INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Konzeption und Implementierung eines Webfrontends für die mandantenfähige Konfiguration eines Delta-Reporting Portscan-Werkzeugs

André Wörner

Draft vom 11. April 2014

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Konzeption und Implementierung eines Webfrontends für die mandantenfähige Konfiguration eines Delta-Reporting Portscan-Werkzeugs

André Wörner

Aufgabensteller:	PD Dr. Wolfgang Hommel
Betreuer:	Stefan Metzger Felix von Eye
Abgabetermin:	14. April 2014

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

München, den 14. April 2014

.....
(Unterschrift des Kandidaten)

Abstract

Mithilfe sogenannter Portscanner können einzelne Systeme, aber auch ganze Netzbereiche auf offene Ports überprüft werden. Ein Port ist stets mit einem netzbasierten Dienst des Hosts verknüpft. Offene Ports können gezielt von Angreifern ausgenutzt werden, um z. B. Zugriff auf ein System zu bekommen. Allerdings können auch Systemadministratoren diese Informationen nutzen, um solchen Sicherheitslücken proaktiv vorzubeugen.

Das am Leibniz-Rechenzentrum entwickelte Tool Dr. Portscan verwendet solche Portscanner und aggregiert die Scan-Ergebnisse in einer Datenbank. Diese können daraufhin gezielt per E-Mail an die zuständigen Systemadministratoren versendet werden. Die Konfiguration von Dr. Portscan wird bisher stets vom sogenannten Dr.-Portscan-Administrator vorgenommen und stellt einen nicht unerheblichen Aufwand dar. Aus diesem Grund wurde im Rahmen dieser Arbeit ein Webfrontend entwickelt, damit die Nutzer von Dr. Portscan eigenständig Konfigurations-, Verwaltungs- und Administrationsaufgaben übernehmen können.

Dem Dr.-Portscan-Administrator wurden Möglichkeiten für die Verwaltung der Nutzer sowie der Konfiguration der Portscans gegeben. Die sogenannten Netzverantwortlichen, den anderen Nutzern des Webfrontends, wird ermöglicht eigenständig Scans zu konfigurieren und die Ergebnisse an die zuständigen Personen weiterzuleiten. Außerdem können sie Scan-Ergebnisse direkt innerhalb des Webfrontends einsehen und ihren eigenen Netzbereich verwalten.

Inhaltsverzeichnis

1 Einleitung

Banken, Firmen, Regierungs- und Forschungseinrichtungen, um nur ein paar zu nennen, sind mit dem Internet verbunden. Auf deren Systemen sind oft vertrauliche oder private Daten vorhanden, die von der Außenwelt abgeschottet werden sollen. Solche Daten stellen eine lukrative Quelle für Angreifer dar. Will sich ein Angreifer Zugriff auf solche Daten verschaffen, kann er Sicherheitslücken in Anwendungen, die auf den Systemen laufen, ausnutzen. Programme die Daten über das Netzwerk empfangen wollen, haben stets einen offenen Port. Über die Adresse des Computers und den Port kann also mit einem Programm auf dem Computer kommuniziert werden. Wissen die Angreifer, welches Programm sich hinter einem offenen Port verbirgt, können sie Schwachstellen dieses Programms ausnutzen, um so Zugriff auf das System zu bekommen.

1.1 Motivation

Mithilfe sogenannter *Portscanner* ist es nicht nur möglich einzelne Computer zu scannen, sondern auch ganze Netzbereiche. Mit solchen Tools können Angreifer ihre Ziele scannen und bekommen so Informationen über offene Ports. Das Scannen von Ports ist nicht nur Angreifern vorbehalten. Auch ein verantwortungsbewusster Netzverantwortlicher[1] oder System-Administrator wird Portscanner nutzen, um die Sicherheit seiner Systeme zu gewährleisten.

Das Optimum für einen Netzverantwortlichen wären Echtzeit-Informationen über gefährliche Veränderungen an seinen Systemen. Ein Portscan benötigt allerdings eine gewisse Zeit, um Ergebnisse zu liefern. Des Weiteren ist ein Netzverantwortlicher nicht nur für einzelne Endsysteme verantwortlich, sondern betreut einen ganzen Netzbereich. Zusätzlich kann er auch noch mit ihm unbekannten, sich stetig ändernden Sollzuständen in Subnetzen[2] konfrontiert werden, wie es in Hochschulnetzen oft der Fall ist. Es würde einen erheblichen Aufwand bedeuten, diese Daten manuell auszuwerten.

Um den Netzverantwortlichen diesen Aufwand abzunehmen, wurde am LRZ[3] ein Open-Source-Werkzeug entwickelt. „Dr. Portscan ist ein Delta-Reporting-Tool, das nahezu beliebige Portscan-Werkzeuge unterstützt, die parallel oder zeitversetzt in beliebigen Zeitintervallen beliebig überlappende Netz- und Portbereiche analysieren. Die Ergebnisse dieser Scan-Läufe werden automatisiert aggregiert und miteinander verglichen.“ [HvGM13, S. 94] Die so registrierten Veränderungen werden in einer Datenbank gespeichert und können gezielt per E-Mail an die für den jeweiligen Netzbereich zuständigen Administratoren geschickt werden.

Bisher kann ein Netzverantwortlicher keinerlei Änderungen an der Konfiguration von Dr. Portscan vornehmen. Dies obliegt nur demjenigen, der für diese Dr.-Portscan-Instanz verantwortlich ist. Würde ein Netzverantwortlicher derzeit Einstellung an den Scannern von Dr.

[1] Ein Netzverantwortlicher ist Ansprechpartner für seinen Netzbereich und hat unter anderem Aufgaben wie die Verwaltung der zugeteilten Namens- und Adressräume oder die Dokumentation über angeschlossene Endgeräte bzw. Netze. [lrz13]

[2] Ein Subnetz ist eine Teilmenge eines bestimmten Netzbereichs.

[3] Leibniz-Rechenzentrum - Hochleistungsrechenzentrum unter anderen für die Münchner Universitäten.

Portscan vornehmen wollen, müssten ihm mindestens Zugriffsrechte für den Server gegeben werden, auf dem Dr. Portscan läuft. Damit könnte er in den Skripten und Datenbanken Änderungen vornehmen. Um seine gewünschten Anpassungen zu erzielen, müsste sich der Netzverantwortliche sehr gut mit dem Aufbau und den Diensten auskennen, die Dr. Portscan nutzt. Selbst dann können schnell Fehler in den Skripten entstehen, welche die Anwendung lahm legen können.

Die Berichterstattung der Portscan-Ergebnisse kann bereits gesondert für den jeweiligen Netzverantwortlichen erfolgen. Die Möglichkeit der Einsichtnahme kann jedoch noch verbessert werden. Eine tägliche E-Mail an den Netzverantwortlichen ist zwar zweckmäßig, bietet jedoch nicht den Komfort, den eine gut strukturierte Website ermöglichen kann.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll ein mandantenfähiges Webfrontend für die Konfiguration von Dr. Portscan erstellt werden.

1.2.1 Mandantenfähigkeit

Verwaltung

Ein Mandant ist ein Netzverantwortlicher. Der Dr.-Portscan-Administrator[4] ist ebenfalls ein Mandant, obwohl er im Normalfall keinen Netzbereich betreut, wie es ein Netzverantwortlicher zur Aufgabe hat, aber trotzdem zur Nutzergruppe des Webfrontends zählt. Die Netzverantwortlichen können, wie es sich bei Netzen anbietet, hierarchisch strukturiert werden. So kann ein Netzverantwortlicher unter sich Subnetzverantwortliche haben, die wiederum unter sich weitere Subnetzverantwortliche, ebenfalls Mandanten, haben können (siehe Abbildung 1.1). Der Dr.-Portscan-Administrator kann als höchste Instanz der Hierarchie angesehen werden, da er die Verwaltungsmacht über alle Netzverantwortlichen hat.

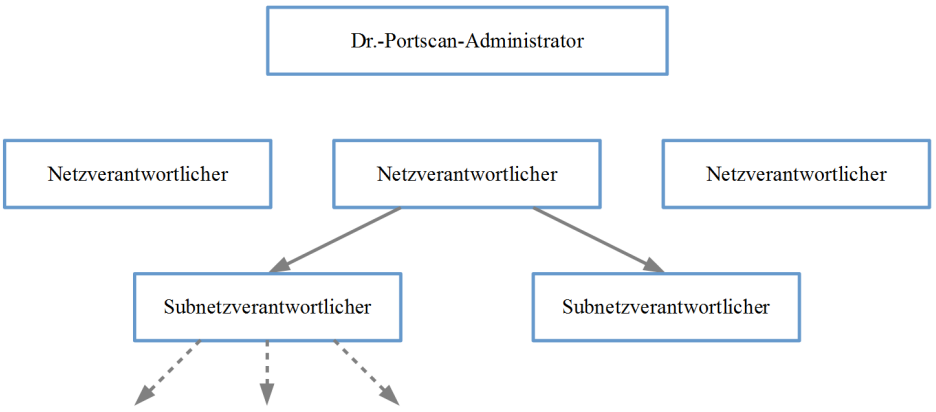

Abbildung 1.1: Mandantenhierarchie, bei der der Dr.-Portscan-Administrator als oberste Instanz betrachtet wird.

[4] Der Dr.-Portscan-Administrator ist die Person, welche die Dr.-Portscan-Instanz betreut und verwaltet.

Als oberste Instanz der Hierarchie kann der Dr.-Portscan-Administrator betrachtet werden, da er die Verwaltungsmacht über alle Netzverantwortlichen hat. Dazu gehört das Hinzufügen neuer Mandanten sowie das Löschen oder Sperren und von Mandanten. Jeder Mandant soll ebenfalls die Option haben, seinen eigenen Netzbereich selbstständig zu verwalten.

Einrichtung

Bei der Einrichtung des Webfrontends soll es möglich sein, bestehende Netzwerkhierarchien, die bereits in einer Datenbank festgehalten sind, automatisiert über einen Datenbank-Import abzubilden. Dadurch kann dem Dr.-Portscan-Administrator der nicht unerhebliche Aufwand erspart werden, diese manuell einzupflegen.

1.2.2 Konfiguration der Berichterstattung und Scans

Jeder Netzverantwortliche soll die im Rahmen von Dr. Portscan gegebenen Möglichkeiten zur Konfiguration der Berichterstattung von Scan-Ergebnissen eigenständig vornehmen können. Dazu gehört die Einrichtung einer E-Mail-Berichterstattung neuer Scan-Ergebnisse. Hierbei soll der Mandant den Inhalt der E-Mail selbst gestalten können.

Außerdem sollen dem Mandanten wichtige Optionen zur Konfiguration der Scans bereitgestellt werden. Dazu gehört das Definieren der zu scannenden Netzbereiche und Ports oder die Aggressivität, mit welcher der Portscanner vorgeht. Alle weiteren Konfigurationen außerhalb dieses Maßes bleiben immer noch Aufgabe des Dr.-Portscan-Administrators.

1.2.3 Berichterstattung der Scan-Ergebnisse

Dem Mandanten soll weiterhin nicht nur eine Berichterstattung per E-Mail ermöglicht werden, sondern auch eine individuelle Berichterstattung direkt auf der Website. Sinnvoll ist ein Report der Veränderungen an den Ports des eigenen Netzbereichs sowie eine Gesamtansicht der aktuellen Portscan-Ergebnisse. Somit gibt es eine zentrale Anlaufstelle für die Mandanten und es entfällt eine Suche im E-Mail-Postfach.

1.3 Struktur der Arbeit

Im Folgenden werden die Inhalte der kommenden Kapitel kurz erläutert.

Kapitel 2 befasst sich mit den nötigen Grundlagen, um diese Arbeit verstehen zu können. So werden Portscanner, Dr. Portscan und benötigte Komponenten für die Umsetzung des Webfrontends erörtert. Außerdem wird eine kurze Einführung in den Themenbereich der Rechnernetze gegeben.

Im Anschluss werden in Kapitel 3 die Anforderungen an das Webfrontend erarbeitet. Hier wird festgelegt, wie genau die Mandantenfähigkeit erreicht werden soll, welche Konfigurationsmöglichkeiten vorhanden sein sollen und welche Optionen es für die Berichterstattung der Ergebnisse von Dr. Portscan geben soll.

Das Kapitel 4, Systementwurf, beinhaltet den Architekturentwurf des Webfrontends und geht auf die benötigte Datenbankstruktur, sowie den Umbau des bisherigen Dr.-Portscan-Konzepts ein.

Die Implementierung wird in Kapitel 5 beschrieben. Hier wird nicht der komplette Code erläutert, sondern auf einige interessante und wichtige Teilkomponenten, wie zum Beispiel

die Umsetzung der Architektur und die Authentifizierung, eingegangen. Außerdem wird zum Schluss ein Vergleich der getätigten Implementierung mit den zuvor gesammelten Anforderungen angestellt.

Daraufhin folgt in Kapitel 6 eine Beschreibung der Inbetriebnahme des Webfrontends für Dr. Portscan und eine Beispielszenario, welches die Verwendung des implementierten Webfrontends vermitteln soll.

Schlussendlich wird in Kapitel 7 diese Arbeit zusammengefasst, um die Ergebnisse dieser darzustellen. Außerdem wird ein Ausblick auf mögliche Erweiterungen und Einsatzmöglichkeiten des Webfrontends gegeben.

2 Grundlagen

Alle wichtigen Komponenten die innerhalb dieser Arbeit Anwendung finden, werden hier, je nach nötigem Tiefgang, mehr oder weniger detailliert erläutert. Dadurch soll jedem Leser ein schneller und einfacher Einstieg in das Themengebiet ermöglicht werden.

2.1 Rechnernetze

Heutzutage ist fast jeder Computer Teil eines riesigen Netzwerks, dem Internet. Theoretisch kann sich jeder Computer, der diesem Netz angehört, mit jedem beliebigen anderen Computer verbinden.

Die Kommunikation innerhalb eines Netzes wird von Protokollen festgelegt. Ein Protokoll ist eine Spezifikation der Vorschriften zum Informationsaustausch in syntaktischer, prozeduraler und semantischer Hinsicht.[KgD12, S. 23] So wird beispielsweise, stark vereinfacht gesehen, ein Gespräch erfolgreich abgehalten, wenn beide Teilnehmer die gleiche Sprache sprechen, sich begrüßen und daraufhin die Hand geben, dann mit dem Informationsaustausch beginnen und sich schlussendlich verabschieden.

Um im Internet Informationen auszutauschen werden verschiedenste Protokolle verwendet (vgl. Abbildung 2.1).

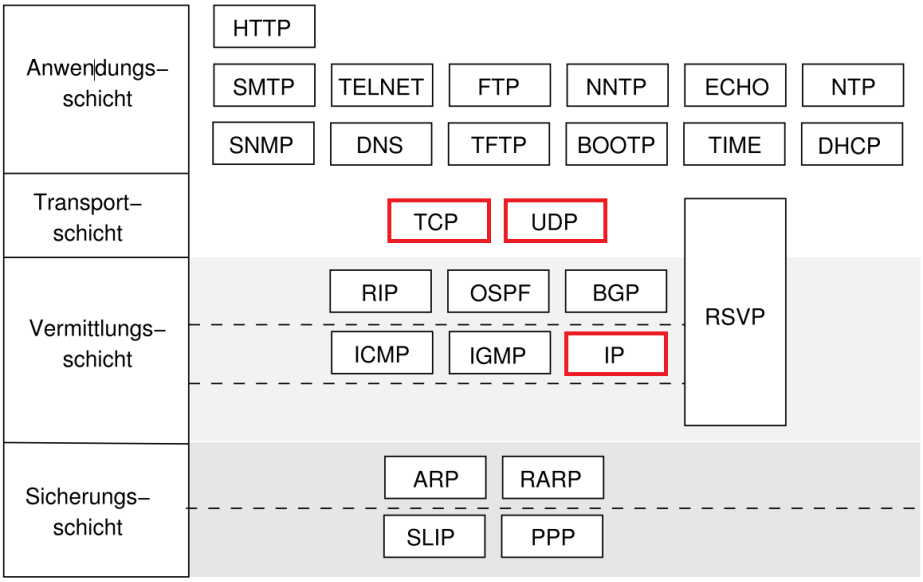

Abbildung 2.1: Das Internet-Modell mit seinen vier Schichten und den zugehörigen Protokollen. [KgD12, S. 31]

In der Sicherungsschicht werden die Bits übertragen, in Blöcke zusammengefasst und gegebenenfalls Fehler korrigiert. Die Vermittlungsschicht beinhaltet das Routing der Pakete und

stellt einen Ende-zu-Ende-Pfad zwischen zwei Systemen her. Der netzunabhängige Ende-zu-Ende Transport wird innerhalb der Transportschicht realisiert. Somit sind die Daten am Ziel angekommen und können der vorgesehenen Anwendung zugeordnet werden.

Das Internet Protocol (IP, siehe Vermittlungsschicht Abbildung 2.1) ermöglicht das Senden von Informationen an eine bestimmte Adresse in einem Netzwerk.

Da auf einem Computer meistens mehrere Anwendungen parallel laufen und diese getrennt voneinander kommunizieren wollen, ohne sich dabei gegenseitig zu beeinflussen, gibt es sogenannte Ports. Jedes Programm, das Nachrichten über ein Netzwerk empfangen will, meldet einen bestimmten Port beim Betriebssystem des Computers an. Das Betriebssystem identifiziert die Portnummer eingehender Nachrichten und gibt sie an das Programm weiter, welches für diese Portnummer registriert ist. Die Adressierung der Ports geschieht nicht etwa im IP sondern in der darüber liegenden Transportschicht (siehe Abbildung 2.1). Zur Transportschicht gehören die Protokolle TCP[1] und UDP[2]. Beide Protokolle besitzen jeweils 2^{16} Ports zur Adressierung. Die ersten 2^{10} Ports sind sogenannte *well-known ports*. Diese Ports sind standardisiert, das heißt, dass man annehmen kann, dass sich bestimmte Programme dahinter verbergen.

2.2 Portscanner

Über Portscanner und Netzwerksicherheit können ganze Bücher verfasst werden (siehe *Nmap Network Scanning: Official Nmap Project Guide to Network Discovery and Security Scanning* von Gordon Lyon), allerdings ist ein grober Überblick für das Verständnis dieser Arbeit vollkommen ausreichend. Deshalb wird im Folgenden nicht auf jede Technik und Möglichkeit von Portscans im Detail eingegangen.

Ein Portscanner ist ein Programm, dass einen über das Netz erreichbaren Computer auf offene Ports überprüft. Ein Port ist eine Adresse, über die ein bestimmtes Programm auf einem Computer, das über das Netzwerk kommunizieren will, erreicht werden kann.

Generell laufen Portscans immer gleich ab. Es werden Daten-Pakete an den zu prüfenden Host[3] gesendet und anhand der Antwort des Hosts eine Entscheidung bezüglich des Zustands des adressierten Ports getroffen.

Insbesondere ist dabei zwischen TCP- und UDP-Scans zu unterscheiden. Da eine TCP-Verbindung auf einem sogenannten *Drei-Wege-Handschlag* zurückgreift, um eine verlustfreie Datenübertragung zu gewährleisten, können TCP-Ports geprüft werden, indem versucht wird, eine Verbindung aufzubauen oder den Aufbau vorzutäuschen. Es gibt noch einige weitere Techniken, auf die hier jedoch nicht eingegangen wird. Beim Scannen von UDP-Ports werden, meist leere, UDP-Datagramme an einen Host geschickt. Kommt keine Antwort zurück, ist der Port offen, kommt eine ICMP-Nachricht[4] „`destination port unreachable`“ zurück, so ist der Port geschlossen. Oft werden allerdings ICMP-Nachrichten von Netzen oder Hosts gefiltert, sodass keine ICMP-Nachricht zurückgeschickt wird. So kann es sein, dass der Port

[1] Transmission Control Protocol - verbindungsorientiertes, zuverlässiges (weil Ankunft der Nachricht überprüft wird) Transportprotokoll.

[2] User Datagram Protocol - verbindungsloses, unzuverlässiges (weil Ankunft der Nachricht nicht überprüft wird) Transportprotokoll.

[3] Ein Host ist ein Computer der Dienste über ein Netzwerk anbietet.

[4] Das Internet Control Message Protocol (ICMP) dient dem Austausch von Meldungen über IP und wird für Status- und Fehlerinformationen verwendet.

geschlossen ist, obwohl keine ICMP-Nachricht empfangen wurde. Das macht die Beurteilung des Zustands von UDP-Ports oft sehr schwierig. [McN08, S. 42ff]

Portscanner bieten oft noch weitere Möglichkeiten, um Subnetze und Hosts zu identifizieren. So kann ein Netz beispielsweise per ICMP auf Subnetze und darin, zum Scan-Zeitpunkt erreichbare Computer getestet werden. Eine weitere Option bietet das *OS-Fingerprinting*, welches eine Vermutung des auf dem gescannten Computer laufenden Betriebssystem liefert. Hierbei wird ausgenutzt, dass jedes Betriebssystem seine Eigenheiten bezüglich bestimmten IP-Standards besitzt. Die ermittelten Eigenschaften des Systems werden mit einer Datenbank abgeglichen, in der die IP-Standards für eine Vielzahl an Betriebssystemen festgehalten werden. Das Betriebssystem des Datenbankeintrags mit den besten Übereinstimmungen wird dann zurückgegeben. [McN08, S. 75f]

Nmap[5] ist der derzeit wahrscheinlich populärste Portscanner. Der folgende beispielhafte Nmap-Aufruf soll abschließend zeigen wie mit einem Portscanner verfahren wird. Die IP-Adressen werden nachfolgend mit „*xxx*“ versehen, welche als Platzhalter für beliebige IP-Adressteile stehen.

```
nmap -pU:53,U:110,T:1-65535 -vv -O -T4 -oX /tmp/result.xml xxx.xxx.xxx.0/26
```

Mit `nmap` wird das Programm aufgerufen. Die folgenden Parameter werden nun der Reihe nach erläutert.

- `-pU:53,U:110,T:1-65535` besagt, dass auf jedem System die UDP-Ports 53 und 110 sowie die TCP-Ports 1 - 65536 gescannt werden.
- Mit `-vv` wird eine detailliertere Ausgabe der gesammelten Daten erreicht.
- `-O` schaltet OS-Fingerprinting ein.
- Mit `-T4` wird der Scan-Durchlauf beschleunigt, weshalb diese Option nur innerhalb verlässlicher Netze, zum Beispiel dem eigenen Netz innerhalb dem sich der Portscanner befindet, aktiviert werden sollte.
- `-oX /tmp/nmapResult.xml` besagt, dass die Ergebnisse im XML-Format in die angegebene Datei gespeichert werden.
- Mit `xxx.xxx.xxx.0/26` wird der Netzbereich adressiert, hier alle Systeme mit den IP-Adressen von xxx.xxx.xxx.0 bis xxx.xxx.xxx.63.

Eine weitere wichtige Option bietet das IPv6-Scanning. Bisher wurde mit IP stets die ältere IP-Version IPv4 vorausgesetzt. Ein Nmap-Aufruf kann IPv4- und IPv6-Adressen nicht gleichzeitig scannen. Daher konnte die IPv6-Option im Beispiel zuvor nicht aufgezeigt werden. Um IPv6-Adressen oder Adressräume zu scannen, wird im Nmap-Aufruf das Flag `-6` verwendet.

[5] `http://nmap.org/`

2.3 Dr. Portscan

Dr. Portscan ist ein Werkzeug mit dem Portscan-Ergebnisse chronologisch verglichen werden. Ergebnisse aus der Vergangenheit werden jeweils für eine bestimmte Maschine und dessen Port in einer Datenbank gespeichert. Diese Daten werden dann mit den aktuellen Ergebnissen verglichen. Treten Änderungen auf, werden diese in der Datenbank festgehalten. Dieses Vorgehen ist insbesondere von Vorteil für Personen, die nicht wissen, welche Ports ihrer Systeme explizit offen sein dürfen. Sie können so beispielsweise sehen, dass ein neuer Port auf einem System offen ist, der zuvor noch nie offen war. Mit dieser Information können dann weitere Details gesammelt werden und gegebenenfalls Maßnahmen ergriffen werden, wie beispielsweise die Benachrichtigung des zuständigen Administrators.

Dr. Portscan kann in die fünf Teilbereiche Datenquellen, Input-Agenten, Delta-Reporter, Datenbank und Output-Agenten gegliedert werden, siehe Abbildung 2.2.

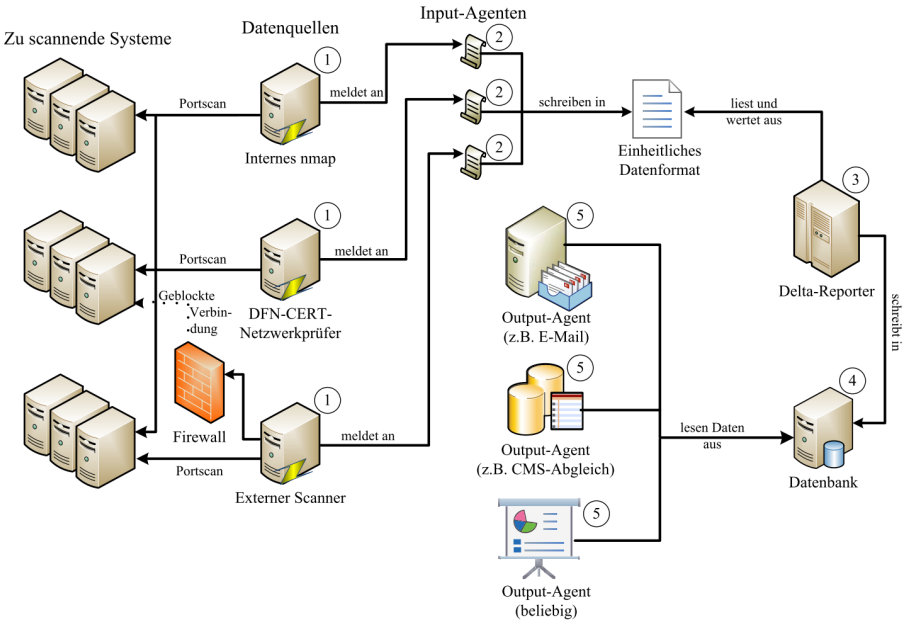

Abbildung 2.2: Die Systemarchitektur von Dr. Portscan. [vHM13, S. 11]

2.3.1 Datenquellen

Die Datenquellen sind die Systeme, auf denen die Portscanner laufen. Diese können im eigenen Netz (intern) oder außerhalb des eigenen Netzes (extern) platziert sein. Auf ihnen wird z. B. ein Nmap-Aufruf gestartet. Nmap sammelt dann die Informationen zu den gegebenen Parametern, die beispielsweise im XML-Format gespeichert werden können. Danach muss der so erstellte Scan-Bericht nur noch, unter Berücksichtigung des Dateinamens, in das richtige Verzeichnis von Dr. Portscan kopiert werden. Ist ein Scanner nicht auf dem gleichen System wie Dr. Portscan selbst platziert, kann der Bericht etwa per *SCP*[6] übertragen werden.

[6] Secure Copy (SCP) ist ein Protokoll zur sicheren Übertragung von Daten innerhalb eines Netzes, von einem Computer zum Anderen.

Um die Scans und/oder die Übertragung der Scan-Ergebnisse in bestimmten Intervallen zu starten, werden sogenannte *Cronjobs* erstellt. Damit können Aufgaben gewissermaßen nach einem Zeitplan ausgeführt werden, vgl. Kapitel 2.9. Weitere Details zur Kommunikation der Datenquellen mit Dr. Portscan folgen in Kapitel 4.2.1.

2.3.2 Input-Agenten

Die Input-Agenten nehmen die Scan-Ergebnisse der Portscanner entgegen. Um den Ablauf zu automatisieren, gibt es einen sogenannten *Input-Watcher*. Er wird regelmäßig in kurzen Zeitintervallen per Cronjob angestoßen. Das Skript sammelt alle Dateien, die von den Datenquellen im Verzeichnis für neue Scan-Ergebnisse abgelegt wurden. Diese Dateien müssen nach einer speziellen Syntax benannt sein, damit der Input-Watcher sie dem richtigen Input-Agenten, dem richtigen Scanner und dem Zeitpunkt zuordnen kann. Alle Scanner werden nämlich in einer Datenbank gespeichert, damit ihre Eigenschaften bei der Auswertung der Daten berücksichtigt werden können, dazu gleich mehr. Der Input-Agent nimmt nun beispielsweise den Bericht eines Scanlaufs von Nmap auf, filtert die notwendigen Daten für Dr. Portscan heraus und bringt diese in ein einheitliches Format, welches vom Delta-Reporter gelesen werden kann.

2.3.3 Delta-Reporter und Datenbank

Dr. Portscan verwendet eine zentrale, relationale Datenbank, um die Scan-Ergebnisse der dezentralen Portscanner zu aggregieren. In dieser Datenbank werden nicht nur die Scan-Ergebnisse gespeichert, sondern auch die einzelnen Scanner, die zum Einsatz kommen. Dadurch können die Ergebnisse immer den jeweiligen Scannern zugeordnet werden. Dies ist von Belang, da jeder Scanner Eigenschaften hat, welche bei der Auswertung berücksichtigt werden, um qualitativ bessere Ergebnisse zu erzielen. So wird beispielsweise die Platzierung der Scanner und auch die Verlässlichkeit gespeichert.

In Abbildung 2.3 ist das Datenbankmodell von Dr. Portscan abgebildet. Es gibt fünf Tabellen in der Dr.-Portscan-Datenbank. Die Tabellen „`hostResults`“, „`portResults`“, „`scanners`“ und „`changeIDs`“ werden für die Portscan-Ergebnisse und die Ergebnisse des Delta-Reporters benötigt. Die Tabelle „`clients`“ dient der Zuweisung von Netzbereichen zu bestimmte Personen. Diese Daten werden z. B. für die E-Mail-Reports verwendet.

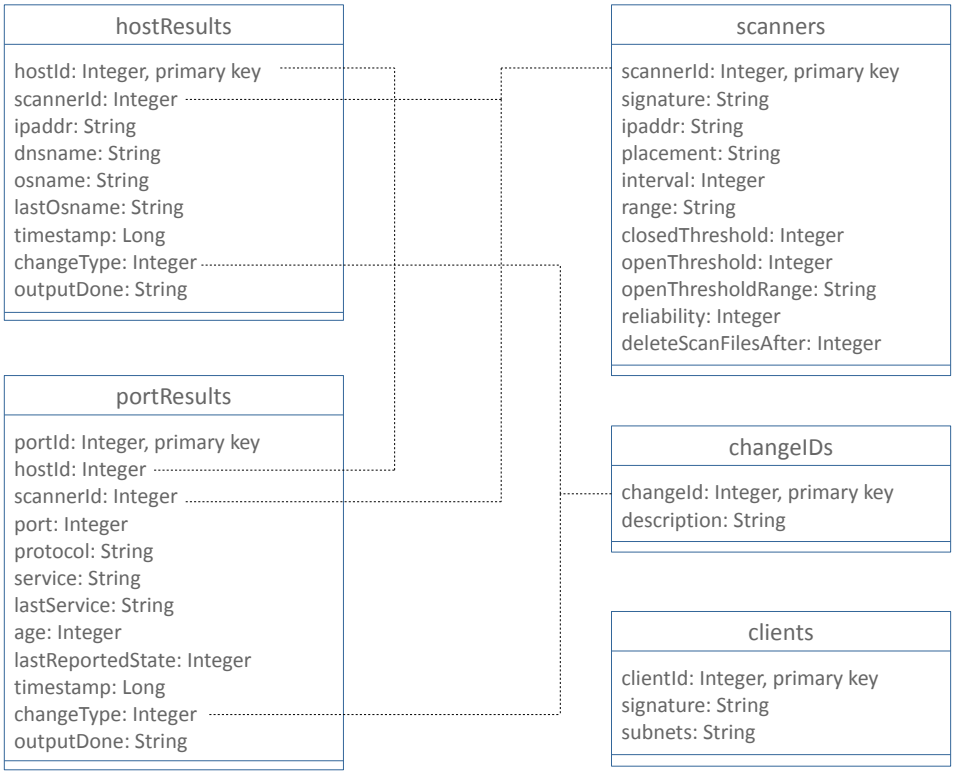

Abbildung 2.3: Das Datenmodell von Dr. Portscan. (Stand: 30.01.2014)

Die Tabelle „`scanners`“ enthält Daten zu den eingesetzten Datenquellen:

Attribut	**Erklärung**
scannerId	Eindeutige Scanner-ID
signature	Eindeutige Bezeichnung des Scanners
ipaddr	Die IP-Adresse des Scanners
placement	Die Platzierung des Scanners, insbesondere ob netzintern oder -extern
interval	Scan-Intervall in Sekunden. Ist der Wert 0 werden die Scans manuell ausgeführt
range	Liste der Netz- und Portbereiche, die gescannt werden
closedThreshold	Schwellenwert, nach wie vielen Portscans ein als geschlossen erkannter Port wirklich als geschlossen gemeldet werden soll
openThreshold	Schwellenwert, nach wie vielen Portscans ein als offen erkannter Port wirklich als offen gemeldet werden soll
openThresholdRange	Ports, welche gemäß *openTheshold* nicht sofort als offen gemeldet werden sollen
reliability	Zuverlässigkeit des Scanners, von 1 (sehr gut) bis 6 (ungenügend)
deleteScanFilesAfter	Zeitintervall in Sekunden, nach dem die Dateien mit den Scan-Ergebnissen vom Server gelöscht werden

Die Tabelle „`hostResults`“ enthält Informationen zu den gescannten Hosts:

Attribut	**Erklärung**
hostId	Eindeutige Host-ID
scannerId	ID des Scanners, der diesen Informationen geliefert hat
ipaddr	IP-Adresse des gescannten Hosts
dnsname	DNS-Name des gescannten Hosts
osname	Bezeichnung des auf dem Scanner laufenden Betriebssystems
lastOsname	Vorherige Bezeichnung des Betriebssystems, falls Änderung vorhanden
timestamp	Zeitpunkt, zu dem der Host gescannt wurde
changeType	ID des Änderungstyps
outputDone	Liste von Output-Agenten, die über den gegebenen Änderungstyp berichtet haben

Die Tabelle „`portResults`“ enthält Informationen zu den gescannten Ports eines Hosts:

Attribut	**Erklärung**
portId	Eindeutige Port-ID
hostId	ID des Hosts, dem dieser Port zugeordnet ist
scannerId	ID des Scanners, der diese Informationen geliefert hat
port	Portnummer auf den sich dieser Eintrag bezieht
protocol	Zugehöriges Protokoll
service	Zugehöriger Dienst
lastService	Voriger Dienst, falls Änderung vorhanden
age	„Zähler, der angibt, seit wie vielen Scan-Läufen desselben Scanners der Port offen oder geschlossen ist. Ein positiver Wert x bedeutet, dass der Port früher einmal offen war, seit x Scan-Läufen aber geschlossen ist. Ein negativer Wert y bedeutet, dass der Port seit y Läufen als offen erkannt wurde.“ [vHM13, S. 12]
lastReportedState	0 wenn Port zuvor geschlossen, 1 wenn Port zuvor offen
timestamp	Zeitpunkt, zu dem der Host gescannt wurde
changeType	ID des Änderungstyps
outputDone	Liste von Output-Agenten, die über den gegebenen Änderungstyp berichtet haben

Die Tabelle „`changeIDs`“ enthält Informationen zu den Änderungstypen die Ports bzw. Hosts vom Delta-Reporter erhalten können:

Attribut	**Erklärung**
changeId	ID des Änderungstyps
description	Beschreibung des Änderungstyps (Ausprägung siehe folgende Tabelle)

changeID	Beschreibung
0	Keine Veränderung gegenüber dem letzten Scan-Lauf
1	Neue Maschine (IP-Adresse ist zum ersten Mal auffällig geworden)
2	Maschine nicht mehr erreichbar
4	Veränderter DNS-Name
8	Neuer offener Port, der vorher noch nie offen war
16	Port inzwischen geschlossen (Port war früher offen)
32	Port wieder geöffnet, nachdem er zwischenzeitlich geschlossen war
64	Änderung des Betriebssystems
128	Änderung des Dienstes der auf dem Port läuft

Die Tabelle „`clients`“ enthält Informationen zu den Nutzern/Empfängern der Ergebnis-Reports:

Attribut	Erklärung
clientId	Eindeutige Klienten-ID
signature	Eindeutige Bezeichnung des Klienten
subnets	Liste der Subnetze, die dem Klienten zugeordnet sind

Nachdem die Input-Agenten die Scan-Ergebnisse ins richtige Format gebracht haben, ist der Delta-Reporter an der Reihe. Er trägt die neuen Ergebnisse, unter Berücksichtigung der aktuell in der Datenbank vorhandenen Einträge, in die Tabellen „`hostResults`“ und „`portResults`“ ein. Wird beispielsweise ein Port für einen vorhandenen Host zum ersten mal als offen erkannt, so trägt der Delta-Reporter den Änderungstyp „8“ in den zugehörigen Eintrag in der Tabelle „`portResults`“ ein. Ebenso werden Attribute wie *`lastOsname`*, *`lastService`* und *`lastReportetState`* durch den Delta-Reporter aktualisiert.

