung 4.8.: Beschreibung der verschiedenen Nfdump-Werkzeuge, Quelle: [nfd13]

nfcapd Ist als Daemon für die Aufzeichnung von Netflows zuständig. Er liest die Daten, die von einem Netflow Exporter gesandt werden und speichert sie im Dateisystem. Für jeden Netflow-Exporter ist ein eigener nfcapd-Prozess notwendig.

nfdump Liest die Netflow-Daten aus den Dateien und zeigt sie dem Benutzer an.

nfprofile Liest die Netflow-Daten aus den Dateien, wendet darauf vom Benutzer spezifizierte Filter an und speichert sie erneut.

Die Installation bzw. Konfiguration der verschiedenen Dienste wird im Anhang B.1 erläutert, die genaue Integration in das beschriebene System werden wir in 5.1.5 erarbeiten.

Bei der Netflow-Variante wertet das System die aufgezeichneten Netflow-Daten aus und gibt eine Liste aller erkannten Hosts zurück. Zusätzlich werden zu jedem Host die erkannten Dienste erfasst. Der Auswertungsprozess (Abbildung 4.9) wird in mehrere Schritte unterteilt, welche nacheinander durchgeführt werden:

Lesen von Netflow-Daten Das Einlesen der Netflow-Daten erfolgt wie in der Datenerfassung in 4.1.6 beschrieben über ein hybrides Modell. So werden die Daten entweder aus einem vom Benutzer angegebenen Ordnerverzeichnis und den darin enthaltenen Dateien oder aus einer zwischengespeicherten Datei geladen, die zuvor wiederum von einem Benutzer hochgeladen wurde. Wenn die Daten nicht geladen werden können, wird die Aufgabe abgebrochen und ein Fehlerstatus mit entsprechender Meldung `Fehler beim Laden der Netflow-Daten` gesetzt.

Anwenden der Filter Wie bei der Wahl vom Scanbereich in 4.1.4 beschrieben, können einzelne auszuwertende Host-Adressen angegeben werden. Es werden also alle Netflow-Zeilen entfernt, die weder bei der `Source`- noch bei der `Target`-IP den gewünschten Hosts entsprechen.

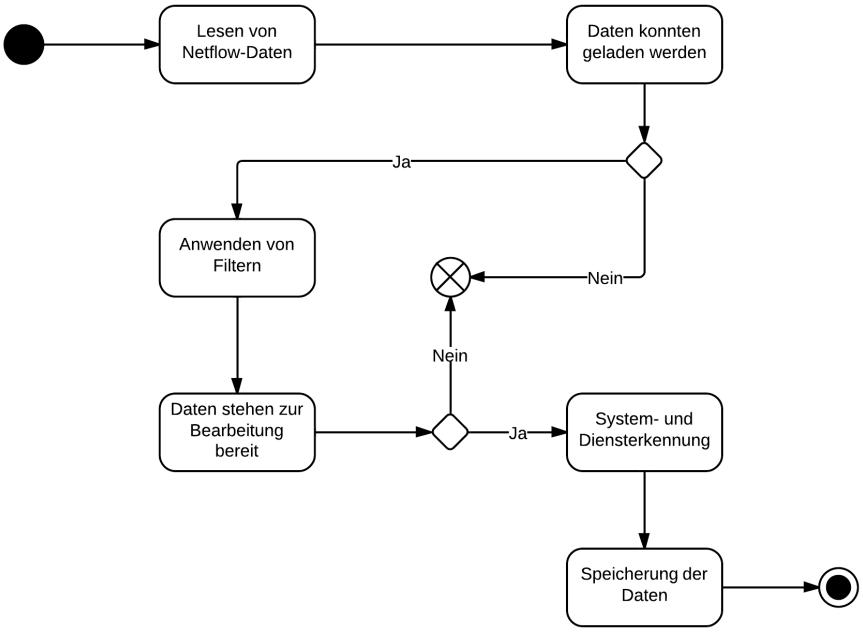


Abbildung 4.9.: Auswertungsprozess bei der passiven Netflow-Technik

Wenn kein Scan-Bereich angegeben ist, wird die nachfolgende Erkennung auf allen Systemen angewandt.

Erkennung Es wird die System- und Diensterkennungsmethode aufgerufen mit den in den vorherigen Schritten erfassten Netflow-Daten und den erstellten Filtern.

System- und Diensterkennung mit flow-dump

Die im Weiteren beschriebene Vorgehensweise zur Erkennung der Systeme und Dienste auf Grundlage der Netflow-Daten ist an den Protoypen `flow-dump` aus der Arbeit von [Kle12] angelehnt. Die grundlegende Arbeitsweise ist in der Abbildung 4.10 beschrieben. Als Eingabe wird eine Netflow-Datei vom oben beschriebenen nfcapd erwartet. Die verschiedenen Betriebsmodi und die für unsere Implementierung gesetzten Parameter werden bei der Implementierung in 5.1.5 näher erläutert.

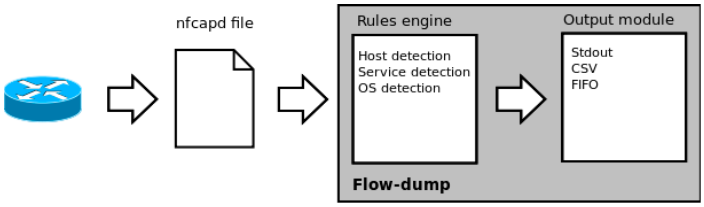


Abbildung 4.10.: Übersicht von flow-dump, Quelle: [Kle12]

Das Programm `flow-dump` nutzt verschiedene Verfahren um Hosts mit dem installierten

Betriebssystem und den darauf laufenden Diensten zu erkennen. Die einfachste Erkennung ist die Host-Erkennung, welche prüft, ob ein Host überhaupt existiert.

Hierfür werden ICMP (Internet Control Message Protocol) Echo Replies (Abbildung 4.11) ausgewertet, welche von einem Host zurückgesandt werden, um zu zeigen, dass das System online ist. Würde man lediglich die ICMP Echo Requests auswerten, hätte man zu viele `False Positives`, da die Zielsysteme nicht zwingend vorhanden sein müssen.

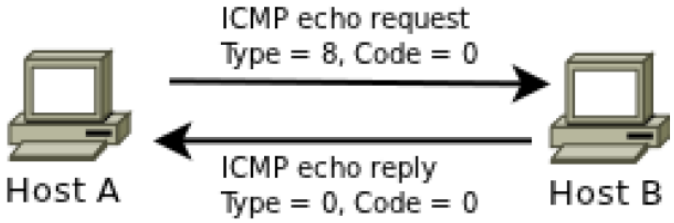


Abbildung 4.11.: ICMP echo reply, Quelle: [Kle12]

Neben Hosts sollen auch die darauf installierten Dienste erkannt werden. [Kle12] stellt fest, dass man sich dafür nicht auf die in den Netflows vorhandenen TCP-Flags verlassen kann. Er begründet dies nachvollziehbar damit, dass nicht der Netzwerkverkehr ausgewertet wird, sondern die Netflow-Einträge und bei dieser Variante die TCP-Flags zusammengefasst werden. Beispielsweise sendet ein Client zum Verbindungsaufbau ein `SYN` Paket und der Server antwortet mit einem `SYN/ACK` Paket. Wenn der Client dann das `ACK` Paket sendet, ist die Verbindung aufgebaut. Normalerweise würde man den Server dann schnell durch dessen `SYN/ACK` Antwort erkennen. Bei der Netflow-Technik werden diese Antworten allerdings zusammengefasst und es gibt zwei Pakete, eines vom Client zum Server mit `SYN/ACK` und eines vom Server zum Client mit `SYN/ACK`. Man kann Client und Server aber dann nicht mehr voneinander unterscheiden und deshalb kommt eine auf TCP-Flags basierende Lösung nicht in Frage.

Es gibt allerdings eine andere Methode, um zu prüfen, welche Dienste auf einem Host laufen. Dieses Verfahren, welches in in Abbildung 4.12 dargestellt ist, basiert darauf, dass man prüft auf welchen Ports entsprechende Systeme kommunizieren. Normalerweise sendet und empfängt ein Client von einem Nutzerport, also von einem Port größer gleich 1024. Um also zu prüfen, welche Dienste auf einem Host aktiv sind, wird geprüft, welche ausgehenden Ports unter 1024 aktiv sind.

Dabei gibt es allerdings zwei Probleme: Zum einen werden Dienste erkannt, welche `False Positives` sind, wie beispielsweise NFS (Network File System). NFS kommuniziert abwechselnd auf einer großen Zahl an verschiedenen Ports. Zum anderen werden Dienste nicht erkannt, welche eigentlich zu erkennen wären - sogenannte `False Negatives`. Dies liegt daran, dass manche Dienste, beispielsweise MySQL mit Port 3306, Ports nutzen, die größer als die oben gewählte Portnummer 1024 ist. Umgehen lässt sich das durch eine zusätzliche Definition von zu erkennenden Services, vergleichbar mit einer Whitelist.

Nach der Erkennung von Hosts und den darauf laufenden Diensten fehlt als letztes noch das Betriebssystem. Eine passive Erkennung eines Betriebssystem nur anhand von Netflow-Einträgen scheint zunächst nicht möglich zu sein, da die vorhandenen Daten auf den ersten Blick nicht ausreichend Informationen enthalten, um daraus ein bestimmtes Betriebssystem

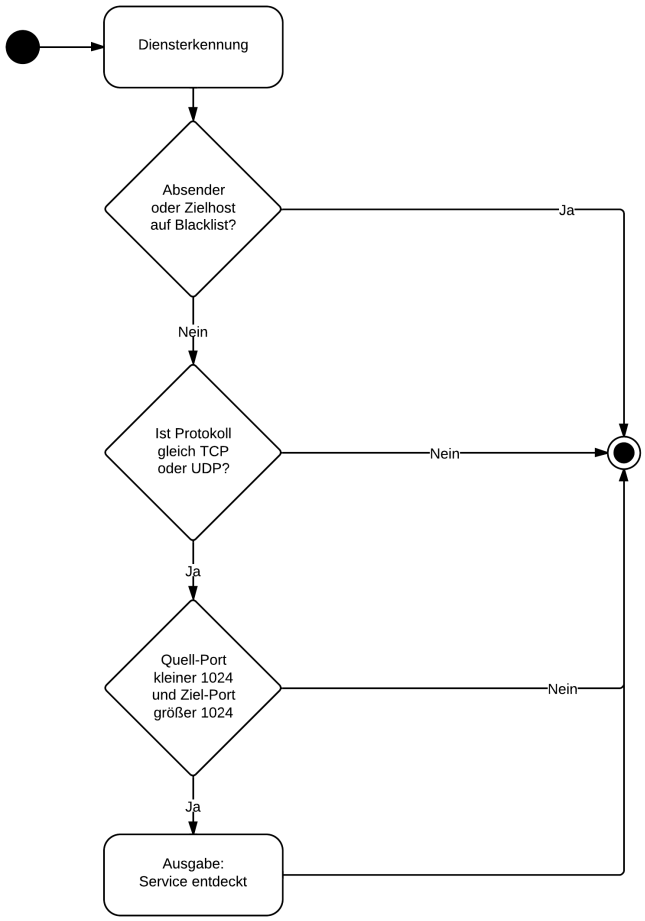


Abbildung 4.12.: Port-basierte Erkennung von Diensten, angelehnt an [Kle12]

abzuleiten. Es gibt nach [Kle12] allerdings eine Möglichkeit, das Betriebssystem über den Update-Mechanismus der jeweiligen Systeme herauszufinden: So prüfen praktisch alle Betriebssysteme stetig ob Updates auf einem Update-Server vorliegen. Dabei wird eine Verbindung zu einem Update-Server im Internet oder Intranet aufgebaut, welcher über eine bestimmte IP-Adresse im Internet erreichbar sind. Um keine Probleme mit Firewall-Regeln zu bekommen, erfolgt diese Anfrage in den allermeisten Fällen über Port 80. Diese Update-Anfragen werden durch die Netflow-Technik erfasst und es kann dann pro Host jeweils anhand der angefragten IP-Adresse bestimmt werden, welches Betriebssystem installiert ist.

Rückgabe und Speicherung der Daten Als Rückgabe von `flow-dump` erhält man eine CSV (Comma-Separated Values)-Datei wie beispielsweise in Quellcode 4.2 dargestellt. Diese werden für eine spätere Speicherung eingelesen, dabei werden Dienste ohne zugehörigen Host ignoriert. Zusätzlich wird zu den erkannten Ports der mögliche Dienst dahinter erfasst, dies erreicht man durch einen Abgleich mit der offiziellen Liste der standardisierten Ports[1]

[1]Die offizielle Liste der Ports ist abrufbar unter: `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`

der Internet Assigned Numbers Authority. Dieser Information ist allerdings keine zu große Bedeutung beizumessen, da sich bei dieser Art der Erkennung immer auch ein anderer Dienst hinter einem solchen Port verbergen kann.

Quellcode 4.2: Beispiel-Ausgabe von flow-dump als CSV

```
1 Service,143.220.7.45,445/TCP,2014-12-13 16:29:08
2 Service,143.220.3.3,53/UDP,2014-12-13 16:29:28
3 Host,143.220.13.7,2014-12-13 16:29:28
4 Host,143.220.240.101,2014-12-13 16:29:32
5 Service,143.220.4.40,53/UDP,2014-12-13 16:29:28
6 OS,143.220.4.5,RedHat,2014-12-13 16:29:35
7 Service,143.220.15.7,80/TCP,2014-12-13 16:29:48
8 Service,143.220.12.40,445/TCP,2014-12-13 16:29:2
9 Service,143.220.65.3,123/UDP,2014-12-13 16:29:40
10 Host,143.220.250.211,2014-12-13 16:29:32
```

Die Ergebnisse werden in die Datenbank geschrieben. Die Hosts jeweils in die Tabelle `System` und die zugehörigen Dienste als einzelner Eintrag in `Dienst`, jeweils verknüpft mit dem zugehörigen `System`. Dabei sind Duplikate in Form von doppelt vorkommenden Hosts (Tabelle `System`) ausdrücklich vorgesehen, da diese jeweils einer Aufgabe zugeordnet sind. So ist eine zeitliche Analyse später problemlos möglich.

Eigene System- und Diensterkennung

Ein Teil der beschriebenen Verfahren wurde von [Kle12] in seinem Prototyp `flow-dump` implementiert. Dieser basiert auf der Skriptsprache Perl und müsste für eine Realisierung mehrerer Anforderungen dieser Arbeit deutlich erweitert werden. Da dieser Prototyp nicht für grundlegende Erweiterungen wie der IPv6-Unterstützung (siehe Abschnitt 4.1.11) entworfen wurde, ist eine Anpassung schwierig. Zudem bedeutet eine Einbindung in die in der Skriptsprache PHP erstellte Arbeitsumgebung, dass zwei verschiedene Skriptsprachen im gleichen Programm zum Einsatz kommen würden.

Betriebssystemerkennung Einige Betriebssysteme führen die im vorherigen Abschnitt beschriebene Prüfung, ob Updates für das System vorliegen, nicht mehr ausschließlich über den Port 80 aus, sondern teilweise ausschließlich oder zusätzlich über den Port 443. Auch für diesen Zielport muss also pro Flow-Eintrag geprüft werden, ob es sich um eine Verbindung zu einem Update-Server eines Betriebssystemes handelt.

Deep Packet Inspection

Ein weiterer Ansatz zur passiven Erkennung von Diensten ist die DPI. Aufgrund der vollständig vorliegenden Paketdaten ist die Wahrscheinlichkeit der korrekten Diensterkennung bei dieser Methode grundsätzlich am höchsten. Das Problem der auf Netflow-basierenden Erkennungsmethoden ist, dass die Verfahren mangels Alternativen eine portbasierte Diensterkennung durchführen müssen. Wenn also ein Dienst von Port 80 sendet und Anfragen erhält, wird er typischerweise als Webserver identifiziert, da für gewöhnlich Webserver wie der Apache auf diesem Dienst laufen. Diese Einteilung hat allerdings die entscheidenden Schwächen, dass Dienste nicht zwingend auf einem fest vorgegebenen Port senden müssen. Ein Beispiel hierfür ist das Programm Skype, welches standardmäßig auf dem Port 80 sendet, um Firewall-Probleme zu umgehen.

Ein bekannter Vertreter im Bereich der DPI ist das Tool `nDPI`[2], das auf dem inzwischen nicht mehr weiterentwickelten `OpenDPI` basiert. Es ist in der Lage, den Netzverkehr live aufzuzeichnen und auszuwerten. Im Gegensatz zu `OpenDPI` erkennt `nDPI` auch immer mehr verschlüsselte Protokolle. Eine Erkennung von Diensten auf Basis von `nDPI` hat derselbe Hersteller in seine Software nProbe™ integriert und stellt diese in einer Basisversion kostenfrei zur Verfügung.

Da in diesem Entwurf keine Echtzeitverarbeitung der anfallenden Daten unterstützt wird (siehe Abschnitt 4.1.6), ist eine Integration nur schwerlich möglich. Eine Zwischenspeicherung und Verarbeitung aller Paketdaten ist bereits bei einem normalen Datenaufkommen in kleineren Netzwerken sehr komplex. Aus diesem Grund wird die Erkennungsmethode nicht angeboten.

4.1.6. Erfassung von Daten

Das Vorhandensein von Daten ist Grundlage für jede Analyse und jede Aufgabe muss dementsprechend Daten hinterlegt haben. In den Anforderungen wurden drei Varianten der Datenerfassung vorgestellt:

- Angabe von Dateipfad
- Datenupload über Weboberfläche
- Selbständiges Erfassen der Daten

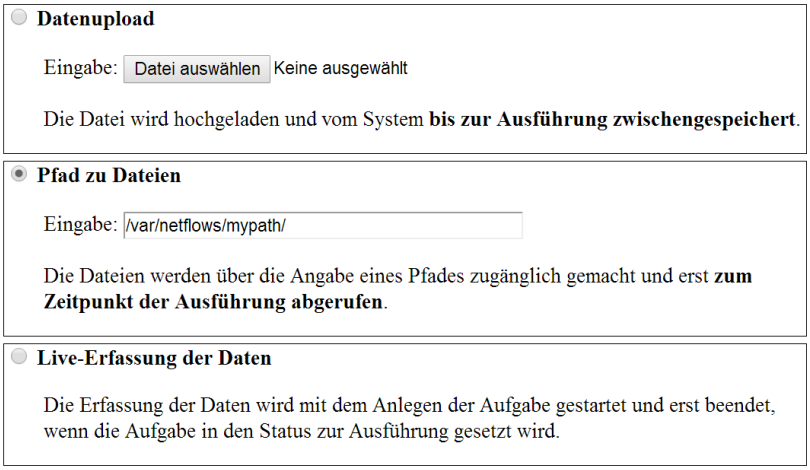


Abbildung 4.13.: Prototyp der verschiedenen Eingabemethoden

Alle drei Varianten haben gemein, dass sie schlussendlich die erfassten Daten dem Aufgabenplaner zur Verfügung stellen. Dabei gibt es zwei verschiedene Speichermöglichkeiten, welche Vor- und Nachteile haben.

Kopieren der Daten Zum einen können die Daten zentral und untergliedert nach der Aufgaben-`ID` gespeichert werden. So ist eine Speicherung im Verzeichnis */var/sdact* vorgesehen und für jede angelegte Aufgabe wird wiederum ein Unterordner erstellt. Gibt ein Benutzer

[2]Hersteller ist ntop, Website: `http://www.ntop.org/products/ndpi/`

also für die Aufgabe mit der `ID` 123 einen Ordnerpfad zu Analysedaten wie beispielsweise Netflow-Dateien an, werden die dort enthaltenen Dateien in den Ordner */var/sdact/*123 kopiert. Genauso verhält es sich, wenn es sich um einen Dateipfad handelt oder wenn der Benutzer eine Datei zu einer Aufgabe hochlädt. Der Lesezugriff auf das vom Benutzer angegebene Verzeichnis bzw. der Schreibzugriff für */var/sdact* muss für die Weboberfläche gewährleistet sein. Für wiederkehrende Aufgaben ist das einmalige Kopieren der Daten in den jeweiligen Aufgabenordner allerdings keine Option, da immer die gleichen Daten wieder und wieder ausgewertet werden würden. Zudem müssen die in die Aufgabenverzeichnisse kopierten Daten regelmäßig gelöscht werden, da andernfalls der Speicherbedarf stetig wächst.

Analyse der Pfadangabe Besonders für wiederkehrende Aufgaben ist es notwendig, dass beim erneuten Aufruf die jeweils aktuellen Daten analysiert werden und nicht die gleichen Daten wie bei der vorherigen Ausführung der Aufgabe. Deshalb werden bei dieser Variante keine Daten kopiert und es ist nur die Erfassung der Daten über einen Ordnerpfad möglich. Externe Programme können dann in diesem Ordnerpfad die jeweils aktuellen Logdateien vor der geplanten Analyse ablegen bzw. austauschen. Als Pfad ist auch ein Netzwerkpfad möglich. Voraussetzung hierfür ist aber der Lesezugriff für das System, von wo aus die Analyse gestartet wird.

Beide Varianten haben jeweils ihre Vor- und Nachteile. So müssen die Daten bei einem Datenupload irgendwohin verschoben werden um dort zum Ausführungszeitpunkt analysiert werden zu können. Bei der Pfadangabe hingegen sollten die Daten direkt von dort jeweils analysiert werden und zwar zu dem Zeitpunkt, wenn die Aufgabe tatsächlich ausgeführt wird. Deshalb wird ein hybrider Ansatz verfolgt, bei dem jeweils einer der beiden Ansätze gewählt wird, je nach dem wie die Dateien erfasst werden soll.

Die dritte und letzte Möglichkeit bei der Datenerfassung (siehe Live-Erfassung in 4.13) sieht vor, dass die Daten selbständig über einen vordefinierten Zeitraum aufgezeichnet werden. Diese Erfassung birgt allerdings einige Herausforderungen. Zum einen ist sie stark von der Umgebung abhängig. Es werden beispielsweise bei der Netflow-Technik auf Router-Ebene viele verschiedene Lösungen eingesetzt und eine Unterstützung vieler verschiedener Geräte ist komplex. Zum anderen werden die Daten häufig bereits regelmäßig erfasst, was die Anforderung in diesem Sinne hinfällig macht. Aus diesem Grund wird die Aufzeichnung der Daten im Rahmen dieser Arbeit nicht weiter verfolgt, ist aber als spätere Erweiterung weiterhin denkbar.

4.1.7. Aktive Erkennung zur Kontrolle

Eine Erkennung des Betriebssystem und den installierten Diensten ist in den meisten Fällen mit den aktiven Erkennungsmethoden problemlos möglich. So erkennt bereits ein einfacher Portscan durch OS-Fingerprinting das Betriebssystem. Des Weiteren lassen offene Ports in Kombination mit weiterführenden Tests auf den jeweiligen Ports auf die installierten Dienste schließen. Im nächsten Beispiel wird gezeigt, dass Nmap nicht nur in der Lage ist, klassische Systeme wie ein Windows- oder Linux-System zu erkennen.

Quellcode 4.3: Nmap-Scan mit Erkennung des Betriebssystems und der Dienste

```
1 nmap -sV -O 192.168.1.1
2
3 Starting Nmap 6.40 ( http://nmap.org ) at 2014-01-20 09:28 Mitteleuropaeische
    Zeit
```

```
Nmap scan report for 192.168.1.1
Host is up (0.00022s latency).
Not shown: 991 closed ports
PORT      STATE SERVICE     VERSION
23/tcp    open  telnet?
53/tcp    open  domain      dnsmasq 2.39
80/tcp    open  http        uhttpd 1.0.0 (Netgear WNDR3700v2 WAP http config)
139/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
3333/tcp  open  dec-notes?
5555/tcp  open  freeciv?

MAC Address: A0:21:B7:9F:45:30 (Netgear)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.27
Network Distance: 1 hop
Service Info: Device: WAP
```

Anhand der erhaltenen Daten ist es möglich, das System sehr genau zu kategorisieren. So gibt Nmap bereits die Information, dass es sich um einen WAP (Wireless Access Point) handelt. Zusätzlich wurden das Betriebssystem und die Dienste mit Angabe der Version korrekt erkannt.

Um die Ausgabedaten von Nmap weiterverarbeiten zu können, kann der Parameter `-oX file.xml` gesetzt werden. Nmap gibt dann die entdeckten Hosts und die erkannten Dienste als XML (Extensible Markup Language)-Datei zurück. Ein Beispiel hierfür ist die Erkennung des Systems `192.168.1.25` in Quellcode 4.4.

Quellcode 4.4: Beispiel-Ausgabe von Nmap in XML

```
<host starttime="1394111815" endtime="1394111855"><status state="up" reason="
    arp-response" reason_ttl="0"/>
<address addr="192.168.1.25" addrtype="ipv4"/>
<address addr="84:1B:5E:40:D8:4A" addrtype="mac" vendor="Netgear"/>
<hostnames>
</hostnames>
<ports><extraports state="closed" count="997">
<extrareasons reason="resets" count="997"/>
</extraports>
<port protocol="tcp" portid="23"><state state="open" reason="syn-ack"
    reason_ttl="64"/><service name="telnet" method="table" conf="3"/></port>
<port protocol="tcp" portid="80"><state state="open" reason="syn-ack"
    reason_ttl="64"/><service name="http" method="table" conf="3"/></port>
<port protocol="tcp" portid="1024"><state state="open" reason="syn-ack"
    reason_ttl="64"/><service name="kdm" method="table" conf="3"/></port>
</ports>
<times srtt="11124" rttvar="2564" to="100000"/>
</host>
```

Diese XML-Daten können dann in einem nächsten Schritt in die Datenbank gespeichert werden. Für das Einlesen der XML-Daten wird ein XML-Parser benötigt, welchen die Skriptsprache PHP mit der Erweiterung `SimpleXML` zur Verfügung stellt.

Der Vorteil an dieser aktiven Erkennungsmethode ist, dass sie relativ leicht und praktisch ohne Konfigurationsaufwand anwendbar sind. Jedoch ist ein Portscan und besonders ein Schwachstellenscan sehr zeitaufwändig und benötigt bei einer hohen Zahl an zu testenden Systemen viele Ressourcen.

