Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Lutz Jankowski

Konzeption und Entwicklung eines Systems zur Inventarisierung und Visualisierung von IP-Netzwerken

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Lutz Jankowski

Konzeption und Entwicklung eines Systems zur Inventarisierung und Visualisierung von IP-Netzwerken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kossakowski
Zweitgutachter: Prof. Dr. Thomas Schmidt

Eingereicht am: 19. September 2017

Lutz Jankowski

Thema der Arbeit
Konzeption und Entwicklung eines Systems zur Inventarisierung und Visualisierung von IP-Netzwerken

Stichworte
Netzwerk, IT-Sicherheit, Inventarisierung, Topologie, Dienste, Portscanning

Kurzzusammenfassung
Um Rechnernetze gegen Angriffe schützen zu können, ist es von entscheidender Bedeutung, zu wissen, welche Geräte sind im Netzwerk befinden, wie sie miteinander verbunden sind und welche Dienste angeboten werden. Da diese Konfiguration ständiger Veränderung unterliegt, bedeutet eine entsprechende händische Dokumentation kontinuierlichen Aufwand, ist somit kaum praktikabel und ihre Zuverlässigkeit dementsprechend zweifelhaft. Diese Arbeit umfasst die Konzeption und Entwicklung eines Systems, das IP-Netzwerke automatisch inventarisiert und damit einen aktuellen Überblick über deren Konfiguration ermöglicht.
Lutz Jankowski

Title of the paper
Design and Implementation of a System that inventories and visualizes IP-Networks

Keywords
network, IT-security, inventory, topology, services, portscanning

Abstract
In order to be able to secure computer networks it is crucial to know what devices are present, how they are connected to each other and what services are provided. As configurations are constantly changing, maintaining a corresponding documentation manually would require continuous efforts, is hardly practicable and its reliability would be doubtful. This thesis covers the design and implementation of a system that automatically inventories IP-networks, allowing an up-to-date overview of their configuration.

Inhaltsverzeichnis

1. Einleitung

"It is important to understand that it is the job of most networks to allow at least some communication to flow into and out of their borders. Networks that exist in complete isolation with no Internet connection and no services like e-mail or web traffic are very rare today. Each service, connection, or route to another network provides a potential foothold for an attacker." [Engebretson, 2013, S. 54]

Die Sicherheit von Computernetzwerken gegen Angriffe hängt maßgeblich von der Konfiguration der Geräte, den darauf laufenden Programmen/Diensten und deren Schwachstellen ab. Daher ist es ein unerlässlicher Schritt, bei der Absicherung eines Netzwerks eine Bestandsaufnahme von Geräten und potentiell angreifbaren Diensten zu machen, und festzustellen, aus welchen anderen Netzsegmenten diese erreichbar sind.
Eine weitere Rolle spielen, Nutzerkonten und -gruppen, die wesentliche Veränderungen an Teilen der Infrastruktur vornehmen könnten.

Dabei ist der stetige Wandel eine weitere Herausforderung, da einmal Festgestelltes schon bald nicht mehr zutreffen kann. Damron berichtet dazu aus der Praxis:
"Enterprise networks are always growing and changing over time. New technologies and new employees create a changing landscape. Major events like a merger or acquisition create large scale changes which must be rapidly understood. Unknown devices can easily become the weakest link in your network." [Damron, 2016]
Oft ist es für Nutzer sogar möglich unautorisierte Geräte zu einem Netzwerk hinzuzufügen, Geräte zu ersetzen oder Software auszuführen bzw. zu installieren.

Erst mit einem vollständigen und aktuellen Überblick lassen sich Schwachstellen ausmachen und es kann sinnvoll mit den erkannten Risiken umgegangen werden. Er dient dann als Einstiegspunkt für weitere sicherheitsbezogene Aktivitäten, wie der Reaktion auf Sicherheitsvorfälle, einer Risikoanalyse oder einem vollständigen Sicherheitsaudit.
Erst wenn unsichere Geräte, Software und Konfigurationen erkannt worden sind, können Maßnahmen getroffen werden, einen den Sicherheitsrichtlinien entsprechenden Zustand herzustellen.
Gegebenenfalls können letztere in Hinblick auf neu erkannte Risiken angepasst werden: Bei der Inventarisierung könnte z.B. auffallen, dass der Fernzugang für privilegierte Nutzer mit einer reinen Passwort-Authentisierung möglich ist. Wenn dies als unsicher bewertet wird, obwohl die bisher vorhandenen Sicherheitsrichtlinien es nicht verbieten, sollte die entsprechende Regel angepasst bzw. eine neue hinzugefügt werden.