2.3.4 Output-Agenten

Es gibt verschiedene Typen von Output-Agenten. Generell sind sie dazu da, die Ergebnisse der Portscans in verwertbare und verständliche Informationen für den Endverbraucher zu bringen. Standardmäßig enthält Dr. Portscan einen XML-Output-Agent, welcher die aktuellen Ergebnisse für einen bestimmten Klienten in XML zusammenfasst, einen XMLToPlaintext-Output-Agent, welcher den XML-Output-Agenten nutzt und dessen Daten in einen individuell formatierbaren Text umwandelt, einen LastStateReport-Output-Agent, welcher alle Ergebnisse des letzten Scans für einen Klienten zusammenfasst und einen E-Mail-Report-Output-Agent, welcher einen E-Mail-Bericht mit individuell gestaltbarem Inhalt an einen Klienten schickt. Weitere Details zur Funktionsweise des E-Mail-Report-Output-Agenten in Kapitel 4.3.1.

2.4 Model-View-Controller Design-Pattern

Um Software möglichst sinnvoll aufzubauen und zu strukturieren gibt es sogenannte Architekturmuster, auch *Design Patterns* genannt. Diese Design-Patterns können als Schablonen für bestimmte softwaretechnische Problemkategorien interpretiert werden. Jedes Design-Pattern definiert also ein Vorgehen, wie an ein spezielles Problem herangegangen werden kann.

Das Model-View-Controller Design-Pattern (MVC-Pattern) eignet sich für jegliche Anwendungen, bei denen es eine grafische Benutzeroberfläche gibt, die auf einen bestimmten Datenbestand zugreift und diesen manipuliert. Das MVC-Pattern sieht für solche Anwendungsfälle eine Einteilung in drei Komponenten vor. Diese Dreiteilung erzielt ein flexibles Programmdesign, welches spätere Änderungen und die Wiederverwendbarkeit einzelner Komponenten ermöglicht. Außerdem sorgt es für eine klare Rollenzuteilung der einzelnen Komponenten sowie für eine gute Übersicht und Ordnung. [Ill07, S. 31]

Die drei Komponenten des MVC-Patterns:

- Das **Model** beschäftigt sich grundsätzlich mit der Datenhaltung. Werden Daten angefordert, geändert oder neu eingepflegt, geschieht dies stets durch das Model. Die Daten können während der Laufzeit vom Programm selber gehalten werden, in Dateien abgelegt oder in einer Datenbank gespeichert werden.
- Die **View** ist für die Darstellung der Daten zuständig. Sie wird mit Daten aus dem Model versorgt und baut mit Hilfe dieser Daten eine grafische Benutzeroberfläche auf.
- Der **Controller** ist für die bidirektionale Vermittlung von Daten zwischen der View und dem Model zuständig. Führt der User eine Aktion innerhalb der View aus, so werden die Daten, welche diese Aktion betreffen, an den Controller weitergegeben. Dieser verarbeitet die Daten und passt falls nötig seine gespeicherten Daten mithilfe des Models an. Der User wird daraufhin mit einer aktualisierten Nutzeroberfläche versorgt, indem der Controller beim Model entsprechende Daten anfordert und diese eigenständig verarbeitet, bevor sie an die View weitergegeben werden.

2.5 SSH und SCP

SSH (Secure Shell) ist ein Protokoll, sowohl als auch eine Implementierung des Protokolls. SSH bietet eine verschlüsselte Kommunikation innerhalb von Netzen mit einer zuverlässigen Authentifizierung des Clients beim Server und umgekehrt. Dabei wird TCP als verlässliches Transportprotokoll verwendet. Häufige Anwendungsfälle für SSH sind der Login auf entfernten Maschinen, die Ausführung von Kommandos auf entfernten Maschinen und das Kopieren von Dateien zwischen zwei Rechnern in einem Netz mittels SCP (Secure Copy).

SSH stellt einige verschiedene Arten der Authentifizierung bereit. Es gibt unter anderen die klassische Variante mit Benutzernamen und Passwort oder auch mittels eines Schlüsselpaars. Solch ein Schlüsselpaar besteht aus einem Public-Key und einem Private-Key. Der Private-Key gehört dem Client, also demjenigen, der sich auf dem Server einloggen will. Der Public-Key wird auf dem Server hinterlegt, auf den von außen zugegriffen werden soll. Will der Client nun Zugriff auf den Server bekommen, so kann er sich mit dem Privat-Key authentifizieren. Für zusätzliche Sicherheit sorgt ein sogenanntes *Passphrase*, also eine Art Passwort, welches den Private-Key vor unbefugtem Missbrauch schützt. [HR09, S. 19f]

2.6 Skriptsprachen

Eine Skriptsprache ist eine Programmiersprache. Klassische Programmiersprachen wie Java oder C++ werden in der Regel vor der Ausführung kompiliert, d. h. in eine für den Computer verständlichen Maschinensprache übersetzt. Skriptsprachen hingegen werden während

der Ausführung des Codes von einem sogenannten *Interpreter* verarbeitet. Dieser analysiert den Quellcode zur Laufzeit des Programms und übersetzt ihn direkt in Maschinenanweisungen. Der Interpreter kann als virtuelle Maschine angesehen werden, die sich zwischen der Programmiersprache und dem Prozessor einordnet.

Skriptsprachen bieten den Vorteil, dass sie oft für bestimmte Anwendungsfälle spezialisiert sind. Dadurch kann durch wenig Code oft eine sehr große Wirkung erzielt und so die Entwicklungszeiten verkürzt werden. Nachteilig ist die meist geringere Geschwindigkeit gegenüber klassischen Programmiersprachen die kompiliert und nicht interpretiert werden. Z. B. der geringere Typisierungsgrad der Variablen bei Skriptsprachen zieht oft deutlich mehr CPU-Instruktionen pro Befehl mit sich, was wiederum zu einem höheren CPU-Ressourcenbedarf führt. [cwS07]

Skriptsprachen werden häufig für dynamische Webseiten genutzt. Dazu schickt ein Client eine Anfrage an einen Server, dieser verarbeitet die Anfrage und sendet die Antwort zurück an den Klienten. Diese Art der Anwendung von Skriptsprachen nennt sich *serverseitige Skriptsprache*. Nun gibt es Skriptsprachen, die auf dem Rechner des Clients ausgeführt werden, sogenannte *clientseitige Skriptsprachen*. Dazu muss der Rechner die nötigen Voraussetzungen erfüllen. Ein Beispiel sind Browser mit JavaScript-Unterstützung. [Hab05, S. 10ff]

2.6.1 Serverseitige Skriptsprachen

Serverseitige Skriptsprachen werden nicht nur für dynamische Webseiten verwendet, siehe Abbildung 2.4, sondern auch um kleinere regelmäßige Arbeiten zu erleichtern. Oft werden Skripte durch einen Cronjob in festgelegten Intervallen ausgeführt um z. B. Datenbanken zu sichern oder bestimmte Daten aktuell zu halten.

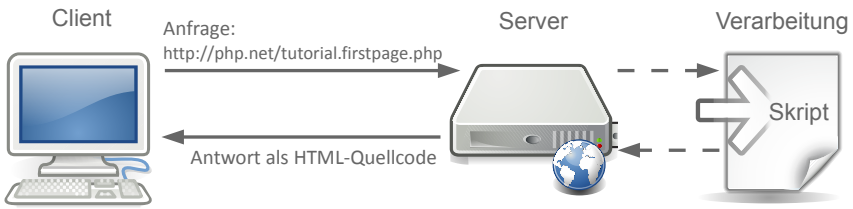

Abbildung 2.4: Ablauf serverseitiger Skriptsprachen am Beispiel einer Webanwendung. (Urheber der Symbole: *RRZEicons* unter der Lizenz: *Attribution-ShareAlike 3.0 Unported*, nicht geändert)

PHP

PHP (PHP Hypertext Preprocessor) ist eine Skriptsprache, die vorrangig im Bereich der Web-Entwicklung Einsatz findet, da sie viele nützliche Funktionen für diesen Zweck bereitstellt. Es werden ohne zusätzliche Installation von weiteren Bibliotheken viele Datenbanken, wie beispielsweise MySQL, PostgreSQL und SQLite, unterstützt. Ebenfalls mit in der

Standard-Installation eingeschlossen sind sogenannte *Sessions*, die bei mandantenfähigen Webanwendungen einige Vereinfachungen bei der Implementierung mit sich bringen. [Hab05, S. 30ff]

Perl

Perl (Practical Extraction and Reporting Language) ist eine universell einsetzbare Skriptsprache, die sehr gut mit Zeichenketten umgehen kann und somit häufig in der Datenauswertung Einsatz findet. Perls Kernfunktionen sind Mengenmäßig überschaubar, allerdings stehen sehr viele sogenannte *Module* zur Verfügung, die den Funktionsumfang stark erweitern. Dr. Portscan ist in Perl implementiert und nutzt ein eigens geschriebenes Perl-Modul, in dem einige Konstanten und häufig verwendete Funktionen definiert sind.[Hab05, S. 19ff]

Shell-Skripte

Shell-Skripte wie `sh` (Shell Command Language) oder `bash` (Bourne-again shell) sind meist ein Werkzeug von Systemadministratoren. Sie erledigen Aufgaben sich immer wiederholender Prozesse wie das Löschen veralteter Dateien in bestimmten Verzeichnissen oder das Übertragen von Dateien auf andere Rechner. Es ist auch noch weitaus mehr möglich mit Shell-Skripten, da sie auch Bedingungen und Schleifen beherrschen.

2.6.2 Clientseitige Skriptsprachen

Clientseitige Skriptsprachen werden, bei Betrachtung von Webseiten, auf dem Computer des Clients ausgeführt. Der Client bekommt also eine Antwort vom Server und diese Antwort wird durch ein mitgeliefertes Skript aufgewertet, vgl. Abbildung 2.5.

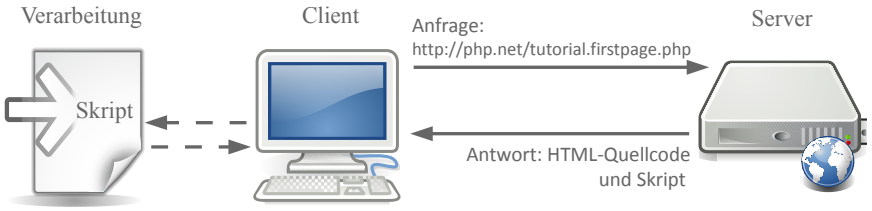

Abbildung 2.5: Ablauf clientseitiger Skriptsprachen am Beispiel des Aufrufs einer Website die ein Skript enthält. (Urheber der Symbole: *RRZEicons* unter der Lizenz: *Attribution-ShareAlike 3.0 Unported*, nicht geändert)

JavaScript

Die wohl bekannteste clientseitige Skriptsprache ist JavaScript. Webseiten mit JavaScript bieten eine Vielzahl an Möglichkeiten im Bezug auf Benutzerinteraktionen, das Verändern von Inhalten oder das Nachladen von Daten. Damit können Webseiten benutzerfreundlicher gestaltet werden und oftmals auch Netzwerkverkehr eingespart werden, indem z. B. nur

relevante Daten nachgeladen werden und dann clientseitig verarbeitet werden, anstatt die Webseite komplett neu zu laden.

2.7 Stylesheets - CSS

Ein CSS (Cascading Style Sheet) besteht aus einer Liste von Regeln zur Formatierung strukturierter Daten. Eine Regel setzt sich aus zwei Teilen zusammen: Dem Selektor, welcher die Elemente auswählt, die für diese Regel zutreffen, und den Deklarationen, welche aus einer Liste von Eigenschaft-Werte-Paaren bestehen, die das Format für die ausgewählten Elemente spezifizieren.

CSS-Stylesheets werden häufig in Verbindung mit HTML[7] verwendet. Öffnet man im Browser beispielsweise eine HTML-Dokument welches CSS-Code enthält, so wendet der Browser beim Darstellen des Dokuments diese Regeln an. Ist kein CSS-Code enthalten, wird nichts desto trotz der browser-interne, voreingestellte CSS-Code zur Darstellung verwendet. HTML ist nur ein Beispiel von vielen, bei denen Stylesheets zur Gestaltung von strukturierten Daten verwendet werden.

2.8 Datenbanken

Datenbanksysteme dienen der Speicherung und Pflege, der Beschreibung und der Wiedergewinnung von Datenmengen, die von verschiedenen Anwendungsprogrammen dauerhaft genutzt werden. [Böh05, S. 25] Im Gegensatz zum normalen Dateisystem bieten Datenbanksysteme Möglichkeiten, Zugriffe zu synchronisieren, so können viele Benutzer gleichzeitig mit den Daten arbeiten. Auch die flexible Zugriffskontrolle, um unautorisierte Zugriffe auszuschließen, ist ein Vorteil.

„Eine Datenbank wird in zwei Ebenen eingeteilt:

- Datenbankschema:
  - Beschreibt den möglichen Inhalt.
  - Struktur- und Typinformationen der Daten (Metadaten).
  - Art der Beschreibung ist durch das Datenmodell vorgegeben.
  - Änderungen sind möglich, aber selten.
- Ausprägung der Datenbank:
  - Tatsächlicher Inhalt.
  - Objektinformationen und Attributwerte.
  - Struktur ist durch Datenbankschema vorgegeben.
  - Änderungen sind häufig.“ [Böh05, S. 28]

Es gibt verschiedene Datenmodelle denen Datenbanken folgen. Diese Datenmodelle beschreiben die Weise, in der Objekte und Beziehungen in der Datenbank dargestellt werden. Beispiele sind das hierarchische und das relationale Datenmodell, siehe Abbildung 2.6.

[7] Hypertext Markup Language (HTML) ist eine Auszeichnungssprache, welche logische Bestandteile eines Dokuments strukturiert. Ein HTML-Dokument wird üblicherweise mit einem Browser betrachtet und als Website dargestellt.

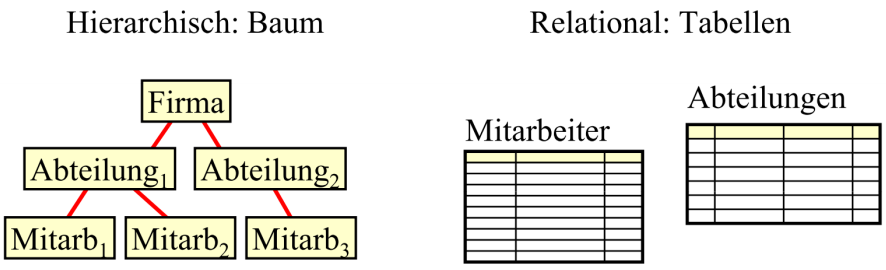

Abbildung 2.6: Das hierarchische und das relationale Datenmodell. (Quelle [Böh05, S. 32])

Um Daten in der Datenbank abzufragen oder zu bearbeiten, werden Datenbanksprachen verwendet. SQL (Structured Query Language) ist eine weit verbreitete Datenbanksprache. Sie wird von vielen Datenbanksystemen, allerdings oft in unterschiedlichem Umfang, verwendet. Der Vorteil dieser häufigen Verwendung liegt in der Austauschbarkeit des Datenbanksystems. Will man beispielsweise eine Anwendung von SQLite auf MySQL (siehe nachfolgende Abschnitte) umstellen, so sind oft nur geringe Änderungen notwendig.

2.8.1 SQLite

Bei SQLite handelt es sich um ein relationales Datenbanksystem. Es wird lediglich durch die SQLite-Bibliothek eingebunden und benötigt keinen Server-Dienst, um eingesetzt werden zu können, was einen großen Unterschied zu den meisten anderen Datenbanksystemen darstellt. Die SQLite-Bibliothek gibt es für alle großen Plattformen und ist lediglich wenige hundert Kilobyte groß. Da es sich nicht um einen eigenen Dienst handelt, gelten für SQLite-Datenbanken die Zugriffsberechtigungen des Dateisystems. Dr. Portscan wird aktuell mit einer SQLite Datenbank betrieben.

2.8.2 MySQL

MySQL ist ebenfalls ein relationales Datenbanksystem, welches als Open-Source-Software sowie als Enterprise-Version für viele Plattformen zur Verfügung steht. Um MySQL nutzen zu können, muss es auf einem Server installiert und der zugehörige Dienst gestartet werden. Danach können MySQL-Clients Anfragen an den Server senden. MySQL ist durch den Dienst, im Gegensatz zu SQLite, also auch über das Netzwerk erreichbar. MySQL ist ein vollwertiges Datenbankmanagementsystem und kommt deshalb, im Gegensatz zu SQLite, auch mit gleichzeitigen Zugriffen mehrerer Benutzer zurecht.

2.9 Cronjobs

Der sogenannte *Cron* ist ein Dienst für Unix-Betriebssysteme, welcher nach einer gegebenen Zeittafel, bestimmte Aufgaben erledigt. Diese Aufgaben werden Cronjobs genannt. Ein Cronjob kann beispielsweise ein bash-Skript sein, das regelmäßig ausgeführt wird um veraltete Log-Dateien vom System zu entfernen. Um einen Cronjob zu erstellen ist ein Eintrag in die dafür vorgesehene Tabelle, den *Crontab*, nötig. Solch ein Eintrag hat folgendes Format:

```
%Minute %Stunde %Tag %Monat %Wochentag %Befehl
```

Alle diese Platzhalter müssen stets gesetzt sein. Am Ende steht der eigentlich auszuführende Befehl. Die Platzhalter davor dienen der Zeitplanung. Zur näheren Erläuterung der Zeitplanung sollen zwei Beispiele dienen:

Minute	**Stunde**	**Tag**	**Monat**	**Wochentag**	**Erläuterung**
*/30	*	15	1,3,5,7,9,11	*	Alle 30 Minuten, am 15. jedes zweiten Monats beginnend mit Januar
0	11,18	*	*	1-5	Montags bis Freitags um 11:00 und 18:00 Uhr

Ein Eintrag der Form `*/15 * * * * skript.sh` führt beispielsweise viertelstündig das Shell-Skript *skript.sh* aus.

3 Anforderungen an das Dr. Portscan Webfrontend

Im Folgenden werden alle Anforderungen an das Webfrontend für Dr. Portscan beschrieben. Um diese Anforderungen zu rechtfertigen, wird anfangs das zugrundeliegende Szenario erläutert. Danach folgen einige sogenannte *Use-Cases*, also Anwendungsfälle, mit denen Anforderungen an das Webfrontend gefunden und gerechtfertigt werden sollen. Diese werden bereits zu Beginn in Kategorien eingeteilt, die dem Aufbau des Webfrontends entgegen kommen. Zum Schluss werden die so gefundenen Anforderungen noch einmal kategorisiert und priorisiert, um eine strukturierte und für den Systementwurf sowie die Implementierung sinnvolle Übersicht darzustellen.

3.1 Szenario

Das zu implementierende Webfrontend für Dr. Portscan soll im Münchner Wissenschaftsnetz (MWN) eingesetzt werden, welches vom Leibniz-Rechenzentrum (LRZ) betrieben wird. Das MWN verbindet viele Hochschulen in und um München, einige Institute der Forschung, Museen und weitere außer-universitäre Einrichtungen wie Bibliotheken. Das MWN unterteilt sich in über 800 Subnetze, an denen insgesamt ca. 100.000 Endgeräte angeschlossen sind. Die Verwaltung der Subnetze erfolgt dezentral. Es gibt ca. 900 Netzverantwortliche in den Fachbereichen der Hochschulen und Institute der angeschlossenen Einrichtungen. Die Entwicklung des Dr.-Portscan-Webfrontends soll primär diesen Netzverantwortlichen dienen.

Um ein besseres Bild über die Arbeit eines Netzverantwortlichen zu bekommen, werden seine Aufgabenbereiche kurz vorgestellt:

- „Verwaltung der zugeteilten Namens- und Adressräume.
- Führung einer Dokumentation über die ans MWN angeschlossenen Endgeräte bzw. Netze.
- Zusammenarbeit mit dem LRZ bei der Planung und Inbetriebnahme von Erweiterungen der Gebäudenetze (neue Anschlusspunkte, neue Netzstrukturen).
- Mitarbeit bei der Fehlerbehebung (z.B. Durchführen von mit dem LRZ abgestimmten Tests zur Fehlereingrenzung).
- Zusammenarbeit mit dem LRZ bei der Eindämmung missbräuchlicher Netznutzung.“ [Hom12, S. 28]

Ein Netzverantwortlicher betreut in den meisten Fällen ein relativ großes Netz und weiß somit nicht immer über die Endgeräte und deren Konfiguration bezüglich offener Ports und

genutzter Dienste Bescheid. Das Erkennen von Schwachstellen erfolgt deshalb in enger Zusammenarbeit mit den Administratoren, welche ein relativ kleines Netz, innerhalb des Netzbereichs eines Netzverantwortlichen, administrieren und über die Konfiguration der Endsysteme Bescheid wissen. Betreut ein Netzverantwortlicher einen großen Netzbereich, so ist es möglich, dass dieser Netzbereich wiederum in mehrere Netzbereiche unterteilt wird, die von weiteren Subnetzverantwortlichen[1] verwaltet werden.

Alle Netzverantwortlichen sind in einer Datenbank, der sogenannten Netzdoku-Datenbank, am Leibniz-Rechenzentrum registriert. Dort werden Informationen wie Netzbereich, E-Mail, Institut usw. gespeichert. Außerdem wurde 2008, im Rahmen des Projekts LRZ-SIM (Secure Identity Management), eine neue Benutzerverwaltung erstellt, die unter anderem Zugangsdaten der Netzverantwortlichen enthält. [Hom12, S. 17]

Dr. Portscan scannt derzeit nur einige wenige LRZ-interne Netzbereiche. Damit die Netzverantwortlichen ihren Netzbereich von Dr. Portscan scannen lassen können, müssen sie beim Dr.-Portscan-Administrator eine Anfrage stellen. Dieser kann daraufhin manuell die Skripte und Datenbanktabellen von Dr. Portscan anpassen, sodass der Netzverantwortliche E-Mail-Berichterstattungen über die Ergebnisse der Scan-Durchläufe erhält. Dieser Ablauf stellt eine relativ unbequeme Einrichtung des Dr.-Portscan-Werkzeugs für Netzverantwortliche als auch den Dr.-Portscan-Administrator dar. Das Webfrontend soll hierfür Abhilfe leisten.

3.2 Beteiligte Rollen

Es gibt zwei verschiedene Gruppen, die das Webfrontend benutzen sollen. Zum einen den Netzverantwortlichen, der nach aktuellem Stand Ergebnisse der Scans per E-Mail-Report einsehen kann. Zum Anderen gibt es den Dr.-Portscan-Administrator, der Zugriff auf den Server hat auf dem Dr. Portscan läuft und dort alles per Texteditor oder Terminal konfigurieren kann.

3.3 Use-Cases der Netzverantwortlichen

Die nachfolgenden Use-Cases werden bereits in Kategorien eingeteilt, die sich aus dem bisherigen Dr.-Portscan-Tool und der Aufgabenstellung ergeben. Für die Netzverantwortlichen kommen somit die Kategorien Konfiguration, Berichterstattung und Verwaltung in Betracht.

3.3.1 Konfiguration

Fall 1: Ein Netzverantwortlicher wird von einigen Administratoren in seinem Netz darauf aufmerksam gemacht, dass in letzter Zeit häufiger Portscans beim Auswerten der Router-Protokolle entdeckt wurden. Daraus schließt er ein mögliches Ausspionieren von Schwachstellen in seinen Systemen. Er weiß, dass am LRZ Dr. Portscan angeboten wird um Netzbereiche automatisiert zu scannen und um Veränderungen zu erfassen. Nun will er, dass sein Netzbereich von Dr. Portscan überwacht wird.

Resultierende Anforderung (1): Netzverantwortliche sollen ihren Netzbereich einfach zum Scannen durch Dr. Portscan hinzufügen können.

[1] Ein Subnetzverantwortlicher ist ein Netzverantwortlicher, der einen Teil eines Netzes von einem anderen Netzverantwortlichen betreut.

Fall 2: Der Netzverantwortliche aus Fall 1 nimmt die entsprechenden Berichte der Router-Protokolle in Augenschein und stellt fest, dass die Portscans nicht von innerhalb des MWN kommen, sondern von unbekannten Rechnern aus dem Internet, also extern. Er will deshalb ebenfalls Portscans von extern ausführen. Nichts desto trotz wäre ein interner Scan ebenfalls interessant für ihn, da er somit gleich die Wirksamkeit seiner Firewall testen kann. Er weiß, dass Dr. Portscan diese Möglichkeit verteilter Portscanner bietet und beschließt sowohl intern als auch externe Scanner einzusetzen.

Resultierende Anforderung (2): Die Möglichkeit sowohl von intern als auch von extern Portscans durchzuführen. Dabei sollen die beiden Scans unabhängig voneinander konfiguriert werden können.

Fall 3: Der Netzverantwortliche lässt nun seinen Netzbereich mit den Standard-Einstellungen der Portscanner scannen und will die Ergebnisse der Scans strukturiert und aufbereitet einsehen können und jeden Freitag einen E-Mail-Bericht erhalten. Diese E-Mails mag er in gewissem Maße formatieren können, damit sie in einen dafür vorgesehenen Ordner seines E-Mail-Programms landen. Außerdem will er die E-Mail-Berichte gleich an die zuständigen Administratoren weiterleiten können.

Resultierende Anforderung (3): Die E-Mail-Berichterstattung soll zu einer bestimmten Zeit versendet werden. Außerdem sollen bestimmte Empfänger für einzelne Netzbereiche festgelegt werden können. Bei dieser E-Mail sollen einige Möglichkeiten der Personalisierung bestehen, wie z. B. das Festlegen des Betreff-Felds.

Fall 4: Ein Netzverantwortlicher verwendet schon seit längerem ein Skript, um seine selbst getätigten Portscans auswerten zu lassen und Statistiken zu erstellen. Nun will er seine eigenen Scans einstellen und auf die Möglichkeit der Portscans von Dr. Portscan zurückgreifen ohne dabei auf die Auswertungen seines Skriptes zu verzichten. Würden die Portscan-Ergebnisse der E-Mail-Berichterstattungen in einem bestimmten Format vorliegen, so könnte er das Skript einfach auf neue E-Mail-Berichterstattungen in seinem dafür vorgesehenen E-Mail-Ordner (siehe Fall 3) ansetzen.

Resultierende Anforderung (4): Festlegen des Formats der einzelnen Scan-Ergebnisse der E-Mail-Berichterstattung, also die Reihenfolge der verfügbaren Attribute der Scan-Ergebnisse, wie IP-Adresse, Portnummer, Protokoll, offen/geschlossen und das Scan-Datum.

Fall 5: Viele Hochschulinstitute bieten Studierenden oder wissenschaftlichen Mitarbeitern den Zugang zum Internet über das sogenannte *eduroam* an. Innerhalb dieses dafür zuständigen Netzbereichs werden IP-Adressen dynamisch vergeben. Einem Netzverantwortlichen bringen Portscans dieses Netzbereiches somit nichts, außer eine Verlangsamung seiner Scan-Durchläufe.

Resultierende Anforderung (5): Netzbereiche gezielt definieren und Ausnahmelisten erstellen können.

Fall 6: Der Netzverantwortliche eines Instituts will seinen Netzbereich IPv6-fähig machen. Daraufhin realisiert das LRZ die IPv6-Fähigkeit über eine sogenannte native Anbindung. Dabei wird, parallel zum bestehenden IPv4-Anschluss, IPv6 aktiviert. Nachdem einige Endsysteme auf das sogenannte *Dualstack*, also sowohl IPv4 als auch IPv6, migriert wurden, möchte der Netzverantwortliche diese nicht nur per IPv4-Scan, sondern auch per IPv6-Scan von Dr. Portscan überprüfen lassen.

Resultierende Anforderung (6): Möglichkeit zum Scannen einzelner IPv6-Endsysteme.

Fall 7: Ein Netzverantwortlicher wird von einem Administrator darüber informiert, dass neuerdings auf einem seiner Server BitTorrent läuft. Neue TCP- und UDP-Ports im Bereich von 6881–6999 werden dafür benutzt. Bisher werden in seinem Netz nur die Standardports von 1–1024 von Dr. Portscan gescannt.

Resultierende Anforderung (7): Netzverantwortliche sollen die Möglichkeit haben, sowohl TCP- als auch UDP-Ports zu scannen. Außerdem sollen die Portbereiche/-nummern individuell eingestellt werden können.

Fall 8: Seitdem ein Netzverantwortlicher sein Netz mit Dr. Portscan scannen lässt, beklagen sich einige Administratoren über ungewöhnliche Abstürze ihrer Systeme. Eine mögliche Ursache dafür könnten zu aggressiv vorgehende Portscans sein.

Resultierende Anforderung (8): Einstellmöglichkeit der Scan-Aggressivität durch den Netzverantwortlichen.

Fall 9: Nach den ersten Ergebnissen von Dr. Portscan fällt dem Netzverantwortlichen auf, dass in der Ergebnisliste unmöglich alle Endsysteme seines Netzbereichs aufgelistet sind. Nach Absprache mit einigen Administratoren kommt er zu dem Schluss, dass der Zeitpunkt der Scans womöglich ungünstig gewählt ist. Einige Systeme werden ab 18 Uhr bis zum nächsten Morgen um 7 Uhr heruntergefahren. Wird zu diesem Zeitpunkt gescannt, werden diese Systeme nicht erfasst.

Resultierende Anforderung (9): Ungefähres Festlegen der Zeitpunkte der eigenen Scanläufe.

Fall 10: Der E-Mail-Ordner eines Netzverantwortlichen, in dem die Dr.-Portscan-Berichte landen, ist mit vielen ungelesenen Berichten gefüllt. Der Netzverantwortliche sieht sich nur jeden Montag und Freitag den aktuellen Bericht an.

Resultierende Anforderung (10): Vorgegebene Scan-Intervalle wie täglich, zwei mal wöchentlich oder wöchentlich.

3.3.2 Berichterstattung

Fall 11: Die E-Mail-Berichte von Dr. Portscan sind einem Netzverantwortlichen zu unübersichtlich. Ihm fehlt eine strukturierte Tabelle in der er vielleicht sogar nach bestimmten Kriterien sortieren oder filtern kann, ohne dass er eigene Skripte auf seine E-Mail-Berichterstattungen ansetzen muss. Er will dort, wie bei den E-Mail-Berichten, die Übersicht über alle aktuellen Ergebnisse sowie eine Übersicht über die Veränderungen gegenüber den letzten Scan-Ergebnissen bekommen.

Resultierende Anforderung (11): Möglichkeit der Einsichtnahme aktueller Scan-Ergebnisse (Veränderungen gegenüber letztem Scan-Ergebnis und Gesamtübersicht) in einer strukturierten und eventuell filterbaren Tabelle, die über eine zentrale Anlaufstelle (Website) erreichbar ist.

Fall 12: Der Netzverantwortliche lässt seinen Netzbereich mit den Standard-Einstellungen der Portscanner scannen und will die Ergebnisse der Scans strukturiert und aufbereitet einsehen können. Außerdem will er per E-Mail eine Berichterstattung erhalten, die neue Scan-Ergebnisse enthält.

Resultierende Anforderung (12): Einsichtnahme der Scan-Ergebnisse für den betreuten Netzbereich und Berichterstattung per E-Mail.

3.3.3 Verwaltung

Fall 13: Der Netzverantwortliche der Rechnerbetriebsgruppe (RBG) der TU-München betreut ein */16-Netz*, d. h. einen sehr großen Netzbereich mit über 60.000 zu vergebenden IP-Adressen. Würde er diesen Bereich scannen wollen, würde dies sehr viel Zeit in Anspruch nehmen und der Netzverantwortliche selbst würde lange für die Betrachtung und Auswertung der Ergebnisse benötigen. Er will diese Aufgabe deswegen an mehrere Netzverantwortliche, die Subnetze seines Netzes betreuen, verteilen. Diese sollen die Portscans eigenständig von Dr. Portscan durchführen lassen können.

Resultierende Anforderung (13): Netzverantwortliche sollen eigenständig ihren Netzbereich verwalten können. Dazu gehört das Hinzufügen neuer Subnetzverantwortlicher, also solchen, die ein Subnetz des Netzverantwortlichen betreuen. Ändert sich beispielsweise eine betreuende Person für ein Subnetz, soll der Netzverantwortliche die nötigen Verwaltungsaufgaben selbstständig durchführen können.

Fall 14: Ein Subnetzverantwortlicher hat einen Scan für ein /28-Netz, d. h. maximal 14 IP-Adressen, bei Dr. Portscan eingestellt. Dieser Netzbereich soll zukünftig durch einen anderen Subnetzverantwortlichen übernommen werden.

Resultierende Anforderung (14): Netzverantwortliche sollen die Möglichkeit haben Subnetzverantwortliche zu sperren oder zu löschen, damit deren Scans nicht mehr durchgeführt werden.

3.4 Use-Cases des Dr.-Portscan-Administrators

Auch hier werden durch Dr. Portscan und die Aufgabenstellung dieser Arbeit bereits einige Kategorien vorgegeben. Die Aufgabenbereiche für den Dr.-Portscan-Administrator unterteilen sich in Verwaltung und Konfiguration. Fortan wird der Dr.-Portscan-Administrator mit DPA abgekürzt.

3.4.1 Verwaltung

Fall 15: Der DPA hat die initiale Einrichtung des Dr.-Portscan-Tools auf einem Server hinter sich. Nun möchte er möglichst vielen Netzverantwortlichen den Zugriff zu Dr. Portscan ermöglichen. Bei Dr. Portscan kann er diese als User in die Datenbank eintragen und sie in den Konfigurationsdateien aufnehmen, damit sie E-Mail-Berichterstattungen der Scan-Ergebnisse erhalten. Das MWN hat ca. 900 Netzverantwortliche, alle manuell hinzuzufügen wäre ein enormer Aufwand. Der DPA weiß allerdings, dass alle Informationen, die er zu den Netzverantwortlichen benötigt, bereits in anderen Datenbanken und Verzeichnissen zu Verfügung stehen.