4.1.8. Regelmäßiges Scanning

Bei einer Aufgabe kann zusätzlich eine weitere Eigenschaft angegeben werden, wodurch ein regelmäßiges Scanning ermöglicht wird. Diese Funktion ist allerdings wenig sinnvoll, wenn sie in Kombination mit einem Dateiupload genutzt wird. Da sich dieser nicht ändert, würde sonst stetig die gleiche Datei auf dem selben Scan-Bereich ausgewertet werden. Deshalb müssen für das regelmäßige Scanning die Daten zwingend über einen Dateipfad zur Verfügung gestellt werden, wie dies bei der Pfadangabe in 4.1.6 der Fall ist. Eine Beispieleingabe dafür wäre */var/netflowsources/*. Die Ergebnisse der jeweiligen Aufgabe können nach deren Ausführung über die Aufgabenliste aufgerufen werden.

Wiederholen Die Eigenschaft erlaubt die Angabe eines Intervalls. Zunächst wird dieses begrenzt auf täglich, wöchentlich und monatlich. Beim Erreichen eines nächsten Ausführungszeitpunktes einer Aufgabe, die wiederkehrend erstellt wurde, wird die Aufgabe dupliziert und der nächste Ausführungszeitpunktes anhand dieser Angabe berechnet.

4.1.9. Mandantenfähigkeit

Für die Mandantenfähigkeit ist eine Zugriffskontrolle unabdingbar, da nur autorisierte Benutzer Zugang zum System erhalten sollen. Die Zugriffskontrolle lässt sich in vier Schritte unterteilen:

Identifizierung Der Benutzer teilt dem System durch die Eingabe einer E-Mailadresse seine Identität mit.

Authentisierung Der Benutzer übergibt Informationen, die eine Verifizierung seiner Identität erlauben. Dabei kommt ein multimodales System zum Einsatz. Ein Zugang soll also nur möglich sein, wenn der Benutzer ein korrektes Passwort angibt (Wissen) und sich von einem bestimmten Host aus anmeldet (Eigenschaft).

Authentifizierung Das System prüft, ob die angegebenen Authentisierungsmerkmale zur Identität passen und erzeugt im Erfolgsfall eine Sitzung. Im Fehlerfall wird er zurück zur Anmeldung geschickt. Dabei werden die übermittelten Daten mit denen in der Datenbank gespeicherten Daten verglichen. Das Passwort soll dabei aus Sicherheitsgründen ausschließlich als Hash in Kombination mit einem Salt gespeichert werden. Dabei wird das Passwort vor der Verwendung als Eingabe einer Hashfunktion um eine zufällig gewählte Zeichenfolge ergänzt. Wörterbuchangriffe durch Rainbow Tables werden dadurch erschwert.

Autorisierung Das System prüft bei jedem Programmaufruf, ob der aktuell angemeldete Benutzer für diese Aktion eine Berechtigung besitzt oder nicht. Hierfür ist für jede Seite jeweils pro Benutzerrolle zu definieren, ob die diese auf die Seite zugreifen darf oder nicht.

Darstellung

Im System gibt es einen Bereich für angemeldete Benutzer und einen Bereich für nicht angemeldete Benutzer in Form einer Anmeldemaske. Jeder Benutzer wird beim Besuch der Seite zunächst auf die Anmeldemaske (siehe Abbildung 4.14) geleitet und zur Eingabe seiner

Benutzerdaten mit Passwort aufgefordert. Da eine Anmeldung außerdem nur von bestimmten Hosts über definierte IPv4- bzw. IPv6-Adressen aus möglich sein soll, wird diese ebenfalls angezeigt.

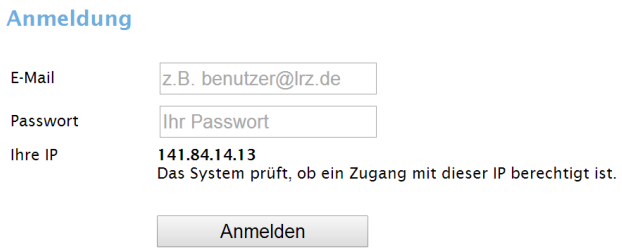


Abbildung 4.14.: Anmeldemaske des SDACT Systems

Nach der erfolgreichen Authentifizierung erhält der Benutzer eine Sitzung zugewiesen und wird auf die Startseite der Anwendung weitergeleitet. Von dort aus kann er eigene Aufgaben anlegen und verwalten. Auch kann er sich die Ergebnisse anzeigen und weitere nicht auf Administratoren begrenzte Funktionen nutzen. Ein Benutzer erhält über die Weboberfläche jeweils nur Zugang zu den Aufgaben, die er selbst erstellt hat. Jede Aufgabe ist dabei einem Benutzer eindeutig zugeordnet.

Für jeden Benutzer muss der Administrator definieren, welche Auswertungen dieser vornehmen darf. Beispielsweise kann der Administrator dem Benutzer den Scan-Bereich `10.156.200.1/28` freischalten. Versucht ein Benutzer einen größeren Scan-Bereich anzugeben, so wird er trotzdem nur die Ergebnisse erhalten, für die er explizit freigeschaltet ist. Die Eingabe von mehreren Scan-Bereichen ist möglich. Ist kein Scan-Bereich gesetzt, darf der Benutzer keine Auswertungen vornehmen.

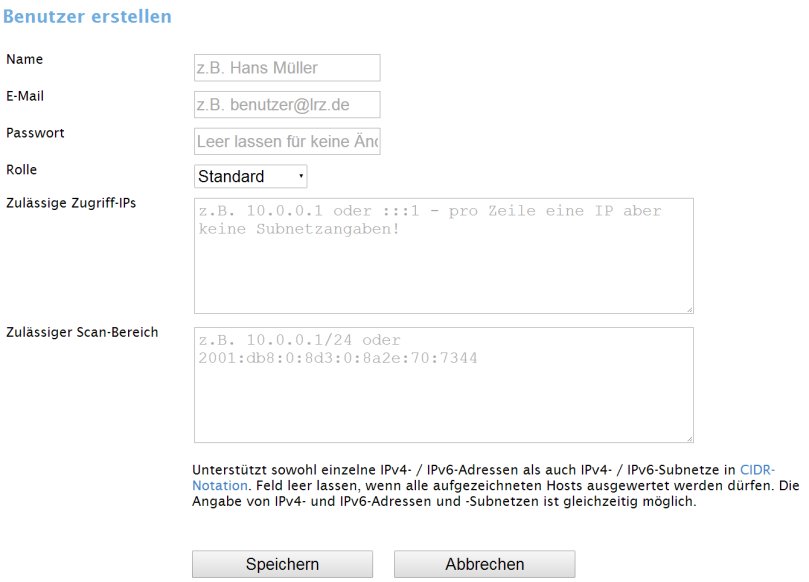


Abbildung 4.15.: Maske zum Anlegen eines Benutzers

Es gibt im beschriebenen System zwei Benutzertypen: einen einfachen Benutzer (Rolle

`Standard`) und einen Administrator (Rolle `Administrator`), der die verschiedenen Benutzer anlegen (siehe Abbildung 4.15) und verwalten kann. Der Benutzertyp wird im Datensatz des jeweiligen Benutzers gespeichert und kann von einem Administrator gesetzt werden. Im Benutzerprofil sind alle relevanten Angaben wie E-Mailadresse oder Benutzertyp zu einem Nutzer gespeichert. Referenziert wird ein Benutzer über seine `ID`, wie es im ERM in 4.1.2 beschrieben ist.

4.1.10. Benachrichtigung per E-Mail

Es gibt bei Aufgaben zusätzlich eine Eigenschaft E-Mail-Benachrichtigung. Bei jeder Statusänderung einer Aufgabe, beispielsweise wenn diese fertiggestellt wurde und Ergebnisse vorliegen, erhält der Benutzer eine E-Mail an die in seinem Profil hinterlegte Adresse zugesandt. Dabei wird die in der eingesetzten Programmiersprache, beispielsweise PHP oder Perl, integrierte Mail-Funktionalität genutzt. Zusätzlich wird in der Konfiguration ein externer Mailserver (Zugang über SMTP) eingetragen, sodass die E-Mails von allen gängigen Anbietern angenommen werden. Der Aufwand zur Einrichtung eines eigenen Mailservers würde in keinem Verhältnis zum Nutzen stehen.

4.1.11. IPv6-Unterstützung

Neben IPv4 sollen nach der Anforderung auch IPv6 Adressen möglich sein. Die Portnummern werden bei IPv6 verwendet wie beim älteren Protokoll. Teilweise wird die Adresse zur Abgrenzung - vor allem bei URL (Uniform Resource Locator)s und HTTP - in eckige Klammern eingeschlossen. So sieht eine IPv6 mit Portnummer in der Form `[2001:0db8:85a3:08d3:1319:8a2e:0370:7344]:8080` aus. Die Weboberfläche muss bei der Ein- und Ausgabe der Daten neben IPv4- zusätzlich auch IPv6-Adressen verarbeiten können, beispielsweise bei der Erfassung des Scan-Bereiches oder bei der Anzeige aller entdeckten Hosts. Das beschriebene System verwendet zudem externe Bibliotheken, welche das Protokoll ebenfalls unterstützen müssen.

Passive Erkennungsmethoden Sowohl die Netflow-Technik als auch die DPI unterstützen grundsätzlich IPv6. Jedoch ist bei beiden Methoden ein Export der Verbindungs- bzw. Paketdaten auf Routerebene vonnöten, was einen IPv6 kompatiblen Router voraussetzt. Zusätzlich müssen die eingesetzten Werkzeuge das Protokoll unterstützen. Snort tut dies beispielsweise seit der Version 2.8.0 und verfügt über integrierte IPv6-DPI-Funktionen. Auch das ebenfalls frei verfügbare Suricata unterstützt spätestens seit der Version 1.4.6 IPv6 vollständig. Auch muss das Netflow-Format v9 als Export-Format genutzt und von den Erkennungsmethoden gelesen werden können, da dies die erste Version eines Netflow-Formates ist, welche IPv6 unterstützt.

Aktive Erkennungsmethoden Bei den aktiven Verfahren ist zwischen einem Port- und einem Schwachstellenscanner zu unterscheiden. Nmap war als Portscanner in der engen Auswahl und unterstützt IPv6 bereits seit 2002. Alle vorgestellten Schwachstellenscanner wie Nessus und Metasploit aber auch PVS von Tenable sind für das neue Protokoll gerüstet.

4.1.12. Kompatibilitätsprüfung der Internetprotokolltypen

Die Anforderung 3.2.12 beschreibt, dass offene Ports gesucht werden, die lediglich über IPv4 erreichbar sind und nicht über IPv6 oder umgekehrt. Dieses Ziel kann grundsätzlich durch alle vorgestellten Erkennungsmethoden erreicht werden, die eine gleichzeitige Verarbeitung von IPv4- und IPv6-Adressen ermöglichen.

Am einfachsten erfüllt man diese Anforderung allerdings, indem man zwei Portscans auf das gleiche System durchführt. Einen auf der IPv4- und einen auf der IPv6-Adresse. Anschließend vergleicht man die erreichbaren Dienste wobei ein Unterschied nicht zwingend eine Fehlkonfiguration oder Schwachstelle darstellen muss. So gibt es immer noch viele Softwareinstallationen, die IPv6 nicht unterstützen. Umgekehrt, also ein Dienst hört auf einen IPv6-Port aber nicht auf denselben IPv4-Port, kann es allerdings auf eine echte Sicherheitslücke hindeuten. Selbstverständlich kann es in beiden Fällen aber auch sein, dass das unterschiedliche Verhalten vom Systemadministrator so konfiguriert wurde und gewollt ist.

Für den Portscan über die beiden Protokolltypen benötigt man sowohl die IPv4- als auch die IPv6-Adresse des Systems. Vom Benutzer würde man aber eventuell lediglich eine der beiden Adressen abfragen müssen, da die jeweils andere über den DNS abgerufen werden könnte.

Im eingeschränkten Rahmen dieser Arbeit ist eine Realisierung der Kompatibilitätsprüfung nicht leistbar, obwohl eine solche Prüfung regelmäßig durchgeführt werden sollte.

4.2. Nichtfunktionale Anforderungen

Nach der Beschreibung der funktionalen Anforderungen wird im Folgenden auf den Entwurf der nichtfunktionalen Anforderungen eingegangen.

4.2.1. Zuverlässigkeit

Die Zuverlässigkeit des Systems unterteilt sich in drei Unterpunkte, welche nachfolgend erläutert werden:

Systemreife Auch wenn das entwickelte System in dieser Arbeit lediglich als Prototyp bezeichnet wird, wird dieser sorgfältig und unter Einhaltung gängiger Standards entwickelt. So schreibt das eingesetzte Wasserfall-Modell eine klare Abgrenzung der einzelnen Projektphasen vor und erleichtert eine sorgfältige Implementierung.

Wiederherstellbarkeit Durch die Speicherung der Daten in einer Datenbank können diese regelmäßig gesichert werden und bei einem Ausfall schnell auf einem anderen System eingespielt werden. Zusätzlich ist der gesamte Programmcode in einem GIT-Repository gesichert.

Fehlertoleranz Aufgrund der großen Anzahl an Daten sollen die Erkennungsverfahren nicht abbrechen, wenn es in einer einzelnen Zeile zu einem Fehlerfall kommt. Im Worst Case sollen aus dieser Zeile dann eher keine Informationen gezogen werden, als die gesamte Auswertung abzubrechen und damit keine Informationen zu erhalten.

4.2.2. Aufbau und Handhabung

Der Entwurf sieht vor, dass der Benutzer alle Programmteile mit wenigen Klicks erreichen kann. Der Aufbau, wie in Abbildung 4.3 beschrieben, ist übersichtlich und intuitiv, der Benutzer kennt diesen von vielen anderen webbasierten Anwendungen bereits. Die Reaktion auf Benutzereingaben erfolgt dabei in kürzester Zeit. Für einen möglichst uneingeschränkten Zugang wurden keine außergewöhnliche Schriftarten eingesetzt und kontraststarke Farben verwendet. Das Design wird für einen besseren Wiedererkennungswert an das Corporate Design des LRZ angelehnt.

Die Steuerung und Benutzung des Programms für eine möglichst breite Anwenderschicht ist durch die Unterstützung von sowohl statischen als auch modernen Browsern möglich. Auch eine Bedienung der Anwendung von einem mobilen Gerät aus sind keine Grenzen gesetzt.

4.2.3. Leistung und Effizienz

Das Programm untergliedert sich in zwei Komponenten. Zum einen die Weboberfläche und zum anderen der Aufgabenplaner, der die Erkennungsmethoden auf die in einer Aufgabe hinterlegten Daten durchführt. Wir betrachten für beide Teile jeweils die Leistung und Effizienz.

Leistung Die Weboberfläche an sich stellt keine besonderen Leistungsanforderungen an das System. Die Ausführungsgeschwindigkeit ändert sich auf verschieden starken Hostsystemen für den Benutzer nicht merklich. Im Gegensatz dazu hängt die Datenverarbeitungsleistung des Aufgabenplaners deutlich von der Installationsumgebung ab. Zum einen können beispielsweise sehr viele Dateizugriffe auf alle Netflow-Dateien in einem Verzeichnis stattfinden. Diese lassen sich durch Pre-Caching oder durch moderne Festplatten wie SSD (Solid-State-Drive) beschleunigen. Zum anderen ist aber auch reine Rechenleistung für die Auswertung der Daten an sich notwendig, beispielsweise für die zeilenbasierte System- und Diensterkennung aber auch bei der Speicherung. Da eine Aufgabe in nur einem Prozess und deshalb auch nur auf einem Prozessor ausgeführt wird, entscheidet dabei allerdings die Geschwindigkeit eines Prozessorkerns und nicht die Anzahl der darauf verbauten Kerne.

Effizienz Wann immer möglich sind bei beiden Programmkomponenten effiziente Verfahren zur Verarbeitung der Datenmengen vorgesehen. Jedoch sind dem Einsatz solcher auch Grenzen gesetzt, da eine Implementierung von einem noch effizienteren Algorithmen eine deutlich höhere Komplexität verursachen kann.

4.2.4. Betrieb und Umgebung

Serverseitig wird eine klassische LAMP (Linux, Apache, MySQL und PHP) - Umgebung zum Einsatz kommen. Hierbei könnte der Webserver oder das Datenbankmanagementsystem prinzipiell beliebig gewählt werden können, in dieser Arbeit wird allerdings auf die bewährte Kombination von Apache und MySQL gesetzt. Seit der Version 5.3 unterstützt PHP alle grundlegenden Konzepte der objektorientierten Programmierung und bietet mit PDO (PHP Data Objects) eine Abstraktionsebene für den Datenbankzugriff. Aber auch an dieser Stelle wäre die Wahl einer anderen Skriptsprache wie Python oder Perl problemlos möglich gewesen. Die Einrichtung des LAMP-Systems wird im Anhang A ausführlich beschrieben. Des Weiteren

sind zur Ausführung der verschiedenen Erkennungsmethoden zusätzliche Programme vonnöten. So wird für ein Portscan beispielsweise Nmap vorausgesetzt und beim Aufzeichnen von Netflows werden Werkzeuge wie `flowd` oder `nfdump` eingesetzt. Letzteres wird dabei auch zum Auswerten der aufgezeichneten Daten benötigt.

4.2.5. Portierbarkeit

Grundsätzlich ist der plattformübergreifende Betrieb des beschriebenen Systems möglich. Wann immer möglich wurden Lösungen ausgewählt, die für die bekanntesten Plattformen wie Windows, Linux oder Mac zur Verfügung stehen. So ist die Weboberfläche ohne größere Einschränkungen auf jedem System, das eine aktuelle Version von Apache, MySQL und PHP installiert hat, lauffähig. Jedoch werden für die Erkennungsmethoden teilweise systemnahe Dienste, welche häufig nur für Linux-Systeme verfügbar sind, eingesetzt. So waren zum Zeitpunkt dieser Arbeit keine Windows-Portierungen von den `nfdump`-Tools oder von PRADS verfügbar.

Durch den Einsatz von PDO bei der Skriptsprache PHP kann das DBMS frei gewählt werden und später auch ohne Probleme gewechselt werden.

4.2.6. Sicherheitsanforderungen

Um die gestellten hohen Sicherheitsanforderungen an die Anwendung zu erfüllen, sind an mehreren Stellen Vorkehrungen zu treffen. Zum einen ist der Zugriff auf die Anwendung abzusichern, da sonst sensible Daten offengelegt werden können. Wie bereits festgehalten, obliegt dies dem Administrator, der den Prototyp einsetzt. Denkbar ist beispielsweise ein Gateway-Konzept. Dabei müssen sich alle Benutzer, die auf das Programm zugreifen möchten, von einem Gateway-Server authentifizieren lassen. Der Zugriff auf die Applikation erfolgt dann ausschließlich über diesen Gateway-Server, ein direkter Aufruf ist nicht mehr möglich.

Bei der Kommunikation der Programmteile ist auf eine gesicherte Verbindung zwischen den verschiedenen Systemen zu achten. Beispielsweise muss die Verbindung zwischen Web- und Datenbankserver über einen SSL-Tunnel erfolgen, wenn diese nicht auf dem gleichen System sind. Aber auch wenn die Programmteile auf dem gleichen System sind, müssen Vorkehrungen getroffen werden, dass die Daten nicht von Unberechtigten gelesen oder manipuliert werden können.

Die auszuwertenden Netflow-Dateien könnten grundsätzlich auch Schadcode enthalten und dürfen deshalb unter keinen Umständen ausgeführt werden können. Bei der Auswertung ist auf eine sichere Behandlung der hochgeladenen Dateien zu achten. Auch sind die ausgewerteten Daten an sich wieder ein schützenswertes und wichtiges Asset, da sie je nach Scan-Bereich eine vollständige Dokumentation des Netzes beinhalten. Für potentielle Angreifer könnten diese Ergebnisse wertvolle Hinweise auf mögliche Schwachstellen beinhalten.

Des Weiteren müssen aufgrund der Gestaltung als Webanwendung folgende Angriffsmöglichkeiten beachtet werden:

Cross Site Scripting (XSS) Ein Angreifer möchte persistente Daten in das System einspeisen um diese dann allen Systembenutzern bei der Benutzung des Systems anzeigen zu lassen. Aufgrund der Vertrauenswürdigkeit des Systems rechnen Benutzer nicht mit diesen Daten.

SQL-Injection Benutzereingaben werden in die Datenbank gespeichert, dabei möchte ein Angreifer die Eingabedaten so manipulieren, dass der dazugehörige SQL-Befehl abgewandelt wird und als Beispiel alle vorhandenen Daten manipuliert oder löscht. Alle Eingabedaten sind entsprechend sorgfältig zu prüfen, bevor sie weiterverarbeitet werden.

4.2.7. Korrektheit

Weder die Vollständigkeit noch die Korrektheit der Ergebnisse kann aufgrund der Aufgabenstellung garantiert werden. Zum einen können Systeme unerkannt bleiben, wenn sie nicht oder nur sehr selten im Netz kommunizieren. Zum anderen können Systeme sich aber auch als etwas anderes ausgeben. Schlussendlich kann die Erkennungsmethode auch einfach nicht in der Lage sein, ein Host, Betriebssystem oder Dienst korrekt zu erkennen, auch wenn dieser sich ganz normal verhält.

Jedoch soll das Auswertungsverfahren für die gleiche Eingabe stets die gleiche Rückgabe zurückliefern. Dabei kann sich die Eingabe einer Aufgabe durch die Angabe eines Ordnerpfades durchaus - auch ohne Änderung der Aufgabe selbst - ändern. Wenn etwa ein Netflow Collector die Daten stetig in das Verzeichnis schreibt, ändert sich die Eingabemenge gewöhnlich alle fünf Minuten.

4.2.8. Skalierbarkeit

Bei der Skalierbarkeit ist zwischen vertikaler und horizontaler Skalierung zu unterscheiden. Das System kann ohne Probleme vertikal skaliert werden, also durch den Einbau einer schnelleren CPU oder einer SSD beschleunigt werden mit direkter Auswirkung auf die Ausführungszeit und Datenverarbeitungsgeschwindigkeit. Bei der horizontalen Skalierung durch das Hinzufügen von neuen Knoten - also beispielsweise statt einem Server mehrere Server - ist zu differenzieren.

So beschleunigt das Hinzufügen von mehreren Knoten die gleichzeitige Ausführung von verschiedenen Aufgaben deutlich. So konnte bisher ein Server eine Aufgabe gleichzeitig in einer Zeitspanne t abarbeiten und bei perfekter horizontaler Skalierung könnten dann neu zehn Server zehn Aufgaben in derselben Zeitspanne t ausführen. Die Verarbeitungsgeschwindigkeit einer einzelnen Aufgabe nimmt dabei aber nicht zu, da eine Aufgabe nach diesem Entwurf nur von einem Prozess gleichzeitig abgearbeitet werden kann.

5. Implementierung

5.1. Funktionale Anforderungen

Die funktionalen Anforderungen treffen Aussagen über die zu erfüllenden Eigenschaften des beschriebenen Systems. Im Folgenden werden wir auf die Implementierung der einzelnen Anforderungen eingehen.

5.1.1. Weboberfläche

Die Weboberfläche wird über einen Browser aufgerufen und von einem LAMP (Linux, Apache, MySQL, PHP)-Server ausgeliefert. Die Einrichtung des LAMP-Systems wird im Abschnitt 5.2.3 weiter erläutert. Dabei nimmt der Webserver Apache die Anfrage entgegen und leitet sie auf Basis von Standard-Regeln an die `index.html` bzw. `index.php` weiter, falls diese im aufgerufenen Verzeichnis existiert. Ruft ein Benutzer also beispielsweise `http://10.0.0.1/sdact/` auf, wird die `index.php` zur Bearbeitung dieser Anfrage ausgeführt.

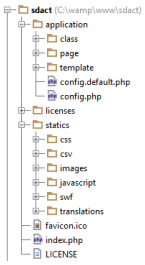


Abbildung 5.1.: Ordnerstruktur der Anwendung

Die Ordnerstruktur der Anwendung (Abbildung 5.1) untergliedert sich in einen dynamischen Programmteil im Verzeichnis `application` und in einen statischen Teil im Verzeichnis `statics`. So finden sich die Dateien, deren Rückgabe sich bei jedem Aufruf ändern können in `application`. Bilder, CSS (Cascading Style Sheets) oder aber JavaScripts befinden sich hingegen in den entsprechenden Unterordnern von `statics`.

Quellcode 5.1: Initialisierung von Programm

```
1 // Include configuration (database connection, navigation, availability of
      functionality)
2 require_once('application/config.php');
3
4 // Initialize database connection
5 require_once('application/class/database.php');
6 $database = new Database();
7 $database->connect($config['database']['host'],$config['database']['port'],
      $config['database']['database'],$config['database']['username'],$config['
      database']['password']);
8
9 // Execute router
```

```
10 require_once('application/class/router.php');
11 $router = new Router();
12 $router->setPages($config['pages']);
13 $router->routePrint();
```

Wie in Quelltext 5.1 ersichtlich, wird das Programm nach und nach initialisiert. Zuerst wird die Konfiguration geladen und mit den Daten daraus anschließend eine Verbindung zur Datenbank aufgebaut, welche während der gesamten Programmlaufzeit aufrecht erhalten wird. Die Klasse Datenbank wird im nachfolgenden Abschnitt 5.1.2 genauer erläutert. Schlussendlich wird der Router, die eigentliche Programmsteuerung, aufgerufen. Dieser bearbeitet die unterschiedlichen Anfragen, welche sich wie folgt voneinander unterscheiden:

Statischer Seitenaufruf Der erste Seitenaufruf lädt das Rahmengerüst, wie in Abbildung 4.3 beschrieben. Zusätzlich wird die gewünschte Seite (Parameter `page`) mit einer Aktion (Parameter `action`), falls diese jeweils gesetzt ist, aufgerufen. Andernfalls wird die Startseite (`?page=start`) geladen. Dadurch wird das Programm auch kompatibel zu Browsern, die kein JavaScript unterstützten bzw. bei denen es nicht aktiviert ist.