Wenn Veränderungen an Netzwerk und -geräten (inkl. verbotener Geräte) nicht automatisiert erfasst werden, existiert ein solcher Überblick oft nicht oder nicht ausreichend. Aufgrund des Aufwandes, den es bedeutet entsprechende Dokumentationen händisch zu erstellen und zu pflegen, darf in diesem Fall angenommen werden, dass eine bestehende Dokumentation Lücken bzw. Fehler aufweist.

1.1 Ziel der Arbeit

Es soll ein System zur Verfügung gestellt werden, das es ermöglicht, mit möglichst geringem Installations- und Konfigurationsaufwand und ohne Rückgriff auf bereits vorhandene Dokumentation oder Asset-Management-Systeme, strukturierte Informationen über Geräte, Software inkl. Dienste sowie die Topologie von IP-Netzwerken zu erlangen. Der Einfachheit halber soll das zu entwickelnde System fortan *ANIS* (Automated Network Inventory System) genannt werden. *ANIS* soll dabei schlank und dennoch möglichst universell einsetzbar und daher nicht auf das Vorhandensein spezieller, bereits diesen Zwecken dienender Systeme, angewiesen sein.
Dabei soll die möglicherweise zeitintensive Erkundung des Netzwerks und deren Mitglieder im Hintergrund erfolgen und die Ergebnisse zentral gesammelt und vorgehalten werden und bei Bedarf abrufbar sein.
Die Ergebnisse sollten auch stets von Menschen ausgewertet werden können. Im Gegensatz zu einer vollständigen, rein textlichen Auflistung der Daten sollten diese daher auch in einer anderen Form dargestellt werden, die intuitiver erfassbar ist und somit einen Einstiegspunkt in den Rest der gesammelten Daten bietet. Aus diesem Grund soll das Netzwerk mit den wichtigsten Kenndaten in Form eines Netzwerkplans visualisiert werden.
Darüber hinaus sollen die gewonnenen Daten auch so repräsentiert werden, dass sie sich einfach automatisiert weiter nutzen lassen.

1.2 Zielgruppe der Arbeit

Diese Arbeit richtet sich zum einen an all jene, die es zur Aufgabe haben, ein Netzwerk auf sicherheitsrelevante Aspekte zu untersuchen. Da aber auch auf die Grundlagen der Netzwerkinventarisierung, und welche Aspekte dabei sicherheitsrelevant sind, eingegangen wird, kann die Arbeit einen guten Überblick für die weitere Beschäftigung mit dem Thema bieten.
Allgemeines Grundlagenwissen über Computernetzwerke, deren Administration, das OSI-Referenzmodell und die wichtigsten Protokolle aus dem TCP/IP-Protokollstack in diesem Zusammenhang, werden zum vollen Verständnis dieser Arbeit vorausgesetzt.

1.3 Voraussetzungen und Abgrenzungen

Behandelt wird im Rahmen dieser Arbeit eine rein digitale Inventarisierung des Netzwerkes ab Schicht 3 des OSI-Referenzmodells.
Zudem wird vorausgesetzt, dass die Zieladressbereiche, die erkundet werden sollen, bereits bekannt sind. Je größer die Adressbereiche sind, desto mehr Zeit und Ressourcen werden für eine gründliche Erkundung benötigt. Daher wird im Folgenden von wenigen tausend Ziel-IP-Adressen ausgegangen.
Durch entsprechende Verteilung der Anwendung könnten zwar auch größere Adressbereiche bedient werden, diese Problematik soll jedoch nicht Gegenstand dieser Arbeit sein.
Zudem wird vorausgesetzt, dass im zu scannenden Bereich eine IP genau einem Gerät zuzuordnen ist.

Bezüglich des IP-Protokolls wird sich auf die Nutzung von IPv4 beschränkt, da es zum gegenwärtigen Zeitpunkt noch verbreiteter als sein Nachfolger IPv6 ist. Beide Protokolle zu unterstützen würde den Implementierungsaufwand erhöhen, ohne jedoch den grundsätzlichen Lösungsansatz zur thematisierten Problemstellung signifikant zu bereichern. Auch was Transportprotokolle betrifft, wird sich auf die verbreitetsten Protokolle, TCP und UDP, beschränkt. Aufgrund seiner führenden Verbreitung als LAN-Technologie wird für die Sicherungsschicht von Ethernet-Netzwerken bzw. den weitgehend kompatiblen IEEE-802.11-Normen ausgegangen.