Resultierende Anforderung (15): Automatischer Datenbank-Import bestehender Netzdoku-Datenbank.

Fall 16: Der DPA wird von einem Netzverantwortlichen kontaktiert, welcher nicht in der Netzdoku-Datenbank enthalten ist und will wissen, ob er nicht auch sein Netz von Dr. Ports-

can scannen lassen kann. Nach der Überprüfung der Korrektheit seiner Angaben spricht nichts dagegen.

Resultierende Anforderung (16): Manuelles Hinzufügen von Netzverantwortlichen, die nicht in der Netzdoku-Datenbank (siehe Fall 15) registriert sind.

Fall 17: Bei einer Überprüfung der Logbucheinträge, die Dr. Portscan erstellt, stellt der DPA fest, dass einige Portscans nicht mehr bis zum Ende durchlaufen und deswegen auch keine Ergebnisse liefern. Außerdem benötigen einige dieser fehlschlagenden Portscans über eine Stunde bis zum Auftreten dieses Fehlers, weshalb viele Ressourcen des Servers verschwendet werden. Er beschließt diese Scans bei den nächsten Durchläufen auszulassen, um Ressourcen zu sparen und auf Fehlersuche zu gehen.

Resultierende Anforderung (17): Verwaltungsmöglichkeit aller eingestellten Scans. Dazu gehört das Blockieren von Scans die von Nutzern von Dr. Portscan eingestellt wurden.

Fall 18: Dr. Portscan findet immer mehr Zuwachs, da mehr und mehr Netzverantwortliche Gefallen an der einfachen Übersicht über Veränderungen an ihren Endsystemen finden. Mittlerweile haben sich über 300 der ca. 900 Netzverantwortlichen Zugang zu Dr. Portscan verschafft. Der DPA überprüft daraufhin die Datenquellen, also die Server auf denen die Portscans durchgeführt werden. Bei dem bisher einzigen internen Scanner überwacht er einige Tage die ungefähre Dauer, die alle Scans täglich benötigen. Die Auswertung ergibt, dass alles Scans zusammen im Durchschnitt 20 Stunden für die Abarbeitung benötigen. Nun stellt sich für den DPA die Frage, ob bei einer so hohen Auslastung noch die Zeitpläne der Scans (siehe Fall 9) eingehalten werden können.

Resultierende Anforderung (18): Übersicht über alle eingestellten Scans mit deren ungefähr benötigten Scan-Dauer.

Fall 19: Der manuell hinzugefügte Mandant aus Fall 16 beendet sein Arbeitsverhältnis. Das Institut hat einen Nachfolger bekannt gegeben, welcher nun die Betreuung des Netzbereichs übernehmen soll.

Resultierende Anforderung (19): Verwalten der nötigen Daten, der manuell hinzugefügten Mandanten, wie z. B. E-Mail-Adresse und Name, oder auch das Löschen von Mandanten.

Fall 20: Der Netzverantwortliche der Rechnerbetriebsgruppe der TU-München betreut ein /16-Netz. Er stellt einen Portscan bei Dr. Portscan ein, der das komplette Netz einmal in der Stunde neu scannt, damit er stets über aktuelle Daten verfügt. Der DPA bemerkt bei einer Konfiguration eines internen Scanners, dass dieser sehr langsam reagiert und geht dem auf den Grund. Er erkennt, dass alle jede Stunde ein Scan für einen großen Netzbereich gestartet wird.

Resultierende Anforderung (20): Der DPA soll die Möglichkeit haben, eingestellte Scans der Netzverantwortlichen zu sperren, sodass diese nicht mehr ausgeführt werden.

3.4.2 Konfiguration

Fall 21: Da ein externer Scanner langsam an seine Grenzen stößt, denkt der DPA über einige ressourcenschonende Möglichkeiten nach. Eine davon wäre, die Scans gezielter auf Systeme anzusetzen, die auch wirklich von Belang sind. Es gibt beispielsweise Netzbereiche

die ihre IP-Adressen dynamisch vergeben, wie z. B. für die Vergabe von IPs um Studenten und Mitarbeitern Internetzugang per WLAN zu ermöglichen (vgl. Fall 5).

Resultierende Anforderung (21): Definieren von Ausnahmelisten durch den DPA. Diese IP-Bereiche werden dann von allen Portscannern ignoriert.

3.5 Sonstige Anforderungen

Einige mögliche Anforderungen ergeben sich durch den Aufbau von Dr. Portscan selbst oder anderen Gegebenheiten und haben keinen rechtfertigenden Use-Case. So könnten beispielsweise Konfigurationseinstellungen, wie unter anderen Pfadangaben zu dem Ordner der neu eingetroffenen Scan-Ergebnisse, über das Webfrontend ermöglicht werden, um dem Dr.-Portscan-Administrator eine zentrale Anlaufstelle zur Konfiguration von Dr. Portscan zu bieten. Eine weitere Anforderung kann die und Konfiguration der Portscanner über das Webfrontend sein, was ebenfalls möglich wäre.

Einige funktionale und sicherheitsrelevante Anforderungen gibt es schließlich noch. Eine Anforderung ist die sichere Authentifizierung der Mandanten, sodass keiner unberechtigt Zugriff auf das Tool bekommt. Weiterhin muss auch stets sichergestellt sein, dass jeder Mandant nur in seinem zuständigen Netzbereich konfigurieren, verwalten und Einsicht die Scan-Ergebnisse erhalten kann. Eine strikte Trennung der Bereiche der Netzverantwortlichen und des Dr.-Portscan-Administrator ist ebenfalls eine grundlegende Anforderung an das Webfrontend. Diese drei Anforderungen sind essenziell für das Realisieren der geforderten Mandantenfähigkeit und werden deshalb Bestandteil der Umsetzung sein.

3.6 Kategorisierung und Priorisierung der Anforderungen

Um die zuvor gesammelten Anforderungen für den nachfolgenden Systementwurf und die Implementierung sinnvoll aufzubereiten, werden nun die bereits bestehenden Kategorien, unter Berücksichtigung der zwei Benutzergruppen, wieder aufgegriffen. Zusätzlich wird eine Priorisierung der erarbeiteten Anforderungen vorgenommen. Da bei der Implementierung möglicherweise nicht alle Anforderungen realisiert werden können, ist die Priorisierung ein wichtiger Teil. Somit können die wichtigen Anforderungen zu erst umgesetzt werden, um am Ende ein sinnvolles und mit nützlichen Funktionen bestücktes Webfrontend vorweisen zu können.

Die Priorisierung wird folgender Maßen abgebildet:

- Prio 1: Anforderungen sind essentiell.
- Prio 2: Anforderungen sind wichtig.
- Prio 3: Anforderungen sind nützlich aber nicht zwingend erforderlich.

In der nachfolgenden Kategorisierung wird jede erarbeitete Anforderung der Anforderungsanalyse mit einer dieser Prioritäten versehen. Außerdem werden die höher priorisierten Anforderungen oben und die niedriger priorisierten Anforderungen unten in den folgenden Listen der Kategorien eingeordnet.

3.6.1 Anforderungen der Netzverantwortlichen

Für die Netzverantwortlichen sollen Scan-Berichte einsehbar sein, sie sollen ihre eigenen Scans konfigurieren und einstellen können, ihre E-Mail-Berichterstattung konfigurieren und sie sollen Personen ihres Netzbereichs das Nutzen von Dr. Portscan ermöglichen können. Daraus ergeben sich folgende drei Kategorien: Konfiguration, Berichterstattung und die Verwaltung des Netzbereichs. Bei der Konfiguration wird noch einmal in die Unterkategorien Scans und Berichterstattung unterteilt. Alle zuvor gefundenen Anforderungen lassen sich in diesen Kategorien unterbringen. Nachfolgend werden alle Anforderungen in diese drei Kategorien der Mandanten eingeteilt.

- Konfiguration
  - Scans
    - Netzverantwortliche sollen ihren Netzbereich einfach zum Scannen durch Dr. Portscan hinzufügen können. (Fall 1) (Prio 1)
    - Netzbereiche gezielt definieren und Ausnahmelisten erstellen können. (Fall 5) (Prio 1)
    - Die Möglichkeit sowohl von intern als auch von extern Portscans durchzuführen. (Fall 2) (Prio 1)
    - Netzverantwortliche sollen die Möglichkeit haben, sowohl TCP- als auch UDP-Ports zu scannen. Außerdem sollen die Portbereiche/-nummern individuell eingestellt werden können. (Fall 7) (Prio 1)
    - Vorgegebene Scan-Intervalle wie täglich, zwei mal wöchentlich oder wöchentlich. (Fall 10) (Prio 2)
    - Einstellmöglichkeit der Scan-Aggressivität durch den Netzverantwortlichen. (Fall 8) (Prio 2)
    - Möglichkeit zum Scannen einzelner IPv6-Endsysteme. (Fall 6) (Prio 3)
    - Ungefähres Festlegen der Zeitpunkte der eigenen Scanläufe. (Fall 9) (Prio 3)
  - Berichterstattung
    - Die E-Mail-Berichterstattung soll zu einer bestimmten Zeit versendet werden. Außerdem sollen bestimmte Empfänger für einzelne Netzbereiche festgelegt werden können. Bei dieser E-Mail sollen einige Möglichkeiten der Personalisierung bestehen, wie z. B. das Festlegen des Betreff-Felds. (Fall 3) (Prio 3)
    - Festlegen des Formats der einzelnen Scan-Ergebnisse der E- Mail-Berichterstattung, also die Reihenfolge der verfügbaren Attribute der Scan-Ergebnisse, wie IP-Adresse, Portnummer, Protokoll, offen/geschlossen und das Scan-Datum. (Fall 4) (Prio 3)
- Berichterstattung
  - Möglichkeit der Einsichtnahme aktueller Scan-Ergebnisse (Veränderungen gegenüber letztem Scan-Ergebnis und Gesamtübersicht) in einer strukturierten und eventuell filterbaren Tabelle, die über eine zentrale Anlaufstelle (Website) erreichbar ist. (Fall 11) (Prio 1)

  - Einsichtnahme der Scan-Ergebnisse für den betreuten Netzbereich und Berichterstattung per E-Mail. (Fall 12) (Prio 1)

- Verwaltung

  - Netzverantwortliche sollen eigenständig ihren Netzbereich verwalten können. Dazu gehört das Hinzufügen neuer Subnetzverantwortlicher, also solchen, die ein Subnetz des Netzverantwortlichen betreuen. Ändert sich beispielsweise eine betreuende Person für ein Subnetz, soll der Netzverantwortliche die nötigen Verwaltungsaufgaben selbstständig durchführen können. (Fall 13) (Prio 1)
  - Netzverantwortliche sollen die Möglichkeit haben Subnetzverantwortliche zu sperren oder zu löschen, damit deren Scans nicht mehr durchgeführt werden. (Fall 14) (Prio 2)

3.6.2 Anforderungen des Dr.-Portscan-Administrators

Für den Dr.-Portscan-Administrator ergeben sich nach Betrachtung der gefundenen Anforderungen zwei Kategorien. Zum Einen hat er die Aufgabe der Verwaltung von Mandanten und Scans, zum Anderen soll er Möglichkeiten für die Konfiguration von Dr. Portscan erhalten. Nachfolgend werden alle Anforderungen in diese zwei Kategorien des Dr.-Portscan-Administrators eingeteilt.

- Verwaltung

  - Automatischer Datenbank-Import bestehender Netzdoku-Datenbank. (Fall 15) (Prio 1)
  - Manuelles Hinzufügen von Netzverantwortlichen, die nicht in der Netzdoku-Datenbank (siehe Fall 14) registriert sind. (Fall 16) (Prio 1)
  - Verwalten der nötigen Daten, der manuell hinzugefügten Mandanten, wie z. B. E-Mail-Adresse und Name, oder auch das komplette Löschen der Mandanten. (Fall 19) (Prio 1)
  - Übersicht über alle eingestellten Scans mit deren ungefähr benötigten Scan-Dauer. (Fall 18) (Prio 2)
  - Verwaltungsmöglichkeit aller eingestellten Scans. Dazu gehört das Blocken von Scans die von Nutzern von Dr. Portscan eingestellt wurden. (Fall 17) (Prio 2)
  - Der DPA soll die Möglichkeit haben, eingestellte Scans der Netzverantwortlichen zu sperren, sodass diese nicht mehr ausgeführt werden. (Fall 20) (Prio 2)

- Konfiguration

  - Definieren von Ausnahmelisten durch den DPA. Diese IP-Bereiche werden dann von allen Portscannern ignoriert. (Fall 21) (Prio 3)
  - Sonstige Anforderungen zu Konfiguration von Dr. Portscan (siehe Kapitel 3.5). (Prio 3)

3.7 Zusammenfassung der Anforderungsanalyse

In Abbildung 3.1 werden alle Anforderungen nach Kategorien und Priorität übersichtlich zusammengefasst. Die erarbeiteten Anforderungen werden durch ihre jeweiligen Fallnummern gekennzeichnet.

Nutzer	Kategorie	Unterkategorie	Prio 1				Prio 2			Prio 3	
Netzverantwortliche	Konfiguration	Scans	1	2	5	7	10	8		6	9
		Berichterstattung								3	4
	Berichterstattung		11	12							
	Verwaltung		13				14				
DPA	Verwaltung		15	16	19		17	18	20		
	Konfiguration									21	sonstige

Abbildung 3.1: Kategorisierte Übersicht der Anforderungen.

4 Systementwurf

In diesem Kapitel wird der grundsätzliche Systementwurf dargestellt. Hierbei wird auf die Architektur und Struktur des Projekts eingegangen, welche den Grundstein für die spätere Implementierung bilden. Danach werden Überlegungen bezüglich des Scanner-Netzes und den in der Anforderungsanalyse erarbeiteten Output-Agenten angestellt. Zum Schluss wird die Datenhaltung erläutert, insbesondere die nötigen Änderungen die vorgenommen werden müssen, um die alte Dr.-Portscan-Datenbank für das neue Webfrontend tauglich zu machen.

4.1 Architektur

Die Architektur setzt auf das Model-View-Controller Architekturmuster, da es im Gegensatz zu einer einfachen Umsetzung mit unabhängigen Skripten für jeden einzelnen Teilbereich der Website, übersichtlicher und somit leichter erweiterbar ist. Des weiteren können Komponenten die häufig Verwendung finden untereinander geteilt werden, was das Projekt wiederum flexibler gegenüber Änderungen macht. Außerdem spricht die weite Verbreitung des Model-View-Controller Design-Pattern (MVC-Pattern) im Bereich der Webentwicklung für seine Vorteile. [Mic11, S. 49]

4.1.1 Model-View-Controller

Das MVC-Pattern ist ein Architekturmuster, welches häufig Gebrauch in der user-orientierten Softwareentwicklung findet, vgl. Kapitel 2.4. Gegenstand dieser Arbeit ist ein Webfrontend, welches über einen Browser durch den User genutzt werden kann. Über diese Website sollen User verschiedene Konfigurationen vornehmen und Daten einsehen können. Diese Daten müssen verarbeitet und aufbereitet werden, um dem User eine benutzerfreundliche Interaktion mit der Anwendung gewährleisten zu können. Diese Tatsachen rechtfertigen den Einsatz des MVC-Patterns für das zu implementierende Webfrontend.

Das Grundkonzept des MVC-Patterns bleibt stets bestehen, allerdings sind je nach Anwendungstyp einige Details unterschiedlich implementiert. Diese Details werden nun erläutert und in einer Grafik 4.1 anschaulich dargestellt.

Vorausgesetzt der User hat bereits eine Ansicht, also eine View, der Website vor sich, so ist der erste Schritt (1) die Interaktion mit der View. Es können Daten in die Form-Felder der Website eingegeben werden und per Submit-Button eine Aktion ausgeführt werden. Der Controller empfängt diese Aktion und leitet sie an den entsprechenden Bereich weiter, der für die Verarbeitung der Anfrage zuständig ist. Dort werden die Daten gegebenenfalls an das Model übergeben (2), welches die Daten einpflegt (3). Außerdem kann der Controller Daten anfordern, die das Model aus der Datenhaltung liefert (4). Diese Daten können vom Controller, falls nötig, noch aufbereitet und angereichert werden. Schlussendlich werden die Daten an die View weitergegeben (5), welche sich daraufhin aufbaut und für den User zur Darstellung bereitgestellt wird (6).

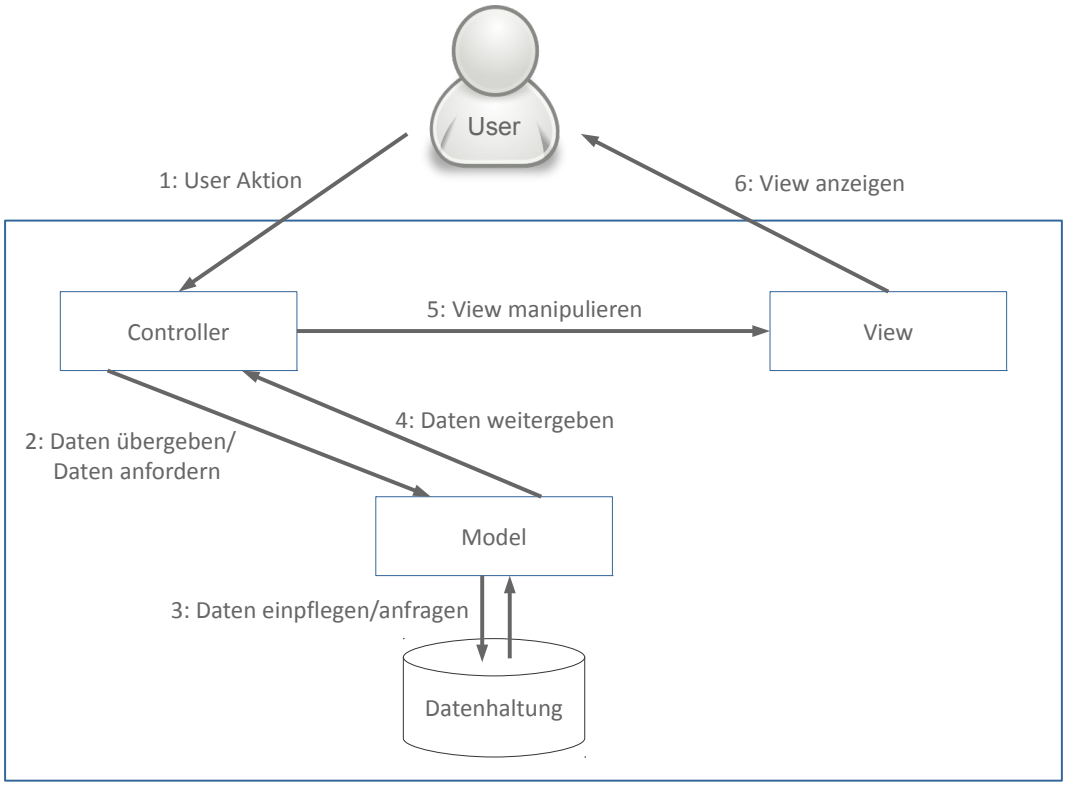

Abbildung 4.1: MVC-Pattern des Webfrontends mit Schrittabfolge.

4.1.2 Webkomponenten und weitere Bereiche

Das Grundkonzept des MVC-Patterns deckt nicht alle nötigen Bereiche ab, die für das Webfrontend nötig sind. Denkt man beispielsweise an den Datenbank-Import der Netzdoku-Datenbank oder die Authentifizierung der User, so können diese Bereiche zwar grundsätzlich dem Controller zugeteilt werden, allerdings lässt sich eine wesentlich übersichtlichere Struktur erreichen, wenn solche großen Aufgabenbereiche einen eigenen Bereich erhalten. Hierfür wird ein *Service-Bereich* eingeführt, welcher solche speziellen Dienste beherbergt.

Zusätzlich bringt ein Webfrontend stets weitere Inhalte mit sich, die ebenfalls nicht in die Struktur des MVC-Patterns passen. Dazu gehören Grafiken, Stylesheets und Javascript-Dateien, die für die Darstellung der Website von Nutzen sind. Grundsätzlich kann dieser Bereich der View zugeordnet werden, allerdings würde dies, wie auch zuvor beim Service-Bereich, die Struktur zu sehr verschachteln, was wiederum für Unübersichtlichkeit sorgen würde. Deshalb wird ein *Web-Bereich* eingeführt, welcher Grafiken, Stylesheets und Javascript enthält.

Wie sich diese zwei Bereiche in die zuvor benutzte Abbildung 4.1 des MVC-Patterns integrieren, zeigt Abbildung 4.2.

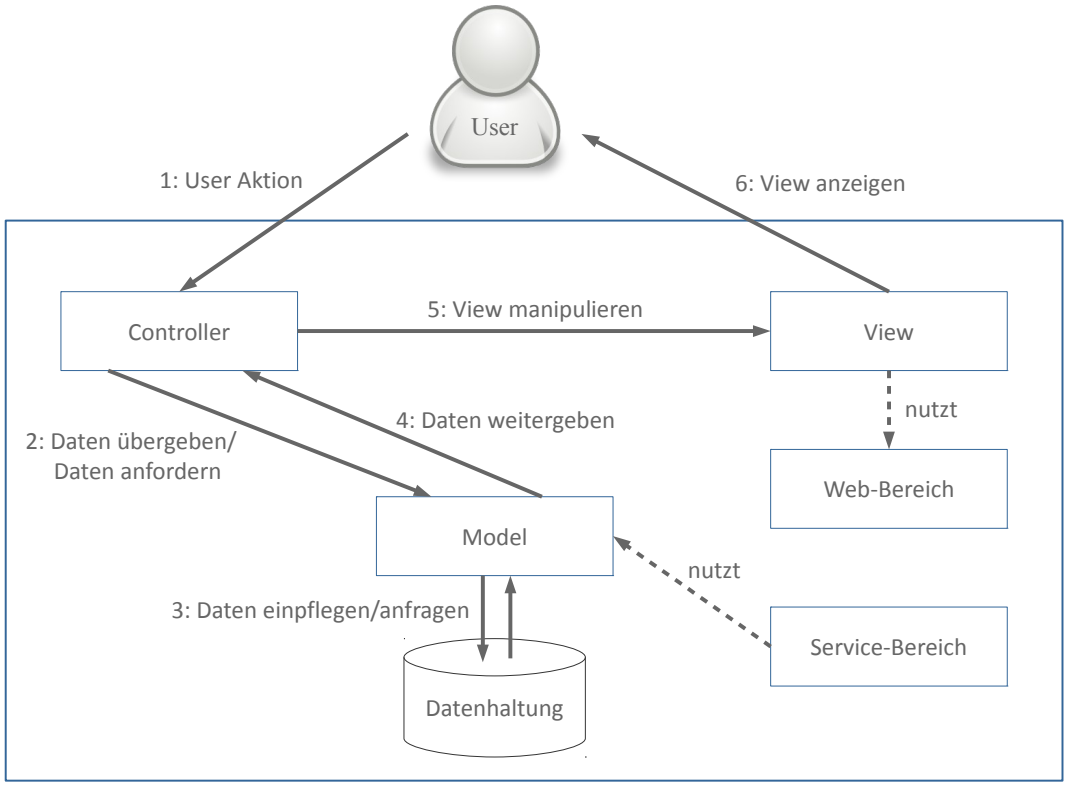

Abbildung 4.2: Eingliederung des Service- und Web-Bereichs in das MVC-Pattern.

4.1.3 Ordnerstruktur und Aktivitätsdiagramm

Nun steht das Grundkonzept des erweiterten MVC-Patterns. Die daraus abgeleitete Ordnerstruktur des Web-Projekts, siehe Abbildung 4.3, wird folgend erläutert.

- Anfangs nimmt stets der Front-Controller, repräsentiert durch die Datei *index.php*, die User-Anfragen entgegen. Diese werden daraufhin an den zuständigen Controller weitergeleitet. Die Controller-Struktur unterteilt sich in Controller für die Netzverantwortlichen, für den Dr.-Portscan-Administrator und gemeinsam genutzte Controller wie z. B. für den Login.
- Das Model wird von sogenannten Data-Access-Objects (DAO) vertreten. Jede Tabelle in der Datenbank wird durch ein DAO repräsentiert. Wird mit anderen Datentypen wie beispielsweise XML-Dateien gearbeitet, haben diese ebenfalls ein eigenes DAO.
- Im Service-Bereich befinden sich Dienste, welche nicht in die zuvor geschilderte Controller-Struktur passen, wie die Authentifizierung, den Datenbank-Import oder die Kommunikation mit anderen Servern.
- Alle möglichen Konstanten, wie beispielsweise die Zugriffsdaten für die Datenbank, befinden sich im Konfigurations-Bereich.
- Die View ist prinzipiell gleich aufgebaut wie die Controller-Struktur.

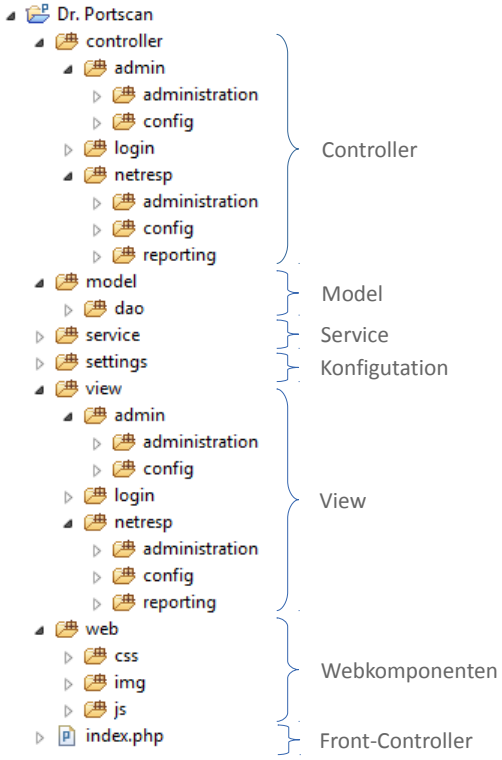

Abbildung 4.3: MVC-Verzeichnisstruktur des Webfrontends.

- Die Webkomponenten sind unterteilt in CSS-Stylesheets, Grafiken und Javascript.

Um die Ordnerstruktur zu verdeutlichen wird, auf Grundlage einer als Beispiel dienenden User-Interaktion und der zuvor gezeigten Schrittfolge des MVC-Patterns, siehe Abbildung 4.1, ein Aktivitätsdiagramm, siehe Abbildung 4.4, erstellt, in dem die einzelnen Schritte erläutert werden. Dabei wird aufgrund der Übersichtlichkeit auf den Web- und Service-Bereich verzichtet. Die als Beispiel dienende User-Interaktion soll das Hinzufügen eines neuen Mandanten durch den Dr.-Portscan-Administrator aufzeigen.

Zu Beginn befindet sich der Dr.-Portscan-Administrator auf der Website im Bereich Verwaltung der Mandanten und will dort einen neuen Mandanten hinzufügen. Dort gibt er die Daten des neu hinzuzufügenden Benutzers in die gegebene HTML-Form ein und übermittelt diese sodann per Klick auf den Submit-Button. Diese Daten werden zusätzlich noch mit Parametern versehen, die den Controllern die Navigation vorgeben. Der Front-Controller liest diese Parameter aus und bestimmt so, an welchen zuständigen Controller die Anfrage weitergeleitet werden soll. Dieser bestimmt daraufhin, ebenfalls auf Grundlage der Navigations-Parameter, die Funktion, welche ausgeführt werden soll, in unserem Fall `createUser()`. Diese Funktion überprüft die syntaktische Korrektheit der übermittelten Daten und reicht diese dann weiter an das Model, welches für die Datenbank-Tabelle „`user`“, in der die Benutzer des Webfrontends gespeichert sind, zuständig ist. Dort wird die Funktion `insert()` aufgerufen, welche die Daten letztendlich in die Tabelle schreibt und `true` bei Erfolg bzw. `false` bei Misserfolg an den Controller zurückgibt. Dieser Rückgabewert wird dann direkt vom Controller an die View weitergegeben. Die View gibt daraufhin den HTML-Code aus, der anhand der

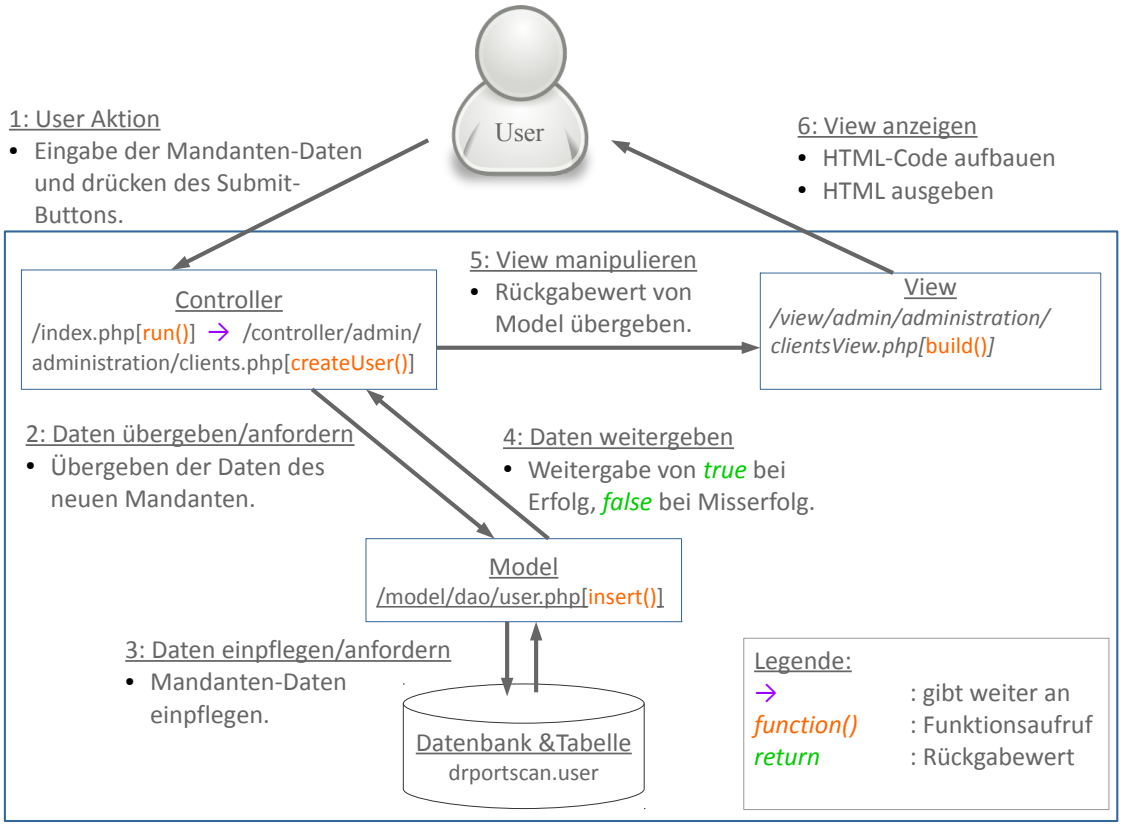

Abbildung 4.4: Aktivitätsdiagramm für die User-Interaktion „*Neuen Mandanten hinzufügen*“.

übergebenen Daten des Controllers individuell aufgebaut wird.

4.2 Scanner-Netz

Das Scanner-Netz stellt die Kommunikation zwischen Dr.-Portscan-Server und den Datenquellen, also den eigentlichen Scannern, dar. Dieses ist bei der aktuellen Dr.-Portscan-Version ohne Webfrontend-Anbindung relativ statisch, d. h. wenn Änderungen an Scans vorgenommen werden, müssen diese fest im Code auf den Datenquellen angepasst werden. Durch das Webfrontend und die dadurch gewünschte Zunahme an Netzverantwortlichen ergibt sich ein neues Szenario, welches eventuell durch ein neues, anpassungsfähigeres Konzept ersetzt werden kann.

4.2.1 Ausgangssituation

Die Scans von Dr. Portscan werden bisher stets vom Dr.-Portscan-Administrator verwaltet. Wurde ein neuer Netzbereich eines Netzverantwortlichen zum Scannen hinzugefügt, wurde dieser manuell in ein bash-Skript auf der Datenquelle eingepflegt. Dieser Netzbereich wurde dann beim nächsten Aufruf des Scripts, durch den zugehörigen Cronjob, beachtet. Da es in

der Regel mehrere Datenquellen (zumindest jeweils eine für interne und externe Scans) für die Ausführung der Scans gibt, mussten immer auch mehrere Skripte angepasst werden.

Solch ein Skript war bisher folgendermaßen aufgebaut:

- Array der zu scannenden Netze in CIDR-Notation.
- Definition der verschiedenen Verzeichnisse für Zwischenspeicherung der Ergebnisse, Logdateien und Zielverzeichnis der endgültigen Ergebnisse.
- Überprüfung ob ein sogenanntes *Lock-File*, zur Blockierung der Ausführung des Skripts existiert, ansonsten Lock-File setzen.
- Bestimmung der Art des Scans aufgrund der Uhrzeit.
- Ausführen des Scans.

Das Skript führt in einer Schleife alle Nmap-Aufrufe nacheinander aus. Diese sequenzielle Ausführung wurde eingeführt, um zu hohen Netzwerk-Verkehr, welcher wiederum zu fehlerhaften Scan-Ergebnissen führen könnte, zu verhindern. Das Lock-File wird verwendet, um das parallele Ausführen des Skripts zur selben Zeit zu unterbinden, welches wiederum zum möglichen Problem des hohen Netzwerk-Verkehrs führen könnte. Wird das Skript nämlich alle zwei Stunden per Cronjob ausgeführt und dauern die Scans der vorigen Ausführung länger als zwei Stunden, so würden wiederum parallele Nmap-Aufrufe auftreten, obwohl das Skript alleine auf sequenzielle Ausführung ausgelegt ist.

Diese Ergebnis-Dateien der Scans werden nach Scan-Ende in das Zielverzeichnis verschoben. Von dort werden sie per rsync[1] von der Datenquelle zum eigentlichen Dr.-Portscan-Server transferiert und dort vom Input-Watcher zur Weiterverarbeitung in Empfang genommen.

4.2.2 Erweitertes Szenario und entstehende Probleme

Das Webfrontend für Dr. Portscan soll dazu dienen, den Netzverantwortlichen einen einfachen Zugang zu Dr. Portscan zu ermöglichen. Ist ein Netzverantwortlicher für den Dienst registriert, kann er dort seinen eigenen Netzbereich scannen. Zusätzlich kann er Subnetzverantwortliche oder sonstige Personen hinzufügen, die innerhalb seines Netzbereichs eigene Scans durchführen können. Die ca. 900 Netzverantwortlichen im MWN und die zusätzlich möglichen Personen die Dr. Portscan über das Webfrontend nutzen können, ergeben eine große Anzahl an Nutzern, mit denen eine ebenso große Anzahl an Scans einhergeht. Diese Scans können jeder Zeit angepasst oder geändert werden. Der Dr.-Portscan-Admin kann Scans sperren oder Nutzer blockieren, was wiederum die Scans des Nutzers sperren würde. Es herrscht also eine hohe Dynamik innerhalb der Scans.