Dynamischer Seitenaufruf Der dynamische Seitenaufruf lädt lediglich einen kleinen Teil der Anwendung nach. So gibt beispielsweise der Aufruf `?page=start` nur die konkret angeforderte Seite, nicht aber nochmals das vollständige Rahmengerüst, zurück. Im Programm wird dies dadurch realisiert, dass alle Verweise und Formularaufrufe automatisch dynamisch ausgeführt werden, also mithilfe der AJAX-Technologie gesandt werden.

Systemaufruf Durch den Parameter `?systemCall=1` wird ein Systemaufruf gekennzeichnet. Dieser wird immer ohne die Ausgabe einer grafischen Oberfläche bearbeitet und führt lediglich die aufgerufene Aktion durch. Ein Beispiel eines solchen Aufrufs ist die regelmäßige Prüfung, ob es Aufgaben gibt, die auszuführen sind: `http://10.0.0.1/sdact/?page=planner&systemCall=1`.

Darstellung

Die erste Seite der Anwendung ist die Startseite, welche eine schnelle Navigation auf die unterschiedlichen Programmfunktionen sicherstellen soll. Wie auf Abbildung 5.2 ersichtlich, befinden sich zu Beginn der Seite die Statistiken, welche einen schnellen Überblick über den Zustand des Systems geben sollen. Beispielsweise, wie viele Aufgaben aktuell auf Ausführung warten oder wann die letzte Aufgabe erfolgreich ausgeführt wurde. Neben den bereits im Entwurf 4.1.1 erwähnten Aktionen wie `Neue Aufgabe anlegen` gibt es zusätzlich die Administratorfunktionen, welche zusätzliche Steuerungsmöglichkeiten der Anwendung anbieten und somit vor allem auch während der Test- und Evaluationsphase nützlich sein können.

Für die effiziente und plattformübergreifende DOM (Document Object Model)-Navigation und -Manipulation wird die freie Javascript-Bibliothek jQuery[1] eingesetzt. Darauf bauen folgende Programmfunktionen auf:

Dynamische Seiten HTML-Hyperlinks werden gewöhnlich mit einem `<a>` Element definiert. Die klassische Art der Definition von Hyperlinks unterstützt auch weiterhin die Webbrowser,

[1] jQuery v1.11.0 von jQuery Foundation, Inc., `http://jquery.org`

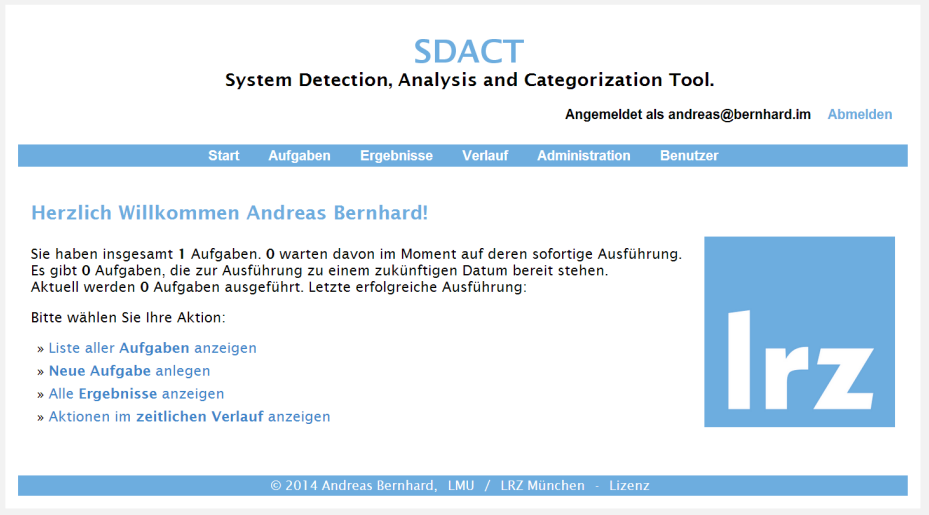


Abbildung 5.2.: Implementierung der Weboberfläche

die beispielsweise aus Sicherheitsgründen kein JavaScript aktiviert haben. Für die zentrale Umwandlung aller in der Weboberfläche so definierter Hyperlinks wurde eine auf jQuery aufbauende Funktion geschrieben, welche in Quellcode 5.3 dargestellt ist. Diese Funktion macht aus allen statischen Links bei aktiviertem JavaScript automatisch dynamische Links und lädt die gewünschte Seite im Hintergrund durch den Aufruf von `loadPage` (siehe Quellcode 5.2) nach.

Quellcode 5.2: loadPage führt Aufruf auf native jQuery load-Funktion aus

```
function loadPage(href) {
    $('#content').load(href);
}
```

Dabei wird auch geprüft, was der eigentliche Zweck des Hyperlinks war. Sollte dieser beispielsweise in einer neuen Seite geöffnet werden oder explizit kein dynamischer Hyperlink (versehen mit einem `nobind='true'` Attribut) sein, wird dieser wie gewünscht ausgeführt. Auch wurde ein zusätzliches Attribut `confirm='Message'` eingeführt, welches beispielsweise eine einfache Sicherheitsabfrage bei einem Link, der eine Aufgabe löscht, realisiert.

Quellcode 5.3: Automatisches Erkennen und dynamisches Umwandeln aller Hyperlinks

```
$('body').on('click',"a, button, input[type='button']",function(event) {
    // If link has a confirm text which a user has to accept
    if($(this).attr('confirm')!=undefined) {
        if(!confirm($(this).attr('confirm'))) {
            return false;
        }
    }
    // If link will be opened in a new window
    if($(this).attr('target')=='_blank' || $(this).attr('target')=='_new') {
        return true;
    }
    // If link is a non dynamic link, marked with HTML nobind attribute
    else if($(this).attr('nobind')=='true') {
        return true;
    }
    // If javascript link, don't use ajax functionality
```

```
17      else if ((typeof($(this).attr('href')) == 'string') && ($(this).attr('href'
            ).search(/^javascript/) != -1)) {
18          return true;
19      }
20      // If href attribute doesn't exist in anchor
21      else if ((typeof($(this).attr('href')) != 'string')) {
22          return true;
23      }
24      // If link contains an anchor #
25      else if(($(this).attr('href')).indexOf('#',0)!=-1) {
26          return true;
27      }
28      // Open link in new window, no ajax action
29      else if(event.ctrlKey) {
30          return true;
31      }
32      // Load page via ajax
33      else {
34          event.preventDefault();
35          loadPage($(this).attr('href'));
36      }
37      return false;
38  });
```

Moderne Tabellen Die klassischen HTML-Tabellen sind statisch und bieten keine Möglichkeiten für den Benutzer, die Inhalte zu durchsuchen oder nach Spalten zu sortieren. Auch ist eine Darstellung vieler Datensätze auf einer Bildschirmseite nicht übersichtlich möglich. Aus diesem Grund wurde für die Weboberfläche eine auf jQuery basierende Lösung namens DataTables[2] integriert, welche eine solche dynamische Tabelle aus einer klassischen HTML-Tabelle erzeugt. Der Aufruf für eine solche Umwandlung ist in Quellcode 5.4 angegeben.

Quellcode 5.4: Aufruf von DataTables zur verbesserten Darstellung statischer HTML-Tabellen

```
1  <script>
2      $(document).ready(function () {
3          $('#historyTable').dataTable({
4              "oLanguage": {
5                  "sUrl": "statics/translations/dataTables.txt"
6              },
7              "aaSorting": [[ 0, "desc" ]]
8          });
9      });
10 </script>
```

Die erste Zeile mit dem Aufruf der `ready`-Funktion bedeutet, dass mit der Ausführung gewartet wird, bis die Seite vollständig geladen ist. Als Datengrundlage für die neue dynamische Tabelle wird eine bestehende HTML-Tabelle gefordert, welche über die ID `historyTable` referenziert wird und in Quellcode 5.5 dargestellt ist. Für eine deutschsprachige Darstellung der Bedienungselemente wird eine Übersetzung über den Parameter `oLanguage` angegeben. Die Sortierung wird festgelegt über die Eigenschaft `aaSorting`, in diesem Beispiel wird absteigend nach der ersten Spalte sortiert[3]. Das Ergebnis dieses Beispiels ist in Abbildung 5.5, einem Verlauf aller Aufgaben, dargestellt.

Quellcode 5.5: Exemplarische Verlaufstabelle, die dynamisch umgewandelt wird

```
1  <table id="historyTable" class="display" cellspacing="0" width="100%">
2      <thead>
```

[2] DataTables 1.10.0 von SpryMedia Ltd, `http://www.datatables.net`
[3] Die Zählung der Spalten beginnt bei `0`.

```
3       <tr>
4           <th>Datum</th>
5           <th>Aufgabe</th>
6           <th>Status</th>
7           <th>Meldung</th>
8       </tr>
9       </thead>
10      <tbody>
11      <tr>
12          <td>11.03.2014</td>
13          <td>Meine Aufgabe (ID 1)</td>
14          <td>Abgeschlossen</td>
15          <td>-</td>
16          </tr>
17          </tbody>
18  </table>
```

Dynamische Eingabefelder Für die schnellere Eingabe von Daten wird ein dynamisches Datums-Eingabefeld[4] verwendet. Zwar gibt es in HTML5 bereits native Datumsfelder, welche browserseitig solche Auswahlmöglichkeiten anbieten, jedoch wird diese noch nicht von allen gängigen Browsern unterstützt.

5.1.2. Datenbank

Das im Entwurf 4.1.2 entwickelte ERM wird bis auf die Benachrichtigung bei Fertigstellung einer Aufgabe wie beschrieben realisiert. Die Benachrichtigung bei Fertigstellung einer Aufgabe sollte, damit sie sinnvoll weitergenutzt werden kann, zusätzlich die Ergebnisse der Aufgabe beinhalten. Diese Ergebnisse sind jedoch vertraulich und sollten nicht als einfacher E-Mail-Inhalt oder -Anhang gesandt werden. Deshalb werden Benachrichtigungen über das Vorliegen von Ergebnissen in dieser Arbeit nicht implementiert.

Das daraus abgeleitete und aktualisierte ERM ist in Abbildung 5.3 dargestellt. Bei der Relation `Aufgabe` wurde aus diesem Grund die Spalte `Benachrichtigung` entfernt.

Die Installation bzw. Konfiguration des DBMS MySQL wird in A bzw. A.2 erläutert und die zugehörigen SQL-Befehle sind im Anhang C nachzulesen.

Der Zugriff von der Weboberfläche auf die Datenbank erfolgt über die Klasse `Database` (Quelltext 5.6). Diese nutzt die seit PHP 5.1 verfügbare Komponente PDO, so kann das zugrundeliegende DBMS leicht ausgetauscht werden. Auch die weiteren Datenbankoperationen werden über die zentrale Klasse `Database` abgewickelt. Falls keine Verbindung zur Datenbank aufgebaut werden kann, wird die aktuelle Programmausführung mit einer entsprechenden Fehlermeldung abgebrochen.

Quellcode 5.6: Ausschnitt der Klasse Database

```
1  class Database {
2      private $db;
3
4      public function connect($host,$port,$database,$username,$password) {
5          try {
6              $this->db = new PDO('mysql:host='.$host.';port='.$port.';dbname='.
                   $database, $username, $password);
7          } catch (PDOException $e) {
8              print "Database Connection Error!: " . $e->getMessage() . "<br/>";
9              die();
```

[4] jQuery DateTimePicker plugin v2.1.9 von Chupurnov Valeriy, `http://xdsoft.net/jqplugins/datetimepicker/`

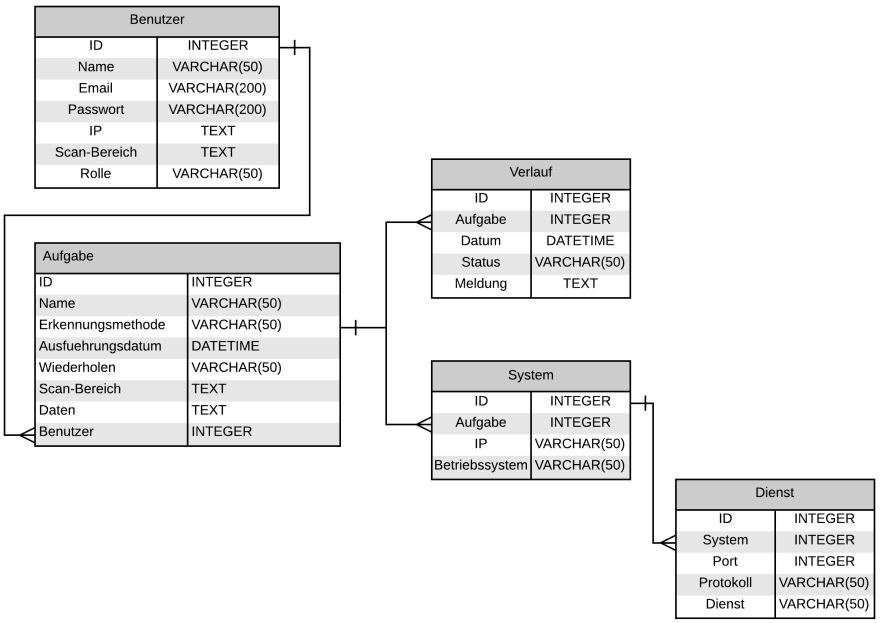


Abbildung 5.3.: Finales ER-Modell der Anwendungsobjekte

```
10          }
11      }
12  }
```

Dem Programm wird kein direkter Zugriff auf die PHP PDO Funktionalität ermöglicht, die Klasse `Database` arbeitet als Wrapper zwischen Programm und PDO. Fehler in Abfragen können so zentral behandelt werden, was in Quellcode 5.7 gezeigt wird.

Quellcode 5.7: Ausführen eines SQL-Query mit Fehlerbehandlung

```
1  public function query($sql) {
2      try {
3          return $this->db->query($sql);
4      } catch (PDOException $e) {
5          print "Database Query Error!: " . $e->getMessage() . "<br/>";
6          die();
7      }
8  }
```

5.1.3. Aufgabenplaner

Der Aufgabenplaner nimmt eine zentrale Rolle im beschriebenen System ein. Zum einen dient er zur Verwaltung der Aufgaben (siehe Eingabemaske Abbildung 5.4) und zur Anzeige der Ergebnisse. Zum anderen führt er die zuvor angelegten Aufgaben auch aus und leitet die notwendigen Schritte ein, um eine Analyse durchführen zu können.

Verwaltung

Die Auflistung aller Aufgaben wurde mithilfe einer DataTable dynamisch implementiert, was eine einfache Durchsuchung und Sortierung für den Benutzer ermöglicht. Die wichtigste

Funktion ist neben der Auflistung der Ergebnisse allerdings das Anlegen und Bearbeiten von Aufgaben, welche in Abbildung 5.4 gezeigt ist. Hierbei müssen die erforderlichen Felder einer Aufgabe vom Benutzer abgefragt werden und nach dem Absenden des Formulars in der Datenbank angelegt bzw. gespeichert werden. Das Formular muss also in der Lage sein, bei der Bearbeitung die aktuellen Werte anzuzeigen und für den Benutzer editierbar zu machen. Durch die Verwendung des HTML-Attributes `placeholder` wird zudem versucht, die Eingabefelder möglichst selbsterklärend darzustellen. Dabei wird bei einem leeren Textfeld im Hintergrund grau schattiert ein Beispielwert angezeigt.

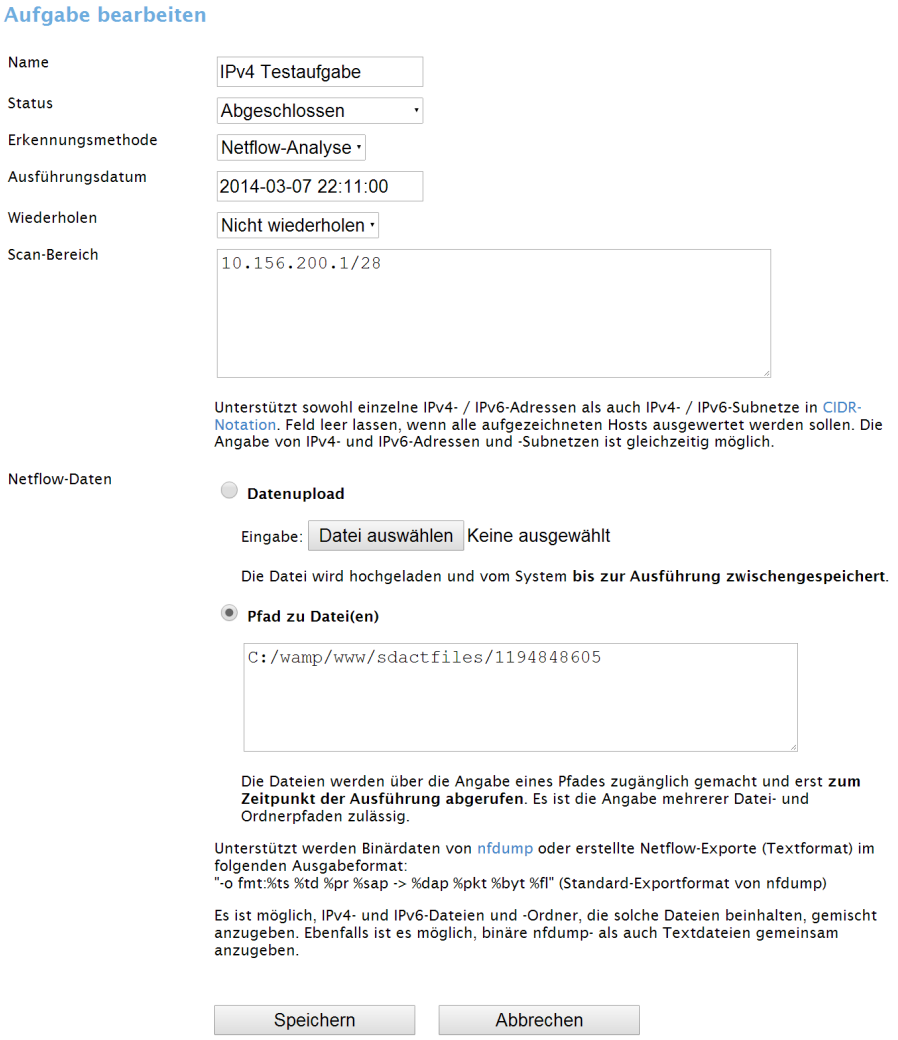


Abbildung 5.4.: Implementierte Bearbeitungsmaske mit Eingabefeldern

Vor dem Absenden des Formulars wird die Benutzereingabe auf Plausibilität geprüft und entsprechend akzeptiert oder im Fehlerfall mit einer Meldung verweigert. Eine Überprüfung der hochgeladenen Dateien bzw. der Dateien im angegebenen Ordnerpfad ist aufgrund fehlender Verfahren nicht trivial und könnte in dem Fall, dass die Dateien zum Zeitpunkt des Anlegens der Aufgabe noch gar nicht existierten, auch nicht gewünscht sein. Bei der Speicherung

ist darauf zu achten, dass die Benutzereingaben in der Hinsicht validiert werden, dass sie weder eine SQL-Injection noch persistente oder nicht-persistente XSS (Cross-Site-Scripting)-Angriffe zulassen. Beide Punkte werden ausführlich im Abschnitt Sicherheitsanforderungen 5.2.5 behandelt.

Zusätzlich wird beim Bearbeiten einer Aufgabe unter dem Formular ein Verlauf in Form einer Tabelle angezeigt. Darin enthalten sind alle bisherigen Statusänderungen, ähnlich wie in Abschnitt 5.1.3 beschrieben.

Aktueller Status

Der aktuelle Status einer Aufgabe ergibt sich nicht durch eine einfache Feldabfrage, sondern muss auf Basis des letzten gesetzten Status im Verlauf ermittelt werden. Die Abfrage ist in Quellcode 5.8 erläutert.

Quellcode 5.8: Laden des aktuellen Status einer Aufgabe

```
# Get current 'Status' from a given task $taskId
$task = $database->query("SELECT *,(SELECT Status FROM Verlauf WHERE Verlauf.
    Aufgabe=Aufgabe.ID ORDER BY ID DESC LIMIT 1) AS Status FROM Aufgabe WHERE
    ID='".$taskId."')->fetch();
```

Da der Status einer Aufgabe an mehreren Stellen im Programm gesetzt wird, wurde die Funktionalität in eine Funktion ausgelagert, welcher über die Klasse `Database` aufgerufen werden kann. Zusätzlich wird in 5.9 geprüft, ob der zu setzende Status ein gültiger Status ist.

Quellcode 5.9: Setzen des aktuellen Status einer Aufgabe

```
// Set 'Status' $state for a given 'Aufgabe' $taskId with an optional 'Meldung'
     $message
public function insertIntoHistory($taskId,$state,$message='') {
    global $config;
    if(!array_key_exists($state,$config['states'])) {
        return false;
    }
    $sth = $this->prepare("INSERT INTO Verlauf SET `Aufgabe`=?,Datum=NOW(),
        Status=?,Meldung=?");
    $sth->execute(array($taskId, $state, $message));
}

// Example call
$database->insertIntoHistory('1', 'done', 'Aufgabe erfolgreich abgeschlossen');
```

Verlauf

Der Verlauf ist ein Navigationspunkt und stellt eine Tabelle der Status-Änderungen aller Aufgaben dar, wie in Abbildung 5.5 erkennbar ist. Dies ermöglicht es, Freigaben, Ausführungen, Ergebnisse und Fehler auf einen Blick zu erkennen und entsprechend darauf zu reagieren. Durch Klick auf den Aufgaben-Namen bzw. die Aufgaben-ID kann direkt in die Bearbeitungsmaske der Aufgabe navigiert werden.

Auch der Verlauf ist wie alle anderen dargestellten Tabellen im Programm dank der Integration der `DataTable`-Komponente sortier- und durchsuchbar.

Anzeige der Ergebnisse

Über die Seite Ergebnisse (`?page=result`) können die erkannten Systeme und die dazugehörigen Dienste aller Aufgaben angezeigt werden. Das Anzeigen der Ergebnisse von nur einer

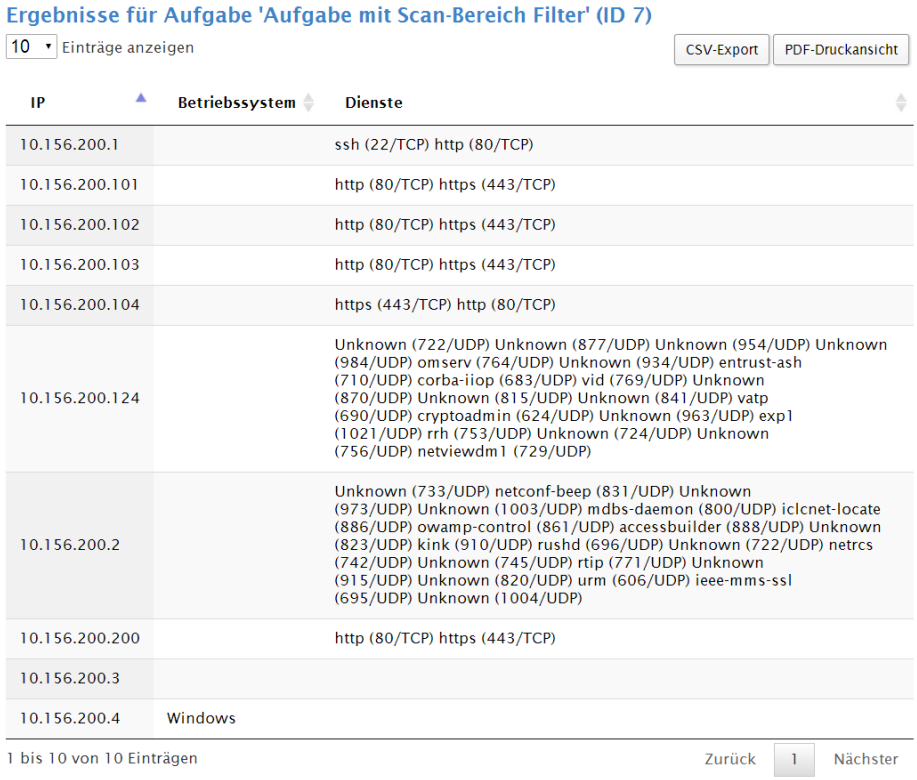
Verlauf

10 Einträge anzeigen — Suchen

Datum	Aufgabe	Status	Meldung
2014-03-07 11:33:21	Aufgabe mit Filter 10.156.200.1/24 auf /var/flows/ (ID 3)	Abgeschlossen	Aufgabe erfolgreich abgeschlossen
2014-03-07 11:32:54	Aufgabe mit Filter 10.156.200.1/24 auf /var/flows/ (ID 3)	Ausführung	Starten der Ausführung
2014-03-07 11:29:04	Aufgabe mit Filter 10.156.200.1/24 auf /var/flows/ (ID 3)	Warten auf Ausführung	

Abbildung 5.5.: Verlauf der Status-Änderungen aller Aufgaben

Aufgabe ist durch den zusätzlichen Parameter `?task=ID` ohne weiteres möglich.



Ergebnisse für Aufgabe 'Aufgabe mit Scan-Bereich Filter' (ID 7)

10 Einträge anzeigen — CSV-Export — PDF-Druckansicht

IP	Betriebssystem	Dienste
10.156.200.1		ssh (22/TCP) http (80/TCP)
10.156.200.101		http (80/TCP) https (443/TCP)
10.156.200.102		http (80/TCP) https (443/TCP)
10.156.200.103		http (80/TCP) https (443/TCP)
10.156.200.104		https (443/TCP) http (80/TCP)
10.156.200.124		Unknown (722/UDP) Unknown (877/UDP) Unknown (954/UDP) Unknown (984/UDP) omserv (764/UDP) Unknown (934/UDP) entrust-ash (710/UDP) corba-iiop (683/UDP) vid (769/UDP) Unknown (870/UDP) Unknown (815/UDP) Unknown (841/UDP) vatp (690/UDP) cryptoadmin (624/UDP) Unknown (963/UDP) exp1 (1021/UDP) rrh (753/UDP) Unknown (724/UDP) Unknown (756/UDP) netviewdm1 (729/UDP)
10.156.200.2		Unknown (733/UDP) netconf-beep (831/UDP) Unknown (973/UDP) Unknown (1003/UDP) mdbs-daemon (800/UDP) iclcnet-locate (886/UDP) owamp-control (861/UDP) accessbuilder (888/UDP) Unknown (823/UDP) kink (910/UDP) rushd (696/UDP) Unknown (722/UDP) netrcs (742/UDP) Unknown (745/UDP) rtip (771/UDP) Unknown (915/UDP) Unknown (820/UDP) urm (606/UDP) ieee-mms-ssl (695/UDP) Unknown (1004/UDP)
10.156.200.200		http (80/TCP) https (443/TCP)
10.156.200.3		
10.156.200.4	Windows	

1 bis 10 von 10 Einträgen — Zurück 1 Nächster

Abbildung 5.6.: Erkannte Systeme und Dienste auf der Ergebnisseite dargestellt

Wie in Abbildung 5.6 ersichtlich, wird für jede Port- / Protokoll-Kombination zugleich der wahrscheinlichste Dienstname auf Basis der offiziellen Liste der standardisierten Ports der IANA (Internet Assigned Numbers Authority) angezeigt. Der Export der angezeigten Daten ist sowohl als CSV- oder als PDF (Portable Document Format)-Datei möglich und umfasst jeweils die abgerufenen Ergebnisse. Werden also nur die Ergebnisse einer bestimmten Aufgabe in der Tabelle dargestellt, werden auch nur diese exportiert. Der Export basiert auf der frei verwendbaren Erweiterung `TableTools` für die eingesetzte Komponente `DataTables`.