Auch wenn für *ANIS* ggf. Schicht 2-Techniken wie ARP direkt genutzt werden könnten, so bleibt der Anspruch an die Ermittlung von Netzwerkkomponenten trotzdem auf IP-Geräte innerhalb des Zieladressbereichs beschränkt.
Dies begründet sich zum einen darin, dass es kaum denkbar ist, unterhalb Schicht 3 initialen Zugang aus der Ferne zu erhalten und zum anderen wären hier im Rahmen einer Inventarisierung andere Techniken erforderlich, deren Behandlung, hinsichtlich des Umfangs dieser Arbeit, ebenfalls nicht geleistet werden kann.
Nichtsdestotrotz ist es im Rahmen eines vollständigen Sicherheitskonzeptes natürlich von großer Bedeutung, auch Zutritt und Zugang zu Netzwerkkomponenten beachten und die Schichten 1 und 2 des OSI-Referenzmodells nicht zu vernachlässigen. Eine treffende Beobachtung aus der Praxis lautet wie folgt:
"If the physical location is unknown, there is no way to know if someone is tapping or viewing the data (confidentiality), modifying the data or the system directly (integrity), or if the server has reliable power, fire suppression, or theft prevention (availability)." [Jorgensen, 2015]
Wichtige Aspekte sind auch auf diesen Ebenen u.a. wo die Geräte stehen, wer Zugang hat und die Möglichkeit auszuschließen, dass die Geräte möglicherweise manipuliert oder bereits unerlaubt hinzugefügt worden sind. Beispielsweise könnten sonst Daten über einen drahtlosen Zugangspunkt unbemerkt nach außen gespiegelt werden. Auch sollten z.B. Vorkehrungen getroffen werden, die Diebstahl verhindern oder, sollte ein solcher doch gelungen sein, diesen möglichst schnell zu erkennen, damit darauf reagiert werden kann.

Des Weiteren wird sich im Rahmen dieser Arbeit darauf beschränkt werden, für viele Systeme nur die Informationen zu sammeln, die z.B. über angebotene Dienste von außen sichtbar sind. Einige Daten über Systeme erhält man nur mit direktem Zugriff darauf. Hier müssen zum einen der Zugang und die entsprechenden Leseberechtigungen vorhanden sein. Zum anderen sind die technischen Zugangsmöglichkeiten zu unterschiedlichen Systemen und die Wege, über die Informationen ausgelesen werden können, häufig nicht einheitlich. Daher ist es für detailliertere Informationen über unterschiedliche Systeme und Konfigurationen oftmals nötig, die Module, die Zugang herstellen und Informationen abfragen, darauf anzupassen bzw. gesondert zu implementieren. In Anbetracht des Anspruchs der Erweiterbarkeit soll letzteres möglich sein, der Fokus soll jedoch auf einer möglichst breiten Einsetzbarkeit liegen.

ANIS soll einen schnellen Überblick über die Netzwerklandschaft geben und als Einstiegspunkt für die weitere Sicherheitsanalyse dienen. Es ersetzt keine bestehenden Netzwerkmanagement-Systeme, Schwachstellen- oder Portscanner und ähnliches. Wie bei allen Werkzeugen, müssen die Ergebnisse *ANIS'* stets vom fachkundigen Menschen ausgewertet und kritisch hinterfragt werden, da eine automatisierte Informationsbeschaffung

in so heterogenen Systemen wie Computernetzwerken auf eine Vielzahl von Schwierigkeiten treffen kann und die Ergebnisse damit nur eingeschränkt zuverlässig sind.
"Good inventory systems are difficult to find, however, and even the best can miss some hosts." [Limoncelli, 2007 S.76]
So sind z.B. auch zum Zeitpunkt der Erkundung abgeschaltete oder ausgefallene Systeme nicht auffindbar. Selbst im Falle von bereits bekannten Systemen ist nicht erkennbar, weshalb sie derzeit nicht ansprechbar sind und ob sie evtl. dauerhaft entfernt wurden.
Des Weiteren können Firewalls die Erkundung des Netzwerks erschweren und IPS können aktiv stören z.B. indem sie Signaturen von Portscans identifizieren und Maßnahmen ergreifen, die dem Scanner falsche Ergebnisse liefern. Je nach Standpunkt und Perspektive, die das System einnehmen soll, sollten solche Sicherungssysteme ggf. so konfiguriert werden, dass sie die Erkundung nicht behindern.