Würden diese Scans manuell, so wie bisher bei Dr. Portscan, angepasst werden, wäre das ein extrem hoher Aufwand. Das Webfrontend soll diese Anpassungen jedoch automatisch vornehmen. Bleibt man bei der bisherigen Methode würde eine Anpassung oder Sperrung eines Scans folgendes mit sich bringen:

- Herausfinden der Datenquelle, auf der die Änderungen vorgenommen werden.

[1] rsync ist nicht nur ein Protokoll sondern auch eine Software für unix-/linuxbasierte Systeme, welche Dateien von einem Server/Quellverzeichnis zu einem anderen Server/Zielverzeichnis überträgt.

- Verbindung z. B. per SSH zur Datenquelle herstellen.
- Vorhandenes Skript auf der Datenquelle auslesen und anpassen oder...
- neues Skript generieren und z. B. per SCP auf die Datenquelle übertragen.

Nun stellt sich die Frage, wie das Skript anzupassen wäre. Ein Lösungsansatz wären sogenannte *Templates*, also Vorlagen für das Skript. Diese könnten bei auftretenden Änderungen verwendet werden, um ein neues, den Änderungen entsprechendes Skript, auf dem Dr.-Portscan-Server zu generieren. Das so erzeugte Skript könnte dann per SCP auf die Datenquelle übertragen werden.

Beim Generieren des Skripts müssten stets alle Scans, die auf der entsprechenden Datenquelle ausgeführt werden sollen, betrachtet werden. D. h. eine Änderung eines Scans bzw. eine Sperrung würde eine Betrachtung aller, für die jeweilige Datenquelle relevanter Scans mit sich bringen, um das neue Skript zu erstellen. Dies ist ohne Probleme möglich, jedoch kein sehr eleganter Lösungsansatz. Es stellt sich also die Frage nach einer dynamischen Lösung, die jeden Scan einzeln betrachtet und behandelt.

Ein weiterer zu betrachtender Punkt ist die eigentliche Ausführung der Nmap-Aufrufe innerhalb des Skripts. Diese werden bisher sequentiell ausgeführt. Die Scans werden also für die einzelnen Netzbereiche nacheinander ausgeführt. Eine Festlegung eines ungefähren Scan-Zeitpunktes, wie in der Anforderungsanalyse gefordert, wäre nicht realisierbar. Außerdem könnten in einem Szenario mit mehr als 900 Netzverantwortlichen und Scans, die gewünschten Scan-Intervalle möglicherweise nicht eingehalten werden. Ist ein Scan zum täglichen Scannen eingetragen, das Skript benötigt zur Ausführung aufgrund der hohen Anzahl an Scans allerdings zwei Tage, so wird das vorgegebene Intervall nicht eingehalten. Dies ist jedoch mehr eine Frage der Aufgabenverteilung der Scans an die Datenquellen als des Aufbaus des Skripts.

4.2.3 Neuer Lösungsansatz

Um die gezielte Betrachtung einzelner Scans und die Beachtung der Scan-Zeitpunkte der Ausführung zu beachten, wurde folgender, neue Lösungsansatz erarbeitet. Dieser Lösungsansatz wird in Abbildung 4.5 dargestellt.

Es handelt sich um eine zentrale Verteilung der Scans vom Dr.-Portscan-Server. Jeder nicht gesperrte Scan wird von einem regelmäßig ausgeführten Cronjob betrachtet. Der betrachtete Scan wird dann ausgeführt, wenn das Scan-Intervall im Vergleich zum letzten Scan-Lauf überschritten ist und die aktuelle Zeit in etwa der gewünschten Scan-Zeit entspricht. Dann wird eine passende, also interne oder externe, Datenquelle ausgesucht und eine SSH-Verbindung aufgebaut. Über diese SSH-Verbindung wird dann der gewünschte Nmap-Aufruf, mit all seinen gewünschten Optionen, ausgeführt.

Ein weiterer Cronjob auf dem Dr.-Portscan-Server sorgt dann dafür, dass alle fertigen Scan-Ergebnisse auf der Datenquelle per SCP in das Verzeichnis für neue Scans auf dem Dr.-Portscan-Server übertragen werden. Die fertigen Scans werden mithilfe eines Skripts auf der Datenquelle identifiziert.

Werden mehrere Scans zur gleichen Zeit, auf der gleichen Datenquelle ausgeführt, sollen diese parallel ablaufen. Ein Test auf einem Server des LRZ hat gezeigt, dass die parallele Ausführung mehrerer Nmap-Aufrufe möglich ist und nicht zu unvorhersehbaren Fehlern auf-

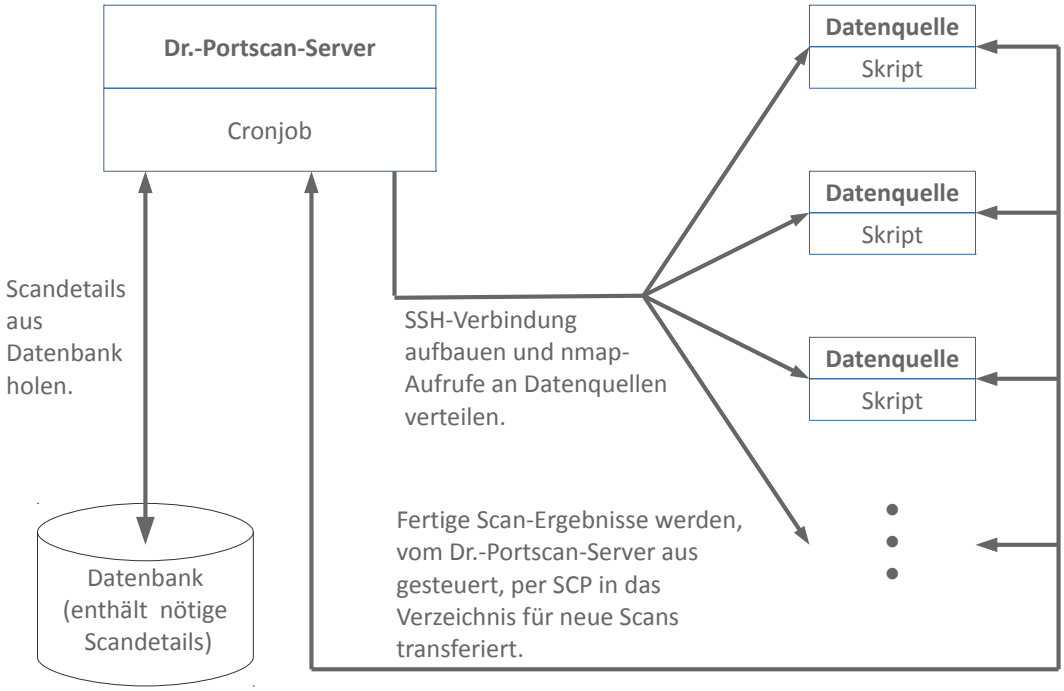

Abbildung 4.5: Ausführung der Scans per SSH.

grund des hohen Netzwerk-Verkehrs führt. Dazu wurde ein Skript erstellt, welches diesen Sachverhalt testet.

Das Skript ist folgendermaßen aufgebaut: Der Code innerhalb einer Schleife wird 200 mal durchlaufen. Dieser Code wählt zufällig einen von vier vorhandenen, unterschiedlichen Nmap-Aufrufen aus. Dieser Nmap-Aufruf wird dann immer für sechs feste IPv4-Adressen, die vom LRZ zum scannen zur Verfügung gestellt wurden, im Hintergrund ausgeführt. D. h. 200 Nmap-Aufrufe werden parallel, jeweils für sechs Netzwerkadressen auf einem Server ausgeführt. Folgende Nmap-Aufrufe wurden verwendet:

```
nmap -pT:1-128 -Pn -oX ...

nmap -pU:53,U:110,T:1-1024 -Pn -oX ...

nmap -pT:1-5000 -vv -Pn -oX ...

nmap -pU:1-1024,T:1-2048 -vv -Pn -oX ...
```

Der Nmap-Parameter `-Pn` wurde hinzugefügt, um die Host-Erkennung abzuschalten. Ohne diesen Parameter kam es zu Problemen, da die gescannten Hosts aufgrund der hohen Scan-Belastung abgeblockt haben. Sie waren also für den Server, von dem die Scans ausgeführt wurden, nicht mehr erkennbar, weshalb die Nmap-Aufrufe zu keinen Ergebnissen führten.

Die Ergebnisse wurden durch den Parameter `-oX` in eine, für den Input-Watcher lesbare, XML-Datei geschrieben.

Das Ergebnis: Nach knapp 1,5 Stunden waren alle Scans beendet. Die 200 XML-Dateien die Nmap erstellt hat wurden so benannt, dass sie mithilfe von Dr. Portscan ausgewertet werden konnten. Dazu wurde der Input-Watcher bemüht. Er hat alle Ergebnisse vom Nmap-Input-Agenten in das richtige Format bringen lassen und daraufhin mithilfe des Delta-Reporters die Änderungen in die Datenbank eingetragen. Die Ausprägung der Tabelle „`portResults`“ (einige Attribute ausgeblendet) nach dem Delta-Report sieht wie folgt aus:

portId	hostId	port	protocol	age	changeType
1	1	22	tcp	-200	0
2	1	111	tcp	-200	0
3	3	22	tcp	-200	0
4	3	80	tcp	-200	0
5	3	111	tcp	-200	0
6	4	22	tcp	-200	0
7	4	80	tcp	-200	0
8	4	111	tcp	-200	0
9	4	443	tcp	-155	0
10	6	80	tcp	-200	0
11	6	443	tcp	-155	0
12	8	22	tcp	-200	0
13	8	111	tcp	-200	0
14	9	135	tcp	-155	0
15	9	445	tcp	-155	0
16	6	2222	tcp	-56	0
17	6	3389	tcp	-56	0
18	9	3389	tcp	-56	0

Für die Auswertung der Ergebnisse ist speziell das Attribut *`age`* interessant. Dieses Attribut gibt nämlich an, seit wie vielen Scans der Port offen bzw. geschlossen ist. Ein negativer Wert von -200 sagt z. B. aus, dass der Port seit 200 Scans geöffnet ist. Betrachtet man die Ergebnisse fällt auf, dass nicht alle Ports den selben Wert von -200 haben. Diese Unterschiede resultieren aus den verschiedenen Nmap-Aufrufen, die zufällig ausgewählt wurden. So werden TCP-Ports größer als 2048 nur von `nmap -pT:1-5000 -vv -Pn` gescannt. Alle Ports, deren Portnummern größer als 2048 sind, wurden in den Ergebnissen 56 mal hintereinander als offen erkannt. Daraus lässt sich schließen, dass diese spezielle Nmap-Aufruf genau 56 mal ausgeführt wurde. Da alle von diesem Scan erfassten Ports 56 mal als offen erkannt wurden, kann mit sehr großer Wahrscheinlichkeit gesagt werden, dass bei jedem Scan die gleichen Ergebnisse erzielt wurden. Das selbe gilt für die Ports mit den *`age`*-Ausprägungen -155 und -200. Wären Abweichungen dieser Werte aufgetreten, würde dies auf Fehler durch zu hohe Netzauslastung beim Scannen von Nmap hinweisen. Da diese nicht aufgetreten sind, stellt die parallele Ausführung von Nmap-Aufrufen offensichtlich kein Problem dar.

4.2.4 Gegenüberstellung der Möglichkeiten und Auswahl

Für das Scanner-Netz wurde nun einmal eine mögliche Anpassung der bisher von Dr. Portscan verwendeten Variante aufgezeigt und einmal ein gänzlich neuer Lösungsansatz präsentiert. Diese zwei Möglichkeiten werden nun mit ihren Vor- und Nachteilen gegenübergestellt:

Möglichkeit	**Pro**	**Contra**
Bisheriges Verfahren mit Erweiterung	• Netzwerkauslastung geringer da sequentielle Ausführung. • Übersichtlicher für kleine Netzbereiche.	• Bei Änderungen an nur einem Scan müssten für die Erstellung per Template, alle Scans der entsprechenden Datenquellen erneut berücksichtigt werden.
Neuer Lösungsansatz	• Möglichkeit die ungefähre Scanzeit einzustellen. • Zentralere Verwaltung, da z. B. keine Cronjobs auf der Datenquelle eingerichtet werden müssen.	• Zu komplex für geringe Anzahl an zu scannenden Netzen.

Damit die Anforderung der Scanzeit-Einstellung erfüllt werden kann und da eine zentrale Verwaltung bei komplexen Systemen stets von Vorteil ist, wird bei der Implementierung der neue Lösungsansatz verfolgt.

4.3 Output-Agent

Die Output-Agenten und insbesondere der E-Mail-Output-Agent werden bisher mittels Konfigurationsdatei, konfiguriert. Diese Methode bedarf ebenfalls einer genauen Überlegung, ob dies weiterhin für das geplante Webfrontend sinnvoll ist oder ob eine andere Methode besser geeignet ist.

4.3.1 Ausgangssituation

Nachdem der Delta-Reporter ausgeführt wurde, wird bisher stets ein Skript namens *autorun-output-agents.list* ausgeführt. Innerhalb dieses Skripts, werden bestimmte Output-Agenten, wie z. B. der E-Mail-Output-Agent, für bestimmte Netzverantwortliche ausgeführt. Es handelt sich um eine Liste von einzelnen Befehlen wie beispielsweise `./output-agents/email-report.pl client1`. Dieser Befehl würde den E-Mail-Output-Agent für den Netzverantwortlichen mit der Kennung „client1“ ausführen.

Der E-Mail-Output-Agent führt daraufhin alle Schritte aus um für diesen bestimmten Netzverantwortlichen einen E-Mail-Bericht zu generieren und zu verschicken. Dazu gehört das Auslesen der Datenbank-Tabelle *clients*, siehe Kapitel 2.3.3, um die Informationen über den betroffenen Netzbereich zu bekommen und daraufhin das Auslesen der Datenbank-Tabellen *portResults* und *hostResults*, um die Scan-Ergebnisse für die Hosts des Netz-

bereichs zu erhalten. Mit Hilfe der Konfigurationsdatei *output-agents.conf* werden dann die Informationen für den Netzverantwortlichen bestimmt, der Bericht generiert und verschickt.

Die Konfigurationsdatei enthält Informationen wie die E-Mail-Adresse des Netzverantwortlichen, die Formatierung der Ergebnisse, den Betreff der E-Mail und den E-Mail-Text. Diese Informationen werden wie in Quellcode 4.1 dargestellt.

```
1  [xml2plaintext]
2  #Legt das Format für Zeitangaben fest.
3  timeFormat = "%R %d.%m.%y"
4
5  #Legt das Format der einzelnen Ergebnis-Zeilen fest.
6  formatStr = "'%time', '%host', '%ip', '%os', '%port/%protocol', '%service', '%
       changeType', '%scanner'"
7
8  [email-report]
9  #E-Mail-Adresse des Absenders.
10 sender = "DrPortScan@localhost"
11
12 #Betreff der E-Mail.
13 subject = "Dr. Portscan - Report"
14
15 #Text des E-Mail.
16 bodytext = "Hallo, dies ist eine von Dr.PortScan automatisch erstelle E-Mail ..."
17
18 [email-report:client1]
19 #Empfänger, mehrere durch ',' trennen.
20 recipients = "root@localhost"
21
22 #Zeitplan für das Versenden der E-Mail-Berichte in cronjob-ähnlichem Format.
23 schedule = "18 * * *"
```

Quellcode 4.1: Beispiel einer Konfigurationsdatei.

Die Konfigurationen für die verschiedenen Output-Agenten werden jeweils durch eckige Klammern abgegrenzt. `[xml2plaintext]` (Zeile 1) gibt an, dass die folgenden Einstellungen für den XMLToPlaintext-Output-Agent vorgesehen sind. Erst durch eine weitere Abgrenzung, wie `[email-report]` (Zeile 8), wird ein neuer Output-Agent konfiguriert. Für den E-Mail-Output-Agent gibt es eine Besonderheit. Für die einzelnen E-Mail-Empfänger können unterschiedliche Parameter definiert werden. Dazu wird eine neue Abgrenzung definiert, wie z. B. `[email-report:client1]`. Alle folgenden Einstellungen sind dann nur für diesen speziellen Empfänger vorgesehen.

Beim XMLToPlaintext-Output-Agent, werden in der Beispieldatei die Formate für Zeitangaben und für das Ergebnis der einzelnen Ports vorgegeben. Das Zeitformat wird hier durch Parameter angegeben die stets mit einem %-Zeichen beginnen. Der Wert „`%R %d.%m.%y`“ (Zeile 3) führt beispielsweise zum dem Datum „19:12 04.02.14“. Nach der gleichen Logik werden die einzelnen Ergebnis-Zeilen der Ports definiert (Zeile 6). Danach folgen die Einstellungen für den E-Mail-Output-Agenten. Anfangs wird der Betreff und der einleitende Text des E-Mail-Berichts definiert. Danach folgen empfänger-spezifische Angaben. Einmal die E-Mail-Adressen der Empfänger und einmal der Zeitpunkt des Versendens des Berichts in cronjob-ähnlichem Format. Cronjob-ähnlich deswegen, weil die Angabe der Minuten fehlt. Der erste Platzhalter ist gleich für die Definition der Stunde zuständig, vgl. 2.9.

4.3.2 Erweitertes Szenario und entstehende Probleme

Das Szenario ändert sich durch das Webfrontend für Dr. Portscan dahingehend, dass eine größere Anzahl an Nutzern mit verschiedensten Konfigurationen für die E-Mail-Berichterstattung

beachtet werden müssen. Das würde die Konfigurationsdatei unübersichtlich machen. Weiterhin sollen die Nutzer die Möglichkeit haben, die Konfiguration ihrer E-Mail-Berichterstattung jederzeit über das Webfrontend anzupassen. Dazu müssten die getätigten Einstellungen per Skript in die Konfigurationsdatei eingepflegt werden.

Um die Konfigurationsdatei für bestimmte E-Mail-Empfänger anzupassen, könnte folgendermaßen vorgegangen werden:

- Per sogenannter *Regular Expression*[2] den entsprechenden Teil des Dokuments finden.
- Wenn kein Eintrag vorhanden ist, neue E-Mail-Empfänger mit entsprechenden Variablen ans Ende des Dokuments anfügen, ansonsten...
- den gefundenen Teil der Regular Expression nach der gesuchten Variable (also den Wert in Anführungszeichen in Quellcode 4.1) durch den neuen Wert ersetzen.

4.3.3 Neuer Lösungsansatz

Dr. Portscan nutzt für die Speicherung der Scan-Ergebnisse, Scanner und E-Mail-Empfänger bereits eine Datenbank. Alle nötigen Konfigurationen für spezielle E-Mail-Reports können ebenfalls in dieser Datenbank gespeichert werden. Der neue Output-Agent müsste dazu nur die Daten für die jeweiligen Empfänger aus der Datenbank holen und kann dann speziell konfigurierte E-Mail-Berichte generieren und verschicken.

Beim Ändern der Konfiguration kann dadurch leicht sichergestellt werden, ob die Änderungen erfolgreich übernommen wurden. Ein weiterer Vorteil der Speicherung in der Datenbank ist das Vermeiden von Inkonsistenzen. Die alte Variante mit der Konfigurationsdatei hat zusätzlich eine Datenbank-Tabelle „`clients`“, in der die Singnatur des Empfängers gespeichert ist. Diese Signatur wird wiederum verwendet um die Empfänger in der Konfigurationsdatei zu identifizieren. Sowohl die Datenbank-Tabelle als auch die Konfigurationsdatei werden gesondert behandelt. Es kann also durchaus vorkommen, dass ein Empfänger in der Datenbank vorhanden ist aber in der Konfigurationsdatei fehlt.

Außerdem könnte die selbe Signatur in der Konfigurationsdatei mehrmals vorkommen, was nicht zu Fehlern führt, allerdings unschön ist und unnötig Speicherplatz benötigt. In einer Datenbank kann dies gänzlich durch eine geeignete Schlüsseldefinition ausgeschlossen werden.

4.3.4 Gegenüberstellung der Möglichkeiten und Auswahl

Für den Output-Agenten wurde nun einmal eine mögliche Anpassung der bisher von Dr. Portscan verwendeten Variante aufgezeigt und einmal ein gänzlich neuer Lösungsansatz präsentiert. Diese zwei Möglichkeiten werden nun mit ihren Vor- und Nachteilen gegenübergestellt:

[2] Mit Regular Expressions können bestimmte Textabschnitte gefunden werden und folgend auch ersetzt werden.

Möglichkeit	Pro	Contra
Bisheriges Verfahren mit Erweiterung	• Gut geeignet für schnelle Implementierung und wenige zu scannende Netze.	• Fehleranfälligkeit der Anpassung per Regular Expression. • Bei einer hohen Anzahl an E-Mail-Empfängern entsteht eine sehr große und somit unübersichtliche Konfigurationsdatei.
Neuer Lösungsansatz	• Vermeiden von Inkonsistenzen, da alle Informationen in einer Datenbank gespeichert werden.	• -

Die Variante mit der Konfigurationsdatei zusätzlich zur Datenbank-Tabelle für die E-Mail-Empfänger, war als schnelle Lösung für eine kleine Menge an Empfängern gedacht. Da das Webfrontend eine Vielzahl neuer Netzverantwortlicher zum Überwachen ihrer Netze bewegen soll und somit auch eine Vielzahl an E-Mail-Empfängern erwartet wird, ist die Umstellung auf eine sowieso bereits vorhandene Datenbank die logische Konsequenz. Bei der Implementierung wird deshalb der neue Lösungsansatz verfolgt.

4.4 Datenbankstruktur

Um die vorhandene Datenbank von Dr. Portscan für ein mandantenfähiges Webfrontend anzupassen, müssen einige Änderungen vorgenommen werden. Damit sich die Benutzer, also der Dr.-Portscan-Administrator und die Netzverantwortlichen, authentifizieren können, wird eine Tabelle mit Zugangsdaten und weiteren Nutzerdetails benötigt. Um die neue Idee der Scan-Ausführung umzusetzen wird ebenfalls eine Tabelle benötigt, in der die Scan-Optionen abgespeichert werden. Diese Scan-Einträge müssen mit den Nutzern verknüpft werden, um die Zuordnung der Scans zu gewährleisten. Im vorigen Kapitel 4.3 wurde beschlossen, die Details für die E-Mail-Berichterstattung ebenfalls in der Datenbank zu speichern. Dafür wird ebenfalls eine neue Tabelle angelegt. Schlussendlich ist die Anforderung für Ausnahmelisten zu beachten, welche ebenfalls eine neue Tabelle erfordert. Somit sind alle Informationen in einer Datenbank festgehalten und es müssen keine weiteren Dateien gepflegt werden.

4.4.1 Anpassung bestehender Tabellen

Die bisherige Datenbankstruktur wird bis auf eine Ausnahme beibehalten. Nur die Tabelle „`clients`“ wird ersetzt. Sie wird lediglich für die Output-Agenten benötigt. Diese werden jedoch durch eine neue Herangehensweise implementiert, siehe Kapitel 4.3. Die neue Tabelle „`recipients`“, welche die E-Mail-Empfänger samt Berichterstattungs-Optionen enthält, wird die Tabelle „`clients`“ ersetzen. Näheres dazu im folgenden Abschnitt.

4.4.2 Neue Tabellen

In den folgenden Tabellenkonzepten werden Schlüsselattribute stets unterstrichen dargestellt. Attribute die eine Auswahl an bestimmten Werten vorgeben, werden im Format „(wert1|wert2|...)“ angegeben.

Die erste neue Tabelle wird für die Nutzer, also die Netzverantwortlichen und den Dr.-Portscan-Administrator, benötigt. Darin werden Informationen zum Login, Nutzer-Typ, Authentifizierungs-Typ, Netzbereich und in der Hierarchie höher angesiedelten Nutzern gespeichert. Das Tabellenkonzept für die Tabelle „`user`“ sieht folgendermaßen aus:

Attribut	**Erklärung**
userId	Eindeutige Nutzer-ID
email	E-Mail-Adresse des Nutzers
hash	Der Hashwert[3](sha256) des Nutzer-Passworts
admin	0 wenn Nutzer ein Netzverantwortlicher ist, 1 wenn Nutzer ein Dr.-Portscan-Administrator ist
cidrRange	Kommaseparierte Liste der Netzbereiche in CIDR-Notation für die der Nutzer verantwortlich ist
rangeName	Name des Netzbereichs des Nutzers
blocked	1 wenn Nutzer geblockt ist und nicht mehr auf das Webfrontend zugreifen kann, 0 sonst
extAuth	1 wenn Nutzer extern über das LRZ-SIM authentifiziert wird, 0 sonst
parentIds	Kommaseparierte Liste der userIds, die in der Hierarchie über dem Nutzer stehen
lastAction	Timestamp des letzten Logins des Nutzers

Als nächstes wird die Tabelle „`scans`“ betrachtet, welche für die Speicherung der Scans samt Scan-Optionen zuständig ist:

[3]Eine Hash-Funktion bildet von einem String beliebiger Länge auf einen String fester Länge (dem Hashwert) ab. Von dem Hashwert kann nicht eindeutig auf den String variabler Länge zurückgeschlossen werden.

Attribut	Erklärung
scanId	Eindeutige Scan-ID
userId	Nutzer-ID des Nutzers, dem dieser Scan zugeordnet werden kann
active	1 wenn Scan aktiv ist, 0 wenn er pausiert (vom Nutzer bestimmt)
netRange	Kommaseparierte Liste der Netzbereiche in CIDR-Notation, welche dieser Scan scannen soll
tcpRange	Kommaseparierte Liste der TCP-Ports, die gescannt werden sollen. Es können auch Portbereiche angegeben werden (z. B. 1-512)
udpRange	Kommaseparierte Liste der UDP-Ports, die gescannt werden sollen. Es können auch Portbereiche angegeben werden (z. B. 1-512)
placement	Ort der Datenquelle, von der aus gescannt werden soll (intern|extern)
aggressiveness	Aggressivität des Scans (aggressive|normal|slow)
interval	Zeitintervall entweder täglich, zwei mal wöchentlich oder wöchentlich (daily|2timesweekly|weekly)
time	Zeitpunkt zu dem der Scan ausgeführt werden soll im 24-Stunden-Format
lastScantime	Timestamp der letzten Ausführung des Scans
blocked	1 wenn Scan blockiert ist, 0 sonst (vom Dr.-Portscan-Administrator oder höherem Netzverantwortlichen bestimmt)

Es folgt das Konzept der Tabelle „`recipients`“, welche Informationen über die E-Mail-Empfänger der Berichterstattung und weiteren Details wie E-Mail-Format und Versand-Zeitpunkt speichert:

Attribut	Erklärung
recipientId	Eindeutige ID des E-Mail-Berichts
userId	Nutzer-ID des Nutzers, dem dieser E-Mail-Bericht zugeordnet werden kann
email	E-Mail-Adresse des Empfänger dieses E-Mail-Berichts
cidrRange	Kommaseparierte Liste der Netzbereiche in CIDR-Notation, für die dieser Bericht erstellt wird
days	Kommaseparierte Liste der Tage (Mon|Tue|Wed|Thu|Fri|Sat|Sun), an denen ein Bericht versendet werden soll
time	Zeitpunkt zu dem der Bericht versendet werden soll im 24-Stunden-Format
subject	Text des Betreff-Felds des E-Mail-Berichts
text	Text der eigentlichen E-Mail des Berichts
format	Format der Port-Ergebnisse des Berichts. Zur Verfügung stehen (%time|%host|%ip|%os|%port|%protocol|%service|%changeType|%scanner) innerhalb eines beliebigen Textes.

Zuletzt noch die Tabelle „`rangeExceptions`“ für die ausgeschlossenen Netzbereiche, also solche, die nie gescannt werden sollen:

Attribut	Erklärung
userId	Eindeutige Nutzer-ID des Erstellers der Ausnahmeliste
exceptions	Kommaseparierte Liste der ausgeschlossenen Netzbereiche in CIDR-Notation

5 Implementierung

Das Kapitel Implementierung beschreibt interessante und wichtige Komponenten des Projekts. Dazu gehören unter anderem die Umsetzung des MVC-Patterns, der Authentifizierung und des Scanner-Netzes. Ebenfalls wird auf einige Sicherheitskomponenten eingegangen. Schlussendlich folgt eine Gegenüberstellung der Implementierung mit der Anforderungsanalyse.

Für die Implementierung des Webfrontends wird die Skriptsprache PHP (Hypertext Preprocessor) in der Version *5.4.4-14+deb7u8* verwendet. Die Wahl hätte, unter Betrachtung der Funktionalität, ebenso gut auf andere Skriptsprachen wie Perl oder Python fallen können. Die Entscheidung für PHP wurde durch folgende Punkte gestützt:

- PHP wird mit einem großen Funktionsumfang, insbesondere auch für die Webentwicklung, ausgeliefert. Bei Perl oder Python müssten einige Module, wie beispielsweise für sogenannte *Sessions* oder die MySQL-Anbindung, zusätzlich manuell installiert werden.
- PHP ist auf einem Standard-LRZ-Webserver vorhanden.
- Weitverbreitet im Bereich der Webentwicklung.
- Persönliche Präferenz.

Die SQLite-Datenbank wird durch eine MySQL-Datenbank ersetzt. Dies bringt folgende Vorteile gegenüber dir SQLite-Variante:

- Bequemere Möglichkeit der entfernten Administration über phpMyAdmin[1].
- MySQL ermöglicht eine detailliertere Definition des Datenbankschemas.
- MySQL ermöglicht gleichzeitigen Zugriff mehrerer Benutzer auf Datenbanken.

5.1 Ausgangspunkt und Vorbereitung

Um den implementierten Code zu testen und ein lauffähiges System aufzustellen wurden vom LRZ zwei Server bereitgestellt, auf denen mit root-Zugriff gearbeitet werden konnte. Diese Server liefen unter Debian in der Version *3.2.54-2*. Einer der Server wurde als Dr.-Portscan-Server genutzt, der andere als Datenquelle, die interne Scans durchführt. Auf die Server wurde entfernt per SSH mit den Anwendungen PuTTY[2] und WinSCP[3] zugegriffen.

Auf der Datenquelle wird lediglich Nmap ausgeführt und ein Shell-Skript, das bei der Übertragung der Ergebnisse hilft, wie später zu sehen. Beide Funktionen sind durch die

[1] phpMyAdmin ist ein Tool zur Administration von MySQL-Datenbanken, welches per HTTP über einen Browser genutzt werden kann.

[2] `http://www.chiark.greenend.org.uk/~sgtatham/putty/`

[3] `http://winscp.net/`

Standardkonfiguration des Servers vorhanden und es müssen keine weiteren Vorkehrungen getroffen werden.

Um den Dr.-Portscan-Server vorzubereiten, müssen einige Schritte vorgenommen werden. Dr. Portscan selbst ist in Perl, welches bereits auf dem Server vorhanden war, implementiert und benötigt einige Module, die per `apt-get install`[4] installiert werden können. Dr. Portscan wird mit einem eigenen Modul *DrPortScan.pm* ausgeliefert. In diesem Modul werden die benötigten Perl-Module aufgelistet. Auf dem Dr.-Portscan-Server fehlten die Module *Curses::UI* sowie *XML::LibXML*, welche installiert werden mussten.

Es wird mit Dr. Portscan in der Version vom 20.12.2013 gearbeitet, siehe `https://git.lrz.de/?p=DrPortScan.git`. Um Dr. Portscan initial einzurichten wird das Perl Skript *setup.pl* ausgeführt, welches sich im root-Verzeichnis von Dr. Portscan befindet. Dieses überprüft ob die benötigten Perl-Module vorhanden sind und erzeugt die benötigten Scan-Verzeichnisse (*./scans/*, *./scans/new/*, *./scans/old/*, *./scans/failed/*). Es werden weitere Schritte, nämlich das Anlegen der SQLite-Datenbank und das Überprüfen auf Vorhandensein der Dateien *./conf/output-agents.conf* und *./autorun-output-agents.list* ausgeführt, die allerdings nicht für das Webfrontend benötigt werden.

Bei der Implementierung des Webfrontends wird die Skriptsprache PHP in Verbindung mit einer MySQL-Datenbank verwendet. Die Datenbank, siehe Kapitel 4.4, wurde mithilfe von phpMyAdmin aufgesetzt und verwaltet. Als Webserver wird Apache 2 genutzt. Diese Pakete werden mit den folgenden Befehlen auf dem Debian-System installiert:

```
apt-get install php5 libapache2-mod-php5
apt-get install mysql-server mysql-client
apt-get install phpmyadmin
apt-get install apache2
```

5.2 MVC-Pattern

Die einzelnen Komponenten wie das Model, die View und der Controller sind stets gleich aufgebaut. Diese Code-Gerüste werden nach einer Beschreibung des Front-Controllers erläutert.

5.2.1 Front-Controller

Der Front-Controller nimmt jede Abwicklung einer Anfrage entgegen. Anfragen, die nicht über den Front-Controller geschehen, wie z. B. das Eingeben einer URL, die direkt zu einem bestimmten Controller führt, sind unerwünscht und werden durch eine sogenannte *.htaccess-Datei*[5] unterbunden, Details dazu in Kapitel 5.10.1. Der Front-Controller ist wie folgt implementiert:

```
1 <?php
2 session_start();
3 require_once 'settings/conf.php';
4 require_once 'service/auth.php';
5
```

[4] Mit dem Befehl `apt-get` können auf Debian-Systemen Pakete wie Perl, PHP, MySQL oder bestimmte Module aus einer gegebenen Liste installiert, aktualisiert und gelöscht werden.

[5] Eine .htaccess-Datei ist eine Server-Konfigurationsdatei welche Zugriffsschutz für Verzeichnisse und weitere Optionen für die vom User aufgerufenen URLs bietet.

```php
class FrontController
{
  private $params;

  public function __construct($params)
  {
    $this->params = $params;
  }

  public function run()
  {
    $auth = new Auth($this->params);

    if($auth->authenticate() == 0)
    {
      require_once 'controller/login/login.php';
      $controller = new Login($this->params);
    }
    else
    {
      if(isset($_SESSION['admin']) && $_SESSION['admin'] == 1)
      {
        $ctl = 'nc_scans';
        if(isset($this->params['ctl']))
        {
          $ctl = $this->params['ctl'];
        }

        switch($ctl)
        {
          case 'aa_clients':
            require_once 'controller/admin/administration/clients.php';
            $controller = new Clients($this->params);
            break;
            ... //Hier folgen weitere Möglichkeiten die aus Platzgründen
                ausgelendet wurden.
        }
      }else
      {
        $ctl = 'nc_scans';
        if(isset($this->params['ctl']))
        {
          $ctl = $this->params['ctl'];
        }

        switch($ctl)
        {
          case 'nc_scans':
            require_once 'controller/netresp/config/scanning.php';
            $controller = new Scanning($this->params);
            break;
            ... //Hier folgen weitere Möglichkeiten die aus Platzgründen
                ausgelendet wurden.
        }
      }
    }
    $controller->run();
  }
}

$fc = new FrontController($_REQUEST);
$fc->run();
?>
```

Quellcode 5.1: Code des Front-Controllers (gekürzt).

Zu Beginn wird eine sogenannte Session gestartet (Zeile 2). Sie dient dazu Daten für einen bestimmten Nutzer der Website serverseitig zu speichern. So kann beispielsweise der Login-Status des Users auf dem Server festgehalten werden, so dass nicht bei jedem Aktualisieren der Website erneut eine Authentifizierung über die Datenbank geschehen muss. Dadurch werden Ressourcen gespart und die Anwendung beschleunigt.

Der Front-Controller ist, wie auch alle anderen Controller, eine Klasse. Von dieser wird Anfangs eine Instanz erstellt (Zeile 64) und die Parameter des *HTTP-GET* bzw. *HTTP-POST*[6] mit zugehörigen Werten übergeben.