Ausführung von Aufgaben

Der Aufruf des ausführenden Aufgabenplaners erfolgt entweder automatisch wie im Entwurf 4.1.3 bereits vorgestellt, beispielsweise über einen Cronjob, oder manuell über die URL `?page=planner`, welche zusätzlich auf der Seite `Start` verlinkt ist. Beim Aufruf über einen Cronjob, muss zusätzlich der Parameter `systemCall=1` gesetzt werden. Außerdem ist der Aufruf zwingend von der IPv4-Adresse `127.0.0.1` bzw. der IPv6-Adresse `::1` auszuführen, da andernfalls der Aufruf aus Sicherheitsgründen blockiert wird.

Die Ausführung von Aufgaben untergliedert sich in folgende Teilaufgaben:

Laden einer Aufgabe Laden einer Aufgabe, die auszuführen ist (Quellcode 5.10). Der Ausführungszeitpunkt muss hierfür kleiner gleich der aktuellen Systemzeit sein und die Aufgabe muss aktuell den Status `Warten auf Ausführung` besitzen.

Quellcode 5.10: Suchen und Laden von auszuführenden Aufgaben

```
$tasks = $database->query("SELECT * FROM Aufgabe WHERE Ausfuehrungsdatum<=NOW()
     AND (SELECT Status FROM Verlauf WHERE Verlauf.Aufgabe=Aufgabe.ID ORDER BY
    ID DESC LIMIT 1)='waiting' ORDER BY Ausfuehrungsdatum ASC LIMIT 1")->
    fetchAll();
```

Aufgabe in Ausführung Markiere geladene Aufgabe mit dem Status `Ausführung`. Dies verhindert, dass weitere Instanzen des Aufgabenplaners die gleiche Aufgabe ausführen. Während dem Ausführen einer Aufgabe darf die Bearbeitung derselben Aufgabe nicht möglich sein, da sonst die Aufrufparameter inmitten der Abarbeitung verändert werden könnten. Dies könnte zu Verfälschungen oder zum unerwünschten Abbruch der aktuellen Programmausführung führen. Aus diesem Grund ist die Eingabemaske innerhalb dieses Zeitraums gesperrt. Sobald eine Aufgabe abgeschlossen ist oder ein Fehler-Status zurückgegeben wurde, kann die Aufgabe wieder regulär bearbeitet werden.

Wiederholen-Modus Prüfe, ob Aufgabe eine Wiederholen-Aufgabe ist und erstelle gegebenenfalls eine identische Aufgabe mit zukünftigem Ausführungszeitpunkt. Die genaue Implementierung wird in Abschnitt 5.1.7 erläutert.

Initialisierung und Ausführen der Erkennungsmethode Im ersten Schritt wird die `Detection`-Klasse initialisiert (Quellcode 5.11), welche zentral alle für eine Erkennungsmethode notwendigen Daten aufbereitet. Dabei werden sowohl die eigentliche Erkennungsmethode als auch alle zu untersuchenden Netzwerk-Adressen und -Subnetze (siehe auch 5.1.4) gesetzt. Auch wird der für den Ersteller der Aufgabe zulässige Scan-Bereiche gesetzt, dieser wird bei der Implementierung der Mandantenfähigkeit (siehe Abschnitt 5.1.8) genauer erläutert.

Quellcode 5.11: Initialisierung der Erkennung durch Setzen der Erkennungsmethode

```
// Init detection with scan sectors
$detection=new Detection();
$detection->setMethod($task['Erkennungsmethode']);

// Set IPv4/IPv6 addresses and subnets
$ips=explode("\n",$task['Scan-Bereich']);
foreach($ips as $ip) {
    $currentIP=escapeshellcmd(trim($ip));
    if(empty($currentIP)) continue;
```

```
10      $detection->addNetwork($currentIP);
11  }
12
13  // Add allowed scan networks for user of this task
14  $ips=explode("\n",$task['User-Scan-Bereich']);
15  foreach($ips as $ip) {
16      $currentIP=escapeshellcmd(trim($ip));
17      if(empty($currentIP)) continue;
18      $detection->addAllowedNetwork($currentIP);
19  }
```

Als nächstes werden die Analysedaten angegeben. Es wird zuerst geprüft, ob die auszuwertenden Daten überhaupt gesetzt sind und anschließend für die einzelnen Dateien und Ordner ein einfacher Test durchgeführt, ob sie existieren. Dabei ist zu unterscheiden, ob es sich um eine einzelne Logdatei oder um einen Ordnerpfad handelt. Je nach dem ist nur eine Datei auszuwerten oder die Analyse muss auf alle Dateien in einem Ordner angewandt werden.

Quellcode 5.12: Hinzufügen aller angegebenen Ressourcen einer Aufgabe

```
1  $dataPaths=explode("\n",$task['Daten']);
2  foreach($dataPaths as $data) {
3          $data=trim($data);
4          if(is_dir($data)) {
5                  $detection->addFolder($data);
6          }
7          elseif(is_file($data)) {
8                  $detection->addFile($data);
9          }
10         else {
11                 $database->insertIntoHistory($task['ID'], 'error', 'Auf
                       angegebenem Pfad '.$data.' keine Daten zum Verarbeiten
                       gefunden oder keine Berechtigung zum Lesen dieser Daten.');
12                 continue;
13         }
14 }
```

Nachdem alle Daten übergeben wurden, kann die Ausführung der Erkennungsmethode (Quellcode 5.13) gestartet werden. Diese wird im Abschnitt 5.1.5 weiter erläutert.

Quellcode 5.13: Ausführen der Erkennungsmethode im Aufgabenplaner

```
1  // Execute detection
2  try {
3          $result = $detection->execute();
4  } catch(Exception $e) {
5          $database->insertIntoHistory($task['ID'], 'error', 'Fehler beim
               Ausfuehren der Aufgabe. Fehlermeldung: '.$e->getMessage());
6          continue;
7  }
```

Verarbeitung der Ergebnisse Je nach Rückgabe des Verfahrens erfolgt entweder das Speichern der Ergebnisse und das nachfolgenden Setzen des `Abgeschlossen`-Status oder das Setzen eines `Fehler-Status`. In Quellcode 5.14 wird zunächst die zeitliche Abfolge aller Aufrufe gezeigt. Wenn keine Ergebnisse vorliegen, wird der `Fehler-Status` mit einer entsprechenden Meldung gesetzt.

Quellcode 5.14: Verarbeitung der Rückgabe der Erkennungsmethode

```
1  if(!$result->hasResults()) {
2          $database->insertIntoHistory($task['ID'], 'error', 'Nach Verarbeitung
               der Daten lagen keine Ergebnisse vor');
```

```
3         continue;
4 }
5
6 // Delete existing results
7 $detection->deleteExistingResults($task['ID']);
8
9 // Save new results
10 $detection->saveTree($task['ID']);
11
12 // Save state 'done'
13 $database->insertIntoHistory($task['ID'], 'done', 'Aufgabe erfolgreich
       abgeschlossen');
```

Wenn die gleiche Aufgabe bereits schon einmal durchgeführt wurde, werden die von dieser Ausführung noch vorhandenen Hostsysteme und Dienste gelöscht (Quellcode 5.15).

Quellcode 5.15: Löschen bereits vorhandener Ergebnisse einer Aufgabe

```
1 /**
2  * Delete results for a given task
3  * @param $taskID
4  */
5 public function deleteExistingResults($taskID) {
6         global $database;
7         // delete old results from same task
8         $systems=$database->query("SELECT * FROM System WHERE Aufgabe='".
              $taskID."'")->fetchAll();
9         foreach($systems as $system) {
10                $database->query("DELETE FROM Dienst WHERE System='".$system['
                      ID'].."'");
11                $database->query("DELETE FROM System WHERE Aufgabe='".$taskID."
                      '");
12        }
13 }
```

Schlussendlich werden alle neuen Daten in der Datenbank gespeichert (Quellcode 5.16), pro System und Dienst jeweils ein eigener Eintrag.

Quellcode 5.16: Speicherung der Systeme und Dienste in der Datenbank

```
1 /**
2  * Save results for a given task
3  * @param $taskID
4  */
5 public function saveTree($taskID) {
6         global $database;
7         // save hosts in database
8         $hosts=$this->hostMapper->getResults();
9         foreach($hosts as $ip => $host) {
10                $sth = $database->prepare("INSERT INTO System SET Aufgabe=?,`IP
                      `=?,Betriebssystem=?");
11                $sth->execute(array($taskID,$ip,implode(', ',$host['os'])));
12                if(count($host['services'])>0) {
13                        $systemId=$database->lastInsertId();
14                        foreach($host['services'] as $service) {
15                                $sth = $database->prepare("INSERT INTO Dienst
                                      SET `System`=?,`Port`=?,`Protokoll`=?,`
                                      Dienst`=?");
16                                $sth->execute(array($systemId,$service['port'],
                                      $service['protocol'],$service['name']));
17                        }
18                }
19        }
20 }
```

5.1.4. Wahl des Scan-Bereichs

Die Eingabe des Scan-Bereichs durch den Benutzer erfolgt wie im Entwurf in 4.1.4 vorgesehen als Textfeld in der Eingabemaske einer Aufgabe. Dabei ist es möglich, mehrere verschiedene IPv4-Adressen bzw. IPv4-Subnetze anzugeben.

Bei der Initialisierung der Erkennungsmethode in Abschnitt 5.1.3 wurden die zu scannenden Adressen und Bereiche durch den Aufruf entsprechender Methoden gesetzt. Wir werden uns diese Methoden nun genauer anschauen. Die `Detection`-Klasse reicht dabei die Netzwerk-Adresse bzw. das Subnetz einfach weiter an die eigentliche Erkennungsmethode. Wie im Entwurf festgehalten, sind alle erfassten Hosts auszuwerten, falls kein Scan-Bereich angegeben wurde. Es ist also bei jedem gefundenen Host zu prüfen, ob dieser erfasst werden soll oder nicht. Dies geschieht mittels der Methode `inNetwork` (Quellcode 5.17), welche als Aufrufparameter die zu untersuchende IP-Adresse und die erlaubten Netzwerkadressen- und Subnetze hat.

Quellcode 5.17: Prüfen ob eine IPv4 / IPv6 Adresse in Liste von Netzwerkadressen und / oder -Subnetzen liegt

```
/**
 * Check if IPv4/IPv6 is in network
 * @param $ip
 * @param $networks
 * @return bool
 */
public function inNetwork($ip,$networks) {
        $ipType=$this->getIPType($ip);
        // If no networks are set, analyze all hosts
        if(!count($networks)) {
                return true;
        }
        foreach($networks as $network) {
                // Match against single IPv4 / IPv6
                if(strpos($network,'/')===false) {
                        if($ip==$network) {
                                return true;
                        }
                }
                // Match against IPv4 / IPv6 subnet
                else {
                        if(($this->getIPType($network)==$ipType) && ($this->
                            cidrMatch($ipType,$ip,$network))) {
                                return true;
                        }
                }
        }
        // Didn't match anything
        return false;
}
```

Als Hilfsfunktionen wurden hier die Methoden `getIPType` und `cidrMatch` eingesetzt. Während erstere intuitiv den Typ der angegebenen IP-Adresse angibt (Quellcode 5.18), ist letztere nicht trivial.

Quellcode 5.18: Erkennen des Typs einer IP-Adresse

```
public function getIPType($ip) {
        if(substr_count($ip,':')>1) {
                return self::IPv6;
        }
        else {
                return self::IPv4;
        }
```

```
8 }
```

Für die Erkennung, ob eine IP-Adresse in einem Subnetz in CIDR-Notation liegt, prüft die Methode `cidrMatch` zunächst um welchen Typ von IP-Adresse es sich handelt und ruft bei einer IPv4-Adresse entsprechend die Methode `cidrMatchIPv4` auf, welche in Quellcode 5.19 dargestellt ist. Der Umgang mit IPv6-Adressen wird im Abschnitt 5.1.9 weiter ausgeführt.

Quellcode 5.19: Prüfen ob eine IPv4 Adresse in angegebenem Subnetz liegt

```
1  /**
2   * Check if IPv4 is in CIDR notation mask
3   * @param $ip
4   * @param $cidr
5   * @return bool
6   */
7  function cidrMatchIPv4($ip, $cidr) {
8          // Extract mask
9          list($network, $mask) = array_pad(explode('/', $cidr), 2, NULL);
10         if( is_null($mask) ){
11                 // No mask specified: range is a single IP address
12                 $mask = 0xFFFFFFFF;
13         } elseif( (int)$mask==$mask) {
14                 // Mask is an integer: it's a bit count
15                 $mask = 0xFFFFFFFF << (32 - (int)$mask);
16         } else{
17                 // Mask is in x.x.x.x format
18                 $mask = ip2long($mask);
19         }
20         $ip = ip2long($ip);
21         $network = ip2long($network);
22         $lower = $network & $mask;
23         $upper = $network | (~$mask & 0xFFFFFFFF);
24         return $ip>=$lower && $ip<=$upper;
25 }
```

Die Prüfung, ob ein Mandant berechtigt ist, einen bestimmten Scan-Bereich auszuwerten, wird bei der Implementierung der Mandantenfähigkeit in Abschnitt 5.1.8 beschrieben.

5.1.5. Passive Erkennung von Systemen und Diensten

Bei der passiven Erkennung von Systemen und Diensten werden wir uns wie bereits im Entwurf 4.1.5 festgehalten auf die Netflow-Technik beschränken. Für diesen Zweck wird ein Prototyp implementiert, der Erkenntnisse von [Kle12] aufgreift und in den erstellten Aufgabenplaner integriert. Dabei werden die von einem Netflow-Collector aufgezeichneten Daten unter Zuhilfenahme mehrerer Werkzeuge ausgewertet und schlussendlich in eine Datenbank gespeichert. Die Auswertung beginnt, wenn der Aufgabenplaner (siehe Abschnitt 5.1.3) eine zu erledigende Aufgabe mit allen Konfigurationsdaten in der `Detection`-Klasse geladen hat und den Status auf in `Ausführung` gesetzt hat.

Die Klasse `NetFlowAnalyzer`, welche nachfolgend die Analyse der Netflow-Daten ausführt, wird über die `Detection`-Klasse aufgerufen, wie in Quellcode 5.20 aufgeführt. Als Parameter sind dabei die zu analysierenden Dateien und die zu filternden IP-Adressen bzw. -Subnetze anzugeben.

Quellcode 5.20: Aufruf der Netflow-Analyse Erkennungsmethode

```
1  public function execute() {
2      if(!isset($this->method)) {
3          throw new Exception("Detection method isn't set.");
4      }
```

```
5       if($this->method=='netflow') {
6           $netflow=new NetFlowAnalyzer($this->files,$this->networks);
7           $this->hostMapper = $netflow->analyze();
8           return $this; // fluent interface
9       }
10      throw new Exception("No action is set for detection method.");
11  }
```

Wir werden uns in den weiteren Schritten ausschließlich mit der Erkennung von Systemen und Diensten auseinandersetzen. Die Daten liegen zur Auswertung vor (siehe Abschnitt Datenerfassung 5.1.6) und der Filter des Scan-Bereichs wird im dortigen Abschnitt 5.1.4 weiter ausgeführt.

Zunächst wird für jede Datei eine separate Analyse gestartet, das heißt es wird jeweils die Methode `parse` (Quellcode 5.21) mit dem absoluten Namen der auszuwertenden Datei als Parameter `$file` aufgerufen. Bei der Auswertung wird jede Zeile der Datei für sich analysiert. Die erste Zeile wird dabei übersprungen, da dort allgemein die Spaltentitel stehen.

Quellcode 5.21: Zeilenweises Parsen der Netflow-Daten

```
1  /**
2   * Parse each line of flow dump file
3   * @param $file
4   */
5  public function parse($file) {
6      $handle = fopen($file,"r");
7      if ($handle) {
8          $count=0;
9          while (($line = fgets($handle)) !== false) {
10             $count++;
11             if($count==1) { continue; }
12             $line = str_replace('->','',$line);
13             $line = str_replace(chr(1),' ',$line);
14             $line = preg_replace('/\s\s+/',' ', $line);
15             if($line[0]!=2) {
16                 continue;
17             }
18             list($date, $time, $duration, $protocol, $source, $target, $packets
                   ) = explode(" ", $line);
19             list($sourceIP, $sourcePort) = $this->splitHost($source);
20             list($targetIP, $targetPort) = $this->splitHost($target);
21
22             // Detect host with source IP
23             $this->detectHost($sourceIP);
24
25             // Detect services
26             $this->detectService($sourceIP,$sourcePort,$targetIP,$targetPort,
                   $protocol);
27
28             // Detect OS
29             $this->detectOS($sourceIP,$sourcePort,$targetIP,$targetPort,
                   $protocol);
30         }
31     }
32 }
```

Pro Zeile sind in einem ersten Schritt die zu erfassenden Daten aufzutrennen. Die Spaltenerkennung orientiert sich an der Standardausgabe von `nfdump` und ist sowohl kompatibel zu IPv4- als auch zu IPv6-Daten. Die Separierung erfolgt durch Leerzeichen. Dabei gibt es die Besonderheit, dass beliebig viele Leerzeichen auch als nur ein Trennsymbol jeweils gelten. Bei IPv6-Daten ist ein zusätzliches Steuerzeichen zu entfernen. Den erkannten Spalten werden entsprechende Bezeichner zugewiesen, die für die weitere Bearbeitung der Zeile gelten.

Die Standard-Spalte `$source` beinhaltet sowohl die Host-Adresse als auch den Port. Zur Auftrennung dieser beiden Informationen wird eine separate Funktion `splitHost` eingesetzt, welche die verschiedenen Repräsentationen von IPv4- und IPv6-Adressen unterstützt und im Abschnitt IPv6-Unterstützung in 5.1.9 genauer erläutert wird. Damit liegen alle Daten vor und die eigentliche Auswertung kann initiiert werden.

Quellcode 5.22: Einfache Host-Erkennung

```
/**
 * Detect host if entry with source IP is given
 * @param $sourceIP
 */
public function detectHost($sourceIP) {
    if($this->inNetwork($sourceIP,$this->networks) && ($this->isAllowed(
        $sourceIP))) {
        $this->hostMapper->addHost($sourceIP);
        return true;
    }
    return false;
}
```

Als erstes wird durch den Aufruf der Methode `detectHost` (Quellcode 5.22) gemeldet, dass der Host aktiv ist. Dies ergibt sich allein durch die Tatsache, dass es einen Flow-Eintrag gibt, der als Absender den Host hat. Für eine Meldung ungeeignet hingegen wäre hingegen die Zieladresse, da dann auch Hosts als aktiv markiert werden, die es vielleicht gar nicht sind. Es wird außerdem bei jeder Erkennung geprüft, ob das erkannte System überhaupt zu erfassen ist oder nicht. Diese Prüfung erfolgt durch den Aufruf von der Methode `inNetwork`, welche in Abschnitt 5.1.4 genauer beschrieben wurde. Zusätzlich muss überprüft werden, ob der Ersteller der Aufgabe dazu berechtigt ist, die erkannte IP-Adresse auszuwerten. Dies geschieht durch den Aufruf der Methode `isAllowed`, welche in Abschnitt 5.1.8 genauer erläutert wird.

Die Ergebnisse werden dabei in einer Instanz der Klasse `HostMapper` (vollständig in Anhang D.4 dargestellt) gespeichert. Diese verfügt über den jeweils aktuellen Ergebnisstand unabhängig von einer Erkennungsmethode und kann deshalb bei einer Integration weiterer Verfahren wiederverwendet werden. In der Klasse wird beim Eintragen eines neuen erkannten Hosts, Betriebssystems oder Dienstes jeweils geprüft, ob es bereits einen solchen Eintrag gibt, womit Duplikate effizient vermieden werden können. Sie stellt zudem Hilfsmethoden wie ein Nachschlageservice der offiziellen IANA-Belegung eines Ports zur Verfügung.

Als nächstes wird durch den Aufruf der Methode `detectService` (Quellcode 5.23) versucht, anhand der Absender- und Empfängerports festzustellen, ob ein Systemdienst darauf läuft. Dabei werden zunächst wieder nur ausgehende Verbindungen untersucht, da sonst nicht sichergestellt werden kann, dass das Zielsystem auch wirklich auf diesem Port sendet. Schließlich könnte die Anfrage auch einfach ins Leere gegangen sein.

Quellcode 5.23: Portbasierte Diensterkennung

```
/**
 * Detect service
 * @param $sourceIP
 * @param $sourcePort
 * @param $targetIP
 * @param $targetPort
 * @param $protocol
 */
public function detectService($sourceIP,$sourcePort,$targetIP,$targetPort,
    $protocol) {
    global $config;
    // Detect via source port, so sender sends really via this port
```

```
12      if(($sourcePort<1024) || $this->inServiceWhitelist($sourcePort,$protocol))
             && !$this->inServiceBlackList($sourcePort,$protocol) && $this->
            inNetwork($sourceIP,$this->networks) && ($this->isAllowed($sourceIP))
            && (!$config['noSystemPortToSystemPortServices'] || ($targetPort>=1024)
            )) {
13          $this->hostMapper->addService($sourceIP,$sourcePort,$protocol);
14      }
15      // Detect via target port which is accessed (can generate a lot of false
            positives!)
16      if($config['detectionViaTargetPorts']) {
17          if(($targetPort<1024) || $this->inServiceWhitelist($targetPort,
                $protocol)) && !$this->inServiceBlackList($targetPort,$protocol) &&
                 $this->inNetwork($targetIP,$this->networks) && ($this->isAllowed(
                $targetIP)) && (!$config['noSystemPortToSystemPortServices'] || (
                $sourcePort>=1024))) {
18              $this->hostMapper->addService($targetIP,$targetPort,$protocol);
19          }
20      }
21  }
```

Wie in Quellcode 5.23 dargestellt ist, werden zunächst nur System-Ports kleiner gleich dem Port 1023 analysiert. Zusätzlich wird aber mittels einer Prüfung ermittelt, ob der Port in einer sogenannten Whitelist (Quellcode 5.24) steht. Falls in der Konfiguration explizit angegeben wurde, dass beispielsweise zusätzlich der Port 3306 (klassischerweise ein MySQL DBMS) zu untersuchen ist, wird dieser zusätzlich berücksichtigt.

Quellcode 5.24: Definition zusätzlicher zu erkennender Dienste

```
1  $config['services']['whiteList'] = array(
2      '3306/TCP' => 'MySQL (TCP)',
3      '3306/UDP' => 'MySQL (UDP)',
4      '5432/TCP' => 'PostgreSQL (TCP)',
5      '5432/UDP' => 'PostgreSQL (UDP)',
6      '3389/TCP' => 'Remote Desktop Protocol (TCP)',
7  );
```

Neben der Whitelist gibt es auch das Gegenteil davon, die Blacklist (Quellcode 5.25). In dieser sind alle zu ignorierenden Dienste aufgeführt. So lässt sich beispielsweise der Echo-Dienst von der Analyse ausnehmen. Falls die Bedingungen erfüllt sind, wird der Dienst an den `HostMapper` gemeldet und damit in das Ergebnis eingetragen.

Quellcode 5.25: Definition von nicht zu erkennenden Diensten

```
1  $config['services']['blackList'] = array(
2      '0/ICMP' => 'ICMP',
3      '7/TCP' => 'Echo',
4      '7/UDP' => 'Echo',
5  );
```

Durch ein optionales Setzen der Variable `$config['detectionViaTargetPorts']` in der Konfigurationsdatei `application/config.php` kann zusätzlich die Erkennung auf Basis von Zielports aktiviert werden. Diese Erkennung ist aber aufgrund oben genannter Gründe fehlerbehaftet und sollte deshalb nur zu Testzwecken genutzt werden.

Als letztes führen wir noch eine Erkennung des Betriebssystems durch den Aufruf von `detectOS` aus. Diese Methode (Quellcode 5.26) prüft, ob eine Verbindung zu einem Update-Server des Herstellers des Betriebssystem aufgebaut wurde. Wenn die IP-Adresse des Zielservers gleich einem in der Konfiguration hinterlegten Update-Server entspricht, wird als Betriebssystem für diesen Host der dort hinterlegte Namen gemeldet. Es werden dabei nur Verbindungen, die an bestimmte Ports gerichtet sind, geprüft. Diese Ports können in der Konfiguration durch das Setzen der Variable `$config['OSUpdatePorts']` definiert werden.

Voreingestellt sind die Ports 80 und 443, welche von den Herstellern der Betriebssysteme standardmäßig genutzt werden, damit die Update-Anfragen nicht von einer Firewall geblockt werden. Da die Eingabe von Subnetzen bei den verfügbaren Update-Servern ermöglicht wurde, muss für eine Zieladresse jeweils geprüft werden, ob diese in einem Subnetz liegt oder nicht. Hierfür nutzen wir die bereits im Abschnitt 5.1.4 entwickelte und vorgestellte Methode `inNetwork`. Auch muss wieder geprüft werden, ob der Benutzer, der die Aufgabe erstellt hat, berechtigt ist, diese IP-Adresse zu analysieren. Dies geschieht durch den Aufruf der Methode `isAllowed`.