Letztlich ist *ANIS* als Prototyp zu betrachten und nicht dazu vorgesehen, ohne weiteres in einer Produktionsumgebung eingesetzt zu werden oder gut zu skalieren. So fließen z.B. auch die Daten zwischen den Komponenten des Werkzeugs unverschlüsselt.
Der Zweck der Implementation ist es, aus den im theoretischen Teil der Arbeit aufgezeigten Möglichkeiten und Ansätzen zur Problemlösung, geeignete auszuwählen und ein mögliches System beispielhaft zu realisieren.

1.4 Struktur der Arbeit

Abschnitt 1 führt in das Thema dieser Arbeit ein, um ein generelles Verständnis dafür zu schaffen, welches Problem im Folgenden bearbeitet wird und welches Ziel mit der Arbeit verfolgt wird. Zudem werden allgemeine Rahmenbedingungen erläutert und das Thema weiter abgegrenzt.

Abschnitt 2 setzt fort mit Grundlagen, die zur Erreichung des Ziels von Nutzen sein könnten. Dabei werden einige Begriffe erklärt und die sich dahinter verbergenden Möglichkeiten erläutert. Weiterhin werden verschiedene Werkzeuge genannt, die diese Möglichkeiten nutzbar machen, ohne alles von Grund auf neu zu entwickeln.

In Abschnitt 3 werden die Anforderungen an eine Lösung aufgestellt und es wird evaluiert inwiefern sich zuvor beschriebene Werkzeuge verwenden lassen, um die jeweiligen Anforderungen als Teilaspekt einer Gesamtlösung zu erfüllen.

In Abschnitt 4 werden sich ergebende Ansätze für eine Gesamtlösung ausgemacht, bewertet und schließlich eine konkrete Lösungsstrategie entwickelt. Auf dieser Basis wird dann mit einer Architektur, Schnittstellen und der grundsätzlichen Funktionsweise ein Konzept für *ANIS* geschaffen.

Abschnitt 5 beschreibt die Umsetzung des zuvor entwickelten Konzepts. Dabei wird auf verwendete Werkzeuge und Technologien eingegangen. Es wird zudem behandelt, die konkreten Ergebnisse des Systems aufgebaut sind, wie das System erweiterbar ist und welche Schwierigkeiten bei der Implementierung aufgetreten sind.

In Abschnitt 6 wird ein Fazit über das Erarbeitete gezogen und ein Ausblick als Ansatzpunkt für weitere Arbeiten gegeben.

2. Grundlagen

Im Folgenden soll theoretisches Wissen und technische Möglichkeiten die für die Zielsetzung von Nutzen sein können, grundlegend erläutert werden. Außerdem werden nutzbare Werkzeuge kurz vorgestellt.

2.1 Topologie

"Network topology is a representation of the interconnection between directly connected peers in a network. A physical topology corresponds to many logical topologies each at a different level of abstraction. At the IP level, peers are hosts or routers one IP hop away from each other and at the workgroup level the peers are hosts connected by a logical link. Network topology constantly changes as nodes and links join a network, personnel move offices, and network capacity is increased to deal with added traffic. Keeping track of network topology manually is a frustrating and often impossible job. Network topology knowledge including the path between endpoints, can play an important role in analyzing, engineering, and visualizing networks." [Nazir, 2007, S.425]

Die Topologie eines Netzwerks auf IP-Ebene zeigt, wie die Geräte logisch miteinander verbunden sind. Stellt man die Daten grafisch dar, können z.B. Subnetze oder Komponenten wie Router und Firewalls veranschaulicht werden und damit ein schneller intuitiv erfassbarer Überblick über das Netzwerk geboten werden.

2.2 Netzwerkanalyse

An dieser Stelle soll ein Überblick über Dienste, Protokolle und Verfahren gegeben werden, die sich dazu nutzen lassen, automatisiert Informationen über Geräte in einem IP-Netzwerk zu sammeln.
Auf oberster Ebene unterteilen sie sich in aktive und passive Methoden.
Passive Methoden zeichnen sich dadurch aus, dass mit dem Ziel selbst nicht interagiert wird. Dadurch belasten sie das Netzwerk nicht. (Für Angreifer eignen sie sich außerdem besonders um unerkannt zu bleiben.)
Aktive Methoden interagieren mit dem Ziel selbst und machen es möglich, systematisch zu sondieren und sind weniger abhängig von Gegebenheiten, wie der Position im Netzwerk oder der Bereitstellung von Daten in Verzeichnissen.
Aus beiden Kategorien gibt es verschiedene Techniken, die Dienste und Protokolle nutzen, welche Informationen über das implementierende System oder andere Systeme freigeben.