Daraufhin wird eine Instanz der Authentifizierungs-Klasse erstellt, welche Auskunft über den Login-Status des Users gibt. Die Authentifizierung wird in einem späteren Kapitel ausführlich behandelt. Ist ein User nicht eingeloggt, wird er stets an den Login-Controller weitergeleitet (Zeile 21f, 60). Ist er eingeloggt folgt die Unterscheidung der beiden Benutzergruppen. Ist ein User authentifiziert, speichert die Authentifizierungs-Klasse die zugeteilte Benutzergruppe in der Session, auf die der Front-Controller zurückgreift. Ist der Dr.-Portscan-Administrator eingeloggt, wird er in den angeforderten Bereich weitergeleitet. Das geschieht durch den Navigations-Parameter `ctl`. Dieser enthält einen eindeutigen Wert, der auf den gewünschten Controller verweist (Zeile 36ff). Von diesem Controller wird eine Instanz erstellt und die GET- bzw. POST-Parameter übergeben. Schlussendlich wird mit `run()` die weitere Abarbeitung gestartet (Zeile 38, 60).

Das Projekt ist vorwiegend objektorientiert aufgebaut, d. h. insbesondere, dass jede Klasse sich in genau einer PHP-Datei befindet. Diese Dateien werden mit `require_once(PFAD)` in das aktuelle Skript eingebunden (Zeile 37), um dann verwendet werden zu können.

5.2.2 Model

Jedes Model repräsentiert eine Datenbank-Tabelle. Mithilfe der Models, welche als sogenannte *Data Access Objects* (DAO) implementiert werden, werden Operationen auf diesen Tabellen ausgeführt. Die Models befinden sich stets im Verzeichnis *./model/dao/*. Das Grundgerüst eines Models ist wie folgt aufgebaut:

```
1  <?php
2  require_once dirname(__FILE__).'/../dao.php';
3
4  class modelClassName extends DAO
5  {
6    const TBLCREATESTATEMENT = "CREATE TABLE `tblName` (
7      `attr1` int(11) NOT NULL AUTO_INCREMENT,
8      `attr2` varchar(50) NOT NULL,
9      PRIMARY KEY (`attr1`)
10     ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8";
11   const TBLINSERTVALUES = "INSERT INTO `tblName` (`attr1`, `attr2`) VALUES
12     (0, 'String1'),(1, 'String2')";
13   const TBLNAME = 'tblName';
14
15   public function __construct()
16   {
17     parent::__construct(self::TBLCREATESTATEMENT, self::TBLNAME, self::
           TBLINSERTVALUES);
18   }
19
20   public function functionName($parameter1, $parameter2, ...)
21   {
```

[6] GET bzw. POST sind Übertragungsmethoden von HTTP, mit denen Daten an den Server übermittelt werden.

```
22      //Funktions-Code.
23    }
24
25    ... //Weitere Funktionen.
26  }
27  ?>
```

Quellcode 5.2: Grundgerüst eines Models.

Jedes Model wird von der Klasse `DAO` abgeleitet (Zeile 4), dazu gleich mehr. Außerdem wird eine Konstante namens `TBLCREATESTATEMENT` definiert, welche das Tabellenschema enthält, eine namens `TBLNAME`, welche den Tabellennamen enthält, und optional eine weitere Konstante namens `TBLINSERTVALUES`, welche die eventuell vorgegebene Ausprägung der Tabelle enthält. Beim erstellen einer Instanz eines Models wird direkt an die Eltern-Klasse `DAO` weitergeleitet, bevor die mit verschiedenen Operationen bestückten Funktionen des Models genutzt werden können. Die `DAO` enthält häufig verwendete Funktionen und baut die Datenbank-Verbindung auf, siehe Quellcode 5.3.

```
1   <?php
2   require_once dirname(__FILE__).'/../settings/conf.php';
3
4   class DAO
5   {
6     protected $link;
7
8     public function __construct($tblCreateStatement, $tblInsertValues = null)
9     {
10      //Aufbau der Verbindung und auf UTF-8 umstellen.
11      $this->link = mysqli_connect(HOST,USER,PASSWORD,DATABASE) or die("Error " .
            mysqli_error($this->link));
12      mysqli_query($this->link, "SET NAMES 'utf8'");
13
14      //Wenn Tabelle nicht existiert, Tabelle erstellen und evtl. vorgegebene Werte
            einfügen.
15      if(!mysqli_query($this->link, "SELECT 1 FROM $tableName"))
16      {
17        mysqli_query($this->link, $tblCreateStatement);
18        if(!empty($tblInsertValues))
19        {
20          mysqli_query($this->link, $tblInsertValues);
21        }
22      }
23    }
24
25    protected function gets($query)
26    {
27      if($result = mysqli_query($this->link, $query))
28      {
29        $res = array();
30        while($row = mysqli_fetch_assoc($result))
31        {
32          $res[] = $row;
33        }
34        mysqli_free_result($result);
35        if(!empty($res)) { return $res; }
36      }
37      return array();
38    }
39
40    protected function get($query)
41    {
```

```
42      //Ähnliche Funktion wie gets(), es wird allerdings nur eine Ergebnis-Spalte
            zurückgeliefert.
43    }
44  }
45  ?>
```

Quellcode 5.3: Implementierung der Klasse `DAO`.

Die Verbindung wird mit zuvor definierten Konstanten aus der Datei *conf.php* (Zeile 2) aufgebaut. Dort sind die Werte für den Host, den Nutzer mit dem auf den MySQL-Dienst zugegriffen wird, dem zugehörigen Passwort und dem entsprechenden Datenbank-Namen abgelegt. Anschließend wird die Verbindung aufgebaut (Zeile 11) und MySQL mitgeteilt, dass die Antworten in UTF-8-Codierung gegeben werden sollen. Dies kann zwar auch über die Konfiguration von MySQL erfolgen, sollten allerdings andere Nutzer Antworten z. B. in ANSI-Codierung bevorzugen, ist dies die elegantere Variante. Schlussendlich wird überprüft, ob die Tabelle existiert (Zeile 15). Ist dies nicht der Fall, wird sie mit dem zuvor erwähnten Tabellenschema erstellt und gegebenenfalls mit vordefinierten Werten gefüllt. Die Funktionen `gets()` und `get()` (Zeile 25, 40) schicken eine Select-Anfrage, um bestimmte Werte einer Tabelle zu erhalten, an MySQL und geben das Ergebnis als assoziativen Array zurück. Ein assoziativer Array, wie er von MySQL geliefert wird, enthält nicht nur die Werte sondern zusätzlich noch die Attributnamen zu den einzelnen Werten, was dem Array eine wesentlich bessere Lesbarkeit verschafft.

5.2.3 View

Jeder Controller hat eine zugehörige View, die nach Abarbeitung aller Schritte, den HTML-Quellcode aufbaut und anzeigt. Eine solche View sieht folgendermaßen aus:

```
1  <?php
2  require_once 'view/masterView.php';
3
4  class viewClassName extends MasterView
5  {
6    public function __construct(&$params, &$data)
7    {
8      parent::__construct($params, $data);
9
10     $this->data['header'] = 'Überschrift der Seite';
11     $this->data['subHeader'] = 'Zusatzinformationen';
12     $this->data['ctl'] = 'controllerId';
13   }
14
15   protected function buildContent()
16   {
17     //Hier wird der eigentliche Content der Seite aufgebaut.
18   }
19 }
20 ?>
```

Quellcode 5.4: Grundgerüßt einer View.

Die View definiert zwei Überschriften und den Navigationsparameter `ctl`, wie bereits im Front-Controller verwendet. Danach wird in der Funktion `buildContent()` stets der eigentliche Inhalt der Seite aufgebaut. Zuvor wird allerdings der Konstruktor der Eltern-Klasse `MasterView` aufgerufen (Zeile 8). Diese abstrakte Klasse dient dazu das Grundgerüst der Website aufzubauen. Dieses Grundgerüst bildet den HTML-Head und den HTML-Body, wie nachfolgend dargestellt:

```php
<?php
abstract class MasterView
{
  protected $params, $data, $html;

  abstract protected function buildContent();

  public function __construct(&$params, &$data)
  {
    $this->params = $params;
    $this->data = $data;
    $this->html = '';
  }

  //Startet das Aufbauen der Seite und zeigt sie an.
  public function build()
  {
    $this->buildHead();
    $this->buildBody();
    $this->display();
  }

  //Baut den Head des HTML-Codes.
  protected function buildHead()
  {
    $this->html.= '<html>';
    $this->html.=   '<head>';
    $this->html.=     '<meta charset="UTF-8">';
    $this->html.=     '<title>Dr. Portscan</title>';
    $this->html.=     '<link rel="shortcut icon" href="./favicon.ico" type="image/x-icon">';
    $this->html.=     '<link rel="stylesheet" type="text/css" href="./web/css/master.css">';
    $this->html.=     '<link rel="stylesheet" type="text/css" href="./web/css/main.css">';
    $this->html.=   '</head>';
  }

  //Baut den Body + Content auf.
  protected function buildBody()
  {
    $this->html.=   '<body>';

    //Hier fängt der Aufbau der Navigationsleiste an.
    $this->html.=     '<nav>';
    $this->html.=       '<div class="menu">';
    $this->html.=         '<ul>';
    $this->html.=           '<li class="heading">';
    $this->html.=             '<a href="./index.php">Dr. Portscan</a>';
    $this->html.=           '</li>';

    if(isset($_SESSION['isLogged']) && $_SESSION['isLogged'] == 1 && $_SESSION['admin'] == 1)
    {
      //Hier werden die Navigationspunkte für den Dr.-Portscan-Administrator aufgebaut.
    }

    if(isset($_SESSION['isLogged']) && $_SESSION['isLogged'] == 1 && $_SESSION['admin'] == 0)
    {
      //Hier werden die Navigationspunkte für die Netzverantwortlichen aufgebaut.
    }

    if(isset($_SESSION['isLogged']) && $_SESSION['isLogged'] == 1)
    {
```

```
61        if(isset($_SESSION['extAuth']) && $_SESSION['extAuth'] == 0)
62        {
63          //Wurde der User manuell hinzugefügt, so enthält die Navigation
64          //einen Punkt "Konto", indem der User z. B. sein Passwort ändern kann.
65        }
66
67        //Hier der Navigationspunkt "Logout" für eingeloggte User.
68      }else
69      {
70        //Hier der Navigationspunkt "Login" für nicht eingeloggte User.
71      }
72
73      $this->html.=          '</ul>';
74      $this->html.=        '</div>';
75      $this->html.=      '</nav>';
76      $this->html.=      '<section class="main">';
77
78      //Hier werden die Überschriften gesetzt.
79
80      $this->html.=        '</header>';
81      $this->html.=        '<div class="content">';
82
83      $this->buildContent();  //Baut den eigentlichen Inhalt der Seite auf.
84
85      $this->html.=        '</div>';
86      $this->html.=      '</section>';
87
88      //Hier der Code für die Fehler- oder Info-Anzeige für den User.
89
90      $this->html.=    '</body>';
91      $this->html.= '</html>';
92    }
93
94    //Gibt den endgültigen HTML-Code aus.
95    protected function display()
96    {
97      echo $this->html;
98    }
99  }
100 ?>
```

Quellcode 5.5: Die `MasterView`, welche HTML-Head und Teile des HTML-Bodys aufbaut (gekürzt).

Die vom User erhaltenen HTTP-GET- und HTTP-POST-Parameter sowie die vom Controller erzeugten Daten werden als Klassenvariablen definiert. Außerdem wird die Klassenvariable `$html`, die nach und nach mit dem gesamten HTML-Code beschrieben wird, als leerer String angelegt.

Nach dem die Instanz der View aufgebaut wurde, wird vom Controller die Funktion `build()` aufgerufen. Von ihr aus werden nun nach und nach die einzelnen HTML-Fragmente zusammengestellt. Zuerst wird der HTML-Head aufgebaut. Dort werden neben der Codierung und dem Titel noch ein Icon und zwei CSS-Stylesheets angegeben (Zeilen 28ff) . Danach folgt der Aufbau des Bodys, bei dem zuerst die Navigationsleiste generiert wird. Ist der User nicht eingeloggt, wird nur der Login angezeigt (Zeile 70). Andernfalls wird zwischen dem Typ des Users unterschieden (Zeilen 49, 54) und daraufhin das entsprechende Navigations-Menü aufgebaut. Es findet ebenso eine Überprüfung der Authentifizierungsmethode des Nutzers statt (Zeile 61). Authentifiziert sich ein Nutzer per LRZ-SIM, so steht ihm der Menüpunkt „Mein Konto“ nicht zur Verfügung, unter dem sich eine Passwortänderung vornehmen lässt. Nachdem die Überschriften dem HTML-Code hinzugefügt wurden, wird der eigentliche Inhalt der View generiert (Zeile 83). Die Funktion `buildContent()` ist nicht Teil der `MasterView`,

sondern der eigentlichen View.

5.2.4 Controller

Die Controller sind die Schnittstelle zwischen Nutzer und Anwendung. Der Nutzer bestimmt durch seine Aktionen die Navigationsparameter. Anhand des Navigationsparameter *ctl* weiß die Anwendung an welchen Controller die User-Aktion weitergeleitet werden soll. Der Navigationsparameter *atn* ist optional. Ein Controller ist stets wie folgt aufgebaut:

```php
<?php

require_once 'controller/common.php';
require_once 'view/path/to/View.php';
...

class controllerClassName extends Common
{
  private $params, $data, ...;

  public function __construct(&$params)
  {
    $this->params = $params;
    $this->data = array();
    ...
  }

  //Leitet an die Funktion des Navigationsparameters 'atn' weiter oder an index()
       falls 'atn' fehlt.
  public function run()
  {
    $atn = 'index';
    if(isset($this->params['atn']))
    {
      $atn = $this->params['atn'];
    }
    if(!method_exists($this, $atn))
    {
      $atn = 'index';
    }
    $this->data['atn'] = $atn;

    $this->$atn();
  }

  private function index()
  {
    //Hier könnten z. B. Daten aus der Datenbank geholt werden.

    $view = new ClassOfView($this->params, $this->data);
    $view->build();
  }

  private function functionName()
  {
    ...
  }

  ... //Hier können weitere Funktionen folgen.
}
?>
```

Quellcode 5.6: Grundgerüst eines Controllers.

Nachdem der Konstruktor aufgebaut wurde, in dem einige Klassenvariablen gesetzt werden, wird vom Front-Controller die Funktion `run()` (Zeile 19) aufgerufen. Diese bestimmt anhand des Navigationsparameters `atn` die auszuführende Aktion. Wird kein solcher Parameter geliefert oder existiert die angegebene Funktion nicht, wird stets `index()` aufgerufen. Ein Controller erweitert die Klasse `Common`. In dieser Eltern-Klasse werden lediglich einige Funktionen definiert, die in mehr als nur einem Controller Verwendung finden.

5.3 Authentifizierung

Die Authentifizierung des Users findet bei jedem Aufruf der Website statt. Ist ein User eingeloggt, wird für ihn eine Session auf dem Server gestartet. Eine Session ist genau einem User zugeordnet. Der Server identifiziert die Session des Users anhand eines sogenannten *Cookies*[7], in dem die PHP-Session-ID gespeichert ist. Die Session an sich erlaubt es Daten in einem Array auf dem Server abzulegen, auf die dann ganz einfach per Session-Variable zugegriffen werden kann. Die Klasse `Auth` ist wie folgt implmentiert:

```
1  <?php
2  require_once 'model/dao/user.php';
3
4  class Auth
5  {
6    private $params;
7    const LOGOUT_TIME = 1200; //Logout, wenn 20 Minuten lang nichts gemacht.
8
9    public function __construct(&$params)
10   {
11     $this->params = $params;
12   }
13
14   //Überprüft ob der User Zugriff erhält.
15   public function authenticate()
16   {
17     if(isset($this->params['atn']) && $this->params['atn'] == 'logout')
18     {
19       unset($_SESSION);
20       session_destroy();
21
22       return 0;
23     }
24
25     if(isset($_SESSION['isLogged']) && $_SESSION['isLogged'] == 1 && isset(
           $_SESSION['userId']) && isset($_SESSION['admin']))
26     {
27       if(time() - $_SESSION['lastAction'] > self::LOGOUT_TIME)
28       {
29         unset($_SESSION);
30         session_destroy();
31
32         return 0;
33       }
34       $_SESSION['lastAction'] = time();
35
36       return 1;
37     }
38
39     if(isset($this->params['atn']) && $this->params['atn'] == 'login' && isset(
           $this->params['email']) && isset($this->params['pass']))
40     {
```

[7] Ein Cookie speichert clientseitig Daten, auf die der Server zugreifen kann.

```
41        $user = new User();
42        $usr = $user->getByEmailNotBlocked($this->params['email']);
43
44        if(!empty($usr))
45        {
46          if($usr['extAuth'] == 1)
47          {
48            //PLATZHALTER ----> Externe Authentifizierung.
49          }else
50          {
51            if($usr['hash'] == hash('sha256', $this->params['pass']))
52            {
53              $_SESSION['userId'] = $usr['userId'];
54              $_SESSION['admin'] = $usr['admin'];
55              $_SESSION['extAuth'] = $usr['extAuth'];
56              $_SESSION['isLogged'] = 1;
57              $_SESSION['cidrRange'] = $usr['cidrRange'];
58              $_SESSION['lastAction'] = time();
59
60              $user->update(array('lastAction'=>$_SESSION['lastAction']), $_SESSION
                    ['userId']);
61
62              return 1;
63            }
64          }
65        }
66      }
67
68      return 0;
69    }
70  }
71  ?>
```

Quellcode 5.7: Klasse `Auth` zur Authentifizierung der User.

Der Authentifizierungsvorgang beginnt mit der Funktion `authenticate()`, welche vom Front-Controller aufgerufen wird. Die Vorgänge in dieser Klasse werden durch den Navigationsparameter `atn` gesteuert. Hat dieser den Wert „logout“, so wird der Nutzer ausgeloggt indem die Session zerstört wird (Zeile 17ff). Ist in der Session die Variable `isLogged` auf 1 gesetzt, bedeutet das, dass der Nutzer sich zuvor bereits erfolgreich authentifiziert hat. Ist dies der Fall, wird die Zeitdifferenz der aktuellen Aktion zur letzten Aktion berechnet. Ist diese größer als der Inaktivitäts-Timeout (Zeile 7), so wird er automatisch ausgeloggt und muss sich erneut authentifizieren. Das bringt den Vorteil, dass sich unbefugte Personen, die sich bei Abwesenheit des eigentlichen Nutzers Zugriff zu dessen Rechner verschaffen, eine kleinere Wahrscheinlichkeit erhalten auch noch Zugriff zum Dr.-Portscan-Webfrontend zu bekommen. Will sich ein Nutzer neu einloggen, wird der Parameter `atn` auf „login“ gesetzt. Dann wird überprüft, ob der Nutzer in der Datenbank-Tabelle „`user`“ mit der angegebenen E-Mail-Adresse gefunden werden kann und ob er nicht geblockt ist (Zeile 42). Nun gibt es zwei mögliche Szenarien (Zeile 46):

Erstens, der User wurde durch den Datenbank-Import hinzugefügt und wird extern über das LRZ-SIM authentifiziert. Dafür ist ein Platzhalter vorgesehen, da diese Option nicht implementiert werden konnte, siehe Kapitel 5.4.

Zweitens, der User wurde manuell zum Dr.-Portscan-Webfrontend hinzugefügt und verfügt über ein hinterlegtes Passwort. In diesem Fall wird der Hash-Wert des übermittelten Passworts erzeugt und geprüft, ob dieser mit dem in der Datenbank hinterlegten übereinstimmt (Zeile 51). Ist die Überprüfung erfolgreich, werden einige Variablen in der Session gespeichert und die Authentifizierung war erfolgreich.

5.4 Datenbankimport und externe Authentifizierung

Der Import der LRZ-Netzdoku-Datenbank, um Netzverantwortlichen die bereits beim LRZ registriert sind direkt Zugriff zum Dr.-Portscan-Webfrontend zu gewähren und die Authentifizierung dieser Netzverantwortlichen über das LRZ-SIM, benötigt einer Konfigurierung und entsprechender Bereitstellung einer Schnittstelle. Die Schnittstelle war bereits vorhanden jedoch konnte eine Anbindung und Konfiguration für das Dr.-Portscan-Webfrontend nicht während der Bearbeitungszeit dieser Arbeit erfolgen. Im Code als auch in der Datenbank sind entsprechende Platzhalter vorhanden, um diese Funktionalität ohne weiteres einzubetten.

Das Skript, welches regelmäßig per Cronjob gestartet wird, die Netzdoku-Datenbank ausliest und den Stand mit dem vorhandenen lokalen Stand des Dr.-Portscan-Webfrontends vergleicht, ist im Verzeichnis *./service/* unter dem Namen *dbImportCron.php* zu finden. Für die externe Authentifizierung ist im Skript *./service/auth.php* ein entsprechender Platzhalter gekennzeichnet.

5.5 Scan-Konfiguration

In diesem Abschnitt wird nicht nur der implementierte Code der Scan-Konfiguration erläutert, sondern ebenso die User-Interaktionen mit dem Webfrontend. Als Beispiel dient das Konfigurieren eines internen Scans durch einen neu hinzugefügten Netzverantwortlichen.

Nachdem sich der Netzverantwortliche eingeloggt hat und den Menüpunkt „Konfiguration - Scans“ erreicht hat, wird ihm die in Abbildung 5.1 dargestellte Website präsentiert. Dort trägt er nach und nach die Konfigurationsdetails des gewünschten Scan-Bereichs ein. Hierbei ist anzumerken, dass aus dem geforderten Scan-Intervall der Anforderungsanalyse Scan-Tage geworden sind. Weshalb diese Änderung vorgenommen wurde, wird in Kapitel 5.7.1 näher erläutert. Am Ende dieses Kapitels wird die daraus resultierende Anpassung des Datenbankschemas der Tabelle „`scans`“ aufgezeigt.

Nachdem der interne Scan vollständig definiert wurde, wird der Button „*Speichern*“ betätigt. Dies leitet die Übermittlung der Daten der Form und der Navigationsparameter ein. Die Navigationsparameter werden per HTTP-GET, die eigentlichen Daten der Form per HTTP-POST übermittelt. Nun leitet der Front-Controller auf Grundlage der Navigationsparameter an den entsprechenden Controller weiter. In diesem Fall entspricht das der Klasse `Scanning`, welche im Skript *./controller/netresp/config/scanning.php* zu finden ist. Der zweite übermittelte Navigationsparameter definiert die auszuführende Funktion innerhalb der Klasse. In diesem Fall wurde der Parameter `atn` mit dem Wert „saveIntern“ übermittelt, der somit die gleichnamige Klassenfunktion aufruft. Diese Funktion ist folgendermaßen aufgebaut:

```
1  //Übermittelte Daten überprüfen
2
3  if(/*Daten nicht Fehlerhaft*/)
4  {
5    $scan = array('placement'=>'intern','userId'=>$_SESSION['userId']);
6    $scan['active'] = $this->params['active'];
7    $scan['netRange'] = $this->params['netRange'];
8    $scan['tcpRange'] = $this->params['tcpRange'];
9    $scan['udpRange'] = $this->params['udpRange'];
10   $scan['aggressiveness'] = $this->params['aggressiveness'];
11   $scan['time'] = $this->params['time'];
12   $scan['days'] = /*Alle gewählten Tage in Kommaseparierter Liste/*;
```

Quellcode 5.8: Funktion `saveIntern()` der Klasse `Scanning`, Teil 1.

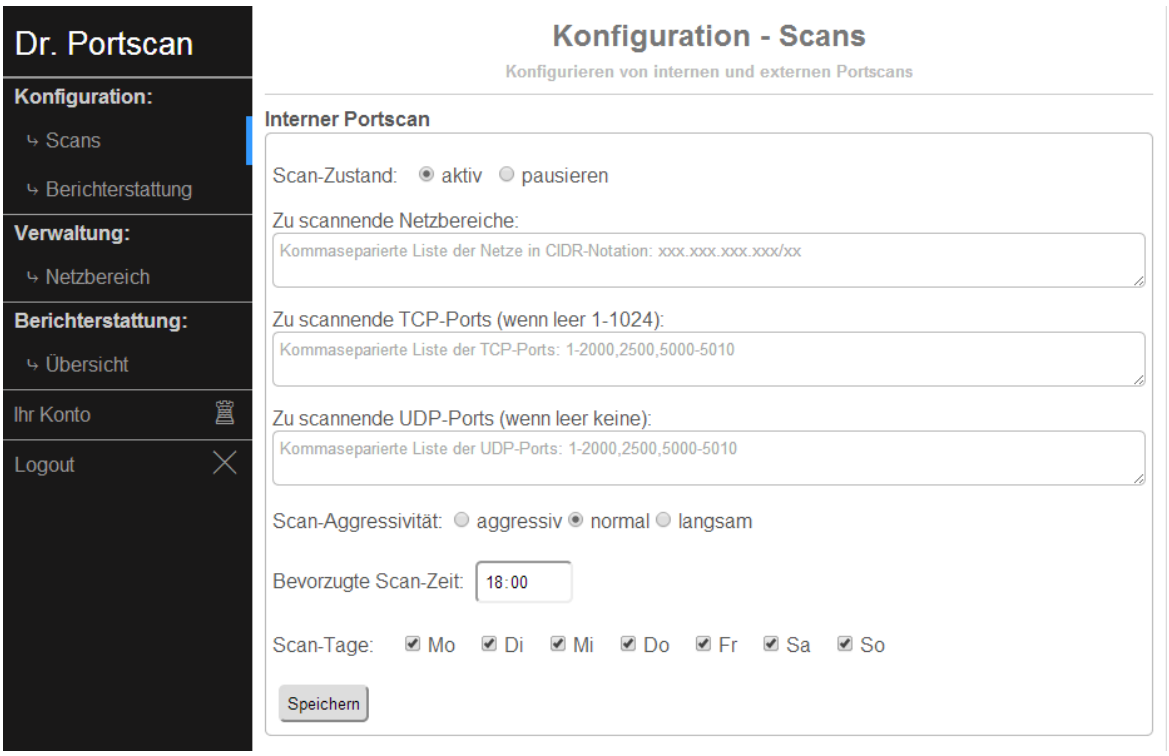

Abbildung 5.1: Konfiguration eines internen Scans durch einen Netzverantwortlichen.

Als erstes findet eine Überprüfung der übermittelten Daten statt. Dort wird unter anderem getestet, ob der Netzbereich in korrekter CIDR-Notation eingetragen wurde. Diese Überprüfung benutzt Funktionen der Eltern-Klasse `Commons`, weshalb diese im Code nicht näher dargestellt werden. Sind fehlerhafte Daten entdeckt worden, wird eine entsprechende Fehlermeldung ausgegeben, ansonsten wird mit dem Speichern der Daten fortgefahren. Dabei wird ein assoziatives Array aufgebaut, welches alle Tabellenattribute der Datenbank-Tabelle „`scans`“, bis auf die *`scanId`* enthält (Zeile 5ff).

```
13 //Überprüfen ob Netzbereich in Verantwortung liegt
14 $user = $this->user->getById($_SESSION['userId']);
15 if(parent::inRange($scan['netRange'], $user['cidrRange']))
16 {
17   $scan['execute'] = 'nmap -vv -sV -p';
18   if(!empty($this->params['tcpRange']))
19   {
20     $tcpRanges = array_map('trim', explode(',', $this->params['tcpRange']));
21     foreach ($tcpRanges as $tcpRange)
22     {
23       $scan['execute'].= 'T:'.$tcpRange.',';
24     }
25     $scan['execute'] = substr($scan['execute'],0,-1);
26   }else
27   {
28     $scan['execute'].= 'T:1-1024';
29   }
30   if(!empty($this->params['udpRange']))
31   {
32     $scan['execute'].= ',';
33     $udpRanges = array_map('trim', explode(',', $this->params['udpRange']));
34     foreach ($udpRanges as $udpRange)
```

```
35      {
36        $scan['execute'].= 'U:'.$udpRange.',';
37      }
38      $scan['execute'] = substr($scan['execute'],0,-1);
39    }
40    if($scan['aggressiveness'] == 'aggressive')
41    {
42      $scan['execute'].= ' -T4';
43    }else if($scan['aggressiveness'] == 'slow')
44    {
45      $scan['execute'].= ' -T2';
46    }
47    $scan['execute'].= " -oX %path%_%scanner%_%date%.xml ";
48    $ranges = array_map('trim', explode(',', $this->params['netRange']));
49    foreach($ranges as $range)
50    {
51      $scan['execute'].= $range.' ';
52    }
53    $scan['execute'] = substr($scan['execute'],0,-1);
```

Quellcode 5.9: Funktion `saveIntern()` der Klasse `Scanning`, Teil 2.

Nun wird verifiziert, ob der angegebene Scan-Bereich in der Verantwortung des Netzverantwortlichen liegt (Zeile 14f). Dazu wird die Funktion `inRange()` der Eltern-Klasse genutzt. Wie diese Verifizierung abläuft, wird später beschrieben. Es folgt die Generierung des, durch die Konfiguration definierten, Nmap-Aufrufs (Zeile 17-53). Dieser Nmap-Aufruf wird ebenfalls in die Datenbank-Tabelle geschrieben. Dazu wurde das Schema der Tabelle „`scans`“ erneut angepasst und das Attribut *`execute`* hinzugefügt. Dies bringt folgenden Vorteil mit sich: Wird der Nmap-Aufruf direkt bei der Konfiguration generiert, muss er nicht jedes mal beim Scan-Aufruf erstellt werden. Es werden somit Ressourcen eingespart.

```
54  $this->data['scans']['intern'] = $this->scans->getByUserIdByPlacement($_SESSION['
        userId'],'intern');
55  if(empty($this->data['scans']['intern']))
56  {
57    if(!$this->scans->insert($scan))
58    {
59      $this->data['msg'] = 'Fehler beim Hinzufügen des Scans!';
60    }
61  }else
62  {
63    if(!$this->scans->update($scan))
64    {
65      $this->data['msg'] = 'Fehler beim Ändern des Scans!';
66    }
67  }
68
69  //Hier folgen weitere Fehlermeldungen und der Aufruf von self::index()
```

Quellcode 5.10: Funktion `saveIntern()` der Klasse `Scanning`, Teil 3.

Nachdem der Nmap-Aufruf generiert wurde, wird die Datenbank nach einem bereits vorhandenen internen Scan des Users durchsucht (Zeile 54). Existiert solch ein Scan bereits, wird er aktualisiert, ansonsten neu erstellt. Zum Schluss wird die Funktion `index()` der Klasse `Scans` aufgerufen, welche die neuen Daten aus der Datenbank holt und der View übergibt. Die View welche den HTML-Quellcode generiert wird hier nicht näher beschrieben. Das Skript besteht aus zu vielen Zeilen für eine übersichtliche Darstellung auf Papier und enthält zudem keinen erwähnenswerten Code.

Um die Arbeitsweise des Models zu veranschaulichen, folgt eine Beschreibung der Funktionsaufrufe `getByUserIdByPlacement()` (Zeile 54) und `insert()` (Zeile 57) des Models `Scans`:

```
1  <?php
2  require_once dirname(__FILE__).'/../dao.php';
3
4  class Scans extends DAO
5  {
6    //Hier wurden Konstruktor und weitere Funktionen ausgeblendet.
7
8    public function getByUserIdByPlacement($userId,$placement)
9    {
10     $query = "SELECT * FROM `scans` WHERE `userId`=$userId AND `placement`='
           $placement' LIMIT 1";
11
12     return parent::daoGet($query);
13   }
14
15   public function insert($scan)
16   {
17     foreach($scan as $key => $value)
18     {
19       $scan[$key] = mysqli_real_escape_string($this->link, $value);
20     }
21     $query = "INSERT INTO `scans` SET ";
22     foreach($scan as $key => $value)
23     {
24       $query.= "`$key`='$value',";
25     }
26     $query = substr($query, 0, -1);
27
28     if(!$result = mysqli_query($this->link, $query))
29     {
30       return false;
31     }
32     return true;
33   }
34 }
35 ?>
```

Quellcode 5.11: Das Model `Scans` mit zwei Funktionsbeispielen.

Der Funktion `getByUserIdByPlacement()` werden beim Aufruf zwei Parameter übergeben. Einmal die User-ID des aktuellen Nutzers und einmal die Platzierung des Scans. Mit diesen Parametern wird eine SQL-Anfrage erstellt, welche in der Datenbank nach einem gespeicherten Scan mit diesen Werten sucht (Zeile 10). Um das Ergebnis der Anfrage zu bekommen, wird die Eltern-Funktion `daoGet()` bemüht. Diese liefert das Ergebnis der Anfrage in einem assoziativen Array zurück. Wird kein passender Scan gefunden wird ein leerer Array zurückgegeben.

Die Funktion `insert()` bekommt bereits ein assoziatives Array mit allen nötigen Scan-Details für die Datenbank übermittelt. Da einige Werte des Arrays vom User übermittelt wurden und diese SQL-Code enthalten könnten, der auf keinen Fall ausgeführt werden soll, sogenannte *SQL-Injections*, wird die PHP-Funktion `mysqli_real_escape_string()` verwendet (Zeile 19). Diese betrachtet die Werte, die in die Datenbank eingefügt werden sollen und entfernt mögliche SQL-Injections. Danach wird die eigentliche SQL-Anfrage generiert (Zeile 21ff) und abgeschickt(Zeile 28). Konnte die Anfrage erfolgreich ausgeführt werden, wird `true`, ansonsten `false` zurückgeliefert.

Die Funktion `inRange()`, deren Beschreibung zuvor bereits angekündigt wurde, ist innerhalb der Eltern-Klasse `Commons` folgendermaßen implementiert:

```
1  protected function inRange(&$childCidrs,&$parentCidrs)
2  {
3    $parentRanges = array_map('trim',explode(',', $parentCidrs));
4    $childRanges = array_map('trim',explode(',', $childCidrs));
5    foreach($childRanges as $childRange)
6    {
7      $inRange = false;
8      $fromToChildRange = self::cidrToRange($childRange);
9      foreach($parentRanges as $parentRange)
10     {
11       $fromToParentRange = self::cidrToRange($parentRange);
12
13       if(ip2long($fromToChildRange['startIp']) >= ip2long($fromToParentRange['
             startIp']) && ip2long($fromToChildRange['endIp']) <= ip2long(
             $fromToParentRange['endIp']))
14       {
15         $inRange = true;
16       }
17     }
18
19     if(!$inRange)
20     {
21       return false;
22     }
23   }
24   return true;
25 }
26
27 protected function cidrToRange($cidr)
28 {
29   $range = array();
30   $cidr = explode('/', $cidr);
31   $range['startIp'] = long2ip((ip2long($cidr[0])) & ((-1 << (32 - (int)$cidr[1]))
         ));
32   $range['endIp'] = long2ip((ip2long($cidr[0])) + pow(2, (32 - (int)$cidr[1])) -
         1);
33
34   return $range;
35 }
```

Quellcode 5.12: Funktionen zum Überprüfen ob Netzbereiche innerhalb anderer Netzbereiche liegen.