Quellcode 5.26: Erkennung des Betriebssystems durch Update-Mechanismus

```
/**
 * Detect OS via update server
 * @param $sourceIP
 * @param $sourcePort
 * @param $targetIP
 * @param $targetPort
 * @param $protocol
 */
public function detectOS($sourceIP,$sourcePort,$targetIP,$targetPort,$protocol)
     {
    global $config;
    if(array_key_exists($targetPort,$config['OSUpdatePorts']) && ($this->
        inNetwork($sourceIP,$this->networks)) && ($this->isAllowed($sourceIP)))
         {
        foreach($config['OSUpdateServers'] as $network => $os) {
            if($this->inNetwork($targetIP,array($network))) {
                $this->hostMapper->addOS($sourceIP,$os);
            }
        }
    }
}
```

5.1.6. Erfassung von Daten

Bei der Erfassung der Daten wird die im Entwurf 4.1.6 vorgestellte Hybridlösung gewählt, was auch in der Abbildung 5.4 auf Seite 61 deutlich wird. Das heißt, die Daten können entweder über die Weboberfläche hochgeladen werden oder es können beliebig viele Datei- oder Ordnerpfade angegeben werden, in dem die auszuwertenden Dateien liegen.

Lädt ein Benutzer eine Datei hoch, wird diese mit einem neuen und zufällig generierten Namen, der nur einmal existiert, in ein Uploadverzeichnis auf den Server kopiert. Der Dateinamen, der einmalig sein soll, wird über eine Zufallszahl generiert und für diese generierte Zahl wird geprüft, ob eine Datei mit diesem Namen im Ordner bereits existiert. Falls ja, wird solange ein zufälliger Dateiname generiert, bis ein freier gefunden ist. Das Uploadverzeichnis kann in der Konfiguration über die Variable `$config['uploadPath']` beliebig gesetzt werden. Für den Zugriff auf die Dateien müssen ausreichend Lese- und Schreibrechte gewährt werden. Nach dem Upload der Datei wird der Dateipfad in das Datenbankfeld `Daten` gespeichert. Der Ablauf für beide möglichen Fälle wird in Quellcode 5.27 dargestellt.

Quellcode 5.27: Verarbeitung der verschiedenen Optionen in der Datenerfassung

```
// Data path given, no action is required
if($_POST['data_transfer']=='path') {
    $dataPath=$_POST['file_path'];
}
// File upload and set data path to uploaded file
else {
```

```
7       // Check for free file name
8       do {
9           $dataPath = $config['uploadPath'].mt_rand();
10      }
11      while(file_exists($dataPath));
12      move_uploaded_file($_FILES['file_upload']['tmp_name'],$dataPath);
13  }
```

Gibt der Benutzer einen oder mehrere Datei- bzw. Ordnerpfade an, werden diese ebenfalls in das Datenbankfeld *Daten* gespeichert. Bei der Ausführung der Aufgabe erkennt das Programm automatisch, ob es sich jeweils um eine Datei oder um einen Ordner (Quellcode 5.28) handelt. Wenn die Daten nicht geöffnet werden können oder leer sind, wird bei der Ausführung der Aufgabe ein Fehler-Status mit einem Fehlerhinweis gesetzt.

Quellcode 5.28: Hinzufügen einer Datei in Dateiliste

```
1   /**
2    * Add file to files list
3    * @param $file
4    */
5   public function addFile($file) {
6       if(!is_file($file)) {
7           throw new Exception($file." is not a file.");
8       }
9       $this->files[] = $file;
10      return true;
11  }
12
13  /**
14   * Scan folder for files and add them to files list
15   * @param $folder
16   */
17  public function addFolder($folder) {
18      if(!is_dir($folder)) {
19          throw new Exception($folder." is not a folder.");
20      }
21      $files = scandir($folder);
22      if(is_array($files)) {
23          foreach($files as $file) {
24              $file=$folder.'/'.$file;
25              if(($file == '.') || ($file == '..')) {
26                  continue;
27              }
28              elseif(!is_file($file)) {
29                  continue;
30              }
31              else {
32                  $this->addFile($file);
33              }
34          }
35      }
36      return true;
37  }
```

Die Daten sind dann für die Auswertung durch eine Erkennungsmethode vollständig erfasst. Jedoch ist für die Netflow-Analyse je nach vorliegendem Datenformat noch eine Umwandlung notwendig. So speichert das Werkzeug `nfdump` die Flows standardmäßig in einer binären Datei, welche für eine Auswertung in ein textbasiertes Format umgewandelt werden muss. Es muss also geprüft werden, ob die vorliegende Datei eine noch zu konvertierende Binärdatei ist oder bereits eine Textdatei. Diese Prüfung ist in Quellcode 5.29 dargestellt.

Quellcode 5.29: Prüfung ob Datei im Binär- oder Textformat vorliegt

```
1   /**
```

```
 * Check if file is text
 * @param $s
 * @return bool
 */
function isTextFile($file) {
    $fp = fopen($file,'rb');
    flock($fp,LOCK_SH);
    $s = fread($fp,512);
    fclose($fp);
    $characters = array_merge(array_map('chr',range(32,127)),array("\012","\015
        ","\t","\b"));
    if(strpos($s,"\0") === true)
        return false;
    if(!$s)
        return true;
    $t = $s;
    foreach($characters as $character) {
        $t = str_replace($character,'',$t);
    }
    if(strlen($t) / strlen($s) > 0.3)
        return false;
    return true;
}

/**
 * Check if netflow data is printed in txt file
 * @param $file
 * @return string
 */
public function checkForTxtFormat($file) {
    // if file is in binary format (e.g. nfdump), try to convert it
    if(!$this->isTextFile($file)) {
        return $this->netflowToTxt($file);
    }
    return $file;
}
```

Wenn eine Datei im binären Format vorliegt, muss diese durch einen Aufruf von `nfdump` in eine lesbare Datei umgewandelt werden. Diese Funktion wird nur auf Linux-Systemen unterstützt, da `nfdump` nicht für andere Plattformen verfügbar ist und lässt sich deshalb in der Konfiguration deaktivieren. Für jede binäre Datei muss eine temporäre Datei angelegt werden, in die die Ergebnisse in Textform geschrieben werden. Diese temporäre Datei wird in das Uploadverzeichnis des Programms gespeichert und nur während der Ausführung beibehalten, wird also vor Beendigung der Aufgabe wieder gelöscht. Nach der Umwandlung der Binärdaten, welche in Quellcode 5.30 gezeigt wird, wird der Name der temporären Datei für die eigentliche Netflow-Analyse zurückgegeben.

Quellcode 5.30: Umwandlung von binärer nfdump Datei in Textdatei

```
/**
 * Convert netflow to txt file (nfdump required)
 * @param $file
 * @return string
 */
public function netflowToTxt($file) {
    global $config;
    if(!$config['nfdump']) {
        throw new Exception("Conversion from nfdump format to txt file isn't
            activated (maybe Windows platform).");
    }
    // Check for free file name
    do {
        $filename = $config['uploadPath'].mt_rand();
```

```
14      }
15      while(file_exists($filename));
16
17      $this->tmpFiles[]=$filename;
18
19      // maybe integrate network filters here
20      system($config['nfdump'].' -o "fmt:%ts %td %pr %sap -> %dap %pkt %byt %fl"
            -r '.$file.' > '.$filename);
21      return $filename;
22  }
```

Um die Anzahl von Netflow-Dateien zu verringern und damit während der Auswertung Zeit zu sparen, da jeweils weniger einzelne Dateien konvertiert werden müssen, empfiehlt es sich in kleineren Netzen, das Intervall für die Generierung einer jeweils neuen Aufzeichnungsdatei (`nfcapd`-Datei) heraufzusetzen. Standardmäßig ist dieses Intervall auf fünf Minuten (= 300 Sekunden) gesetzt und kann über den Parameter `-t` verändert werden.

5.1.7. Regelmäßiges Scanning

Das beschriebene System implementiert das geforderte regelmäßige Scanning über das Feld `Wiederholen`. Dieses Feld kann bei jeder Aufgabe gesetzt werden und bietet drei Auswahlmöglichkeiten: `täglich`, `wöchentlich` und `monatlich`. Beim Ausführen einer Aufgabe wird automatisch geprüft, ob das Feld gesetzt ist. Falls es gesetzt ist, wird eine neue Aufgabe mit den gleichen Eigenschaften aber einem neu berechneten Ausführungsdatum erstellt (Quellcode 5.31). Für das Rechnen mit Daten wird die in PHP 5.2 eingeführte Klasse `DateTime` eingesetzt, welche einfaches Addieren und Subtrahieren von Zeitintervallen ermöglicht.

Quellcode 5.31: Duplizieren der aktuellen Aufgabe als neue wiederkehrende Aufgabe

```
1  $executingDate=new DateTime($task['Ausfuehrungsdatum']);
2  switch($task['Wiederholen']) {
3      case 'daily':
4          $executingDate->modify('+1 day');
5          break;
6      case 'weekly':
7          $executingDate->modify('+1 week');
8          break;
9      case 'monthly':
10         $executingDate->modify('+1 month');
11         break;
12     default:
13         $database->insertIntoHistory($task['ID'], 'error', 'Falscher
               Wiederholen-Modus gefunden, Aufgabe abgebrochen');
14         continue;
15         break;
16 }
17 $sth = $database->prepare("INSERT INTO Aufgabe SET `Name`=?,Erkennungsmethode
       =?,Ausfuehrungsdatum=?,Wiederholen=?,`Scan-Bereich`=?,Daten=?");
18 $sth->execute(array($task['Name'], $task['Erkennungsmethode'], $executingDate->
       format('Y-m-d H:i:s'), $task['Wiederholen'], $task['Scan-Bereich'], $task['
       Daten']));
19 $taskId=$database->lastInsertId();
20 $sth = $database->insertIntoHistory($taskId, 'waiting', 'Wiederkehrende Aufgabe
       angelegt');
```

Dabei wird der Status automatisch auf `Warten auf Ausführung` gesetzt, damit die Aufgabe zum definierten Zeitpunkt ohne weitere Benutzerinteraktion ausgeführt werden kann.

5.1.8. Mandantenfähigkeit

Für die Unterstützung von Mandanten ist neben der Zuordnung von Aufgaben an einen bestimmten Benutzer, welche trivial ist, eine Zugriffskontrolle erforderlich. Diese wird im Folgenden erarbeitet. Im letzten Unterpunkt werden noch die zusätzlichen Funktionen in der Administration, unter anderem auch die Benutzerverwaltung, vorgestellt.

Identifikation und Authentisierung

Der Benutzer gibt seine Login-Daten in die dafür vorgesehene Anmeldemaske (siehe Abbildung 4.14) ein. Identifikationsmerkmal ist dabei seine E-Mailadresse. Er authentisiert sich durch Klick auf den Button `Anmelden`.

Authentifizierung

Nach dem Absenden des Anmeldeformulars werden die eingegebenen Daten geprüft und der Benutzer authentifiziert (siehe Quellcode 5.32). Das eingegebene Passwort wird dabei um eine in der Konfiguration hinterlegten Zeichenkette (dem sogenannten Salt) vor Eingabe in die Hashfunktion `SHA512` ergänzt. In einem ersten Schritt wird dann geprüft, ob die Kombination `E-Mail` plus `Passwort` im System gespeichert ist und der Benutzer so eine korrekte Eingabe gemacht hat. In einem weiteren Schritt wird dann geprüft, ob die IP-Adresse des Benutzers als erlaubte IP-Adresse in seinem Profil hinterlegt ist.

Quellcode 5.32: Authentifizierung eines Benutzers anhand E-Mail, Passwort und IP

```
/**
 * Authenticate user with email and ip
 * @param $email
 * @param $password
 * @param $ip
 * @return bool
 */
public function authenticate($email,$password,$ip) {
    global $database, $router, $config;
    usleep(500000); // slow down process for 0.5s
    $query=$database->prepare("SELECT * FROM Benutzer WHERE Email=:mail AND
        Passwort=:password");
    $query->bindParam(':mail',$email);
    $hashed=hash('sha512',$password.$config['salt']);
    $query->bindParam(':password',$hashed);
    unset($hashed);
    $query->execute();
    $user=$query->fetch();
    if(!$user) {
        return false;
    }
    elseif(empty($ip) || (strpos($user['IP'],$ip)===false)) {
        return false;
    }
    else {
        $_SESSION['user_id']=$user['ID'];
        $router->forward('start');
        return true;
    }
}
```

Durch das Erzeugen einer Sitzung und das Setzen der Benutzer-`ID` kann der Benutzer bei allen weiteren Anfragen innerhalb einer gewissen Zeitspanne erneut authentifiziert werden.

Dies ist aufgrund der Zustandslosigkeit der Verbindungen bei Webanwendungen erforderlich. Bei einem erneuten Aufruf wird die Sitzung dann entsprechend wiederhergestellt.

Ein Benutzer wird nach einer gewissen Zeitspanne, welche in der Konfiguration der Skriptsprache PHP verändert werden kann, automatisch abgemeldet, indem die zugehörige Sitzung gelöscht wird. Standardmäßig wird die Sitzung nach 1440 Sekunden gelöscht. Führt der Benutzer innerhalb dieser Zeit keine neue Aktion mehr aus, wird er abgemeldet. Möchte er erneut eine Aktion ausführen, muss er sich erneut authentisieren. Ein Benutzer kann sich allerdings auch selbst abmelden, die Sitzung wird dann mit sofortiger Wirkung gelöscht und er wird auf die Anmeldeseite weitergeleitet.

Autorisierung

Die Autorisierung wird mithilfe einer rollenbasierten Zugriffskontrolle implementiert. Dabei sind die Berechtigungen additiv und der jeweilige Benutzer mit einer höheren Rolle erbt automatisch die Rechte aller niedrigeren Rollen. Die im System vergebenen Berechtigungen pro Rolle sind in Quelltext 5.33 dargestellt. Ein Gast bekommt dabei die Rolle `default`, damit er die Anmeldemaske und die Lizenzseite aufrufen kann.

Quellcode 5.33: Access Control List des Systems

```
// User acl - which role can access which page (additive roles: privileges for
    an administrator = default + standard + administrator
$config['acl'] = array(
    'default'=>array(
        'login',
        'license',
    ),
    'standard'=>array(
        'start',
        'task',
        'result',
        'history',
        'logout',
    ),
    'administrator'=>array(
        'admin',
        'user',
        'planner',
    ),
);
```

Bei jedem Aufruf prüft der `Router` (Komponente ist in Abschnitt 5.1.1 beschrieben), ob der aktuell angemeldete Benutzer bzw. Gast berechtigt ist, diesen Aufruf auszuführen. Die Autorisierung ist in Quellcode 5.34 dargestellt. Die Rollen lassen sich dabei in der Konfiguration beliebig erweitern.

Quellcode 5.34: Autorisierung bei Seitenaufruf

```
global $session, $config;
$privileges = $config['acl']['default'];
if($session->getRole()) {
    $privileges = array_merge($privileges,$config['acl']['standard']);
    if($session->getRole()=='administrator') {
        $privileges = array_merge($privileges,$config['acl']['administrator']);
    }
}
if (!in_array($pageName, $privileges)) {
    $pageName = 'not-found';
}
```

Beim Anlegen einer Aufgabe wird zusätzlich automatisch der Benutzer erfasst, sodass die Aufgabe von keinem anderen Benutzer mit der Rolle `Standard` eingesehen oder verändert werden kann.

Da der Scan-Bereich pro Benutzer eingeschränkt ist, muss dies beim Ausführen einer Aufgabe überprüft werden. Beim Entdecken eines Hosts, eines Dienstes oder eines Betriebssystems ist deshalb jeweils zu prüfen, ob der Benutzer, der die Aufgabe erstellt hat, das Ergebnis erhalten darf. Dies geschieht durch die Methode `isAllowed`, welche in Quelltext 5.35 dargestellt ist. Dabei wird für eine erfasste IP geprüft, ob diese in irgendeinem der erlaubten Scan-Bereiche des Benutzers liegt.

Quellcode 5.35: Prüfung ob Benutzer ein System analysieren darf

```
/**
 * Check if scanning of a given host is allowed
 * @param $ip
 * @return bool
 */
public function isAllowed($ip) {
    $ipType=$this->getIPType($ip);
    foreach($this->allowedNetworks as $network) {
        // Match against single IPv4 / IPv6
        if(strpos($network,'/')===false) {
            if($ip==$network) {
                return true;
            }
        }
        // Match against IPv4 / IPv6 subnet
        else {
            if(($this->getIPType($network)==$ipType) && ($this->cidrMatch(
                $ipType,$ip,$network))) {
                return true;
            }
        }
    }
    return false;
}
```

Administration

Administratoren hingegen können alle Benutzer und Aufgaben betrachten. Sie können Benutzer bearbeiten, nicht jedoch Aufgaben. Allerdings können sie Aufgaben anderer Benutzer löschen. Es gibt dabei eine spezielle Seite Benutzer (Abbildung 5.7), auf der alle im System angelegten Benutzer angezeigt werden. Eine selbstständige Registrierung von Benutzern ist nicht vorgesehen, ein Administrator muss diese jeweils anlegen.

Bei der Bearbeitung von Benutzern (Abbildung 4.15) kann der Administrator alle Daten eines Benutzers ändern. Möchte er dabei das Passwort eines Benutzers nicht ändern, kann er das Feld entsprechend leer lassen.

Zusätzlich haben Administratoren Zugriff auf eine spezielle Administrations-Seite (siehe Abbildung 5.8), worüber die Systemstatistiken und weitere Funktionen wie das Löschen aller vorhandenen Ergebnisse zugänglich sind. Außerdem wird an dieser Stelle auch die geladene Konfiguration des Systems als Werkzeug zum Diagnostizieren von Fehlern angezeigt.

5.1.9. IPv6-Unterstützung

Eine IPv6-Unterstützung vom beschriebenen System ist allein schon aufgrund des Anwendungsszenarios vorgesehen. Verschiedene Programmteile müssen dabei IPv6-Adressen und

Abbildung 5.7.: Anzeigen aller im System angelegten Benutzer

Abbildung 5.8.: Administrationsseite mit erweiterten Funktionen und Konfiguration

-Subnetze unterstützen.

Dateneingabe Bei der Eingabe und Verwaltung der Daten sind grundsätzlich keine Grenzen gesetzt. So sind die Datenbankfelder in der Relation `System` für IP-Adressen von der zulässigen Spaltengröße her so ausgelegt, dass sie auch IPv6-Adressen erfassen können. Auch die Eingabe des Scan-Bereiches bei einer Aufgabe oder die Darstellung der Hosts sind jeweils kompatibel.

Datenerfassung Bei der Datenerfassung müssen die IPv6-Daten aufgezeichnet werden können. Das ist beispielsweise bei einem Portscanner wie Nmap kein Problem, da dieser bereits seit langem IPv6 unterstützt. Anders sieht es bei der Netflow-Technik aus: Das noch weit verbreitete Netflow v5 Format unterstützt ausschließlich IPv4-Hosts. Für die Erfassung von IPv6-Hosts ist also ein Netflow Exporter notwendig, der das aktualisierte Netflow v9 Format beherrscht, welches IPv6 vollständig unterstützt.

Datenverarbeitung Bei der Verarbeitung der Daten müssen diese von der Erkennungsmethode erkannt werden. Bei der vorgestellten Netflow-Analyse in Abschnitt 5.1.5 geschieht

dies durch eine jeweilige Erkennung, um welchen Typ von IP-Adresse es sich handelt. Für die Unterstützung von IPv6-Subnetzen ist es allerdings zusätzlich nötig, dass geprüft werden kann, ob eine IPv6-Adresse in einem IPv6-Subnetz in CIDR-Notation liegt. Dieses Ziel wird durch die Implementierung einer zusätzlichen Methode `cidrMatchIPv6` erreicht, wie sie in Quellcode 5.36 dargestellt ist.

Quellcode 5.36: Erkennung ob eine IPv6-Adresse in IPv6-Subnetz liegt

```
/**
 * Inet to Bits
 * @param $inet
 * @return string
 */
function inetToBits($inet) {
        $unpacked = unpack('A16', $inet);
        $unpacked = str_split($unpacked[1]);
        $binaryIP = '';
        foreach ($unpacked as $char) {
                $binaryIP .= str_pad(decbin(ord($char)), 8, '0', STR_PAD_LEFT);
        }
        return $binaryIP;
}

/**
 * Check if IPv6 is in CIDR notation mask
 * @param $ip
 * @param $cidr
 * @return bool
 */
function cidrMatchIPv6($ip, $cidr) {
        $ip = inet_pton($ip);
        $binaryIP=$this->inetToBits($ip);
        list($net,$maskBits)=explode('/',$cidr);
        $net=inet_pton($net);
        $binaryNet=$this->inetToBits($net);
        $IPNetBits=substr($binaryIP,0,$maskBits);
        $netBits =substr($binaryNet,0,$maskBits);
        if($IPNetBits!==$netBits) {
                return false;
        } else {
                return true;
        }
}
```

Zusätzlich war es bisher einfach möglich, den Port einer IP-Adresse beispielsweise bei `10.0.0.1:80` anhand des Doppelpunktes als Trennzeichen zu erkennen. Bei einer IPv6-Adresse werden im Gegensatz dazu bereits die einzelnen Blöcke mit dem Doppelpunkt getrennt und der Port standardmäßig im Netflow v9 Format mit einem Punkt angehängt. Es ist also bei der Aufspaltung der Adresse vom Port eine Fallunterscheidung einzuführen, ob es sich dabei um eine IPv4- oder IPv6-Adresse handelt (Quellcode 5.37).

Quellcode 5.37: Fallunterscheidung IPv4/IPv6 bei Host-/Porterkennung

```
/**
 * Split IPv4/IPv6 address from port
 * @param $host
 * @return array
 */
public function splitHost($host) {
        if($this->getIPType($host)==self::IPv6) {
                return explode('.',$host);
        } else {
                return explode(':',$host);
```

```
11            }
12    }
```

Weitere Anpassungen sind für die IPv6-Unterstützung nicht notwendig, die weitere Verarbeitung erfolgt analog zu IPv4.

5.2. Nichtfunktionale Anforderungen

Nach der Beschreibung der funktionalen Anforderungen wird im Folgenden auf die Implementierung der nichtfunktionalen Anforderungen eingegangen.

5.2.1. Zuverlässigkeit

Ähnlich zum Entwurf werden die drei Unterpunkte der Zuverlässigkeit im Bezug auf die Implementierung erneut analysiert:

Systemreife Selbstverständlich wurde bei der Implementierung darauf geachtet, möglichst wenig Fehlerzustände zu verursachen. Jedoch kann aufgrund der Größe und der Komplexität einer solchen Anwendung von einem Betrieb ohne zusätzliche und tiefergehende Überprüfungen des Programmcodes in sensiblen Bereichen nur abgeraten werden.

Wiederherstellbarkeit Die im Entwurf erläuterte Wiederherstellbarkeit ist durch die Speicherung der ausgewerteten Daten in einer Datenbank und durch die Sicherung des Programmcodes in einem GIT-Repository implementiert. Die auszuwertenden Rohdaten sind gegebenenfalls vom Systemadministrator zusätzlich zu sichern.

Fehlertoleranz Das System gerät bei fehlerhaften Eingabedaten nicht in einen Fehlerzustand, sondern setzt jeweils nur bei der Aufgabe, die den Fehler verursacht hat, einen Fehlerzustand. Damit Benutzer diese Fehler nachvollziehen können, wird zu jedem Fehler-Status eine genaue Beschreibung mitgeliefert. Alle fehlerhaften Aufgaben können in einem Verlauf angezeigt werden.

5.2.2. Aufbau und Handhabung

Die Weboberfläche wurde wie im Entwurf beschrieben realisiert. Dabei traten keine Probleme bei der Implementierung auf. Der Benutzer erhält Zugriff auf ein klar strukturiertes und deshalb leicht erlernbares System. Die Programmsteuerung ist typisch für eine Webanwendung. Wie auf anderen Websites auch ist eine reine Steuerung über die Tastatur möglich, wenn auch aufwändig.

5.2.3. Betrieb und Umgebung

Die im Entwurf 4.2.4 vorgestellten Voraussetzungen für den Betrieb der Software haben sich in der Implementierung bewährt und gelten deshalb uneingeschränkt weiter.

Zusätzlich ist für den CSV- bzw. PDF-Export über die Weboberfläche clientseitig eine Flash-Installation erforderlich. Falls diese nicht vorhanden ist, wird die Funktionalität automatisch ausgeblendet.

Die Konfiguration der Anwendung ist in eine Standard-Konfiguration und eine angepasste Konfiguration unterteilt. Beide Konfigurationsdateien liegen im Anwendungsverzeichnis `application`. In der Standard-Konfiguration `config.default.php` (siehe Anhang D.1) sind die für den Betrieb notwendigen Variablen mit voreingestellten Werten definiert. Möchte ein Administrator diese anpassen, kann er diese in der `config.php` überschreiben. Von einer direkten Bearbeitung der Standard-Konfiguration wird abgeraten, da in diesem Fall die Kompatibilität bei zukünftigen Updates nicht mehr gewährleistet werden kann.

5.2.4. Portierbarkeit

Der Prototyp wurde auf einem Windows-System entwickelt und in einer Linux-Umgebung getestet. Er besitzt außer den bereits im Entwurf aufgeführten Abhängigkeiten keine weiteren externen Komponenten und ist deshalb in jeder modernen Apache, MySQL und PHP-Umgebung lauffähig.

Auf Windows-Systemen ist aufgrund der Nichtverfügbarkeit einer Portierung der `nfdump`-Tools die Konfigurationsvariable `$config['nfdump']` auf den Wert `false` zu setzen. Eine Umwandlung von `nfcapd`-Dateien ist dann deaktiviert und das Programm setzt einen Fehler-Status, falls solche Dateien zur Auswertung übergeben wurden. Es können also auf Systemen, die nicht auf Linux basieren, lediglich Dateien verarbeitet werden, die von `nfdump` im Textformat exportiert wurden.

5.2.5. Sicherheitsanforderungen

Bei der Implementierung des Prototypen wurde auf die im Entwurf vorgestellten Gefahren berücksichtigt und entsprechende Schutzmaßnahmen integriert, von denen ein Auszug nachfolgend erläutert wird.

Schutz vor SQL-Injections

Sämtliche Eingabedaten werden durch verschiedene Verfahren geprüft und gegebenenfalls unschädlich gemacht, bevor sie in einen SQL-Befehl eingefügt werden. So wird beim einfachen Auswählen und Laden einer Aufgabe die ID mit `(int)` zum Datentyp Integer umgewandelt, wie in Quellcode 5.38 gezeigt wird.

Quellcode 5.38: SQL-Injections durch sicheres Type Casting verhindern

```
1 $task = $database->query("SELECT *,(SELECT Status FROM Verlauf WHERE Verlauf.
      Aufgabe=Aufgabe.ID ORDER BY ID DESC LIMIT 1) AS Status FROM Aufgabe WHERE
      ID='".(int)$_GET['id']."')->fetch();
```

Zusätzlich wird beim Einfügen und Ändern von Einträgen stets die PDO eigene `prepare`-Funktionalität (Quellcode 5.39) genutzt, welche sämtliche Eingabewerte so maskiert, dass diese keine SQL-Injection mehr verursachen können.

Quellcode 5.39: SQL-Injections durch PDO-Funktionalität verhindern

```
1 $sth = $database->prepare("INSERT INTO Aufgabe SET `Name`=?,Erkennungsmethode
      =?,Ausfuehrungsdatum=?,Wiederholen=?,`Scan-Bereich`=?,Daten=?");
2 $sth->execute(array($_POST['name'], $_POST['detectionMethod'], $_POST['
      executionDate'], $_POST['repeatingMode'], $_POST['scanSector'], $dataPath))
      ;
```

Nicht-persistentes Cross-Site-Scripting

Aufgrund des dynamischen Einbindens von Programmteilen besteht die Gefahr einer RFI (Remote File Inclusion). Die Sicherheitslücke ermöglicht es einem Angreifer, unkontrolliert Programmcode in den Webserver einzuschleusen und dort auszuführen. So wird im beschriebenen System beim Aufruf `?page=start` die Datei `start.php` im Verzeichnis `application/pages/` ausgeführt. Ein Angreifer könnte also über den Parameter `page` eine beliebige PHP-Datei auf dem System ausführen, also sich auch beispielsweise die Datenbankkonfiguration mit `?page=../config.php` ausgeben lassen. Wir begegnen diesem durch mehrere Schutzmaßnahmen. Zum einen wird der Eingabeparameter `page` in 5.40 gefiltert, sodass außer Groß- und Kleinbuchstaben sowie Ziffern keine weitere Eingabe möglich ist. Dadurch ist es nicht mehr möglich, den Pfad zu wechseln.

Quellcode 5.40: Filter von page-Parameter

```
1 private function filterPageName($pageName) {
2         if(!empty($pageName)) {
3             $pageName = preg_replace("/[^A-Za-z0-9_]/", "", $pageName);
4         }
5         else {
6             $pageName=false;
7         }
8         return $pageName;
9     }
```

Zum anderen wird in der Konfiguration (Quellcode 5.41) definiert, welche Seitenaufrufe überhaupt nur erlaubt sind.

Quellcode 5.41: Erlaubte Seiten werden explizit konfiguriert

```
1 // Definition of allowed pages
2 $config['pages'] =
3     array(
4         'start' => 'Welcome',
5         'task' => 'Tasks',
6         'result' => 'Results',
7         'user' => 'Users',
8     );
```

Bei jedem Aufruf wird dann in Quellcode 5.42 geprüft, ob der angefragte Seitenname erlaubt ist. Falls ja, wird die angefragte Seite beibehalten. Im anderen Fall wird der Seitenname auf `not-found` gesetzt und dem Benutzer damit eine Fehlerseite angezeigt.

Quellcode 5.42: Überprüfung des Vorhandenseins der angefragten Seite

```
1 private function checkPageName($pageName) {
2         if (!array_key_exists($pageName, $this->getPages())) {
3             $pageName = 'not-found';
4         }
5         return $pageName;
6     }
```

5.2.6. Korrektheit

Es wurden keine Funktionen oder Heuristiken implementiert, die lediglich einen zufällig gewählten Teil der Eingabedaten analysieren. Somit ist das Ergebnis bei jeder Ausführung gleich, sofern die Eingabedaten unverändert sind und der Algorithmus der Erkennungsmethode in der Zwischenzeit nicht verändert wurde. Ein Beispiel dafür kann ein Programmupdate für

eine verbesserte Dienst-Identifizierung sein. Dieses Ergebnis lässt sich durch manuelle Tests verifizieren.

6. Test und Evaluation

In diesem Kapitel wird zunächst die für die Entwicklung und zum Testen genutzte Umgebung beschrieben. Anschließend werden verschiedene Eingaben vorgestellt, um die Ergebnisse bewerten zu können.

6.1. Beschreibung der Testumgebung

Das LRZ hat im Rahmen dieser Arbeit eine Testumgebung zum Probebetrieb eines Prototyps zur Verfügung gestellt. Diese setzt sich aus folgenden Systemen zusammen, welche alle miteinander wie in Abbildung 6.1 in einem privaten Netzwerk verbunden sind:

- System 1: Debian 7.3 (dient als Master-System)
- System 2: SUSE Linux Enterprise Server 11
- System 3: Debian 7.3
- System 4: Windows 7
- System 5: Windows 2008
- System 6: Windows 2012 (deutsche Sprachversion)
- System 7: Windows 2012 (englische Sprachversion)

Dabei dient das Master-System als Analyse-System. Es übernimmt deshalb den Betrieb des in dieser Arbeit beschriebenen Softwaresystems und stellt dafür eine LAMP-Umgebung bereit. Die Installation dieser Umgebung wird in Anhang A beschrieben. Zusätzlich dient es als sogenannter Netflow-Collector (siehe Abbildung 2.6) als Empfänger von Analysedaten. Die Einrichtung des Netflow-Collectors wird in Anhang B.1 beschrieben.

Das System 2 soll keine zusätzlichen Dienste installiert haben und lediglich als SUSE Linux Enterprise Server erkannt werden. System 3 verfügt über die identische LAMP-Umgebung wie System 1.

Die Systeme 4 bis 7 haben das von Microsoft erstellte OS Windows in den verschiedensten Versionen installiert und sind zusätzlich mit folgenden Diensten ausgestattet, welche mittels der vorgestellten Methoden zu erkennen sind:

- System 4: Apache 2.4.7 und MySQL 5.5.34
- System 5: Apache 2.4.7
- System 6: FileZilla FTP Server 0.9.41
- System 7: Mercury Mail Transport System v4.62

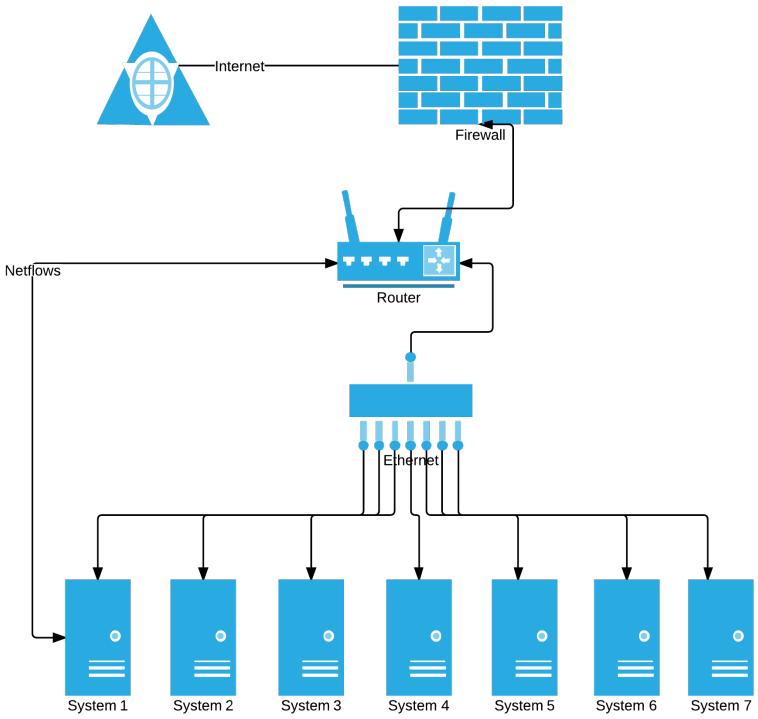


Abbildung 6.1.: Beschreibung der System- und Netzwerkumgebung

Dabei kommt das vom Apache Friends Team entwickelte XAMPP[1] zum Einsatz, welches eine schnelle Einrichtung und Konfiguration der Dienste auf Windows Systemen ermöglicht. XAMPP ist für verschiedene Betriebssysteme verfügbar und beinhaltet neben den klassischen LAMP-Diensten auch einen FTP- (FileZilla) und einen Mailserver (Mercury). Die Installation sowie die Bedienung erfolgt über eine intuitive GUI und ist selbsterklärend.

Die Systeme 4 und 5 werden während der Tests zusätzlich als einfache Windows-Systeme ohne zusätzliche Serverdienste betrieben, um eine Erkennung von Client-Systemen zu prüfen. Es werden dabei auch typische Client-Aktivitäten wie Surfen im Internet oder Up- und Download von Dateien über BitTorrent ausgeführt.

Als Client für die Serverdienste kommen die jeweils anderen Systeme zum Einsatz. So kann beispielsweise System 4 eine Website vom Webserver auf System 1 anfordern oder System 2 eine SMTP-Verbindung zum Mailserver auf System 7 aufbauen.

6.2. Auswertung

Die Aufzeichnung der Netflow-Daten erfolgte über mehrere Tage, bei der Auswertung lagen über 2.500 einzelne Netflow-Dateien vor. Das ist durchaus eine realistische Größe und unterfordert das implementierte Programm für einen ersten Test mit Sicherheit nicht. Die Netflow-Dateien werden durch Pfadangabe des entsprechenden Ordners `/var/flows/` allesamt einzeln analysiert. Der Aufwand für die Auswertung der Dateien wächst linear mit der Anzahl

[1] Projektwebsite: http://www.apachefriends.org/de/xampp.html

und Größe der Daten. Für das Erkennen des oben erläuterten Testsystems hat die Auswertung 20 Sekunden gedauert. Als Scan-Bereich ist der Bereich `10.156.200.1/28` gewählt.

Ergebnisse für Aufgabe 'Aufgabe mit Scan-Bereich Filter' (ID 7)

10 Einträge anzeigen — CSV-Export — PDF-Druckansicht

IP	Betriebssystem	Dienste
10.156.200.1		ssh (22/TCP) http (80/TCP)
10.156.200.2		Unknown (733/UDP) netconf-beep (831/UDP) Unknown (973/UDP) Unknown (1003/UDP) mdbs-daemon (800/UDP) iclcnet-locate (886/UDP) owamp-control (861/UDP) accessbuilder (888/UDP) Unknown (823/UDP) kink (910/UDP) rushd (696/UDP) Unknown (722/UDP) netrcs (742/UDP) Unknown (745/UDP) rtip (771/UDP) Unknown (915/UDP) Unknown (820/UDP) urm (606/UDP) ieee-mms-ssl (695/UDP) Unknown (1004/UDP)
10.156.200.3		
10.156.200.4	Windows	
10.156.200.5		ftp (21/TCP)
10.156.200.6	Windows	ftp (21/TCP)
10.156.200.7	Windows	ftp (21/TCP)

1 bis 7 von 7 Einträgen — Zurück 1 Nächster

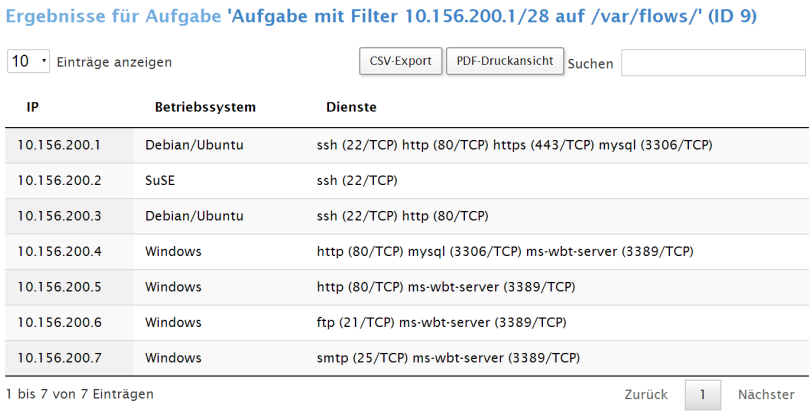
Abbildung 6.2.: Erste Ergebnisse von der Netflow-Auswertung der Test-Umgebung

Wie in Abbildung 6.2 ersichtlich, wurden zunächst einmal alle sieben Systeme identifiziert. Bei drei von vier Windows-Systemen wurde zudem festgestellt, dass es sich um Windows-Systeme handelt. Bei System 1 konnte der SSH, sowie der HTTP-Dienst ausgemacht werden. Bei den Systemen 5, 6 und 7 konnte ein jeweils ein FTP-Server festgestellt werden. Dies ist kein Fehler, da während des Testens von Firewall-Regeln auf den Windows-Servern mit dem Zu- und Abschalten eines FTP-Servers jeweils gespielt wurde. Wie erklären sich nun aber die fehlenden Dienste, welche nicht erkannt wurden? Zum einen werden Dienste mit einer Portnummer größer 1023 aufgrund der Einschränkung auf Systemports nicht erfasst. Diese müssen explizit in der sogenannten Whitelist in der Konfiguration angegeben werden, beispielsweise der Dienst MySQL auf Port 3306. Zum anderen werden Dienste allein durch deren Installation nicht immer gleich aktiv und warten zunächst einmal auf eingehende Anfragen auf ihrem Port. Wenn es aber keine eingehenden Anfragen auf diesem Port gibt, können auch keine aufgezeichnet werden. Auffallend sind zudem die vielen einzelnen Dienste auf System 2, welche auf Linux-Systemdienste hindeuten. Diese sind allerdings nicht als Serverdienste im eigentlichen Sinne zu interpretieren, sondern sie haben lediglich über diesen Port als Absender einmal eine Anfrage gestellt und wurden aus diesem Grund erkannt.

Um ein besseres Ergebnis zu erhalten werden wir im Folgenden die Konfiguration anpassen und gezielt Aktionen ausführen, damit Dienste und Betriebssysteme aktiv werden. Anschließend werden wir die Analyse erneut durchführen und die Ergebnisse vergleichen.

Durch das Einpflegen von Diensten, die nicht erkannt werden sollen, in eine Blacklist erhält man ein übersichtlicheres Ergebnis. So nehmen wir beispielsweise systemnahe Dienste wie NTP (Network Time Protocol) oder die Linux-Systemdienste vom Debian-System 2 auf. Zusätzlich kann durch das Setzen der Konfigurationsvariable `$config['noSystemPortToSystemPortServices']` auf `true` deaktiviert werden, dass Dienste erkannt werden, die von einem Port kleiner gleich 1023 zu einem anderen Port kleiner gleich 1023 senden. Damit auch die Dienste, die seither keine Verbindungen zu verzeichnen hatten, erfasst werden können, werden wir eine Mail versenden, den FTP-Server nutzen und weitere einzelne Dienste ansteuern. Außerdem werden wir manuell die Update-Routinen der Systeme anstoßen und

fehlende interne Update-Server des LRZ nachtragen.



Ergebnisse für Aufgabe 'Aufgabe mit Filter 10.156.200.1/28 auf /var/flows/' (ID 9)

10 Einträge anzeigen | CSV-Export | PDF-Druckansicht | Suchen

IP	Betriebssystem	Dienste
10.156.200.1	Debian/Ubuntu	ssh (22/TCP) http (80/TCP) https (443/TCP) mysql (3306/TCP)
10.156.200.2	SuSE	ssh (22/TCP)
10.156.200.3	Debian/Ubuntu	ssh (22/TCP) http (80/TCP)
10.156.200.4	Windows	http (80/TCP) mysql (3306/TCP) ms-wbt-server (3389/TCP)
10.156.200.5	Windows	http (80/TCP) ms-wbt-server (3389/TCP)
10.156.200.6	Windows	ftp (21/TCP) ms-wbt-server (3389/TCP)
10.156.200.7	Windows	smtp (25/TCP) ms-wbt-server (3389/TCP)

1 bis 7 von 7 Einträgen | Zurück | 1 | Nächster

Abbildung 6.3.: Aktualisierte Ergebnisse von der Netflow-Auswertung der Test-Umgebung

Alles in allem zeigt sich nach dem manuellen Ausführen von Anfragen an den Mailserver oder an den Webserver auf System 2, dass die Hosts vollständig erkannt wurden. Auch wurden die aktiven Ports, die auf bestimmte Dienste hinweisen können, zum größten Teil entdeckt. Zusätzlich wurde auf den Systemen 1 bis 3 ein offener Port 22 entdeckt, welcher auf das Netzwerkprotokoll SSH (Secure Shell) hindeutet. Tatsächlich wird über dieses Protokoll auch auf die Systeme zugegriffen.

Wie in Abbildung 6.3 deutlich wird, ist die Erkennung von Betriebssystemen grundsätzlich möglich. Es wurden alle vier Windows-Systeme als solche erkannt und es konnte auch festgestellt werden, dass es sich bei System 1 und 3 jeweils um ein Debian-System handelt. Bei der Identifizierung des Systems 2 konnte zunächst das Betriebssystem SUSE Linux Enterprise Server nicht erkannt werden, da Online-Updates nicht aktiviert waren. Nach dem Hinzufügen des entsprechenden Update-Servers und der Aktivierung der Online-Updates, konnte auch dieses Betriebssystem identifiziert werden.

Zudem wurde ursprünglich das DBMS MySQL auf keinem der Systeme erkannt. Dies war der Tatsache geschuldet, dass der Dienst auf Port 3306 läuft und deshalb aufgrund der Einschränkung auf Systemports (Ports kleiner 1024) nicht erkannt werden konnte. Abhilfe schaffte ein zusätzlicher Konfigurationseintrag in der Whitelist, sodass auch Port 3306 untersucht wird. Ebenfalls auf diese Liste hinzugefügt haben wir den Port 3389, da wir mittels RDP (Remote Desktop Protocol) auf die Windows-Systeme zugegriffen haben, was auch erkannt wurde.

Ohne Eingrenzung des Scan-Bereiches konnten zudem weitere im Netz vorhandene Systeme erkannt werden, so beispielsweise Server mit aktiviertem HTTPS, Kerberos- oder LDAP (Lightweight Directory Access Protocol)-Server und viele mehr. Prinzipiell ist festzuhalten, dass die Erkennung umso besser wird, je länger der Beobachtungszeitraum ist und / oder je größer die Netzaktivität der zu untersuchenden Systeme ist.

Wir möchten dieses Ergebnis zusätzlich durch einen aktiven Scan verifizieren. Dazu installieren wir das in Kapitel 2 vorgestellte Werkzeug Nmap und rufen dieses wie in Quellcode 6.1 dargestellt auf.

Quellcode 6.1: Kontrolle mit aktiver Erkennungsmethode Nmap

```
1 # Installieren von Nmap
```

```
apt-get install nmap

# Betriebssystem- und Diensterkennung für System 1
nmap -o -sV 10.156.200.1

Starting Nmap 6.00 ( http://nmap.org ) at 2014-03-16 18:07 CET
Nmap scan report for lxdpi-master.srv.lrz.de (10.156.200.1)
Host is up (0.000010s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 6.0p1 Debian 4 (protocol 2.0)
80/tcp  open  http    Apache httpd 2.2.22 ((Debian))
111/tcp open  rpcbind
Service Info: OS: Linux; CPE: cpe:/o:linux:kernel

# Betriebssystem- und Diensterkennung für System 4
nmap -o -sV 10.156.200.4

Starting Nmap 6.00 ( http://nmap.org ) at 2014-03-16 18:04 CET
Nmap scan report for BADWLRZ-TWEYE1.ads.mwn.de (10.156.200.4)
Host is up (0.00031s latency).
Not shown: 992 filtered ports
PORT     STATE SERVICE        VERSION
80/tcp   open  http           Apache httpd 2.4.7 ((Win32) OpenSSL/1.0.1e PHP
    /5.5.6)
135/tcp  open  msrpc          Microsoft Windows RPC
139/tcp  open  netbios-ssn
443/tcp  open  ssl/http       Apache httpd 2.4.7 ((Win32) OpenSSL/1.0.1e PHP
    /5.5.6)
445/tcp  open  netbios-ssn
2701/tcp open  sms-rcinfo?
3306/tcp open  mysql          MySQL (unauthorized)
3389/tcp open  ms-wbt-server?

MAC Address: 00:50:56:8F:0E:F8 (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Es zeigt sich, dass man mit Nmap als aktive Erkennungsmethode zu den gleichen Ergebnissen kommt wie mit dem passiven Verfahren, das auf der Netflow-Technik basiert. Zusätzlich erkennt Nmap auch Dienste wie beispielsweise Windows-Dienste auf den Ports 135 und 139, die aufgrund fehlender Verbindungsdaten mit der Netflow-Technik nicht erkannt werden konnten. Auch die beiden Betriebssysteme, Linux auf System 1 und Windows auf System 4 können mit dem Nmap-Ergebnis bestätigt werden.

6.2.1. IPv6-Hosts

Um zu zeigen, dass der Prototyp auch IPv6-Hosts und darauf laufende Dienste vollständig erkennt, wird zusätzlich eine Netflow-Datei mit Aufzeichnung von IPv6-Verkehr ausgewertet. Eine exemplarische Datei wurde vom LRZ zur Verfügung gestellt und der Analyse über den Webupload zur Verfügung gestellt.

Wie in Abbildung 6.4 ersichtlich, wurden insgesamt 119 Hosts entdeckt. Auch die Diensterkennung funktioniert augenscheinlich. Der Grund, warum die Betriebssystemerkennung nicht einwandfrei durchgeführt werden konnte, besteht darin, dass in ausschließlichen IPv6-Netzen auch IPv6-Updateserver zum Einsatz kommen. Damit die Betriebssysteme also korrekt erkannt werden können, müssen die Update-Server noch in der Liste in der Konfiguration entsprechend hinzugefügt werden.

Eine stichprobenartige Überprüfung der Hostsysteme mit einer aktiven Erkennungsmethode konnte leider nicht durchgeführt werden, da die aufgezeichneten Hosts nicht von den zur

Ergebnisse für Aufgabe 'IPv6 Auswertung' (ID 8)

10 Einträge anzeigen | CSV-Export | PDF-Druckansicht | Suchen

Datum	IP	Betriebssystem	Dienste
2014-03-06 11:13:00	2001:4ca0:0:100:0:53:1:1		domain (53/UDP)
2014-03-06 11:13:00	2001:4ca0:0:100:0:53:1:2		domain (53/UDP)
2014-03-06 11:13:00	2001:4ca0:0:101:0:53:0:4		domain (53/UDP)
2014-03-06 11:13:00	2001:4ca0:0:101:0:80:21:1		ftp (21/TCP)
2014-03-06 11:13:00	2001:4ca0:0:103::81bb:fe49		imaps (993/TCP) https (443/TCP)
2014-03-06 11:13:00	2001:4ca0:0:103::81bb:fee1		https (443/TCP)
2014-03-06 11:13:00	2001:4ca0:0:103::81bb:ff93		https (443/TCP)
2014-03-06 11:13:00	2001:4ca0:0:106:0:53:0:1		
2014-03-06 11:13:00	2001:4ca0:0:109::a9c:817		domain (53/UDP)
2014-03-06 11:13:00	2001:4ca0:0:116::a9c:621		domain (53/UDP)

1 bis 10 von 119 Einträgen | Zurück 1 2 3 4 5 … 12 Nächster

Abbildung 6.4.: Ergebnisse der IPv6-Auswertung

Verfügung gestellten Systemen erreichbar waren.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde zunächst eine Kategorisierung erstellt, bei der Hostsysteme anhand von spezifischen Eigenschaften in eine Gruppe eingeordnet werden. Anschließend wurden sowohl aktive Erkennungsmethoden wie Portscanner oder Schwachstellenscanner untersucht als auch passive Erkennungsmethoden wie die Netflow-Technik oder die DPI betrachtet und bewertet. Ausgehend von diesen Erkenntnissen wurde ein Prototyp erstellt, der eines der vorgestellten Verfahren, die Netflow-Technik, zur Erreichung der erstellten Kategorisierung von Hostsystemen implementiert. Dieses Verfahren wurde gewählt, weil es als passive Erkennungsmethode eine ressourcenschonende und unbemerkte Auswertung ermöglicht.

Dieser Prototyp wurde dabei in eine Weboberfläche eingebunden, wodurch es möglich ist, Analyse-Aufgaben anzulegen und auszuführen. Die Eingabedaten können dem Programm manuell per Webupload oder automatisch über eine Pfadangabe übergeben werden. Dabei können Auswertungsfilter wie zu untersuchende Netzwerkbereiche angegeben werden und die Aufgaben können zeitgesteuert durchgeführt werden. Zusätzlich ist es möglich, regelmäßig wiederkehrende Aufgaben anzulegen. Die Ergebnisse können übersichtlich dargestellt werden, aber auch als Roh- oder Druckdaten exportiert werden.

Das Programm ist dabei vollständig mandantenfähig, das heißt, dass jeder Benutzer eine eigene Arbeitsumgebung mit Aufgaben und Ergebnissen besitzt. Ein Mandant wird dabei von einem Administrator angelegt und kann für verschiedene Scan-Bereiche berechtigt werden.

7.1. Diskussion

Die Ziele der Aufgabenstellung wurden vollständig erreicht. Zusätzlich wurde das Gros der Anforderungen von Kapitel 3 realisiert. Von insgesamt 20 Anforderungen wurden 15 implementiert, davon alle mit der Priorität `unbedingt notwendig` und `wichtig`. Eine Übersicht über die funktionalen Anforderungen erhält man in Tabelle 3.2 und über die nichtfunktionalen Anforderungen in Tabelle 3.3.

Die eigentliche Erkennungsrate des Prototypen ist im Großen und Ganzen zuverlässig. Eine genaue und vollständig korrekte Darstellung der in einem Netz agierenden Systeme ist mit nur einer eingesetzten Erkennungsmethode aber nicht möglich. Allerdings wäre selbst bei einem gleichzeitigen Einsatz mehrerer verschiedener Erkennungsverfahren und der Betrachtung über einen längeren Zeitraum keine eindeutige Bestimmung aller Hosts, Betriebssysteme und Dienste möglich. Das liegt darin begründet, dass Dienste - sei es beabsichtigt oder nicht - zum einen nicht immer auf einem bekannten Port operieren oder aber zum anderen verschleiert sind.

Systeme in bestimmten IP-Bereichen können durch Filtertechniken gezielter auf Existenz und vorhandene Dienste hin analysiert werden. Die Steuerung und Verwaltung der Aufgaben über die Weboberfläche ist wie geplant bequem und die Ergebnisse können übersichtlich dargestellt werden. Auch eine Weiterverarbeitung der Ergebnisse durch externe Werkzeuge ist durch die vorhandene Exportfunktionalität ohne weiteres möglich. Administratoren können

durch die wiederkehrende Aufgaben regelmäßig ihren gewünschten Netzbereich überprüfen und daraus in einem stetigen Prozess wieder und wieder aktualisierte Sicherheitsmaßnahmen ableiten.

Aufgrund der Einsatzmöglichkeiten ergibt sich großes Potential für Erweiterungen in den verschiedensten Bereichen. Hieraus entstanden sind einige Ideen und Verbesserungen, von denen lediglich ein kleiner Teil im nachfolgenden Abschnitt 7.2 berücksichtigt werden kann.

7.2. Ausblick

In weiteren Arbeiten kann der in dieser Arbeit entwickelte Prototyp verbessert werden. Zum einen kann die Darstellung der Ergebnisse noch übersichtlicher gestaltet und die übertragene Datenmenge reduziert werden. Bei einer sehr großen Zahl von erkannten Systemen benötigt die Anzeige und das Suchen in den Ergebnissen aktuell viel Zeit. Zum anderen könnten die Filter benutzerfreundlicher integriert werden, sodass man sich beispielsweise mit einem Klick alle Systeme mit einem bestimmten geöffneten Port anzeigen lassen kann. Im Moment ist dies nur über eine detaillierte Suche möglich.

Die in diesem Prototyp gewonnenen Ergebnisse können in den verschiedensten Anwendungen weitergenutzt werden und liegen dafür je nach Bedarf als CSV-Export vor oder können direkt über die Datenbank abgefragt werden. So können die Firewall-Regeln je nach Resultat neu konfiguriert werden oder neu erkannte Dienste können einer IDS zur zusätzlichen Überprüfung hinzugefügt werden.

Im Zuge der integrierten Mandantenfähigkeit wäre es vor allem auch für länger andauernde oder regelmäßig ausgeführte Aufgaben für Benutzer von Vorteil, dass sie darüber per E-Mail informiert werden. Eine solche Mail könnte beispielsweise versandt werden, wenn die Aufgabe fertiggestellt wurde und entsprechend Ergebnisse vorliegen. Selbstverständlich sollte im Fehlerfall äquivalent eine Benachrichtigung erfolgen, damit darauf reagiert werden kann. Insbesondere für ein zuverlässiges Monitoring wäre diese Funktionalität wichtig.

Für eine verbesserte Erkennungsrate ist es schlussendlich unabdingbar, weitere Erkennungsmethoden einzuführen. PRADS ist dabei ein interessantes Projekt und kann durch zusätzliche Daten vorhandene Systeme und Dienste verifizieren oder weitere Informationen hinzufügen.

Auch die aktive Erkennung zur Kontrolle, basierend beispielsweise auf einem Portscanner, wie im Entwurf vorgestellt, bietet sich als Datenquelle für zusätzliche bisher nicht verfügbare Informationen an. Denkbar sind dabei zwei Varianten: Einerseits ein stichprobenartiger Ansatz, wobei jeweils zufällig ein paar gefundene Hosts und deren jeweiliges Scan-Ergebnis mit einem aktiven Verfahren kontrolliert werden. Andererseits, dass bei Hosts, bei denen nichts oder keine eindeutigen Hinweise entdeckt wurden, mit einem aktiven Portscan gezielt weitere Informationen gesucht werden können.

Anhang

A. Einrichtung der LAMP-Umgebung

Wie im Namen schon enthalten beinhaltet eine typische LAMP-Umgebung ein Linux-System, einen Apache-Webserver, einen MySQL-Server und PHP. Im Nachfolgenden wird sowohl auf die Installation als auch auf die Konfiguration eines solchen Systems eingegangen.

A.1. Installation

Das der LAMP-Umgebung zugrundeliegende Linux-System ist frei wählbar und stand in dieser Arbeit mit Debian 7.3 bereits fertig eingerichtet zur Verfügung.

Zur Installation der weiteren Dienste nutzen wir das standardmäßig bei Debian-basierten OS mitgelieferte APT (Advanced Packaging Tool), welche eine einfache Paketverwaltung zur Verfügung stellt. Zunächst installieren wir den Apache Webserver und richten PHP in der Version 5 als Apache2 Modul ein.

Quellcode 7.1: Installation von Apache mit PHP