Die Funktion `inRange()` bekommt zwei kommaseparierte Listen mit Netzbereichen übergeben. Einmal dem so benannten Eltern-Bereich und einmal dem Kind-Bereich. Die Funktion überprüft, ob der Kind-Bereich innerhalb des Eltern-Bereichs liegt. Dazu werden die Listen anfangs in Arrays überführt (Zeile 3f), damit sie dann in Schleifen einzeln durchgegangen werden können. Sodann wird über den Kind-Bereich iteriert. Der betrachtete Kind-Bereich wird dann von der CIDR-Notation mithilfe der Funktion `cidrToRange()` in ein assoziatives Array mit den Schlüsselwerten `startIp` und `endIp` überführt. Die `startIp` enthält die erste mögliche IP-Adresse des Netzbereichs und `endIp` die letzte mögliche. Danach wird innerhalb der Schleife über die Kind-Bereiche eine weitere Schleife gestartet, welche über die Eltern-Bereiche iteriert. Der betrachtete Eltern-Bereich wird ebenfalls in die Form des gerade beschriebenen Arrays überführt. Schlussendlich folgt der Vergleich der beiden Netzbereiche (Zeile 13). Liegt ein Kind-Bereich nicht innerhalb einer der Eltern-Bereiche, wird `false` zurückgegeben, ansonsten `true`. Um die IP-Adressen vergleichen zu können, wird die PHP-Funktion `ip2long()` genutzt. Sie dient dazu, die IPs in numerische Werte zu überführen, mit denen größer-/kleiner-Vergleiche möglich sind.

Wie vorher bereits erwähnt, wird nun die Änderung des Datenbankschemas für die Tabelle

„scans“ dargestellt. Das Attribut *interval* wurde entfernt. Folgende zwei Attribute sind hinzugekommen:

Attribut	Erklärung
days	Kommaseparierte Liste der Scan-Tage (Mon|Tue|Wed|Thu|Fri|Sat|Sun)
execute	Der Nmap-Aufruf

5.6 Berichterstattung per Webfrontend

Als nächstes wird die Berichterstattung per Webfrontend beschrieben. Die Netzverantwortlichen sollen laut Anforderungsanalyse eine Möglichkeit erhalten, die Scan-Ergebnisse direkt im Webfrontend betrachten zu können. Diese Implementierung wird folgend erläutert. Die einzig interessanten Code-Bereiche in diesem Zusammenhang betreffen einmal das Model, in dem aufgrund des Netzbereichs des Netzverantwortlichen die Scan-Ergebnisse aus der Datenbank gefunden werden müssen, und einmal die Filterfunktion für die Ergebnis-Tabelle.

Um die spezifischen Ergebnisse für die jeweiligen Netzverantwortlichen zu erhalten, wird die Funktion `getsByStartIpEndIp()` des Models `HostResults` aufgerufen, welche im folgenden Quellcode abgebildet ist:

```
1  <?php
2  require_once dirname(__FILE__).'/../dao.php';
3
4  class HostResults extends DAO
5  {
6    //Hier wurden Konstruktor und weitere Funktionen ausgeblendet.
7
8    public function getsByStartIpEndIp($startIps, $endIps)
9    {
10     $query = "SELECT * FROM `hostResults` WHERE ";
11     for($i = 0; $i < count($startIps); $i++)
12     {
13       if($i == 0)
14       {
15         $query.= "INET_ATON(`ipaddr`)
16           BETWEEN INET_ATON('$startIps[$i]')
17           AND INET_ATON('$endIps[$i]') ";
18       }else
19       {
20         $query.= "OR INET_ATON(`ipaddr`)
21           BETWEEN INET_ATON('$startIps[$i]')
22           AND INET_ATON('$endIps[$i]') ";
23       }
24     }
25
26     return parent::daoGets($query);
27   }
28 }
29 ?>
```

Quellcode 5.13: Das Model `HostResults` mit Funktion zum abfragen der Hosts eines Netzbereichs.

Dieser Funktion werden zwei Arrays mit gleicher Länge übergeben. Diese Arrays enthalten jeweils die Start- und End-IP eines Netzbereichs beim gleichen Index des Arrays. Danach wird die SQL-Anfrage generiert. Hierbei wird die Funktion `INET_ATON()` von MySQL zur Hilfe gezogen. Sie berechnet einen numerischen Wert, der die IPv4-Adresse repräsentiert. Mit diesen Werten ist es nun möglich die betreffenden Hosts aus der Tabelle zu erhalten. Es

wird lediglich geprüft, ob die Host-IPs der Einträge in der Tabelle „`hostResults`“ innerhalb der angegebenen Netzbereiche liegen. Die so generierte Anfrage wird dann mit der Funktion `daoGets()` der Eltern-Klasse `DAO` des Models ausgewertet (Zeile 26).

Danach werden die einzelnen Ports der Hosts per ***`hostId`*** gefunden. Dies geschieht dann allerdings erst im weiteren Verlauf des Controllers. Dieser übergibt die erhaltenen Daten dann wieder an die zuständige View, welche eine Tabelle mit den Port-Ergebnissen erstellt. Ein Beispiel eines Ergebnisses im Browser ist in Abbildung 5.2 abgebildet. Die Scan-Ergebnisse sind zum Teil frei erfunden und sollen nur als Darstellungsbeispiel dienen.

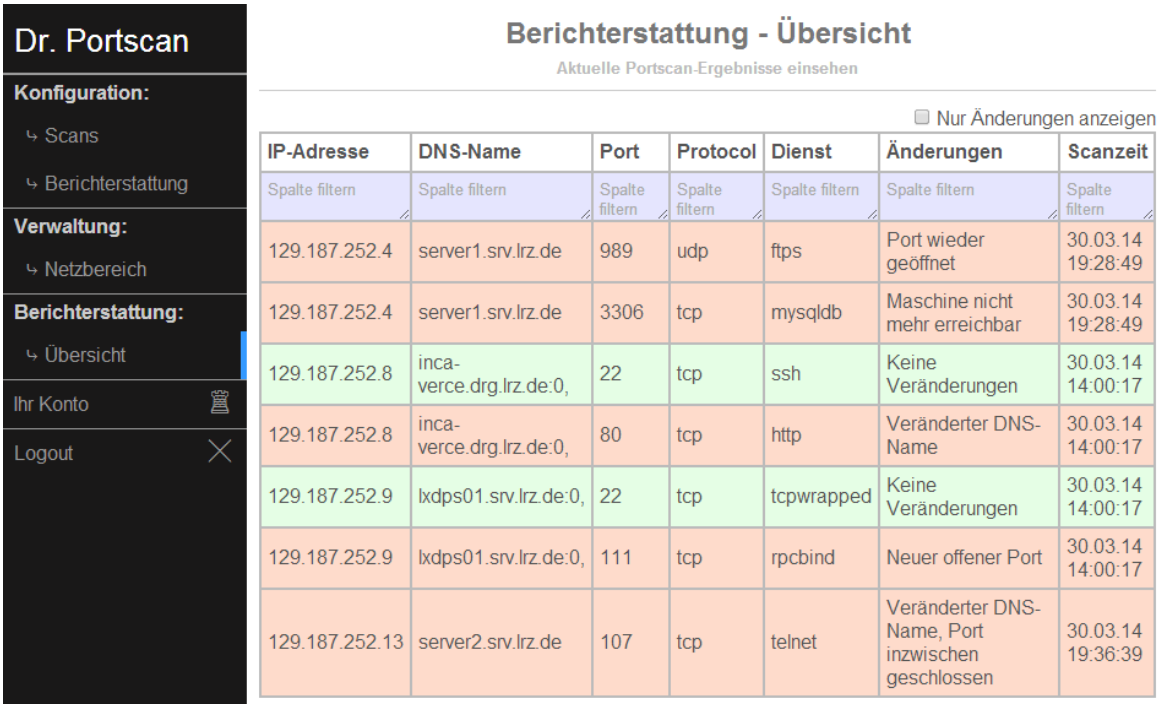

Abbildung 5.2: Beispiel einer Berichterstattung per Webfrontend (erfundene Scan-Ergebnisse).

Diese Tabelle wurde mit zwei JavaScript-Funktionen zum Filtern der Ergebnisse versehen. Einmal können per Checkbox nur die Ergebnisse angezeigt werden, bei denen Änderungen aufgetreten sind, ansonsten wird eine Gesamtübersicht der aktuellen Ergebnisse angezeigt. Die zweite Funktion ermöglicht eine spaltenweise Filterung der Tabelle. Dazu wurde die zweite Zeile der Tabelle mit Textboxen versehen. In diese kann der User beliebige Zeichenketten eingeben. Die Funktion blendet dann alle Zeilen aus, die diese Zeichenkette nicht in der entsprechenden Spalte enthalten. Nachfolgend ist der JavaScript-Code abgebildet, der diese Funktionalität liefert:

```
1 var filters = document.getElementsByTagName("textarea");
2 for(var i = 0; i < filters.length; i++)
3 {
4   filters[i].addEventListener("keyup",filterTable);
5 }
6
7 function filterTable()
8 {
9   var filterValues = new Array();
```

```
10    var filterRow = this.parentNode.parentNode;
11    var filters = filterRow.childNodes;
12    for(var i = 0; i < filters.length; i++)
13    {
14      if(filters.item(i).firstChild.value != "")
15      {
16        filterValues[i] = filters.item(i).firstChild.value;
17      }
18    }
19
20    var rowsToHide = new Array();
21    var rows = document.getElementsByTagName("tbody")[0].children;
22    for(var j = 0; j < rows.length; j++)
23    {
24      if(document.getElementById("changes").checked == 1 && rows[j].className == "
          green")
25      {
26        rowsToHide[j] = 1;
27        continue;
28      }
29
30      rowsToHide[j] = 0;
31      var cells = rows[j].childNodes;
32      for(var k = 0; k < filterValues.length; k++)
33      {
34        var patt = new RegExp(filterValues[k],"i");
35        if(!patt.test(cells.item(k).textContent))
36        {
37          rowsToHide[j] = 1;
38        }
39      }
40    }
41
42    for(var i = 0; i < rowsToHide.length; i++)
43    {
44      if(rowsToHide[i] == 1)
45      {
46        rows[i].style.display = "none";
47      }else{
48        rows[i].style.display = "table-row";
49      }
50    }
51  }
```

Quellcode 5.14: Die Funktion `filterTable()` zum Filtern per Zeichenkette.

Das JavaScript wird am Ende des HTML-Bodys geladen. Daraufhin werden alle Textfelder, in welche die Zeichenketten eingetragen werden können, mit einem Event-Listener der auf das sogenannte *keyup-Event* reagiert, versehen (Zeile 1ff). D. h., dass wenn ein User ein Zeichen in ein solches Feld eingibt, wird die Funktion `filterTable()` ausgeführt. Im Funktionsrumpf wird dann zuerst die Filter-Zeile der Tabelle ausgewertet (Zeile 9ff). Das Ergebnis ist ein Array, welches als Schlüsselwert die Spaltennummer der Filter-Zelle und die dazugehörige Zeichenkette enthält. Danach wird ermittelt, welche Zeilen ausgeblendet werden sollen. Dazu werden alle Zeilen des Tabellen-Bodys durchgegangen (Zeile 22). Wurde eine Zeile bereits durch die zuvor erwähnte Checkbox ausgeblendet, wird sie gesondert betrachtet (Zeile 24ff). Ansonsten wird für jede Zelle der Zeile überprüft, ob die eingegebene Zeichenkette in der entsprechenden Filter-Zelle in der betrachteten Zelle der Zeile vorkommt (Zeile 35). Ist dies nicht der Fall, wird die Zeilennummer in einem Array festgehalten, der als Schlüsselwert die Zeilennummer hat und den Wert 1, wenn die Zeile ausgeblendet werden soll, sonst 0. In einer letzten Iteration über alle Zeilen wird dann der entsprechende Befehl zum Ausblenden oder

Einblenden ausgeführt (Zeile 42ff).

5.7 Scanner-Netz

Das Konzept des Scanner-Netzes wurde bereits in Kapitel 4.2 vorgestellt, nun folgt die Implementierung. Dabei wird zwischen zwei Aufgabentypen unterschieden. Einmal der Scan-Ausführung, die für die Übermittlung der Scans an die Datenquellen zuständig ist, und einmal der Transferierung der Scan-Ergebnisse von den Datenquellen zurück zum Dr.-Portscan-Server.

5.7.1 Scan-Ausführung

Auf dem Dr.-Portscan-Server wird in viertelstündigen Intervallen per Cronjob ein Skript namens *cron.php* ausgeführt. Dort werden die eingestellten Scans durchgegangen und diejenigen gestartet, die zum Zeitpunkt der Ausführung des Skriptes und zu diesem Tag an der Reihe sind.

Bei der Implementierung ist eine Unregelmäßigkeit aufgetreten, welche die Konfiguration der Scans betrifft. Die Scans, die beim Aufruf des Skripts an der Reihe sind, wurden anhand der eingestellten Scan-Uhrzeit und des Scan-Intervalls bestimmt. Wurde ein Scan gestartet, wurde der aktuelle Unix-Timestamp in die Datenbank-Tabelle „`scans`“, im Attribut *`lastScantime`* eingetragen. Nun war dieser aktuelle Unix-Timestamp stets ein paar Sekunden älter als die Uhrzeit des letzten Scan-Zeitpunkts. Ist nun der letzte Scan-Zeitpunkt kurz vor Mitternacht und der Scan wurde erneut gestartet, sprang der letzte Scan-Zeitpunkt um einen Tag nach vorne. Dies hatte zur Folge, dass der Scan nach einer gewissen Zeit immer mal wieder einen Tag unerwünscht aussetzt. Eine programmatische Lösung wäre die letzte Scan-Zeit schlicht um einen Tag zu erhöhen und nicht den aktuellen Unix-Timestamp einzutragen. Diese Überlegungen führten allerdings zu einem anderen zu beachtenden Punkt. Die Konfiguration der Ausführung per Scan-Intervalle ist nicht transparent für den Netzverantwortlichen. Er weiß nämlich nicht zu welchen Tagen exakt die Ausführung erfolgt, vorausgesetzt das Scan-Intervall ist nicht auf tägliche Scans eingestellt. So kann es beispielsweise sein, dass ein wöchentlicher Scan immer Sonntags stattfindet, obwohl dort einige der zu scannenden Hosts ausgeschaltet sind und die Ergebnisse somit unvollständig wären.

Da es kein Zufall sein soll, wann die Scans stattfinden, wird ein anderer Lösungsansatz verfolgt. Der Netzverantwortliche kann beim Konfigurieren der Scans nicht das Intervall einstellen, sondern die Wochentage an denen gescannt werden soll. Dadurch wird die zuvor genannte Unregelmäßigkeit umgangen und der Scan-Ablauf für den Netzverantwortlichen transparenter. Das resultierende Skript ist in Quellcode 5.15 dargestellt.

```
1  <?php
2  ini_set('error_reporting', E_ALL);
3  require_once dirname(__FILE__).'/../model/dao/scans.php';
4  require_once dirname(__FILE__).'/../model/dao/scanners.php';
5  require_once dirname(__FILE__).'/../model/dao/user.php';
6  require_once dirname(__FILE__).'/../model/dao/rangeExceptions.php';
7
8  class Cron
9  {
10   private $day,$date,$scanners,$scans,$users,$rangeExceptions,$exclude;
11
12   const ONEHOUR = 10000; const TWENTYMINUTES = 2000;
13   const MONDAY = 'Mon'; const TUESDAY = 'Tue'; const WEDNESDAY = 'Wed';
```

```
14    const THURSDAY = 'Thu'; const FRIDAY = 'Fri'; const SATURDAY = 'Sat';
15    const SUNDAY = 'Sun';
16
17    private $scannerConnections = array(
18      'lxdps02'=>array('user'=>'root','pub'=>'/root/.ssh/id_rsa.pub','priv'=>'/root
          /.ssh/id_rsa','path'=>'/opt/DrPortscan/scans/nmap-xml')
19    );
20
21    public function __construct()
22    {
23      $this->day = date('D');
24      $this->date = date('Y-m-d');
25      $this->scanners = new Scanners();
26      $this->scans = new Scans();
27      $this->users = new User();
28      $this->rangeExceptions = new RangeExceptions();
29
30      $exceptions = '';
31      $admins = $this->users->getsAdmins(); //Alle User die Admins sind.
32      foreach($admins as $admin)
33      {
34        $rangeExceptions = $this->rangeExceptions->getByUserId($admin['userId']);
35        if(empty($exceptions))
36        {
37          $exceptions = str_replace(' ', '', $rangeExceptions['exceptions']);
38        }else
39        {
40          $exceptions = ','.str_replace(' ', '', $rangeExceptions['exceptions']);
41        }
42      }
43
44      $this->exclude = empty($exceptions) ? '':'--exclude '.$exceptions;
45    }
46
47    public function run()
48    {
49      //Details folgen.
50    }
51
52    private function execute(&$scan,$scanners)
53    {
54      //Details folgen.
55    }
56  }
57
58  $cron = new Cron();
59  $cron->run();
60  ?>
```

Quellcode 5.15: Skript zur regelmäßigen Ausführung der eingestellten Scans.

Anfangs wird eine Instanz der Klasse `Cron` erzeugt (Zeile 58). Die Klasse enthält neben den Klassenvariablen auch einige Konstanten (Zeile 12ff), die unter anderen die möglichen Werte für die Scan-Tage enthalten. Die Klassenvariable `$scannerConnections` enthält Details zum Verbindungsaufbau mit den Datenquellen. In diesem Array werden für jeden Scanner Daten abgelegt, die für eine SSH-Verbindung per Public-Key-Verfahren benötigt werden. Zusätzlich ist das Verzeichnis der Datenquelle angegeben, in dem die Ergebnisse der Scans im XML-Format gespeichert werden sollen. Im Konstruktor wird der aktuelle Tag und das aktuelle Datum in die entsprechenden Klassenvariablen geschrieben (Zeile 23f). Danach werden Instanzen der benötigten Models erstellt. Nun gilt es die vom Dr.-Portscan-Administrator definierte Ausnahmeliste zu erhalten und in die Klassenvariable zu schreiben (Zeile 30ff). Dieser Codebereich ist so ausgelegt, dass auch mehrere Ausnahmelisten verschiedener Dr.-

Portscan-Administratoren beachtet werden, für den Fall, dass es nicht nur einen geben sollte. Nun wird der eigentliche Teil der Abarbeitung durch den Aufruf der Funktion `run()` gestartet (Zeile 59). Diese Funktion ist folgendermaßen aufgebaut:

```
1  public function run()
2  {
3    $scanners = $this->scanners->gets();
4    $scans = $this->scans->getsActiveNotBlocked();
5
6    foreach($scanners as &$scanner)
7    {
8      if(array_key_exists($scanner['signature'], $this->scannerConnections))
9      {
10       $scanner['auth'] = $this->scannerConnections[$scanner['signature']];
11     }
12   }
13
14   foreach($scans as $scan)
15   {
16     $lastScanDate = date('Y-m-d',$scan['lastScantime']);
17     $comparableTime = 1000000 + (int)str_replace(':', '', $scan['time']);
18     $nowTime = 1000000 + (int)date('His');
19     if($nowTime >= $comparableTime && $nowTime < $comparableTime + self::ONEHOUR)
20     {
21       if(strpos($scan['days'],$this->day) !== false && $lastScanDate != $this->
           date)
22       {
23         self::execute($scan,$scanners);
24       }
25     }
26   }
27 }
```

Quellcode 5.16: Funktion `run()` der Klasse `Cron`.

Zu Beginn werden alle Scanner und alle Scans, die weder geblockt noch pausiert wurden, aus der Datenbank geholt. Danach werden die Scanner mit den passenden Daten aus der Klassenvariable `$scannerConnections` angereichert. Jetzt werden in einer Schleife alle Scans durchgegangen. Es werden einige Variablen beschrieben, die bei der Überprüfung des Ausführungsstatus des Scans benötigt werden. Als erstes wird überprüft ob der eingestellte Scan-Zeitpunkt ungefähr dem aktuellen entspricht (Zeile 19). Die gewählte Zeitspanne, welche für die Ausführung des Scans spricht, startet bei der aktuellen Uhrzeit und endet bei der aktuellen Uhrzeit plus einer Stunde. Das späteste Ende der Ausführung wurde um eine Stunde nach hinten verlegt, um einen Puffer zu schaffen. Dieser Puffer soll verhindern, dass Scans eventuell, aufgrund von Serverausfällen oder Wartungsarbeiten, nicht ausgeführt werden. Sodann erfolgt noch die Überprüfung des Wochentags und ob der Scan nicht schon einmal an diesem Tag ausgeführt wurde (Zeile 21). Sind alle Bedingungen erfüllt, wird der Scan ausgeführt:

```
1  public function execute(&$scan,$scanners)
2  {
3    shuffle($scanners);
4    foreach($scanners as &$scanner)
5    {
6      if($scan['placement'] == $scanner['placement'])
7      {
8        if(!isset($scanner['ssh2Auth']))
9        {
10         $scanner['connection'] = ssh2_connect($scanner['ipaddr'],22,array('
             hostkey' => 'ssh-rsa'));
```

```
11          if(ssh2_auth_pubkey_file($scanner['connection'],$scanner['auth']['user'],
                $scanner['auth']['pub'],$scanner['auth']['priv']))
12          {
13            $scanner['ssh2Auth'] = 1;
14          }else
15          {
16            echo date('Y-m-d H:i:s').' - Fehler bei Verbindungsaufbau zu
                  Datenquelle "'.$scanner['signature'].'" und Scan mit id "'.$scan['
                  scanId'].'"'.PHP_EOL;
17          }
18        }
19        $call = $scan['execute'];
20        $call = str_replace('%path%', $scanner['auth']['path'], $call);
21        $call = str_replace('%scanner%', $scanner['signature'], $call);
22        $call = str_replace('%date%', date('YmdHis'), $call);
23
24        echo date('Y-m-d H:i:s')." - ".$call.' '.$this->exclude.PHP_EOL;
25        ssh2_exec($scanner['connection'],$call.' '.$this->exclude.' > /dev/null &')
              ;
26
27        $newScan = array('userId'=>$scan['userId'],'placement'=>$scan['placement'],
              'lastScantime'=>time());
28        $this->scans->update($newScan);
29
30        break;
31      }
32    }
33  }
```

Quellcode 5.17: Funktion `execute()` der Klasse `Cron`.

Die Funktion startet damit, dass ein passender Scanner (intern oder extern) für den Scan zufällig ausgewählt wird. Das zufällige Auswählen dient der einfachen Skalierbarkeit des Scanner-Netzes. Ist z. B. ein interner Scanner aufgrund von zu vielen durchzuführenden Scans überlastet, kann die Last relativ einfach verteilt werden. Es müssen lediglich eine neue Datenquelle gefunden und konfiguriert, diese in die Datenbank-Tabelle „`scanners`“ eingetragen und die Klassenvariable `$scannerConnections` entsprechend gefüllt werden. Das zufällige Auswahlverfahren wird durch ein Mischen des Arrays der Scanner und anschließendem Auswählen des ersten passenden, also internen bzw. externen Scanners, erreicht. Nachdem der Scanner ausgewählt wurde, wird geprüft ob bereits eine SSH-Verbindung zu dieser Datenquelle besteht (Zeile 8ff). Wenn nicht wird versucht sie aufzubauen. Gelingt der Verbindungsaufbau nicht wird eine Fehlermeldung ausgegeben. Besteht die Verbindung wird der vorgefertigte Nmap-Aufruf des Scans, welcher in der Datenbank gespeichert ist, mit weiteren Details gefüllt (Zeile 19ff). Dieser Nmap-Aufruf wird dann zur Dokumentation in das vorher erwähnte Logbuch eingetragen und per SSH auf der entfernten Datenquelle ausgeführt. Schlussendlich wird der letzte Scan-Zeitpunkt des Scans in der Datenbank aktualisiert. Durch die direkte Aktualisierung des letzten Scan-Zeitpunkts nach dem Aufruf auf der Datenquelle wird erreicht, dass der Scan bei der nächsten Ausführung des Cronjobs nicht erneut gestartet wird, was der Fall wäre, wenn die letzte Scan-Zeit erst nach Beendigung des Scans eingetragen werden würde.

Dieses Skript und der Input-Watcher, welcher die Scan-Ergebnisse der Datenquellen in Empfang nimmt, werden etwas zeitversetzt, um die Auslastung des Servers zu verteilen, viertelstündlich mit folgenden Cronjobs ausgeführt:

```
*/15 * * * * php5 /var/www/DrPortscan/service/cron.php >>
/opt/DrPortscan/log/cron.log
```

```
5,20,35,55 * * * * cd /opt/DrPortscan/ && perl input-watcher.pl >>
/opt/DrPortscan/log/input-watcher.log
```

5.7.2 Transferieren der Ergebnisse

Da die Scan-Aufrufe auf den Datenquellen entfernt vom Dr.-Portscan-Server gestartet und alle Ergebnisse im gleichen Verzeichnis abgelegt werden, muss nun noch ein Skript erstellt werden, welches die Scan-Ergebnisse zurück zum Dr.-Portscan-Server transferiert. Zu beachten ist, dass Nmap die XML-Dateien, in denen die Ergebnisse gespeichert werden, kontinuierlich, also über die komplette Dauer der Scan-Ausführung, beschreibt. Es muss somit eine Überprüfung stattfinden, ob die Ergebnisse bereits vollständig sind, bevor mit der Übertragung begonnen wird. Diese Aufgabe übernimmt ein Shell-Skript auf der Datenquelle, welches die Pfade zu den fertigen Scan-Dateien ausgibt und wie folgt implementiert ist:

```
1  #!/bin/bash
2
3  FILES=$1'*'
4
5  for f in $FILES
6  do
7    if [[ "$f" == *\.* ]]
8    then
9      LASTLINE=$(tail -1 $f)
10     if [[ $LASTLINE =~ ^\</nmaprun\>$ ]]
11     then
12       echo "$f"
13     fi
14   fi
15 done
```

Quellcode 5.18: Shell-Skript zur Identifizierung fertiger Scan-Dateien.

Dem Skript wird ein Argument übergeben, welches den Pfad der Scan-Dateien auf der Datenquelle angibt. Weitere Angaben sind nicht nötigt. Es kann somit universell auf allen Datenquellen eingesetzt werden. Das Skript geht in einer Schleife alle Dateien des Verzeichnisses durch und überprüft die letzte Zeile im Dokument. Nmap-XML-Ergebnisse werden nämlich stets mit dem XML-Closing-Tag „</nmaprun>“ beendet. Entspricht die letzte Zeile diesem String (Zeile 10), was per Regular-Expression getestet wird, so wird der absolute Pfad dieser Datei ausgegeben.

Dieses Skript wird nun als Helfer, innerhalb des eigentlichen Transfer-Skripts auf dem Dr.-Portscan-Server verwendet, welches wie folgt implementiert ist:

```
1  <?php
2  ini_set('error_reporting', E_ALL);
3  define('NEW_SCANS_PATH','/opt/DrPortscan/scans/new/');
4
5  class ScpCron
6  {
7    private $scannerConnections = array(
8          'lxdps02'=>array('ipaddr'=>'129.187.252.10','user'=>'root',
9              'pub'=>'/root/.ssh/id_rsa.pub','priv'=>'/root/.ssh/id_rsa',
10             'script'=>'/opt/DrPortscan/getFinishedScans.sh','path'=>'/opt/
                   DrPortscan/scans/')
11       );
12
13   public function run()
14   {
```

```
15      foreach($this->scannerConnections as $key => &$scanner)
16      {
17        $scanner['connection'] = ssh2_connect($scanner['ipaddr'],22,array('hostkey'
              => 'ssh-rsa'));
18        if($scanner['connection'] !== false)
19        {
20          if(ssh2_auth_pubkey_file($scanner['connection'],$scanner['user'],$scanner
                ['pub'],$scanner['priv']))
21          {
22            $stdout_stream = ssh2_exec($scanner['connection'], 'bash '.$scanner['
                  script'].' '.$scanner['path']);
23            sleep(1);
24            $stderr_stream = ssh2_fetch_stream($stdout_stream, SSH2_STREAM_STDERR);
25            stream_set_blocking($stdout_stream, true);
26            while($line = fgets($stdout_stream))
27            {
28              flush();
29              $fullPath = str_replace(array("\r", "\n"), '', $line);
30              echo date('Y-m-d H:i:s').' - '.$fullPath.' übertragen'.PHP_EOL;
31              $fileName = basename($fullPath);
32              if(ssh2_scp_recv($scanner['connection'], $fullPath, NEW_SCANS_PATH.
                    $fileName))
33              {
34                ssh2_exec($scanner['connection'], "rm $fullPath");
35              }
36            }
37          }else { echo 'ssh2_auth_pubkey_file Fehler für '.$key.PHP_EOL; }
38        }else { echo 'ssh2_connect Fehler für '.$key.PHP_EOL; }
39      }
40    }
41  }
42
43  $cron = new ScpCron();
44  $cron->run();
45  ?>
```

Quellcode 5.19: Skript zum Transferieren fertiger Scan-Dateien.

Zu Beginn wird der Ziel-Pfad der Scan-Ergebnisse definiert (Zeile 3). Danach eine Instanz der Klasse `ScpCron` erstellt und die Funktion `run()` gestartet. In der Klassenvariable `$scannerConnections` sind die Details für jede Datenquelle hinterlegt, welche zum Transferieren benötigt werden. Diese Datenquellen werden dann in einer Schleife durchlaufen. Für jede Datenquelle wird dann eine SSH-Verbindung aufgebaut und eine Authentifizierung per Public-Key-Verfahren durchgeführt. Danach wird der Befehl zur Ausführung des zuvor erläuterten bash-Skripts an die Datenquelle geschickt (Zeile 22). Das resultierende Ergebnis, ein sogenannter *Stream*, ist die Folge. Dieser Stream enthält pro Zeile jeweils einen absoluten Pfad zu einer fertigen Scan-Dateien. Dieser Stream wird nun in einer Schleife Zeile für Zeile ausgelesen (Zeile 26). Für jede Zeile, also jeden Pfad einer fertigen Scan-Datei, wird eine Übertragung per SCP an den Dr.-Portscan-Server beantragt (Zeile 32). War die Übertragung erfolgreich, wird die Datei auf der Datenquelle entfernt.

Dieses Skript wird ebenfalls viertelstündlich mit folgendem Cronjob, welcher die Ausgaben des Skripts in einer Log-Datei mitschreibt, ausgeführt:

```
12,27,42,57 * * * * php5 /var/www/DrPortscan/service/scpCron.php >>
/opt/DrPortscan/log/scpCron.log
```

5.8 E-Mail-Berichterstattung

Dieses Kapitel befasst sich mit der Konfiguration und dem Report-Versand der E-Mail-Berichterstattung. Bei der Konfiguration wird die Bedieneroberfläche für den User anhand einiger Screenshots der Website erläutert. Der Report-Versand behandelt im Anschluss die Implementierung des Cronjobs, der die Berichte erstellt und versendet.

5.8.1 Konfiguration

Befindet sich der Netzverantwortliche im Bereich „Konfiguration - Berichterstattung“ des Webfrontends, wird er die Website aus Abbildung 5.3 vorfinden, auf der allerdings seine eigenen E-Mail-Empfänger angezeigt werden.

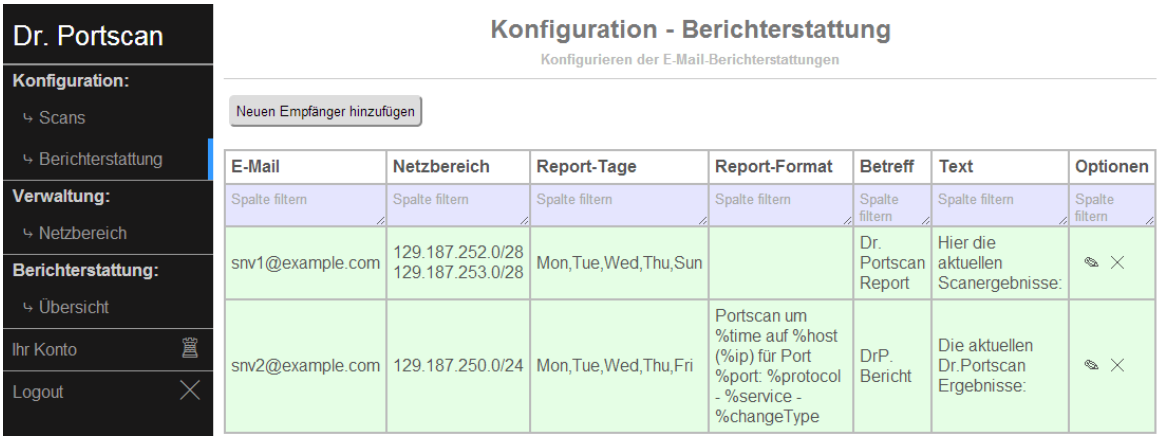

Abbildung 5.3: Startseite des Bereichs „Konfiguration - Berichterstattung“ des Webfrontends.

Auf dieser Seite sind alle E-Mail-Empfänger des jeweiligen Netzverantwortlichen mit allen eingestellten Details in einer Tabelle dargestellt. Hat der Netzverantwortliche viele Empfänger eingestellt und will einen bestimmten finden, um beispielsweise dessen geänderte E-Mail-Adresse einzutragen, kann die selbe Filterfunktion wie aus Kapitel 5.6 genutzt werden. Dazu muss nur die gesuchte Zeichenkette in die entsprechende Spalte der blauen Zeile eingegeben werden. Für jeden Empfänger stehen zwei Optionen bereit. Der Netzverantwortliche kann die Empfänger bearbeiten oder löschen. Soll ein neuer Empfänger hinzugefügt werden, ist der Button `Neuen Empfänger hinzufügen` zu betätigen. Klickt der Netzverantwortliche auf diesen Button, erhält er die in Abbildung 5.4 abgebildete Ansicht.

Dort trägt der Netzverantwortliche alle Details zum Versand ein. Es werden unter anderem die Wochentage und Uhrzeit des Versands, die Netzbereiche für die der Bericht erstellt werden soll und Details zur Formatierung der E-Mail angegeben. Nachdem der Netzverantwortliche auf den Button `Hinzufügen` geklickt hat, wird er zurück zur Übersicht weitergeleitet in der nun bereits der neue Empfänger enthalten ist, siehe Abbildung 5.5. Außerdem wird eine Meldung am oberen Rand angezeigt, dass alles wie gewünscht abgelaufen ist und der Empfänger hinzugefügt werden konnte. Auch hier findet wieder eine Überprüfung der angegebenen Netzbereiche statt. Sind diese außerhalb des Verantwortungsbereichs des Netzverantwortlichen, wird der Empfänger nicht hinzugefügt und eine Fehlermeldung ausgegeben.

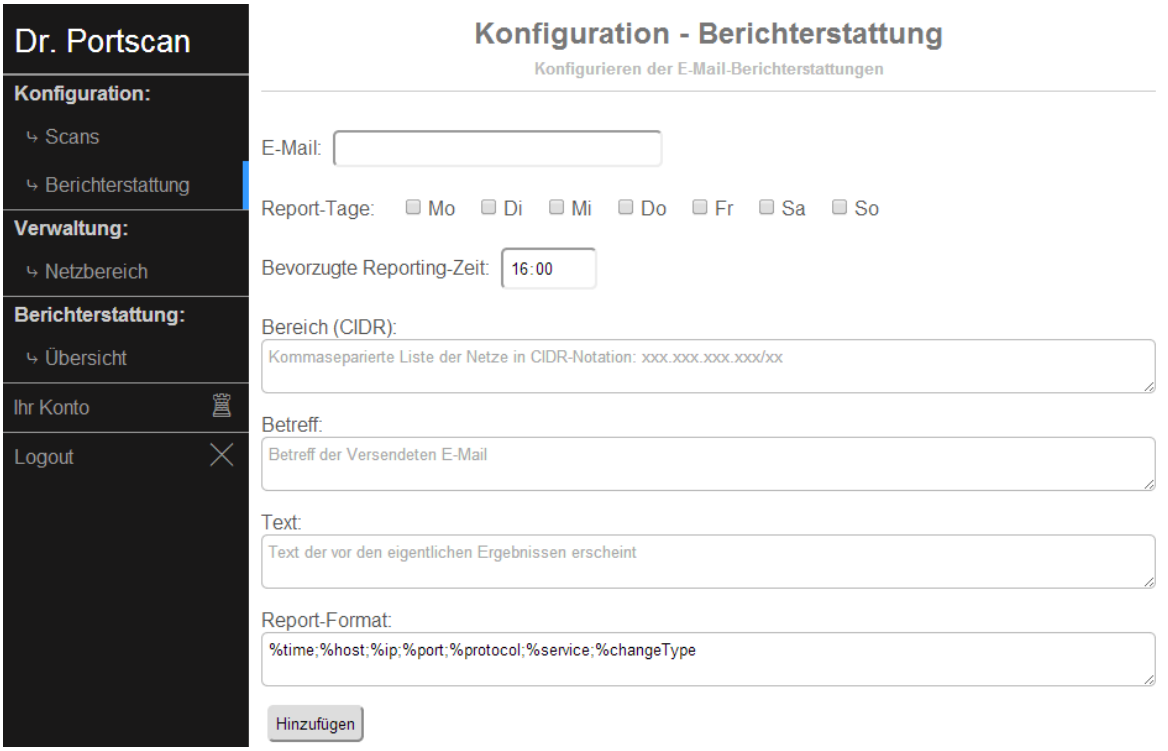

Abbildung 5.4: Website auf der Netzverantwortliche neue E-Mail-Empfänger für die Berichterstattung hinzufügen können.

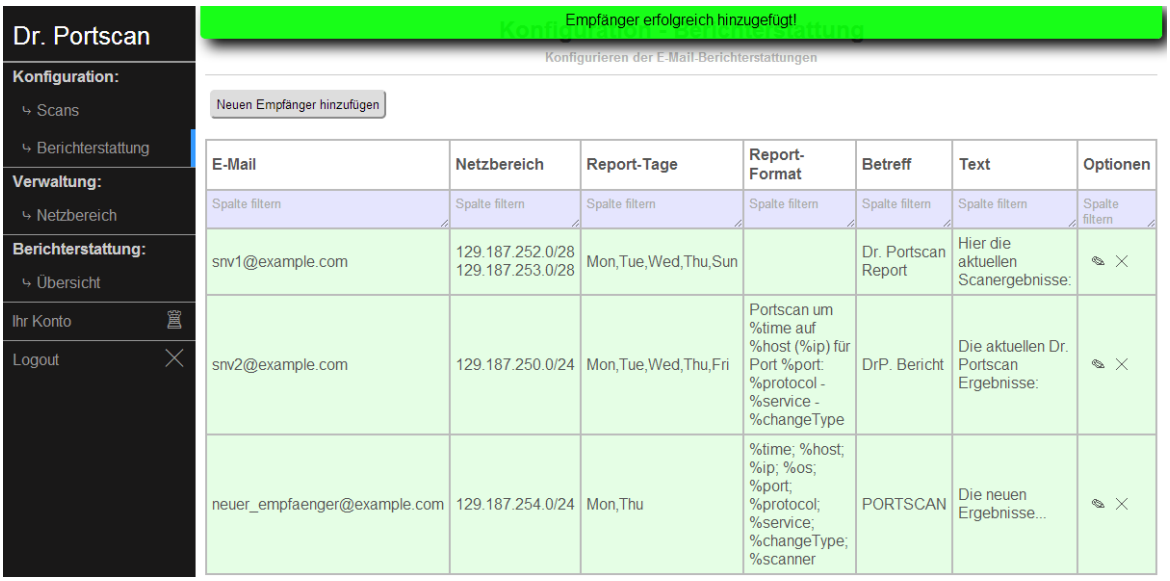

Abbildung 5.5: Startseite des Bereichs „Konfiguration - Berichterstattung“ des Webfrontends direkt nach Hinzufügen eines neuen Empfängers.

5.8.2 Report-Versand

Nun wird erläutert, wie der E-Mail-Versand implementiert ist. Dazu wird ein Cronjob verwendet, der grundsätzlich genau so aufgebaut ist wie der zum Verteilen der Nmap-Aufrufe, siehe Kapitel 5.7.1. Dieser findet auf die gleiche Art und Weise heraus, ob zum Ausführungszeitpunkt des Skripts ein Report für den betrachteten Empfänger verschickt werden soll oder nicht. Ist der Empfänger an der Reihe, werden die Scan-Ergebnisse für den eingestellten Netzbereich des Reports aus der Datenbank geholt. Danach folgt die eigentliche Zusammenstellung der zu versendenden E-Mail. Diese Funktionalität wird durch die Funktion `sendMail()` implementiert, welche im folgenden Quellcode ersichtlich ist:

```
1  private function sendMail(&$recipient, &$hosts)
2  {
3    $message = "<html><head><title>Dr. Portscan Report</title></head><body>".
         $recipient['text']."<br /><br />";
4    $csv = '"time";"host";"ip";"port";"protocol";"service";"changeType";"
         changeDescription"'."\r";
5    $csvFormat = '"%time";"%host";"%ip";"%port";"%protocol";"%service";"%changeType
         ";"%changeDescription"';
6    foreach($hosts as $host)
7    {
8      foreach($host['ports'] as $port)
9      {
10       if(!empty($recipient['format']))
11       {
12         $tmpMsg = $recipient['format'];
13         $tmpMsg = str_replace('%time', date('Y-m-d H:i:s', $port['timestamp']),
               $tmpMsg);
14         $tmpMsg = str_replace('%host', $host['dnsname'], $tmpMsg);
15         $tmpMsg = str_replace('%ip', $host['ipaddr'], $tmpMsg);
16         $tmpMsg = str_replace('%port', $port['port'], $tmpMsg);
17         $tmpMsg = str_replace('%protocol', $port['protocol'], $tmpMsg);
18         $tmpMsg = str_replace('%service', $port['service'], $tmpMsg);
19         $tmpMsg = str_replace('%changeType', $port['changeDescription'], $tmpMsg)
               ;
20
21         $message.= $tmpMsg."<br />";
22       }
23
24       $tmpCsv = $csvFormat;
25       $tmpCsv = str_replace('%time', date('Y-m-d H:i:s', $port['timestamp']),
             $tmpCsv);
26       $tmpCsv = str_replace('%host', $host['dnsname'], $tmpCsv);
27       $tmpCsv = str_replace('%ip', $host['ipaddr'], $tmpCsv);
28       $tmpCsv = str_replace('%port', $port['port'], $tmpCsv);
29       $tmpCsv = str_replace('%protocol', $port['protocol'], $tmpCsv);
30       $tmpCsv = str_replace('%service', $port['service'], $tmpCsv);
31       $tmpCsv = str_replace('%changeType', $port['changeType'], $tmpCsv);
32       $tmpCsv = str_replace('%changeDescription', $port['changeDescription'],
             $tmpCsv);
33
34       $csv.= $tmpCsv."\r";
35     }
36   }
37   $message.= "</body></html>";
38
39   $csv = chunk_split(base64_encode($csv));
40   $uid = md5(uniqid(time()));
41   $header= "From: DrPortscan <do-not-reply@drpotscan.de>" . "\r\n";
42   $header .= "MIME-Version: 1.0\r\n";
43   $header .= "Content-Type: multipart/mixed; boundary=\"".$uid."\"\r\n\r\n";
44   $header .= "This is a multi-part message in MIME format.\r\n";
45   $header .= "--".$uid."\r\n";
46   $header .= "Content-type:text/html; charset=utf-8\r\n";
```

```
47    $header .= "Content-Transfer-Encoding: 7bit\r\n\r\n";
48    $header .= $message."\r\n\r\n";
49    $header .= "--".$uid."\r\n";
50    $header .= "Content-Type: text/csv; name=\"".drPortscanReport_".date('d_m_Y').
          ".csv"."\"\r\n";
51    $header .= "Content-Transfer-Encoding: base64\r\n";
52    $header .= "Content-Disposition: attachment; filename=\"".drPortscanReport_".
          date('d_m_Y').".csv"."\"\r\n\r\n";
53    $header .= $csv."\r\n\r\n";
54    $header .= "--".$uid."--";
55
56    $subject = $recipient['subject'];
57
58    if(mail($recipient['email'], $subject, $message, $header))
59    {
60      echo date('Y-m-d H:i:s').' - Report gesendet an: '.$recipient['email'].
            PHP_EOL;
61      $this->recipients->updateLastReportTime($recipient['recipientId']);
62    }else
63    {
64      echo date('Y-m-d H:i:s').' - Report fehlgeschlagen für: '.$recipient['email'
            ].PHP_EOL;
65    }
66  }
```

Quellcode 5.20: Funktion zum Generieren und Versenden der E-Mail-Berichte.

Der Funktion wird der betroffene Empfänger in der Variable `$recipient` und die betroffenen Scan-Ergebnisse in der Variable `$hosts` übergeben. Die E-Mail wird als HTML-Mail versendet, weshalb die aufgebaute Nachricht mit HTML-Code beginnt in dem auch gleich der vom Netzverantwortlichen konfigurierte Text steht (Zeile 3). Danach wird die Kopfzeile der CSV-Datei[8] und die Formatierung der einzelnen CSV-Zeilen definiert. Jeder E-Mail-Bericht enthält nämlich einen CSV-Anhang, da dieses Datei-Format von vielen Programmen unterstützt wird und viele Möglichkeiten der Weiterverarbeitung der Daten bietet. Anschließend werden die Portscan-Ergebnisse in zwei Schleifen durchgegangen und so der E-Mail-Text (Zeile 12ff) sowie der CSV-Text (Zeile 24ff) generiert. Nun wird der Header nach RFC 2045[9] erstellt, welche den Aufbau von E-Mails mit Anhang erläutert. Die angehängte CSV-Datei wird stets nach dem Format „`drPortscanReport_TAG_MONAT_JAHR`“ benannt, wobei `TAG_MONAT_JAHR` mit dem aktuellen Datum gefüllt wird. Schlussendlich wird die E-Mail versendet (Zeile 58). Kann diese erfolgreich verschickt werden, wird dies in einer Log-Datei festgehalten und die *`lastReportTime`* in der entsprechende Eintrag in der Datenbank-Tabelle „`recipients`“ mit dem aktuellen Unix-Timestamp versehen. Treten Probleme auf wird dies ebenfalls in der Log-Datei gespeichert.

5.9 Datenbank und Stylesheets

Damit auch die kleineren Teilebereiche der Implementierung Erwähnung finden, werden nun einige Anmerkungen zu den Bereichen Datenbank und Stylesheets gemacht.

Die Datenbank wurde wie im Systementwurf beschrieben implementiert. Die einzige Ausnahme stellt die Tabelle „`scans`“ dar, welche Details zu den eingestellten Scans enthält. Dort wurde das Attribut *`interval`* durch das Attribut *`days`* ersetzt. Außerdem wurde das Attribut *`execute`* hinzugefügt, welches den zu tätigen Scan-Aufruf für Nmap enthält. Die

[8] Eine CSV-Datei enthält die Daten einer Tabelle in reinem Textformat.
[9] `http://www.ietf.org/rfc/rfc2045.txt`

Datenbankstruktur sieht noch weitere, nicht im Entwurf vorgestellte Elemente vor. Das sind Beziehungen zwischen verschiedenen Tabellen. Einmal wird die Tabelle „`scans`“ über die *`userId`* und einmal die Tabelle „`recipients`“ ebenfalls über die *`userId`* mit der Tabelle „`user`“ verbunden. Dadurch wird vermieden, dass die Datenbank unnötig alte Datensätze behält. Löscht nun beispielsweise der Dr.-Portscan-Administrator einen Netzverantwortlichen, so werden auch dessen eingestellte Scans und E-Mail-Empfänger gelöscht.

Die eingesetzten Stylesheets dienen lediglich der Verschönerung des Webfrontends. Auf eine Erläuterung und Beschreibung dieser, wurde Aufgrund der Austauschbarkeit und nicht vorhandener Relevanz für die Programmlogik verzichtet. Nichts desto trotz soll der Einsatz kurz erläutert werden. Es werden zwei CSS-Stylesheets eingesetzt. Das eine ist für die Darstellung der `MasterView` und somit des Navigationsmenüs und der Überschriften zuständig. Das andere enthält alle Style-Regeln für die Darstellung weiterer, auf den eigentlichen Inhalt bezogener Elemente.

5.10 Sicherheit

Dieses Kapitel befasst sich mit zwei sicherheitsrelevanten Themen. Einmal wird die Zugriffskontrolle des Webservers auf das Webfrontend erklärt und einmal die Umsetzung der SSH-Verbindungen, mit denen der Dr.-Portscan-Server mit den Datenquellen kommuniziert.

5.10.1 Zugriffskontrolle

User sollen über den Webserver nur Zugriff auf den Front-Controller und die Web-Komponenten erhalten. Dazu wurde eine .htaccess-Datei eingerichtet. Um solche Dateien für die Zugriffskontrolle bestimmter Verzeichnisse einsetzen zu können, muss die Konfigurationsdatei des Apache-Webservers wie folgt angepasst werden:

```
<Directory /var/www/DrPortscan/>
        AllowOverride All
</Directory>
```

Dadurch kann nun im Verzeichnis */var/www/DrPortscan/* eine .htaccess-Datei eingesetzt werden, die folgendermaßen aussieht:

```
RewriteEngine on
RewriteRule .*controller/.* - [F]
RewriteRule .*model/.* - [F]
RewriteRule .*service/.* - [F]
RewriteRule .*settings/.* - [F]
RewriteRule .*view/.* - [F]
```

Durch das Einschalten der sogenannten *RewriteEngine* können nun Regeln für die Zugriffskontrolle erstellt werden. Will der User eine Seite aufrufen, auf die eine der hinterlegten Regeln anwendbar ist, werden bestimmte Aktionen durchgeführt. Es können beispielsweise alle HTTP-Anfragen umgeleitet werden. Die implementierte .htaccess-Datei sorgt dafür, dass Nutzer, die auf die Verzeichnisse *./controller/*, *./model/*, *./service/*, *./settings/* oder *./view/* des Dr.-Portscan-Webfrontends zugreifen wollen, auf eine Fehlerseite (HTTP-Statuscode 403)

weitergeleitet werden. So werden unerwünschte Aufrufe auf die Unterverzeichnisse des Dr.-Portscan-Webfrontends verhindert. Ohne diese Zugriffskontrolle könnte ein Angreifer z. B. das Skript *./service/cron.php* ausführen und so eventuelle Fehlermeldungen auslesen, die ihm wiederum Einblick in die Konfiguration des Servers liefern könnten.

5.10.2 SSH-Verbindungen

Alle SSH-Verbindungen werden stets vom Dr.-Portscan-Server aus aufgebaut und niemals von den Datenquellen, da besonders externe Datenquellen nicht vom LRZ betrieben werden und somit die Sicherheitsstandards unbekannt sind. Die Authentifizierung erfolgt immer per Public-Key-Verfahren. Dabei ist anzumerken, dass keine Passphrase für den Private-Key verwendet wird. Die Verwendung einer Passphrase für den Private-Key ist zwar sehr zu empfehlen, allerdings in dem Szenario von Dr. Portscan nicht von Vorteil. Damit der Dr.-Portscan-Server die Verbindungen automatisch aufbauen kann, müsste die Passphrase irgendwo, auf dem Server oder für den Server erreichbar, abgelegt sein. Hat ein Angreifer den Private-Key gefunden, was voraussetzt das er Zugriff zum Server hat, ist es für ihn nur eine Frage der Zeit, bis er den entsprechenden Codeabschnitt findet, in dem die Passphrase hinterlegt ist. Deshalb wird bei allen Authentifizierungen mit nicht passphrase-geschützten Private-Keys gearbeitet.

5.11 Abgleich mit der Anforderungsanalyse

Innerhalb dieses Abschnitts wird die erfolgte Implementierung mit den Anforderungen der Anforderungsanalyse verglichen. Dazu werden die Punkte erwähnt, bei denen Änderungen im Vergleich zur anfänglichen Anforderung vorgenommen wurden und solche, die gar nicht erfüllt wurden. Somit sind alle Anforderungen erfüllt, die hier nicht erwähnt werden.

Der erste nicht gänzlich erfüllte Anforderung betrifft die Konfiguration der Scans durch die Netzverantwortlichen. Aus Fall 5 resultierte die Anforderung Netzbereiche gezielt definieren und Ausnahmelisten erstellen zu können. Der Netzbereich kann für die Netzbereiche gezielt definiert werden, allerdings ohne Ausnahmelisten. Ausnahmelisten sind in der Implementierung nur vom Dr.-Portscan-Administrator zu erstellen. Will ein Netzverantwortlicher einen bestimmten Netzbereich nicht scannen, muss er den eingestellten Scan-Bereich lediglich genauer definieren. Dadurch ist der gleiche Effekt wie mit Ausnahmelisten zu erzielen. Netzbereiche die auf gar keinen Fall gescannt werden sollen, also auch nicht in der Zukunft, können dem Dr.-Portscan-Administrator mitgeteilt werden. Dieser kann den Netzbereich daraufhin in seine globale Ausnahmeliste aufnehmen.

Die nächste Änderung betrifft ebenfalls die Kategorie Scan-Konfiguration. Aus Fall 10 resultierte das Definieren von Scan-Intervallen wie täglich und wöchentlich. Da diese Scan-Intervalle den eigentlichen Ausführungszeitpunkt der Scans nicht sonderlich transparent für die Netzverantwortlichen machen, wurde auf Wochentage umgestellt. Die Netzverantwortlichen können nun exakt die Wochentage definieren, zu denen ihre Scans ausgeführt werden sollen.

Der nächste Punkt betrifft Fall 6 und somit das Scannen einzelner IPv6-Endsysteme. Diese Anforderung wurde aus folgendem Grund nicht implementiert: Das Scannen einzelner IPv6-Systeme würde eine Übergangslösung darstellen. Die vergebenen IPv6-Adressbereiche für die Netzverantwortlichen sind sehr groß und können leicht mehrere Millionen mögliche Endsysteme beherbergen. Diese können nicht in angemessener Zeit mithilfe eines Portscanners

gescannt werden. Dazu müsste ein komplett neues Konzept entworfen werden. Nun werden nach und nach immer mehr Systeme auf IPv6 umgestellt. Diese alle in die Scan-Listen einzutragen stellt keine zufriedenstellende Lösung dar, weshalb gänzlich darauf verzichtet wurde und somit keine IPv6-Unterstützung implementiert wurde.

Die Anforderung aus Fall 15, der automatische Datenbank-Import der Netzdoku-Datenbank und die damit verbundene externe Authentifizierung, konnte nicht implementiert werden, wie bereits in Kapitel 5.4 erläutert.

In der Anforderung aus Fall 18 wurde eine Übersicht der Scans mit ungefährer Scan-Dauer für den Dr.-Portscan-Administrator gefordert. Die Übersicht der Scans wurde implementiert, jedoch ohne zusätzliche Information zur ungefähren Scan-Dauer. Diese zu bestimmen hängt von sehr vielen Parametern ab. Insbesondere die parallele Ausführung der Scans kann zu einer stark variierenden Scan-Dauer führen, weshalb diese zusätzliche Information nicht in das Webfrontend integriert wurde.

Alle weiteren Anforderungen wurden, wie in der Anforderungsanalyse beschrieben, implementiert. In Abbildung 5.6 ist eine Übersicht der Anforderungen abgebildet. Grün markierte Anforderungen wurden erfüllt, blaue wurden etwas abgeändert erfüllt und rote wurden nicht erfüllt.

Nutzer	Kategorie	Unterkategorie	Prio 1				Prio 2			Prio 3	
Netzverantwortliche	Konfiguration	Scans	1	2	5	7	10	8		6	9
		Berichterstattung								3	4
	Berichterstattung		11	12							
	Verwaltung		13				14				
DPA	Verwaltung		15	16	19		17	18	20		
	Konfiguration									21	sonstige

Abbildung 5.6: Vergleich der Anforderungsanalyse mit dem Zustand nach der Implementierung.

6 Inbetriebnahme und Verwendung

Dieses Kapitel behandelt die Inbetriebnahme des Webfrontends. Es wird erläutert, welche Schritte befolgt werden müssen, damit ein reibungsloser Einsatz gewährleistet werden kann. Im Anschluss wird ein kleines Szenario durchgespielt. Dabei fügt der Dr.-Portscan-Administrator einen neuen Netzverantwortlichen hinzu, welcher daraufhin weitere Subnetzverantwortliche hinzufügt. Diese stellen jeweils einen Scan ein. Die Ergebnisse werden dann mithilfe des Webfrontends eingesehen und eine E-Mail-Berichterstattung eingerichtet. Dieses Schritte werden mit einer Vielzahl an Screenshots des eigentlichen Webfrontends bebildert um einen Eindruck der Verwendung zu vermitteln.

6.1 Inbetriebnahme

Zur Inbetriebnahme des Webfrontends für Dr. Portscan gehören die Anpassungen von Dr. Portscan, die Umstellung auf MySQL, die Einrichtung der nötigen SSH-Verbindungen sowie der Cronjobs. Die bereits in Kapitel 5.1 aufgeführten Vorbereitungen, wie z. B. PHP, Perl, PuTTY oder phpMyAdmin werden im Folgenden vorausgesetzt.

Zuerst werden die zwei Pakete für Dr. Portscan und das Webfrontend benötigt. Dr. Portscan wurde bei der Implementierung im Verzeichnis */opt/DrPortscan/* entpackt, es kann jedoch auch ein beliebiges anderes Verzeichnis gewählt werden. Danach wird die im Paket enthaltene Datei *setup.pl* ausgeführt, welche unter anderem die Verzeichnisse für die Scans anlegt. Anschließend wird das Webfrontend in einem Apache Webordner entpackt. Bei der durchgeführten Implementierung war dies */var/www/DrPortscan/*. Danach wird, wie in Kapitel 5.10.1 beschrieben, das Verwenden der .htaccess-Datei erlaubt.

6.1.1 Datenbank

Als nächstes wird die MySQL Datenbank angelegt. Dazu wird phpMyAdmin verwendet. Beim Anlegen der Datenbank wird die Kollation „*utf8_general_ci*“ gewählt und ein beliebiger Name für die Dr.-Portscan-Datenbank gewählt. Anschließend wird ein neuer Benutzer samt Passwort hinzugefügt, dem alle Rechte für die Dr.-Portscan-Datenbank gewährt werden. Diesem Benutzer wird der Zugriff ausschließlich vom sogenannten *localhost*[1] erlaubt, da alle Skripte, die eine Datenbankverbindung benötigen, auf dem gleichen Server laufen, auf dem auch Dr. Portscan liegt. Nun wird der neu erstellte Benutzer im Skript *./settings/conf.php* des Webfrontends eingetragen. Diese Datei ist wie folgt aufgebaut:

```
define('HOST','localhost');
define('USER','Name_des_Benutzers');
define('PASSWORD','Passwort');
define('DATABASE','Name_der_Datenbank');
```

[1] In der Netzwerktechnik bezeichnet der localhost den aktuell verwendeten Host, also die IP-Adresse des Rechners, auf dem gerade gearbeitet wird.

Die Variablen `USER`, `PASSWORD` und `DATABASE` müssen nun mit den zuvor gewählten Werten gefüllt werden. Um das Datenbankschema bei der Installation nicht manuell eintragen zu müssen, wurde das Skript *setupDb.php* erstellt, welches sich im Verzeichnis *./service/* befindet. Dieses nutzt die in den Models hinterlegten Tabellenschemata, um die Datenbankstruktur aufzubauen. Die Ausprägung der Tabelle „`changeIds`“ wird ebenfalls importiert. Mit dem Aufruf `php5 setupDb.php` ist die Einrichtung der MySQL-Datenbank abgeschlossen.

Der nächste Schritt ist die Anpassung der Dr.-Portscan-Skripte. Da hier zuvor eine SQLite-Datenbank verwendet wurde, muss nun auf MySQL umgestellt werden. Die betroffenen Skripte sind *delta-reporter.pl* und *input-watcher.pl*. In diesen Skripten, muss folgende Zeile gefunden werden:

```
my $dbh = DBI->connect("dbi:SQLite:dbname=".$sqlite_db_file,"","");
```

Diese Zeile wird dann mit einer „#“ am Anfang der Zeile auskommentiert und darunter folgende Zeile, mit den Nutzer- und Datenbankinformationen für MySQL, hinzugefügt:

```
my $dbh = DBI->connect("DBI:mysql:Name_der_Datenbank:host=localhost:3306",
"Name_des_Benutzers","Passwort");
```

6.1.2 Datenquellen

Nun werden die Datenquellen eingerichtet. Auf jeder Datenquelle muss das Skript *getFinishedScans.sh* vorhanden sein. Es ist im Verzeichnis *./service/* des Webfrontends zu finden und wurde bei der Implementierung im Verzeichnis */opt/DrPortscan/* auf der Datenquelle untergebracht. Im selben Verzeichnis wurde ein Ordner *./scans/* erstellt, in dem dann die neuen Scan-Ergebnisse zwischengespeichert werden.

Damit vom Dr.-Portscan-Server später die SSH-Verbindungen zu den Datenquellen hergestellt werden können, muss ein SSH-Schlüsselpaar generiert werden. Das wird mithilfe von PuTTY und dem Befehl `ssh-keygen -b 2048 -t rsa` gemacht. Will man eine längere Schlüssellänge kann der Wert des Parameters `-b` variiert werden. Eine kürzere Schlüssellänge ist nicht zu empfehlen. Dieses Schlüsselpaar wird nach Ausführung des Befehls (als User *root*) im Verzeichnis */root/.ssh/* abgelegt. In der Datei *id_rsa* befindet sich der Private-Key und in *id_rsa.pub* der Public-Key. Letzterer muss nun noch zu den autorisierten Keys auf den Datenquellen hinzugefügt werden. Dazu wird auf der Datenquelle ins Verzeichnis */root/.ssh/* navigiert (greift man als User *root* zu). Dort befindet sich die Datei *authorized_keys*. In dieser wird eine neue Zeile mit dem vorher generierten Public-Key hinzugefügt.

6.1.3 Scanner

In drei Schritten werden als nächstes die Scanner auf dem Dr.-Portscan-Server eingerichtet. Dazu werden diese anfangs per phpMyAdmin in die Datenbank-Tabelle „`scanners`“ eingetragen. Im nächsten Schritt wird das Skript *./service/cron.php* angepasst. Hier müssen die Scanner in die Klassenvariable `$scannerConnections` mit allen nötigen Details eingetragen werden. Das gleiche muss für das Skript *./service/scpCron.php* erfolgen.

Anschließend wird, ebenfalls mit phpMyAdmin, ein Administrator in die Tabelle „`user`“ eingetragen. Dabei wird das Feld *`admin`* auf 1 gesetzt. Als *`cidrRange`* wird 0.0.0.0 eingetragen. Um den Passwort-Hash für das Feld *`admin`* zu erzeugen, wird der Befehl `echo -n PASSWORT | sha256sum` per PuTTY ausgeführt.

6.1.4 Cronjobs

Als nächstes werden die Cronjobs auf dem Dr.-Portscan-Server eingerichtet. Da die Cronjobs alle Ausgaben der Skripte in Log-Dateien festhalten sollen, wird noch ein geeignetes Verzeichnis benötigt. Bei der Implementierung wurde dazu ein Ordner *./log/* im Dr.-Portscan-Verzeichnis unter */opt/DrPortscan/* erstellt. Zum Einrichten der Cronjobs wird nun mithilfe von PuTTY auf den Server zugegriffen. Mit dem Befehl `crontab -e` kann der Crontab bearbeitet werden. Die folgenden Cronjobs werden nun, unter Berücksichtigung der eventuell anders gewählten Verzeichnisse, in den Crontab eingetragen:

```
*/15 * * * * php5 /var/www/DrPortscan/service/cron.php >>
/opt/DrPortscan/log/cron.log
5,20,35,55 * * * * cd /opt/DrPortscan/ && perl input-watcher.pl >>
/opt/DrPortscan/log/input-watcher.log
12,27,42,57 * * * * php5 /var/www/DrPortscan/service/scpCron.php >>
/opt/DrPortscan/log/scpCron.log
*/15 * * * * php5 /var/www/DrPortscan/service/recipientCron.php >>
/opt/DrPortscan/log/recipientCron.log
```

6.1.5 Checkliste

Zur Überprüfung der Schritte kann die Checkliste in Abbildung 6.1 verwendet werden. Wurden alles korrekt konfiguriert, kann das Webfrontend nun vom Administrator mit der angegebenen E-Mail-Adresse und dem gewählten Passwort benutzt werden.

1. Pakete für Dr. Portscan und das Webfrontend in Zielverzeichnisse entpacken und .htaccess-Datei für den Apache Webordner erlauben.	☑
2. MySQL-Datenbank anlegen, *./settings/conf.php* anpassen und danach *./service/setupDb.php* ausführen.	☑
3. Pakete für Dr. Portscan und das Webfrontend in Zielverzeichnisse entpacken und .htaccess-Datei für den Apache Webordner erlauben.	☑
4. Anpassung der Dr.-Portscan Skripte für MySQL.	☑
5. Datenquellen einrichten. Dazu das Shell-Skript *getFinishedScans.sh* auf die Datenquellen kopieren und eine Verzeichnis für fertige Scans erstellen.	☑
6. SSH-Schlüsselpaar erstellen und den Public-Key auf den Datenquellen hinterlegen.	☑
7. Die Scanner (Datenquellen) in die Datenbank und die Skripte *./service/cron.php* sowie *./service/scpCron.php* eintragen.	☑
8. Administrator in die Datenbank-Tabelle „*user*“ eintragen.	☑
9. Die vier Cronjobs auf dem Dr.-Portscan-Server einrichten.	☑

Abbildung 6.1: Checkliste zur Inbetriebnahme des Dr.-Portscan-Webfrontends.

6.2 Verwendung

Nun wird ein kleines Szenario durchlaufen um die Verwendung des Webfrontends zu erläutern. Dazu wurden vom LRZ zwei /24-Netzbereiche bereitgestellt.

6.2.1 Benutzerverwaltung

Begonnen wird mit einem eingeloggten Dr.-Portscan-Administrator, der sich im Bereich „Verwaltung - Mandanten“ befindet, Abbildung 6.2. Mit einem Klick auf den Button `Neuen Benutzer hinzufügen` wird er an die Eingabe-Form weitergeleitet, Abbildung 6.3. Dort kann er einen neuen Netzverantwortlichen einpflegen und ihm so Zugriff zum Webfrontend gewähren. Hier muss er lediglich die E-Mail-Adresse, den Netzbereich in einer kommaseparierten Liste in CIDR-Notation und den Namen des Netzbereichs eingeben. Sind alle Eingaben korrekt, wird er wieder an die Übersicht weitergeleitet, in der nun der neu hinzugefügte Netzverantwortliche zu sehen ist, Abbildung 6.4.

Nach dem Hinzufügen den neuen Nutzers, erhält dieser eine E-Mail mit seinen Zugangsdaten, die wie folgt aussieht:

```
Ihr Dr. Portscan Zugang wurde erstellt. Sie können sich ab sofort mit
folgendem Passwort einloggen:

p4s5_w0rT

Am besten ändern Sie ihr Passwort nach dem ersten Einloggen unter dem
Menüpunkt "Mein Konto"!
```

Nun hat der Netzverantwortliche einen Zugang zum Webfrontend. Hier kann er sich mit seiner E-Mail-Adresse und dem zugesandten Passwort einloggen, Abbildung 6.5. Als nächstes wird, wie in der E-Mail angemerkt, das Passwort geändert. Dazu navigiert der Netzverantwortliche in den Bereich „Mein Konto“, wo er eine kleine Kontoübersicht erhält und die Möglichkeit hat sein Passwort zu ändern. War die Änderung erfolgreich, wird ihm an oberen Bildrand der Webseite eine Infobox mit der positiven Nachricht angezeigt. Nun kann er mit der Verwaltung seines Netzbereichs fortfahren. Dazu navigiert er zum Bereich „Verwaltung - Netzbereich“. Dieser Bereich ist exakt so aufgebaut wie der Bereich „Verwaltung - Mandanten“ des Dr.-Portscan-Administrators. Der Netzbereich des Netzverantwortlichen ist in zwei /24-Netze unterteilt, die wiederum von weiteren Netzverantwortlichen betreut werden. Deshalb fügt er diese zwei Netzverantwortlichen hinzu, damit sie ebenfalls Zugriff auf das Webfrontend erhalten. Nach dem Hinzufügen sind die zwei Subnetzverantwortlichen in seiner Übersicht aufgelistet, Abbildung 6.6. Diesen neuen Usern wird daraufhin ebenfalls eine E-Mail mit den Zugangsdaten zugeschickt.

Abbildung 6.2: Bereich „Verwaltung - Mandanten“ vor Hinzufügen neuer Benutzer.

Abbildung 6.3: Form um neue Netzverantwortliche hinzuzufügen.

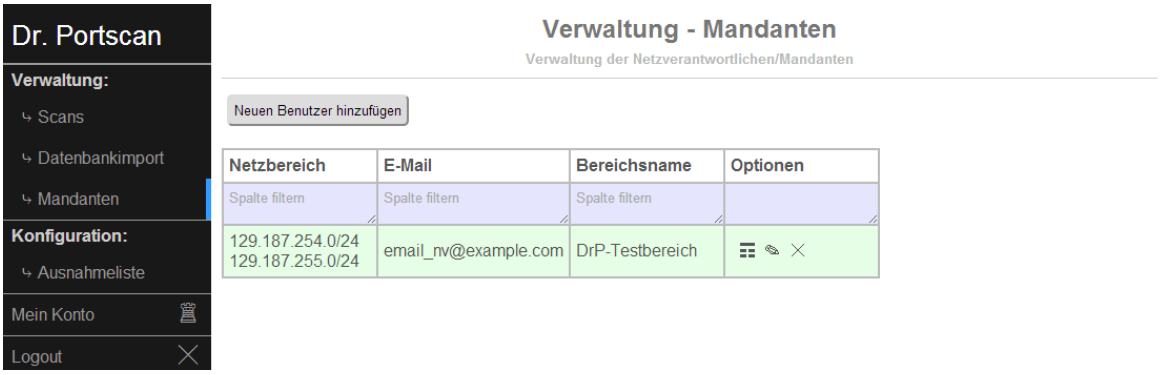

Abbildung 6.4: Bereich „Verwaltung - Mandanten“ nach Hinzufügen neuer Benutzer.

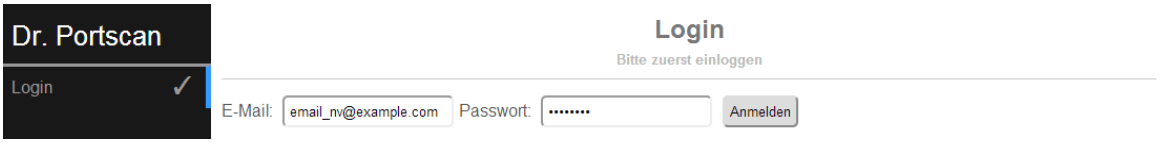

Abbildung 6.5: Login eines Netzverantwortlichen.