```
1 # Installation von Apache 2
2 apt-get install apache2
3
4 # Installation von PHP5 als Apache2 Modul
5 apt-get install php5 libapache2-mod-php5
```

Der Webserver ist direkt nach der Installation bereits gestartet und wenn alles erfolgreich war, kann eine Testseite über die Adresse des Systems abgerufen werden, z.B. `http://10.156.200.1/`.

Als nächstes wird das DBMS installiert. Man muss sowohl für das DBMS als auch für phpMyAdmin jeweils ein Kennwort für den Administratorzugang (root) angeben.

Quellcode 7.2: Installation von MySQL

```
1 # Installation von MySQL-Server
2 apt-get install mysql-server
3
4 # Installation von MySQL-Client und PHP5-Plugin
5 apt-get install mysql-client php5-mysql
6
7 # Installation von phpMyAdmin zur Verwaltung der MySQL-Datenbanken
8 apt-get install phpmyadmin
```

MySQL ist ebenfalls bereits gestartet und wird auch nach einem Systemneustart genauso wie der Apache Webserver automatisch gestartet. Da wir zusätzlich phpMyAdmin zur Verwaltung des DBMS über eine Weboberfläche installiert haben, kann man direkt nach der Installation über z.B. `http://10.156.200.1/phpmyadmin` auf MySQL zugreifen. Der Zugang erfolgt über das vorher vergebene MySQL-Kennwort in Kombination mit dem Benutzernamen root.

A.2. Konfiguration

Neben der reinen Installation des Webservers sind noch einige Konfigurationen anzupassen, sodass das Tool einwandfrei ausgeführt werden kann.

Webserver

Wir richten das Verzeichnis `/var/www/sdact` ein und holen uns den Quellcode des Programms von einem Git-Repository, das vom LRZ freundlicherweise zur Verfügung gestellt wurde. Dies hat den Vorteil, dass Updates relativ leicht eingespielt und auch wieder rückgängig gemacht werden können.

Quellcode 7.3: Einrichtung des Programmverzeichnisses und der Versionsverwaltung

```
# Anlegen des Verzeichnisses sdact
cd /var/www
mkdir sdact

# Installation von Git
apt-get install git-core

# Klonen des zentralen Repository
git clone gitosis@git.lrz.de:lxdpi.git /var/www/sdact

# Aktualisieren auf die neueste Revision vom zentralen Repository
git pull /var/www/sdact
```

Für das `cgi` Verzeichnis müssen auf Linux-Systemen entsprechende Ausführungsrechte gesetzt sein.

Datenbank

Nachdem das DBMS der Wahl, in unserem Falle ein MySQL-System, installiert wurde, melden wir uns mit dem root-Benutzer und dem oben angegebenen Passwort am Server an. Hierfür bietet sich phpMyAdmin zur Verwaltung des DBMS an, welches wir ebenfalls bereits eingerichtet haben.

Zunächst richten wir eine neue Datenbank ein, wir nennen sie `sdact`. Für den Zugriff auf diese Datenbank ist ein MySQL-Benutzeraccount anzulegen, der aus Sicherheitsgründen ausschließlich auf die angelegte Datenbank Schreib- und Leserechte erhält.

Quellcode 7.4: Einrichtung der Datenbank mit eigenem Benutzer

```
CREATE DATABASE sdact;

CREATE USER 'sdact_user'@'localhost' IDENTIFIED BY  '***';

# The USAGE privilege specifier stands for "no privileges."
GRANT USAGE ON * . * TO  'sdact_user'@'localhost' IDENTIFIED BY  '***' WITH
    MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0
    MAX_USER_CONNECTIONS 0 ;

# Grant data rights (select, insert, update, delete) to database sdact for user
     sdact_user
GRANT SELECT , INSERT , UPDATE , DELETE ON  `sdact` . * TO  'sdact_user'@'
    localhost';
```

Die entsprechenden Zugangsdaten sind in der Konfiguration in `config.php` im Ordner `application` einzufügen, eine Beispieleingabe ist:

Quellcode 7.5: PHP-Konfiguration für Datenbankverbindung

```
$config['database'] =
    array(
        'host' => '127.0.0.1',
        'port' => '3306',
        'username' => 'sdact_user',
        'password' => 'sdact_password',
        'database' => 'sdact',
    );
```

Die Konfigurationsdatei `config.php` im Ordner `application` überschreibt dabei die in der `config.default.php` hinterlegten Werte, welche eine Datei des GIT-Repository ist. Zusätzlich kann ein Administrator einen Auszug der Konfiguration im Administrationsbereich des Programms ansehen. Nicht angezeigt werden an dieser Stelle allerdings Zugangsdaten, beispielsweise zum MySQL-Server.

Dateiverzeichnis

Durch den verfügbaren Dateiupload müssen die Dateien auf dem Zielsystem an irgendeiner Stelle gespeichert werden. Dieses Verzeichnis geben wir in der Konfiguration `config.php` im Ordner `application` an, so wie es in Quellcode 7.6 beschrieben ist. Das Verzeichnis muss existieren und es müssen Schreibberechtigungen für den zugreifenden Programmprozess vorliegen.

Quellcode 7.6: PHP-Konfiguration für Dateiverzeichnis mit Schreibrechten

```
// Path where all uploaded files are saved (write privileges required)
$config['uploadPath'] = 'C:/wamp/www/sdactfiles/'; // don't forget trailing
    slash
```

B. Einrichtung der Erkennungsmethoden

Für die einwandfreie Ausführung der Erkennungsmethoden sind gesondert Vorbereitungen erforderlich, welche nachfolgend beschrieben werden.

B.1. Netflow

Damit eine Aufzeichnung der Netflow-Daten überhaupt erst möglich ist, muss eine vorgelagerte Komponente als Netflow-Exporter fungieren und die Verbindungsdaten an unser System senden. Diese Konfiguration ist stark davon abhängig, welche physischen oder virtuellen Geräte eingesetzt werden und wird hier als gegeben angesehen.

Zur Aufzeichnung der empfangenen Netflow-Daten wird nfcapd aus dem `nfdump`-Toolset als Netflow Collector eingesetzt. Anschließend wird ein Ordner angelegt, in dem alle Flows abgelegt werden sollen, angelegt.

Quellcode 7.7: Installation von nfdump als Netflow Collector

```
# Installation von nfdump
apt-get install nfdump

# Ordner zum Ablegen aller netflow-Daten
mkdir /var/flows
```

Wir starten `nfcapd` anschließend, nutzen als Ausgabeverzeichnis das oben erzeugte und als Port den vom Netflow Exporter verwendeten Port 4242. Zusätzlich ist die Angabe des Parameters $-T$ in Kombination mit dem Wert `nsel` notwendig, wenn ein Cisco-Router eingesetzt wird. `nsel` steht hierfür für NSEL (NetFlow Security Event Logging). Ohne diesen Parameter wurden weder Zeitstempel noch die Anzahl der Pakete korrekt erfasst.

Quellcode 7.8: Initialisierung von nfcapd

```
# Starten von nfcapd
nfcapd -p 4242 -l /var/flows -T nsel
```

Um zu prüfen, ob dies korrekt funktioniert, lassen wir uns die Aufzeichnungen mit nfdump ausgeben.

Quellcode 7.9: Ausgabe mit nfdump

```
nfdump -R /var/flows
```

Zusätzlich ist für die Netflow-Erkennungsmethode der Pfad zu `nfdump` zu setzen, der vom Programm aufgerufen wird. Dies machen wir exemplarisch in Quellcode 7.10.

Quellcode 7.10: Setzen der Konfigurationsvariable für nfdump

```
// Path to nfdump tool, e.g. /usr/bin/nfdump or just nfdump
$config['nfdump'] = 'nfdump';
```

C. Anlegen der Datenbankstruktur

Nachdem das DBMS MySQL vollständig konfiguriert wurde in A.2, können wir in die erstellte Datenbank folgende Tabellen mittels der in Quellcode 7.11 aufgeführten SQL-Befehle importieren.

Quellcode 7.11: SQL-Befehle zum Erstellen der Datenbank

```

-- --------------------------------------------------------

--
-- Table structure for table `aufgabe`
--

CREATE TABLE IF NOT EXISTS `Aufgabe` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(50) NOT NULL,
  `Erkennungsmethode` varchar(50) NOT NULL,
  `Ausfuehrungsdatum` datetime NOT NULL,
  `Wiederholen` varchar(50) NOT NULL,
  `Scan-Bereich` text NOT NULL,
  `Daten` text NOT NULL,
  `Benutzer` int(11) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `Benutzer` (`Benutzer`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Table structure for table `benutzer`
--

CREATE TABLE IF NOT EXISTS `Benutzer` (
```

```
28    `ID` int(11) NOT NULL AUTO_INCREMENT,
29    `Name` varchar(50) NOT NULL,
30    `Email` varchar(200) NOT NULL,
31    `Passwort` varchar(200) NOT NULL,
32    `IP` text NOT NULL,
33    `Scan-Bereich` text NOT NULL,
34    `Rolle` varchar(50) NOT NULL,
35    PRIMARY KEY (`ID`)
36  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
37
38  -- --------------------------------------------------------
39
40  --
41  -- Table structure for table `dienst`
42  --
43
44  CREATE TABLE IF NOT EXISTS `Dienst` (
45    `ID` int(11) NOT NULL AUTO_INCREMENT,
46    `System` int(11) NOT NULL,
47    `Port` int(11) NOT NULL,
48    `Protokoll` varchar(50) NOT NULL,
49    `Dienst` varchar(50) NOT NULL,
50    PRIMARY KEY (`ID`),
51    KEY `System` (`System`)
52  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
53
54  -- --------------------------------------------------------
55
56  --
57  -- Table structure for table `system`
58  --
59
60  CREATE TABLE IF NOT EXISTS `System` (
61    `ID` int(11) NOT NULL AUTO_INCREMENT,
62    `Aufgabe` int(11) NOT NULL,
63    `IP` varchar(50) NOT NULL,
64    `Betriebssystem` varchar(50) NOT NULL,
65    PRIMARY KEY (`ID`),
66    KEY `Aufgabe` (`Aufgabe`)
67  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
68
69  -- --------------------------------------------------------
70
71  --
72  -- Table structure for table `verlauf`
73  --
74
75  CREATE TABLE IF NOT EXISTS `Verlauf` (
76    `ID` int(11) NOT NULL AUTO_INCREMENT,
77    `Aufgabe` int(11) NOT NULL,
78    `Datum` datetime NOT NULL,
79    `Status` varchar(50) NOT NULL,
80    `Meldung` text NOT NULL,
81    PRIMARY KEY (`ID`),
82    KEY `Aufgabe` (`Aufgabe`)
83  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
84
85  --
86  -- Constraints for dumped tables
87  --
88
89  --
90  -- Constraints for table `aufgabe`
91  --
92  ALTER TABLE `Aufgabe`
```

```
93    ADD CONSTRAINT `aufgabe_ibfk_1` FOREIGN KEY (`Benutzer`) REFERENCES `Benutzer
          ` (`ID`);
94
95  --
96  -- Constraints for table `dienst`
97  --
98  ALTER TABLE `Dienst`
99    ADD CONSTRAINT `dienst_ibfk_1` FOREIGN KEY (`System`) REFERENCES `System` (`
          ID`);
100
101 --
102 -- Constraints for table `system`
103 --
104 ALTER TABLE `System`
105   ADD CONSTRAINT `system_ibfk_1` FOREIGN KEY (`Aufgabe`) REFERENCES `Aufgabe`
          (`ID`);
106
107 --
108 -- Constraints for table `verlauf`
109 --
110 ALTER TABLE `Verlauf`
111   ADD CONSTRAINT `verlauf_ibfk_1` FOREIGN KEY (`Aufgabe`) REFERENCES `Aufgabe`
          (`ID`);
```

D. Quelltexte

Nachfolgend werden die wichtigsten Quelltexte für die passive Erkennung von Systemen und Diensten dargestellt.

D.1. Standard-Konfiguration der Anwendung

Quellcode 7.12: Voreingestellte Belegung der Konfigurationsvariablen

```
1  <?php
2  /**
3   * Default configuration file, you can set and overwrite these values with your
          own values in config.php
4   */
5
6  // Database configuration
7  $config['database'] =
8      array(
9          'host' => '127.0.0.1',
10         'port' => '3306',
11         'username' => 'root',
12         'password' => '',
13         'database' => 'sdact',
14     );
15
16 // User roles
17 $config['userRoles'] = array(
18     'standard' => 'Standard',
19     'administrator' => 'Administrator',
20 );
21
22 // User acl - which role can access which page (additive roles: privileges for
       an administrator = default + standard + administrator
23 $config['acl'] = array(
24     'default'=>array(
25         'login',
26         'license',
27     ),
```

```
28      'standard'=>array(
29          'start',
30          'task',
31          'result',
32          'history',
33          'logout',
34      ),
35      'administrator'=>array(
36          'admin',
37          'user',
38          'planner',
39      ),
40  );
41
42  // Password salt used for hashing
43  $config['salt'] = 'testpasswortsalt'; // please overwrite
44
45  // Path where all uploaded files are saved (write privileges required)
46  $config['uploadPath'] = '/var/www/sdact/files/'; // don't forget trailing slash
47
48  // List of available detection methods
49  $config['detectionMethods'] =
50      array(
51          'netflow' => 'Netflow-Analyse',
52      );
53
54  // List of available repeating modes
55  $config['repeatingModes'] =
56      array(
57          'daily' => 'täglich',
58          'weekly' => 'wöchentlich',
59          'monthly' => 'monatlich',
60      );
61
62  // List of available states
63  $config['states'] =
64      array(
65          'normal' => 'Neu',
66          'waiting' => 'Warten auf Ausführung',
67          'executing' => 'Ausführung',
68          'done' => 'Abgeschlossen',
69          'error' => 'Fehler-Status',
70      );
71
72  // Update ports for OS
73  $config['OSUpdatePorts']= array(
74      '80'=>'HTTP',
75      '443'=>'HTTPS',
76  );
77
78  // Update servers for OS
79  $config['OSUpdateServers'] = array(
80      '129.240.2.25' => 'RedHat',      # yum.uio.no
81      '65.55.0.0/16' => 'Windows',     # update.microsoft.com
82      '65.54.0.0/16' => 'Windows',     # update.microsoft.com
83      '65.53.0.0/16' => 'Windows',     # update.microsoft.com
84      '65.52.0.0/16' => 'Windows',     # update.microsoft.com
85      '207.46.0.0/16' => 'Windows',     # update.microsoft.com
86      '157.56.0.0/16' => 'Windows',     # update.microsoft.com
87      '157.54.0.0/16' => 'Windows',     # update.microsoft.com
88      '129.240.12.27' => 'Windows',    # wsus.uio.no
89      '17.250.248.95' => 'Darwin',     # swscan.apple.com
90      '129.241.93.37' => 'Ubuntu',     # no.archive.ubuntu.com
91      '10.156.10.105' => 'SuSE',       # lxinst.srv.lrz.de
92      '10.156.84.30' => 'Windows',     # sus.lrz.de
93      '129.187.10.100' => 'Debian/Ubuntu' # debmirror.lrz.de
```

```
);

// White and blacklist of services
$config['services'] = array();
$config['services']['whiteList'] = array(
    '3306/TCP' => 'MySQL (TCP)',
    '3306/UDP' => 'MySQL (UDP)',
    '5432/TCP' => 'PostgreSQL (TCP)',
    '5432/UDP' => 'PostgreSQL (UDP)',
    '3389/TCP' => 'Remote Desktop Protocol (TCP)',
);
$config['services']['blackList'] = array(
    '0/ICMP' => 'ICMP',
    '7/TCP' => 'Echo',
    '7/UDP' => 'Echo',
);

// Insecure detection via targeted ports (can generate a lot of false positives
    )
$config['detectionViaTargetPorts'] = false; // default false

// Path to nfdump tool, e.g. /usr/bin/nfdump or just nfdump
$config['nfdump'] = 'nfdump';

// No system port to system port detection
$config['noSystemPortToSystemPortServices'] = true; // default true

// Uncomment this on Windows platforms
//$config['nfdump'] = false;
```

D.2. Initialisierung der Erkennungsmethode(n)

Quellcode 7.13: Aufbereitung der Daten für Erkennungsmethode(n)

```
<?php
/**
 * Class Detection
 */
class Detection {
    private $method;
    private $networks = array();
    private $allowedNetworks = array();
    private $files = array();
    private $hostMapper;

    /**
     *
     * @param $method
     * @throws Exception
     */
    public function setMethod($method) {
        global $config;
        if(!array_key_exists($method,$config['detectionMethods'])) {
            throw new Exception("Detection method doesn't exist.");
        }
        $this->method = $method;
        return true;
    }

    /**
     * Add IPv4/IPv6 address / subnet to network list
     * @param $net
     */
    public function addNetwork($network) {
        $this->networks[] = $network;
        return true;
```

```
33      }
34
35      /**
36       * Add IPv4/IPv6 address / subnet to allowed network list
37       * @param $net
38       */
39      public function addAllowedNetwork($network) {
40          $this->allowedNetworks[] = $network;
41          return true;
42      }
43
44      /**
45       * Add file to files list
46       * @param $file
47       */
48      public function addFile($file) {
49          if(!is_file($file)) {
50              throw new Exception($file." is not a file.");
51          }
52          $this->files[] = $file;
53          return true;
54      }
55
56      /**
57       * Scan folder for files and add them to files list
58       * @param $folder
59       */
60      public function addFolder($folder) {
61          if(!is_dir($folder)) {
62              throw new Exception($folder." is not a folder.");
63          }
64          $files = scandir($folder);
65          if(is_array($files)) {
66              foreach($files as $file) {
67                  $file=$folder.'/'.$file;
68                  if(($file == '.') || ($file == '..')) {
69                      continue;
70                  }
71                  elseif(!is_file($file)) {
72                      continue;
73                  }
74                  else {
75                      $this->addFile($file);
76                  }
77              }
78          }
79          return true;
80      }
81
82      /**
83       * Execute analyzing method(s)
84       */
85      public function execute() {
86          if(!isset($this->method)) {
87              throw new Exception("Detection method isn't set.");
88          }
89          if($this->method=='netflow') {
90              $netflow=new NetFlowAnalyzer($this->files,$this->networks,$this->
                    allowedNetworks);
91              $this->hostMapper = $netflow->analyze();
92              return $this; // fluent interface
93          }
94          throw new Exception("No action is set for detection method.");
95      }
96
97      /**
```

```
98          * Check if detection method has found results
99          * @return bool
100         */
101     public function hasResults() {
102         if($this->hostMapper->hasResults()) {
103             return true;
104         }
105         return false;
106     }
107
108     /**
109      * Delete results for a given task
110      * @param $taskID
111      */
112     public function deleteExistingResults($taskID) {
113         global $database;
114         // delete old results from same task
115         $systems=$database->query("SELECT * FROM System WHERE Aufgabe='".
                $taskID."'")->fetchAll();
116         foreach($systems as $system) {
117             $database->query("DELETE FROM Dienst WHERE System='".$system['ID'].
                    "'");
118             $database->query("DELETE FROM System WHERE Aufgabe='".$taskID."'");
119         }
120     }
121
122     /**
123      * Save results for a given task
124      * @param $taskID
125      */
126     public function saveTree($taskID) {
127         global $database;
128         // save hosts in database
129         $hosts=$this->hostMapper->getResults();
130         foreach($hosts as $ip => $host) {
131             $sth = $database->prepare("INSERT INTO System SET Aufgabe=?,`IP`=?,
                    Betriebssystem=?");
132             $sth->execute(array($taskID,$ip,implode(', ',$host['os'])));
133             if(count($host['services'])>0) {
134                 $systemId=$database->lastInsertId();
135                 foreach($host['services'] as $service) {
136                     $sth = $database->prepare("INSERT INTO Dienst SET `System
                            `=?,`Port`=?,`Protokoll`=?,`Dienst`=?");
137                     $sth->execute(array($systemId,$service['port'],$service['
                            protocol'],$service['name']));
138                 }
139             }
140         }
141     }
142 }
```

D.3. Netflow-Analyse (IPv4 / IPv6 kompatibel)

Quellcode 7.14: Auswertungsprogramm für Netflows-Dateien (Original- und Exportformat)

```
1  <?php
2  /**
3   * Class NetFlowAnalyzer
4   */
5  class NetFlowAnalyzer {
6      private $files;
7      private $networks;
8      private $allowedNetworks;
9      private $hostMapper;
10     private $tmpFiles = array();
11
```

```
12      const IPv4 = 'IPv4';
13      const IPv6 = 'IPv6';
14
15      /**
16       * @param array $files
17       * @param array $networks
18       */
19      public function __construct($files=array(),$networks=array(),
            $allowedNetworks=array()) {
20          $this->files=$files;
21          $this->networks=$networks;
22          $this->allowedNetworks=$allowedNetworks;
23          $this->hostMapper=new HostMapper();
24      }
25
26      /**
27       * Convert netflow to txt file (nfdump required)
28       * @param $file
29       * @return string
30       */
31      public function netflowToTxt($file) {
32          global $config;
33          if(!$config['nfdump']) {
34              throw new Exception("Conversion from nfdump format to txt file isn'
                    t activated (maybe Windows platform).");
35          }
36          // Check for free file name
37          do {
38              $filename = $config['uploadPath'].mt_rand();
39          }
40          while(file_exists($filename));
41
42          $this->tmpFiles[]=$filename;
43
44          // maybe integrate network filters here
45          system($config['nfdump'].' -o "fmt:%ts %td %pr %sap -> %dap %pkt %byt %
                fl" -r '.$file.' > '.$filename);
46          return $filename;
47      }
48
49      /**
50       * Check if file is text
51       * @param $s
52       * @return bool
53       */
54      function isTextFile($file) {
55          $fp = fopen($file,'rb');
56          flock($fp,LOCK_SH);
57          $s = fread($fp,512);
58          fclose($fp);
59          $characters = array_merge(array_map('chr',range(32,127)),array("\012","
                \015","\t","\b"));
60          if(strpos($s,"\0") === true)
61              return false;
62          if(!$s)
63              return true;
64          $t = $s;
65          foreach($characters as $character) {
66              $t = str_replace($character,'',$t);
67          }
68          if(strlen($t) / strlen($s) > 0.3)
69              return false;
70          return true;
71      }
72
73      /**
```