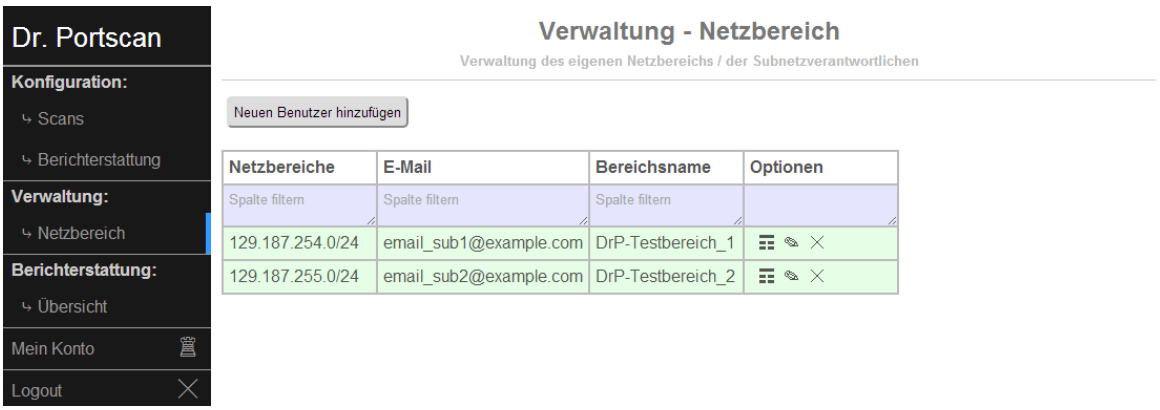

Abbildung 6.6: Übersicht des Netzbereichs eines Netzverantwortlichen im Bereich „Verwaltung - Netzbereich“.

6.2.2 Scan-Konfiguration und Berichterstattung

Nun stellt der Subnetzverantwortliche einen internen Scan für seinen Netzbereich ein. Dazu navigiert er nach dem Login in den Bereich „Konfiguration - Scans“. Dort findet er eine Form, in der er seinen Scan konfigurieren und durch klicken auf den Button `Speichern` speichern kann, Abbildung 6.7. Nun ist alles erledigt damit der Scan zum nächsten zutreffenden Zeitpunkt ausgeführt wird.

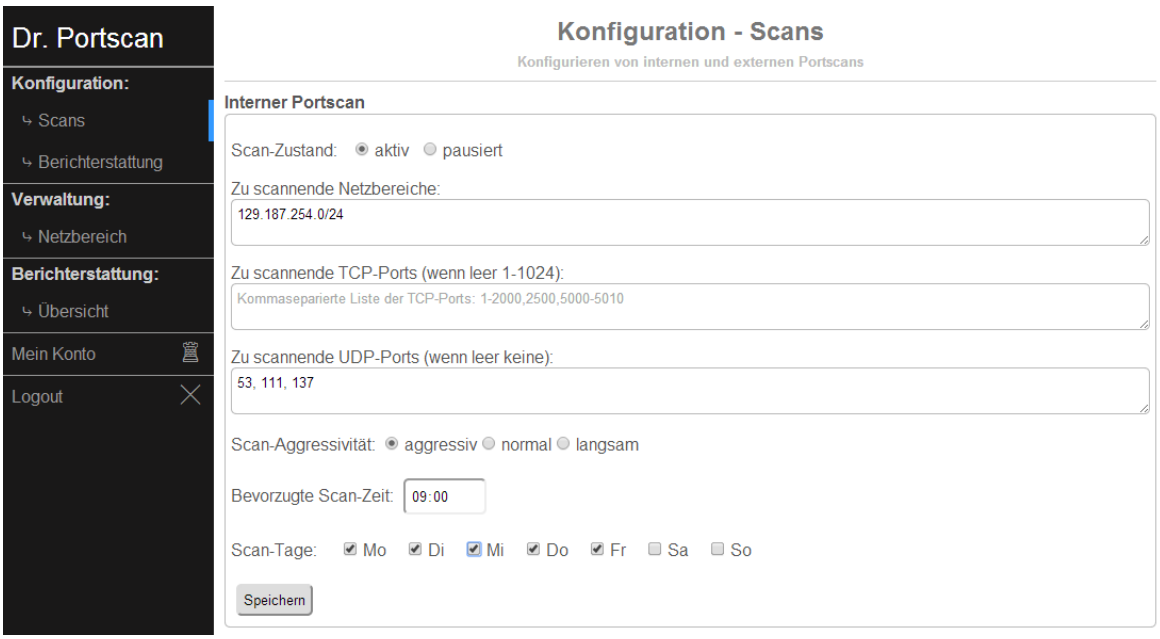

Abbildung 6.7: Bereich „Konfiguration - Scans“ der Netzverantwortlichen zum Hinzufügen neuer Scans.

Nachdem der Scan einige Male ausgeführt wurde will der Netzverantwortliche die Ergebnisse auf dem Webfrontend einsehen und Navigiert dazu zum Bereich „Berichterstattung - Übersicht“. Er bekommt eine Tabelle mit allen Ergebnissen dargestellt, Abbildung 6.8. Oben

rechts bekommt er eine Statistik der vorhandenen Hosts, der registrierten Ports und der aktuellen Änderungen angezeigt. Sind Änderungen an den Ports vorhanden, werden die entsprechenden Zeilen orange, ansonsten grün markiert. Oben links befindet sich die Checkbox, um nur Ports mit Veränderungen anzuzeigen. In der zweiten Spalte kann er für alle Spalten einen Filter setzen. In Abbildung 6.9 sind beispielsweise nur Hosts zu sehen, deren DNS-Namen die Zeichenkette „tum“ enthalten.

Nun will der Netzverantwortliche einem der zuständigen Systemadministratoren regelmäßig die Scan-Ergebnisse per E-Mail zukommen lassen. Dazu wechselt er in den Bereich „Konfiguration - Berichterstattung“ und klickt auf den Button `Neuen Empfänger hinzufügen`. Dort gibt er alle Details für die Berichterstattung ein und klickt auf `Hinzufügen`, Abbildung 6.10. Danach wird er zur Übersicht der E-Mail-Berichterstattung weitergeleitet, wo nun der neu hinzugefügte Empfänger zu sehen ist, Abbildung 6.11.

Jetzt wird auch der zuständige Systemadministrator stets über den Zustand seiner Systeme auf dem Laufenden gehalten. Die E-Mail für den eingestellten Empfänger sieht nun wie in Abbildung 6.12 aus. Jeder E-Mail wird eine CSV-Datei angehängt, die alle aktuellen Ergebnisse enthält, Abbildung 6.13. Somit können mithilfe von Tabellenkalkulationsprogrammen wie Excel weitere Auswertungen erfolgen oder Statistiken gepflegt werden.

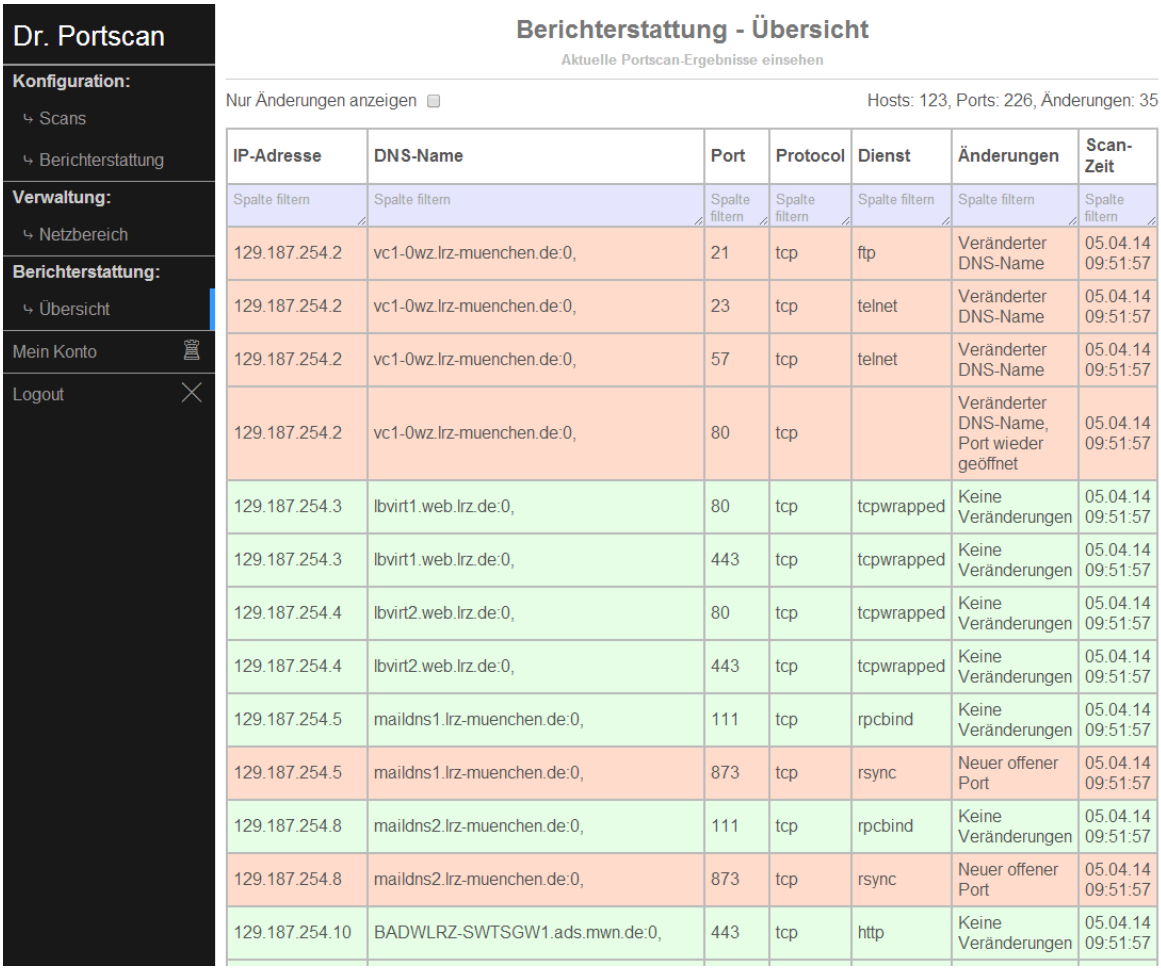

Abbildung 6.8: Einsichtnahme der Scan-Ergebnisse per Webfrontend.

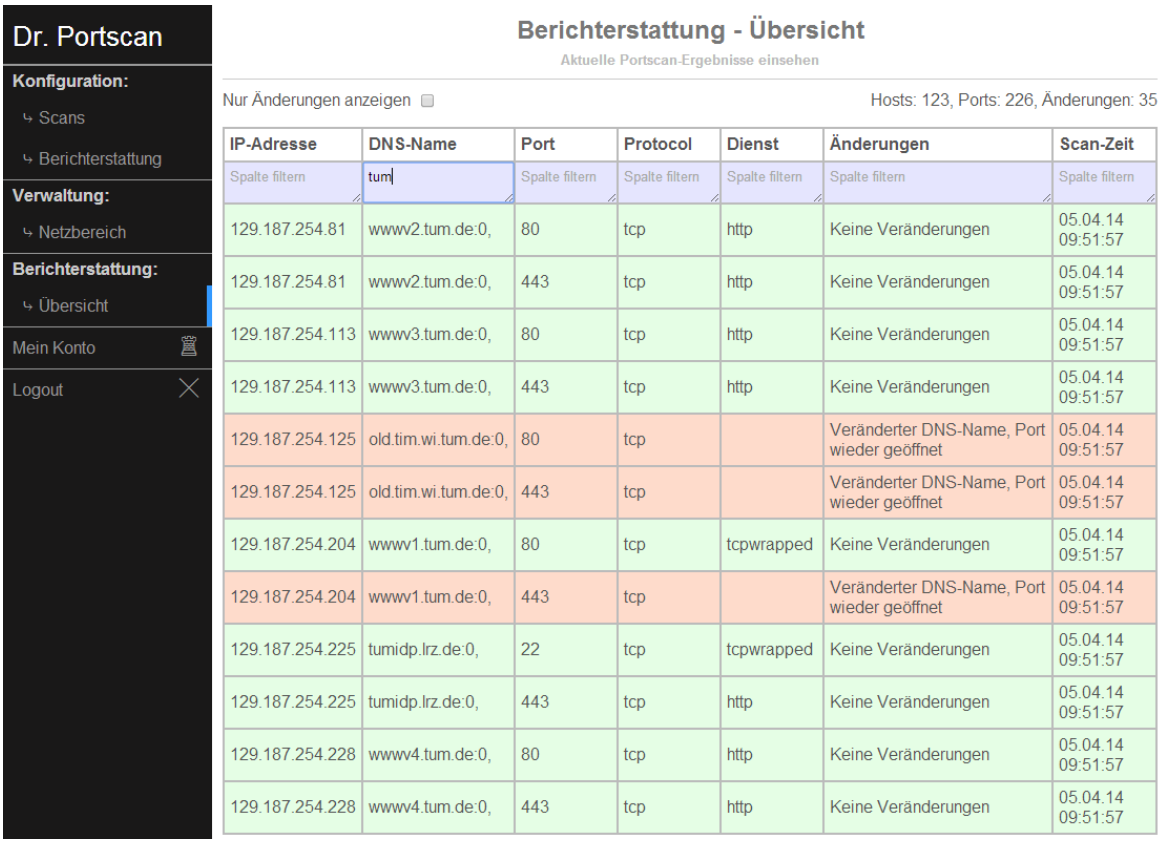

Abbildung 6.9: Filterung der Scan-Ergebnisse auf dem Webfrontend.

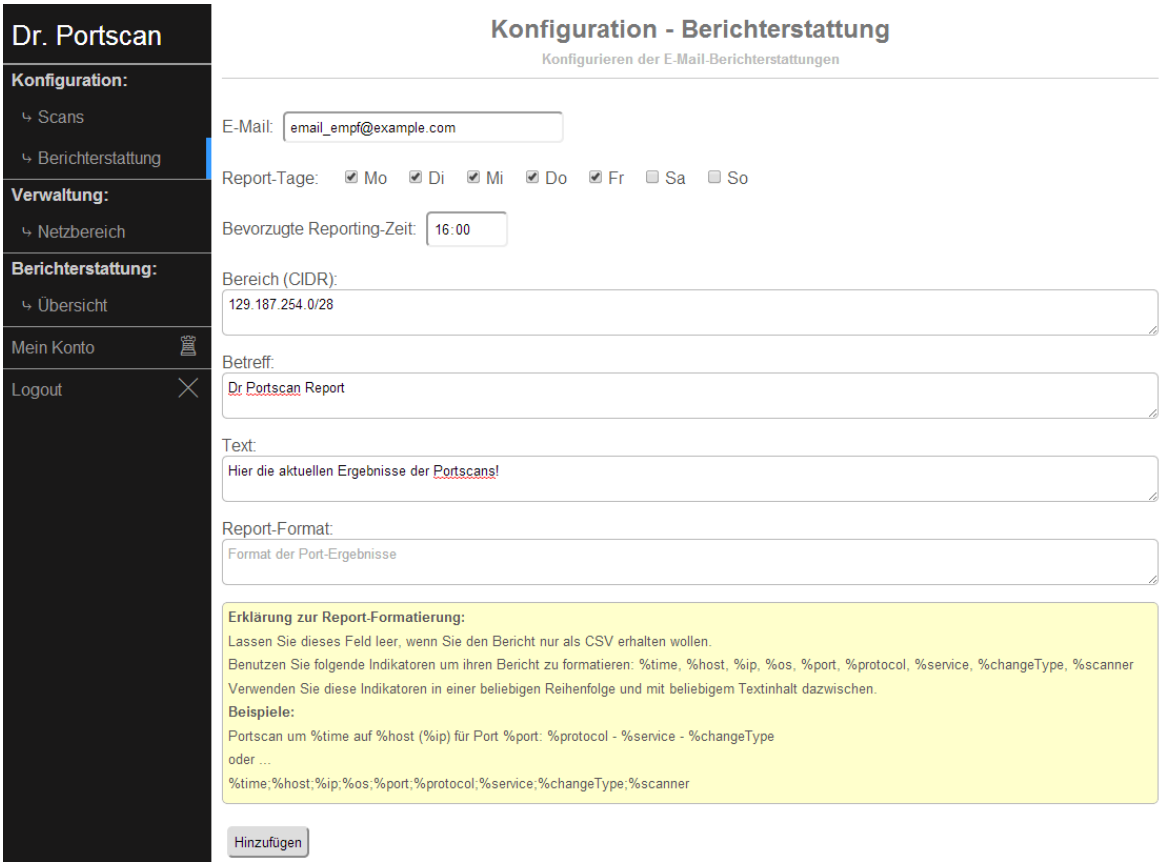

Abbildung 6.10: Hinzufügen eines E-Mail-Empfängers für die E-Mail-Berichterstattung.

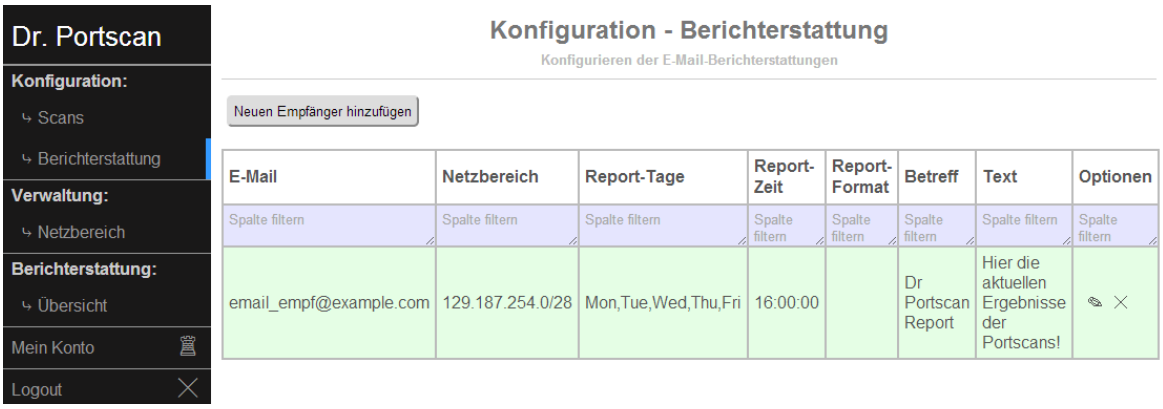

Abbildung 6.11: Übersicht der E-Mail-Empfänger im Bereich „Konfiguration - Berichterstattung“.

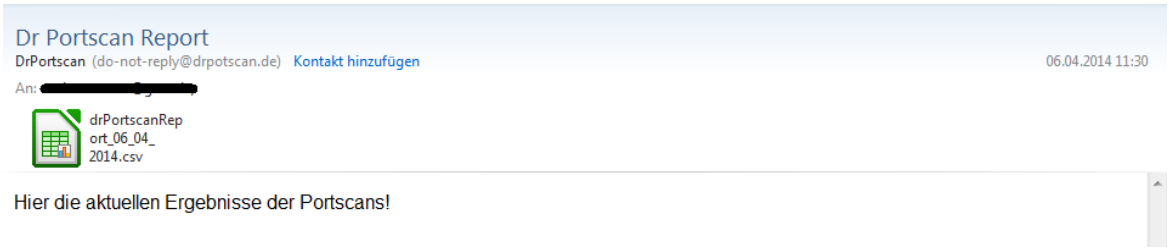

Abbildung 6.12: Versendeter E-Mail-Bericht.

	A	B	C	D	E	F	G	H
1	time	host	ip	port	protocol	service	changeType	changeDescription
2	2014-04-06 09:19:02	vc1-0wz.lrz-muenchen.de:0,	129.187.254.2	21	tcp	ftp	32	Port wieder geöffnet
3	2014-04-06 09:19:02	vc1-0wz.lrz-muenchen.de:0,	129.187.254.2	23	tcp	telnet	8	Neuer offener Port
4	2014-04-06 09:19:02	vc1-0wz.lrz-muenchen.de:0,	129.187.254.2	57	tcp	telnet	8	Neuer offener Port
5	2014-04-06 09:19:02	vc1-0wz.lrz-muenchen.de:0,	129.187.254.2	80	tcp		0	Keine Veränderungen
6	2014-04-06 09:19:02	lbvirt1.web.lrz.de:0,	129.187.254.3	80	tcp	tcpwrapped	0	Keine Veränderungen
7	2014-04-06 09:19:02	lbvirt1.web.lrz.de:0,	129.187.254.3	443	tcp	tcpwrapped	0	Keine Veränderungen
8	2014-04-06 09:19:02	lbvirt2.web.lrz.de:0,	129.187.254.4	80	tcp	tcpwrapped	0	Keine Veränderungen
9	2014-04-06 09:19:02	lbvirt2.web.lrz.de:0,	129.187.254.4	443	tcp	tcpwrapped	0	Keine Veränderungen
10	2014-04-06 09:19:02	maildns1.lrz-muenchen.de:0,	129.187.254.5	111	tcp	rpcbind	0	Keine Veränderungen
11	2014-04-06 09:19:02	maildns1.lrz-muenchen.de:0,	129.187.254.5	873	tcp	rsync	32	Port wieder geöffnet
12	2014-04-06 09:19:02	maildns2.lrz-muenchen.de:0,	129.187.254.8	111	tcp	rpcbind	0	Keine Veränderungen
13	2014-04-06 09:19:02	maildns2.lrz-muenchen.de:0,	129.187.254.8	873	tcp	rsync	32	Port wieder geöffnet
14	2014-04-06 09:19:02	BADWLRZ-SWTSGW1.ads.mwn.de:0,	129.187.254.10	443	tcp	http	0	Keine Veränderungen
15	2014-04-06 09:19:02	wlan.lrz.de:0, apstat.srv.lrz.de:0,	129.187.254.11	22	tcp	tcpwrapped	0	Keine Veränderungen
16	2014-04-06 09:19:02	wlan.lrz.de:0, apstat.srv.lrz.de:0,	129.187.254.11	80	tcp	http	0	Keine Veränderungen
17	2014-04-06 09:19:02	wlan.lrz.de:0, apstat.srv.lrz.de:0,	129.187.254.11	111	tcp	rpcbind	0	Keine Veränderungen
18	2014-04-06 09:19:02	wlan.lrz.de:0, apstat.srv.lrz.de:0,	129.187.254.11	443	tcp	http	0	Keine Veränderungen
19	2014-04-06 09:19:02	wobinich.netz.lrz.de:0, dnsstat.lrz.de:0, dnsstat.lrz-muenchen.de:0,	129.187.254.12	22	tcp	tcpwrapped	0	Keine Veränderungen
20	2014-04-06 09:19:02	wobinich.netz.lrz.de:0, dnsstat.lrz.de:0, dnsstat.lrz-muenchen.de:0,	129.187.254.12	80	tcp	http	0	Keine Veränderungen
21	2014-04-06 09:19:02	wobinich.netz.lrz.de:0, dnsstat.lrz.de:0, dnsstat.lrz-muenchen.de:0,	129.187.254.12	111	tcp	rpcbind	0	Keine Veränderungen
22	2014-04-06 09:19:02	mailout.lrz.de:0,	129.187.254.15	25	tcp	smtp	0	Keine Veränderungen
23	2014-04-06 09:19:02	mailout.lrz.de:0,	129.187.254.15	465	tcp	smtp	0	Keine Veränderungen
24	2014-04-06 09:19:02	mailout.lrz.de:0,	129.187.254.15	587	tcp	smtp	0	Keine Veränderungen
25								

Abbildung 6.13: Aufbau der Scan-Ergebnisse innerhalb der CSV-Datei.

7 Zusammenfassung und Ausblick

Das finale Kapitel fasst die Ausgangssituation und den kompletten Ablauf der Arbeit kurz zusammen. Im Anschluss wird ein Ausblick möglicher Erweiterungen diskutiert, mit denen eine Optimierung der Implementierung des Webfrontends für Dr. Portscan erreicht werden kann.

7.1 Zusammenfassung

Mithilfe sogenannter Portscanner kann ermittelt werden, welche netzbasierten Dienste ein Rechner anbietet. Solche Dienste können Sicherheitslücken darstellen, weshalb Systemadministratoren häufig Portscan-Ergebnisse verwenden, um ihre Systeme nach unerwünscht offenen Ports oder der richtigen Konfiguration der eingesetzten Firewall überprüfen. Mit Dr. Portscan, ein vom Leibniz-Rechenzentrum (LRZ) entwickeltes Werkzeug, können regelmäßig durchgeführte Portscans automatisch ausgewertet werden. Die räumlich und zeitlich versetzten Portscan-Ergebnisse werden dann in einer Datenbank aggregiert. Die Report-Skripte von Dr. Portscan werden dann dazu verwendet, die zuständigen Personen über die aktuellen Zustände ihrer Systeme auf dem Laufenden zu halten. Durch dieses proaktive Vorgehen kann Angriffen auf die eigenen Systeme vorgebeugt werden.

Das LRZ ist Betreiber des Münchner Wissenschaftsnetz (MWN), welches bayrische Hochschul- und Forschungseinrichtungen verbindet. Dieses Netz wird in viele Subnetze unterteilt, die von sogenannten Netzverantwortlichen betreut werden. Um diesen Netzverantwortlichen einen einfachen Zugang zu Dr. Portscan zu ermöglichen, damit sie ihre Netzbereiche scannen können, wurde ein mandantenfähiges Webfrontend entwickelt, mit dessen Hilfe jeder Netzverantwortliche eigene Portscans konfigurieren und die Ergebnisse einsehen kann.

Um den Funktionsumfang des Webfrontends zu definieren wurde in einem ersten Schritt eine Anforderungsanalyse durchgeführt. Dabei wurden typische, in der Praxis auftretende Use-Cases der Netzverantwortlichen und des Dr.-Portscan-Administrators betrachtet. Diese resultierten in einer Sammlung an Anforderungen, mit denen im zweiten Schritt ein neuer Systementwurf für das Zusammenspiel von Webfrontend und Dr. Portscan entwickelt wurde. Insbesondere das Verteilen der Scans auf die Datenquellen, also den Servern, von denen aus die Portscans gestartet werden, und die Berichterstattung per E-Mail wurden neu konzipiert. Dr. Portscan nutzt statische Skripte auf den Datenquellen, um die Scans durchzuführen. Dies stellte für ein dynamisch ausgelegtes Konfigurationswebfrontend keine Option dar. Deshalb werden die Scan-Aufrufe nun zentral vom Dr.-Portscan-Server an die Datenquellen verteilt. Das Reporting der Scan-Ergebnisse wurde ebenfalls neu konzipiert. Die Ergebnisse können weiterhin per E-Mail an verschiedene Empfänger verschickt werden. Hinzugekommen ist allerdings die Möglichkeit der direkten Einsichtnahme auf dem implementierten Webfrontend. Um diesen Systementwurf umzusetzen, wurde das von Dr. Portscan bisher verwendete Datenbankschema um einige Tabellen erweitert und von einer SQLite- auf eine MySQL-Datenbank umgestellt.

Anschließend folgte die Implementierung des Webfrontends mit der Skriptsprache PHP. Grundlegend wurde das Projekt nach dem Model-View-Controller-Design-Pattern strukturiert. Dabei werden die Datenhaltung und der -zugriff (Model), die Generierung und Ausgabe der Benutzeroberfläche (View) und die eigentliche Programmlogik (Controller) in getrennten, unabhängigen und somit austauschbaren Bereichen implementiert. Das Ergebnis war ein mandantenfähiges Webfrontend für die Konfiguration von Dr. Portscan, welches zwischen zwei Benutzergruppen unterscheidet. Es gibt eine Sektion für den Dr.-Portscan-Administrator und eine für die Netzverantwortlichen.

Der Dr.-Portscan-Administrator kann nun mithilfe des Webfrontends Aufgaben der Verwaltung und der Konfiguration vornehmen. Er kann neue Netzverantwortliche hinzufügen, sie temporär sperren oder wieder löschen. Er hat eine Übersicht der eingestellten Scans der Netzverantwortlichen und somit ein Tool zur Überwachung der Auslastung der Datenquellen. Des Weiteren kann er eine globale Ausnahmeliste pflegen, in der Netzbereiche gespeichert werden, die von keinem der eingestellten Scans beachtet werden soll.

Die Netzverantwortlichen haben nun eine zentrale Anlaufstelle für die eigenständige Konfiguration von Scans innerhalb ihres Netzbereiches. Sie können Portscans sowohl von internen (innerhalb des MWN) als auch von externen (außerhalb des MWN) Datenquellen veranlassen. Auch die Berichterstattung per E-Mail ist individuell konfigurierbar. Es können Reports an beliebige Empfänger geschickt werden, sodass z. B. ein Systemadministrator stets die Scan-Ergebnisse für seine Endsysteme erhält und schnell reagieren kann, falls unerwünschte Ports geöffnet sind. Die Netzverantwortlichen haben ebenso die Möglichkeit, die aktuellen Ergebnisse direkt auf dem Webfrontend einzusehen. Dabei kann zwischen einer Gesamtansicht aller Ergebnisse und einer Ansicht, bei der nur die durch Änderungen auffällig gewordenen Ports sichtbar sind, gewählt werden. Um den Verwaltungsaufwand des Dr.-Portscan-Administrators zu verringern, wird den Netzverantwortlichen die autonome Verwaltung ihres Netzbereichs gewährt. Sie können Subnetzverantwortliche innerhalb ihres Netzbereiches hinzufügen, die dadurch selbst Zugriff auf das Dr.-Portscan-Webfrontend erhalten.

7.2 Ausblick

Das implementierte Webfrontend ist ein erster Prototyp. Erst der reale Einsatz wird zeigen, ob es allen Anforderungen genügt und ob die Skalierbarkeit hinsichtlich der Menge der Netzverantwortlichen und der Netzbereiche ausreicht. Mit diesen Erfahrungswerten kann die bestehende Version optimiert und angepasst werden.

Die nicht erfüllten Anforderungen, insbesondere der Datenbank-Import der LRZ-Netzdoku-Datenbank, in der viele Netzverantwortliche des MWN gespeichert sind, und die damit verbundene Authentifizierung über das LRZ-SIM sind aussichtsreiche Optionen, welche die Verwendung des Webfrontends weiter optimieren können. Auch die Möglichkeit IPv6-Adressen zu scannen sollte in Zukunft ermöglicht werden. Da hier allerdings die großen Netzbereiche der Netzverantwortlichen zu beachten sind, muss ein neues Konzept für IPv6-Scans konzipiert werden. Es könnten nur einzelne IPv6-Endsysteme gescannt oder aber komplexere Systeme entwickelt werden. Dazu kann beispielsweise im Rahmen des DNS-Lookups (Domain Name System) der IPv4-Adressen der gescannten Systeme überprüft werden, ob für den DNS-Eintrag auch ein IPv6-Eintrag vorliegt. Diese können daraufhin verknüpft werden und zusätzlich ein IPv6-Scan veranlasst werden. Insbesondere in der aktuellen Übergangszeit von IPv4 zu IPv6 ist solch ein Vergleich sehr interessant um z. B. Inkonsistenzen in der

Firewall-Konfiguration aufzudecken.

Ebenfalls können weitere Leistungen von Portscannern genutzt werden. Es besteht die Möglichkeit, die installierten Betriebssysteme der jeweiligen Hosts zu erkennen. Der Portscanner Nmap, welcher bei der Implementierung des Webfrontends eingesetzt wurde, bietet solch eine Option an. Diese Option hätte ohne weiteres in die eingesetzten Scan-Aufrufe integriert werden können. Dr. Portscan kann diese Informationen bereits mit in die Datenbank aufnehmen. Allerdings bringt solch eine Betriebssystemerkennung eine erhöhte Ausführungszeit der Scans mit sich. Den eingesetzten Datenquellen dadurch verfügbare Ressourcen zu nehmen, ist keine optimale Lösung. Insbesondere ist es nicht sinnvoll, täglich einen Betriebssystemscan durchzuführen. Ein möglicher Lösungsansatz wäre eine eigene, nur für die Betriebssystemerkennung zuständige Datenquelle einzusetzen. Diese könnte beispielsweise einmal die Woche alle in der Datenbank vorhandenen Hosts scannen, um so deren aktuelle Betriebssysteme zu erkennen.

Dr. Portscan kann ebenfalls als ganzes oder auch nur teilweise in das bestehende Portal für Netzverantwortliche namens *NeSSI* integriert werden. NeSSI (Network Self Service Interface) bietet derzeit die Möglichkeit der Einsichtnahme gespeicherter Informationen über den jeweiligen Netzverantwortlichen, der vom LRZ-DHCP-Server zugeteilten IP-Adressen sowie der angeschlossenen Endgeräte unter Berücksichtigung des betreuten Netzbereichs. NeSSI ist durch den ähnlichen Aufbau und die gleiche Nutzergruppe optimal für eine Erweiterung durch das entwickelte Dr.-Portscan-Webfrontend geeignet. Denkbar wäre die Einsichtnahme der Scan-Ergebnisse durch eine Kopplung mit der Dr.-Portscan-Datenbank oder die komplette Integration aller Komponenten der Netzverantwortlichen des Dr.-Portscan-Webfrontends. Dadurch kann den Netzverantwortlichen eine zentrale Anlaufstelle für Informationen rund um ihren Netzbereich geboten werden.

Abbildungsverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[Böh05] Böhm, Christian: *Datenbanksysteme I.* Vorlesungsskript Kapitel 1, 2005.

[cwS07] *Skriptsprachen: Einfach und doch sehr mächtig*, September 2007. `http://www.computerwoche.de/a/skriptsprachen-einfach-und-doch-sehr-maechtig,538267`.

[Hab05] Habets, Corinna: *Skriptsprachen*, 2005. Lehrstuhl Informatik IX - RWTH Aachen.

[Hom12] Hommel, Wolfgang und Reiser, Helmut: *Das Münchner Wissenschaftsnetz (MWN) Konzepte, Dienste, Infrastrukturen, Management.* Organisationsbeschreibung, Leibniz-Rechenzentrum, 2012. `http://www.lrz.de/services/netz/mwn-netzkonzept/mwn-netzkonzept.pdf`.

[HR09] Hommel, Wolfgang und Helmut Reiser: *IT-Sicherheit.* Vorlesungsskript Kapitel 13, 2009.

[HvGM13] Hommel, Wolfgang, Felix von Eye, Michael Grabatin und Stefan Metzger: *Automatisierte Portscan-Auswertung in großen Netzen - Leider geöffnet.* ADMIN, 2013.

[Ill07] Illik, Johan Anton: *Verteilte Systeme: Architekturen und Software-Technologien.* expert Verlag, 2007.

[KgD12] Kranzlmüller, Dieter, Nils gentschen Felde und Vitalian Danciu: *Rechnernetze und verteilte Systeme.* Vorlesungsskript, 2012.

[lrz13] *LRZ: Netzverantwortliche in den Instituten*, Dezember 2013. `http://www.lrz.de/services/netz/netzverantwortliche/`.

[McN08] McNab, Chris: *Network Security Assessment.* O'reilly, 2008.

[Mic11] Michelsen, Björn: *Grundlagen von Webframeworks.* Web-Technologien, Lehrstul für Medieninformatik an der Otto-Friedrich-Universität Bamberg, Seiten 45–60, 2011.

[vHM13] von Eye, Felix, Wolfgang Hommel und Stefan Metzger: *Dr. Portscan: Ein Werkzeug für die automatisierte Portscan-Auswertung in komplexen Netzinfrastrukturen.* Technische Hintergrundbericht, Leibniz-Rechenzentrum, 2013. DFN-Workshop: Sicherheit in vernetzten Systemen.