```
74          * Check if netflow data is printed in txt file
75          * @param $file
76          * @return string
77          */
78         public function checkForTxtFormat($file) {
79             // if file is in binary format (e.g. nfdump), try to convert it
80             if(!$this->isTextFile($file)) {
81                 return $this->netflowToTxt($file);
82             }
83             return $file;
84         }
85 
86         /**
87          * Check if IPv4 is in CIDR notation mask
88          * @param $ip
89          * @param $cidr
90          * @return bool
91          */
92         function cidrMatchIPv4($ip, $cidr) {
93             // Extract mask
94             list($network, $mask) = array_pad(explode('/', $cidr), 2, NULL);
95             if( is_null($mask) ){
96                 // No mask specified: range is a single IP address
97                 $mask = 0xFFFFFFFF;
98             } elseif( (int)$mask==$mask) {
99                 // Mask is an integer: it's a bit count
100                $mask = 0xFFFFFFFF << (32 - (int)$mask);
101            } else{
102                // Mask is in x.x.x.x format
103                $mask = ip2long($mask);
104            }
105            $ip = ip2long($ip);
106            $network = ip2long($network);
107            $lower = $network & $mask;
108            $upper = $network | (~$mask & 0xFFFFFFFF);
109            return $ip>=$lower && $ip<=$upper;
110        }
111 
112        /**
113         * Inet to Bits
114         * @param $inet
115         * @return string
116         */
117        function inetToBits($inet) {
118            $unpacked = unpack('A16', $inet);
119            $unpacked = str_split($unpacked[1]);
120            $binaryIP = '';
121            foreach ($unpacked as $char) {
122                $binaryIP .= str_pad(decbin(ord($char)), 8, '0', STR_PAD_LEFT);
123            }
124            return $binaryIP;
125        }
126 
127        /**
128         * Check if IPv6 is in CIDR notation mask
129         * @param $ip
130         * @param $cidr
131         * @return bool
132         */
133        function cidrMatchIPv6($ip, $cidr) {
134            $ip = inet_pton($ip);
135            $binaryIP=$this->inetToBits($ip);
136            list($net,$maskBits)=explode('/',$cidr);
137            $net=inet_pton($net);
138            $binaryNet=$this->inetToBits($net);
139            $IPNetBits=substr($binaryIP,0,$maskBits);
```

```
140         $netBits =substr($binaryNet,0,$maskBits);
141         if($IPNetBits!==$netBits) {
142             return false;
143         } else {
144             return true;
145         }
146     }
147 
148     /**
149      * Call IPv4/IPv6 matching methods
150      * @param $type
151      * @param $ip
152      * @param $cidr
153      * @return bool
154      */
155     function cidrMatch($type, $ip, $cidr) {
156         if($type==self::IPv4) {
157             return $this->cidrMatchIPv4($ip, $cidr);
158         }
159         else {
160             return $this->cidrMatchIPv6($ip, $cidr);
161         }
162     }
163 
164     /**
165      * Check if scanning of a given host is allowed
166      * @param $ip
167      * @return bool
168      */
169     public function isAllowed($ip) {
170         $ipType=$this->getIPType($ip);
171         foreach($this->allowedNetworks as $network) {
172             // Match against single IPv4 / IPv6
173             if(strpos($network,'/')===false) {
174                 if($ip==$network) {
175                     return true;
176                 }
177             }
178             // Match against IPv4 / IPv6 subnet
179             else {
180                 if(($this->getIPType($network)==$ipType) && ($this->cidrMatch(
                        $ipType,$ip,$network))) {
181                     return true;
182                 }
183             }
184         }
185         return false;
186     }
187 
188     /**
189      * Check if IPv4/IPv6 is in network
190      * @param $ip
191      * @param $networks
192      * @return bool
193      */
194     public function inNetwork($ip,$networks) {
195         $ipType=$this->getIPType($ip);
196         // If no networks are set, analyze all hosts
197         if(!count($networks)) {
198             return true;
199         }
200         foreach($networks as $network) {
201             // Match against single IPv4 / IPv6
202             if(strpos($network,'/')===false) {
203                 if($ip==$network) {
204                     return true;
```

```
205                 }
206             }
207             // Match against IPv4 / IPv6 subnet
208             else {
209                 if(($this->getIPType($network)==$ipType) && ($this->cidrMatch(
                        $ipType,$ip,$network))) {
210                     return true;
211                 }
212             }
213         }
214         // Didn't match anything
215         return false;
216     }
217 
218     /**
219      * Get Type of IP address
220      * @param $ip
221      * @return string
222      */
223     public function getIPType($ip) {
224         if(substr_count($ip,':')>1) {
225             return self::IPv6;
226         }
227         else {
228             return self::IPv4;
229         }
230     }
231 
232     /**
233      * Split IPv4/IPv6 address from port
234      * @param $host
235      * @return array
236      */
237     public function splitHost($host) {
238         if($this->getIPType($host)==self::IPv6) {
239             return explode('.',$host);
240         } else {
241             return explode(':',$host);
242         }
243     }
244 
245     /**
246      * Detect host if entry with source IP is given
247      * @param $sourceIP
248      */
249     public function detectHost($sourceIP) {
250         if($this->inNetwork($sourceIP,$this->networks) && ($this->isAllowed(
                $sourceIP))) {
251             $this->hostMapper->addHost($sourceIP);
252             return true;
253         }
254         return false;
255     }
256 
257     /**
258      * Check if service is on white list
259      * @param $port
260      * @param $protocol
261      * @return bool
262      */
263     public function inServiceWhiteList($port,$protocol) {
264         global $config;
265         if(array_key_exists($port.'/'.$protocol,$config['services']['whiteList'
                ])) {
266             return true;
267         }
```

```
268         return false;
269     }
270 
271     /**
272      * Check if service is on black list
273      * @param $port
274      * @param $protocol
275      * @return bool
276      */
277     public function inServiceBlackList($port,$protocol) {
278         global $config;
279         if(array_key_exists($port.'/'.$protocol,$config['services']['blackList']
                ])) {
280             return true;
281         }
282         return false;
283     }
284 
285     /**
286      * Detect service
287      * @param $sourceIP
288      * @param $sourcePort
289      * @param $targetIP
290      * @param $targetPort
291      * @param $protocol
292      */
293     public function detectService($sourceIP,$sourcePort,$targetIP,$targetPort,
            $protocol) {
294         global $config;
295         // Detect via source port, so sender sends really via this port
296         if((($sourcePort<1024) || $this->inServiceWhitelist($sourcePort,
                $protocol)) && !$this->inServiceBlackList($sourcePort,$protocol) &&
                 $this->inNetwork($sourceIP,$this->networks) && ($this->isAllowed(
                $sourceIP)) && (!$config['noSystemPortToSystemPortServices'] || (
                $targetPort>=1024))) {
297             $this->hostMapper->addService($sourceIP,$sourcePort,$protocol);
298         }
299         // Detect via target port which is accessed (can generate a lot of
                false positives!)
300         if($config['detectionViaTargetPorts']) {
301             if((($targetPort<1024) || $this->inServiceWhitelist($targetPort,
                    $protocol)) && !$this->inServiceBlackList($targetPort,$protocol
                    ) && $this->inNetwork($targetIP,$this->networks) && ($this->
                    isAllowed($targetIP)) && (!$config['
                    noSystemPortToSystemPortServices'] || ($sourcePort>=1024))) {
302                 $this->hostMapper->addService($targetIP,$targetPort,$protocol);
303             }
304         }
305     }
306 
307     /**
308      * Detect OS via update server
309      * @param $sourceIP
310      * @param $sourcePort
311      * @param $targetIP
312      * @param $targetPort
313      * @param $protocol
314      */
315     public function detectOS($sourceIP,$sourcePort,$targetIP,$targetPort,
            $protocol) {
316         global $config;
317         if(array_key_exists($targetPort,$config['OSUpdatePorts']) && ($this->
                inNetwork($sourceIP,$this->networks)) && ($this->isAllowed(
                $sourceIP))) {
318             foreach($config['OSUpdateServers'] as $network => $os) {
319                 if($this->inNetwork($targetIP,array($network))) {
```

```
320                     $this->hostMapper->addOS($sourceIP,$os);
321                 }
322             }
323         }
324     }
325
326     /**
327      * Parse each line of flow dump file
328      * @param $file
329      */
330     public function parse($file) {
331         $handle = fopen($file,"r");
332         if ($handle) {
333             $count=0;
334             while (($line = fgets($handle)) !== false) {
335                 $count++;
336                 if($count==1) { continue; }
337                 $line = str_replace('->','',$line);
338                 $line = str_replace(chr(1),' ',$line);
339                 $line = preg_replace('/\s\s+/',' ', $line);
340                 if($line[0]!=2) {
341                     continue;
342                 }
343                 list($date, $time, $duration, $protocol, $source, $target,
                        $packets) = explode(" ", $line);
344                 list($sourceIP, $sourcePort) = $this->splitHost($source);
345                 list($targetIP, $targetPort) = $this->splitHost($target);
346
347                 // Detect host with source IP
348                 $this->detectHost($sourceIP);
349
350                 // Detect services
351                 $this->detectService($sourceIP,$sourcePort,$targetIP,
                        $targetPort,$protocol);
352
353                 // Detect OS
354                 $this->detectOS($sourceIP,$sourcePort,$targetIP,$targetPort,
                        $protocol);
355             }
356         }
357     }
358
359     /**
360      * Delete temporary files which were generated during program execution
361      */
362     public function deleteTmpFiles() {
363         foreach($this->tmpFiles as $file) {
364             @unlink($file);
365         }
366     }
367
368     /**
369      * Analyze each given file
370      * @return HostMapper
371      */
372     public function analyze() {
373         // Iterate through given files
374         foreach($this->files as $file) {
375             // Convert nfdump file if necessary from binary to text
376             $file=$this->checkForTxtFormat($file);
377             $this->parse($file);
378         }
379
380         // Delete created files add the end
381         $this->deleteTmpFiles();
382
```

```
383         return $this->hostMapper;
384     }
385 }
```

D.4. HostMapper

Quellcode 7.15: Zuordnung der Systeme mit Dienste und Betriebssystem sowie Speicherung

```
1  <?php
2  /**
3   * Class HostMapper
4   */
5  class HostMapper {
6      private $hosts=array();
7      private $ports=array();
8
9      /**
10      * Add Host
11      * @param $ip
12      */
13     public function addHost($ip) {
14         if(!array_key_exists($ip,$this->hosts)) {
15             $this->hosts[$ip]=array();
16             $this->hosts[$ip]['services']=array();
17             $this->hosts[$ip]['os']=array();
18         }
19     }
20
21     /**
22      * Add Service
23      * @param $ip
24      * @param $port
25      * @param $protocol
26      */
27     public function addService($ip,$port,$protocol) {
28         $this->addHost($ip);
29         if(!array_key_exists($port.'/'.$protocol,$this->hosts[$ip]['services'])
               ) {
30             $this->hosts[$ip]['services'][$port.'/'.$protocol]=array(
31                 'port'=>$port,
32                 'protocol'=>$protocol,
33                 'name'=>$this->getServiceName($port,$protocol),
34             );
35         }
36     }
37
38     /**
39      * Add OS
40      * @param $ip
41      * @param $os
42      */
43     public function addOS($ip,$os) {
44         $this->addHost($ip);
45         if(!in_array(ucfirst($os),$this->hosts[$ip]['os'])) {
46             $this->hosts[$ip]['os'][]=ucfirst($os);
47         }
48     }
49
50     /**
51      * Get service name for given port / protocol combination
52      * @param $port
53      * @param $protocol
54      * @return string
55      */
56     public function getServiceName($port,$protocol) {
57         // Init list with all ports and names (protocols included)
```

```
58          if(empty($this->ports)) {
59              $sourceFile=file_get_contents('statics/csv/port-names.csv');
60              $rows=explode("\n",$sourceFile);
61              foreach($rows as $row) {
62                  $columns=explode(',',$row);
63                  // Skip lines with invalid port number or if no service name is
                        set
64                  if(!isset($columns[1]) || !is_numeric($columns[1]) || empty(
                        $columns[0])) {
65                      continue;
66                  }
67                  $columns[2]=strtoupper($columns[2]);
68                  // Only add to list if not already on list
69                  if(!array_key_exists($columns[1].'/'.$columns[2],$this->ports))
                         {
70                      $this->ports[$columns[1].'/'.$columns[2]]=$columns[0];
71                  }
72              }
73          }
74          // Found service name
75          if(array_key_exists($port.'/'.$protocol,$this->ports)) {
76              return $this->ports[$port.'/'.$protocol];
77          } else {
78              // Unknown service
79              return 'Unknown';
80          }
81      }
82
83      /**
84       * Return all results
85       * @return array
86       */
87      public function getResults() {
88          return $this->hosts;
89      }
90
91      /**
92       * Check if there are any results
93       * @return bool
94       */
95      public function hasResults() {
96          if(!count($this->hosts)) {
97              return false;
98          }
99          return true;
100     }
101 }
```

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
APT	Advanced Packaging Tool
BIOS	Basic Input Output System
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DBMS	Database Management System
DNS	Domain Name System
DOM	Document Object Model
DoS	Denial of Service
DPI	Deep Packet Inspection
DPM	Deep Packet Modification
ERM	Entity-Relationship-Modell
FIFO	First In – First Out
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDS	Intrusion Dedection System

IIS	Microsoft Internet Information Services
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IPS	Intrusion Prevention System
IT	Informationstechnik
LAMP	Linux, Apache, MySQL, PHP
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LMU	Ludwig-Maximilians-Universität München
LRZ	Leibniz-Rechenzentrum
MWN	Münchner Wissenschaftsnetz
NAS	Network Attached Storage
NFS	Network File System
NIDS	Netzwerk-Basierte IDS
NIPS	Network Intrusion Prevention System
NSEL	NetFlow Security Event Logging
NTP	Network Time Protocol
OS	Operating System
PDF	Portable Document Format
PDO	PHP Data Objects
PRADS	Passive Real-time Asset Detection System
PVS	Passive Vulnerability Scanner
RDP	Remote Desktop Protocol
RFI	Remote File Inclusion
RIA	Rich Internet Application
SMTP	Simple Mail Transfer Protocol
SSD	Solid-State-Drive
SSH	Secure Shell

SSL	Secure Sockets Layer
TB	Terabyte
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
WAP	Wireless Access Point
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XSS	Cross-Site-Scripting

Abbildungsverzeichnis

1.1. Anbindung des MWN an das X-WiN, Quelle: [Lei13] 1

2.1. Kategorisierung anhand der Betriebssysteme 7
2.2. Kategorisierung anhand des Einsatzzweckes und Diensten 7
2.3. Kategorisierung anhand einer Schwachstelle 8
2.4. Kategorisierung der unterschiedlichen Systeme 8
2.5. Nmap-Scan inklusive Dienst- und Versionserkennung 9
2.6. Architektur der Netflow-Technik, angelehnt an [hel07] 12
2.7. Netflow-Daten von www.lrz.de 13
2.8. Angriffserkennung durch Musterdatenbank 19

3.1. Kategorisierung von Sicherheitsmaßnahmen, Quelle: [HR13] 22
3.2. Funktionale Anforderungen mit Referenzen 28
3.3. Nichtfunktionale Anforderungen mit Referenzen 29

4.1. Weboberfläche im Client-Server-Modell, Quelle: [Fra06] 31
4.2. Vergleich einer herkömmlichen Web-Anwendung mit Ajax Web-Anwendung, Quelle: [Dan07] 32
4.3. Entwurf der Weboberfläche 33
4.4. Entwurf einer Startmaske für das System 34
4.5. Entwurf der Tabelle zur Verwaltung von Aufgaben 34
4.6. Entwurf eines ER-Modell der Anwendungsobjekte 35
4.7. Exemplarisches Eingabefeld für Scan-Bereich 39
4.8. Beschreibung der verschiedenen Nfdump-Werkzeuge, Quelle: [nfd13] 40
4.9. Auswertungsprozess bei der passiven Netflow-Technik 41
4.10. Übersicht von flow-dump, Quelle: [Kle12] 41
4.11. ICMP echo reply, Quelle: [Kle12] 42
4.12. Port-basierte Erkennung von Diensten, angelehnt an [Kle12] 43
4.13. Prototyp der verschiedenen Eingabemethoden 45
4.14. Anmeldemaske des SDACT Systems 49
4.15. Maske zum Anlegen eines Benutzers 49

5.1. Ordnerstruktur der Anwendung 55
5.2. Implementierung der Weboberfläche 57
5.3. Finales ER-Modell der Anwendungsobjekte 60
5.4. Implementierte Bearbeitungsmaske mit Eingabefeldern 61
5.5. Verlauf der Status-Änderungen aller Aufgaben 63
5.6. Erkannte Systeme und Dienste auf der Ergebnisseite dargestellt 63
5.7. Anzeigen aller im System angelegten Benutzer 79
5.8. Administrationsseite mit erweiterten Funktionen und Konfiguration 79

6.1. Beschreibung der System- und Netzwerkumgebung 86
6.2. Erste Ergebnisse von der Netflow-Auswertung der Test-Umgebung 87
6.3. Aktualisierte Ergebnisse von der Netflow-Auswertung der Test-Umgebung . . 88
6.4. Ergebnisse der IPv6-Auswertung 90

Quellcodeverzeichnis

2.1. Auswertung von Netflow-Daten hinsichtlich Dauer der Übertragung 14
2.2. Ergebnis der Auswertung von Netflow-Daten hinsichtlich der Übertragungsdauer 15
2.3. Ergebnis der Auswertung von Netflow-Daten hinsichtlich der übertragenen Bytes 15
2.4. Ergebnis der Auswertung von Netflow-Daten hinsichtlich der Anzahl der Pakete 16
2.5. Lauschen mit PRADS auf Netzwerkadapter 18
2.6. PRADS Aufzeichnung in .log-Datei 18

4.1. Exemplarische Crontab-Datei auf Linux-System 38
4.2. Beispiel-Ausgabe von flow-dump als CSV 44
4.3. Nmap-Scan mit Erkennung des Betriebssystems und der Dienste 46
4.4. Beispiel-Ausgabe von Nmap in XML 47

5.1. Initialisierung von Programm 55
5.2. loadPage führt Aufruf auf native jQuery load-Funktion aus 57
5.3. Automatisches Erkennen und dynamisches Umwandeln aller Hyperlinks . . . 57
5.4. Aufruf von DataTables zur verbesserten Darstellung statischer HTML-Tabellen 58
5.5. Exemplarische Verlaufstabelle, die dynamisch umgewandelt wird 58
5.6. Ausschnitt der Klasse Database 59
5.7. Ausführen eines SQL-Query mit Fehlerbehandlung 60
5.8. Laden des aktuellen Status einer Aufgabe 62
5.9. Setzen des aktuellen Status einer Aufgabe 62
5.10. Suchen und Laden von auszuführenden Aufgaben 64
5.11. Initialisierung der Erkennung durch Setzen der Erkennungsmethode 64
5.12. Hinzufügen aller angegebenen Ressourcen einer Aufgabe 65
5.13. Ausführen der Erkennungsmethode im Aufgabenplaner 65
5.14. Verarbeitung der Rückgabe der Erkennungsmethode 65
5.15. Löschen bereits vorhandener Ergebnisse einer Aufgabe 66
5.16. Speicherung der Systeme und Dienste in der Datenbank 66
5.17. Prüfen ob eine IPv4 / IPv6 Adresse in Liste von Netzwerkadressen und / oder -Subnetzen liegt 67
5.18. Erkennen des Typs einer IP-Adresse 67
5.19. Prüfen ob eine IPv4 Adresse in angegebenem Subnetz liegt 68
5.20. Aufruf der Netflow-Analyse Erkennungsmethode 68
5.21. Zeilenweises Parsen der Netflow-Daten 69
5.22. Einfache Host-Erkennung 70
5.23. Portbasierte Diensterkennung 70
5.24. Definition zusätzlicher zu erkennender Dienste 71
5.25. Definition von nicht zu erkennenden Diensten 71
5.26. Erkennung des Betriebssystems durch Update-Mechanismus 72
5.27. Verarbeitung der verschiedenen Optionen in der Datenerfassung 72

5.28. Hinzufügen einer Datei in Dateiliste 73
5.29. Prüfung ob Datei im Binär- oder Textformat vorliegt 73
5.30. Umwandlung von binärer nfdump Datei in Textdatei 74
5.31. Duplizieren der aktuellen Aufgabe als neue wiederkehrende Aufgabe 75
5.32. Authentifizierung eines Benutzers anhand E-Mail, Passwort und IP 76
5.33. Access Control List des Systems 77
5.34. Autorisierung bei Seitenaufruf 77
5.35. Prüfung ob Benutzer ein System analysieren darf 78
5.36. Erkennung ob eine IPv6-Adresse in IPv6-Subnetz liegt 80
5.37. Fallunterscheidung IPv4/IPv6 bei Host-/Porterkennung 80
5.38. SQL-Injections durch sicheres Type Casting verhindern 82
5.39. SQL-Injections durch PDO-Funktionalität verhindern 82
5.40. Filter von page-Parameter 83
5.41. Erlaubte Seiten werden explizit konfiguriert 83
5.42. Überprüfung des Vorhandenseins der angefragten Seite 83

6.1. Kontrolle mit aktiver Erkennungsmethode Nmap 88

7.1. Installation von Apache mit PHP 93
7.2. Installation von MySQL 93
7.3. Einrichtung des Programmverzeichnisses und der Versionsverwaltung 94
7.4. Einrichtung der Datenbank mit eigenem Benutzer 94
7.5. PHP-Konfiguration für Datenbankverbindung 95
7.6. PHP-Konfiguration für Dateiverzeichnis mit Schreibrechten 95
7.7. Installation von nfdump als Netflow Collector 95
7.8. Initialisierung von nfcapd 96
7.9. Ausgabe mit nfdump 96
7.10. Setzen der Konfigurationsvariable für nfdump 96
7.11. SQL-Befehle zum Erstellen der Datenbank 96
7.12. Voreingestellte Belegung der Konfigurationsvariablen 98
7.13. Aufbereitung der Daten für Erkennungsmethode(n) 100
7.14. Auswertungsprogramm für Netflows-Dateien (Original- und Exportformat) . 102
7.15. Zuordnung der Systeme mit Dienste und Betriebssystem sowie Speicherung . 109

Literaturverzeichnis

[Bed09] Bedner, Mark: *Rechtmäßigkeit der Deep Packet Inspection.* Technischer Bericht, Universität Kassel, 2009. http://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:34-2009113031192/5/BednerDeepPacketInspection.pdf (26. November 2013).

[Dan07] DanielSHaischt: *A traditional, stateless web application and it's ajaxified counterpart on the right site*, 2007. https://commons.wikimedia.org/wiki/File:Ajax-vergleich.svg (17. Januar 2014).

[Fra06] Franke, Gerd: *Die Webanwendung im Client-Server-Modell*, 2006. https://de.wikipedia.org/wiki/Datei:Webanwendung_client_server_01.png (14. Januar 2014).

[Fyo98] Fyodor: *Das Erkennen von Betriebssystemen mittels TCP/IP Stack FingerPrinting.* 1998. http://nmap.org/nmap-fingerprinting-article-de.html (18. Oktober 1998).

[Gel12] Gelbmann, Matthias: *jQuery now runs on every second website*, 2012. http://w3techs.com/blog/entry/jquery_now_runs_on_every_second_website (17. Januar 2014).

[hel07] helix84: *Netflow architecture diagram, with English labels*, 2007. https://commons.wikimedia.org/wiki/File:Netflow_architecture_en.svg (13. Dezember 2013).

[HR13] Hommel, Wolfgang und Helmut Reiser: *IT-Sicherheit - Sicherheit vernetzter Systeme.* 2013. http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2013ws/itsec/_skript/itsec-k2-v9.1.pdf (09. Januar 2014).

[Kle12] Klepsland, Mats Erik: *Passive Asset Detection using NetFlow.* Technischer Bericht, University of Oslo, Department of Informatics, 2012. http://urn.nb.no/URN:NBN:no-30783 (28. Januar 2014).

[Lei13] Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften: *Das Leibniz-Rechenzentrum IT-Dienste für Forschung und Lehre in München, Bayern, Deutschland und Europa.* 2013. http://www.lrz.de/wir/Einfuehrung-LRZ.pdf (12. November 2013).

[nfd13] nfdump.sourceforge.net: *NFDUMP tools overview.* 2013. http://nfdump.sourceforge.net/ (13. Januar 2014).

[Nma13] Nmap.org: *Nmap Network Scanning - Chapter 7. Service and Application Version Detection.* 2013. http://nmap.org/book/vscan.html (25. November 2013).

[Nto13] NTOP.ORG: *Open and Extensible GPLv3 Deep Packet Inspection Library.* 2013. `http://www.ntop.org/products/ndpi/` (11. Dezember 2013).

[vEMH13] EYE, FELIX VON, STEFAN METZGER und WOLFGANG HOMMEL: *Dr. Portscan: Ein Werkzeug für die automatisierte Portscan-Auswertung in komplexen Netzinfrastrukturen.* In: PAULSEN, CHRISTIAN (Herausgeber): *Sicherheit in vernetzten Systemen: 20. DFN Workshop*, Seiten C-1–C-21, Norderstedt, Deutschland, Januar 2013. Books on Demand.

[Wik13] WIKIPEDIA: *OS-Fingerprinting - TCP/IP-Protokollstapel.* 2013. `http://de.wikipedia.org/w/index.php?title=OS-Fingerprinting&oldid=116460485` (29. Dezember 2013).