2.1.1 Netzwerk-Sniffing

Als Netzwerk-Sniffing bezeichnet man das Mitlesen von (insbesondere fremdem) Datenverkehr. Auch hier wird zwischen einer aktiven und einer passiven Vorgehensweise unterschieden.
Passives Sniffing bietet sich besonders bei der Verwendung von Broadcast-Medien wie Hubs oder WLAN an, da hier auch fremde Pakete ohne weitere Maßnahmen an der eigenen

Netzwerkschnittstelle ankommen. Letztere muss lediglich in den Promiscuous Mode versetzt werden, was dafür sorgt, dass fremde Pakete auch angenommen werden.
In voll geswitchten Netzwerken werden an Hosts nur in Ausnahmefällen Pakete ankommen, die nicht für sie speziell bestimmt sind. Die gewonnenen Daten würden sich dann vor allem auf erhaltene Broadcast-Nachrichten wie z.B. ARP-Anfragen beschränken. [vgl. Pompon, 2016, S. 202ff]

Aktive Vorgehensweisen, nutzen z.B. ARP um Zugriff auf fremden Verkehr zu erhalten.
ARP (Address Resolution Protocol) ist in Schicht 2 des OSI-Referenzmodells angesiedelt und dient vor allem der Ermittlung von Hardware-Adressen zu gegebenen IPv4-Adressen innerhalb eines Ethernet-Netzes: Der Sender schickt dazu ein ARP-"Request"-Paket mit der gesuchten IP-Adresse an die Broadcast-Hardware-Adresse 0xFF:FF:FF:FF:FF:FF, wodurch es allen Teilnehmern im Subnetz zugestellt wird. Der Empfänger, der feststellt, dass es sich bei der gesuchten Adresse um seine eigene handelt, antwortet an die Hardware-Adresse des Senders mit einem ARP-"Reply"-Paket mit seiner IP-Adresse und der Hardware-Adresse als Absender. [vgl. Wilhelm, 2013, S.181]
Durch eine Protokollschwachstelle lässt sich ARP für einen Man-in-the-Middle-Angriff nutzen: Auch bei nicht angeforderten ARP-Antworten wird stets angenommen, dass sie vom korrekten Gerät kommen. Die Kommunikation zwischen zwei Geräten kann daher belauscht werden, indem beiden ungefragt ARP-Reply-Nachrichten geschickt werden, welche die IP-Adresse, des jeweils anderen beinhalten. Um die Kommunikation der beiden nicht zu unterbrechen sollte IP-Forwarding eingeschaltet werden.

Eine weitere aktive Möglichkeit, Zugriff auf fremden Netzwerkverkehr zu erhalten wird MAC-Flooding genannt. Hier wird das Verhalten einiger Switches ausgenutzt, bei einer Überflutung mit ARP-Replies und daher einhergehender Überlastung des Adressspeichers, in den Hub-Modus zu gehen und alle Pakete an alle weiterzuleiten. Viele neuere Switches sollen einen eingebauten Schutz gegen eine solche Attacke haben.

Ist erst einmal der Zugriff auf den Datenverkehr gegeben, gibt es verschiedene Werkzeuge zur Auswertung. Zu beachten ist hierbei natürlich, dass die Informationen, die sich aus verschlüsseltem Netzwerkverkehr gewinnen lassen nur begrenzt aussagekräftig sind. Aber auch hier lassen sich zumindest IP-Adressen, Ports und Datenmengen analysieren. Es gibt weitere Ansätze, die den Verkehr manipulieren, sichere Verbindungen vortäuschen etc. sollen hier jedoch nicht weiter beleuchtet werden, da sie sich vom Thema der Arbeit entfernen. [vgl. Baloch, 2015, S. 139ff; Gregg, 2015, S. 83]

2.1.2 Portscanning

Beim Portscanning handelt es sich um eine aktive Aufklärungsmethode, bei der definierte Ports von Zielsystemen mit Hilfe von Sondierungspaketen und den erhaltenen Antworten daraufhin untersucht werden, ob sich dahinter ein Dienst befindet, der Verbindungen annimmt. Zudem kann Portscanning dazu genutzt werden, Firewallregeln zu erkennen, indem analysiert wird, welche Pakete durchgelassen bzw. geblockt werden.

Die wichtigsten Zustände in denen sich ein Port befinden kann sind:

- offen - ein Dienst nimmt Verbindungen an

- geschlossen - der Port ist erreichbar, nimmt jedoch keine Verbindungen an

Oftmals reichen diese beiden Zustände nicht aus, um das Ergebnis des Scans wiederzugeben, daher unterscheidet der beliebte Portscanner Nmap weiterhin zwischen folgenden Zuständen:

- gefiltert - entweder es gibt eine ICMP-Nachricht, die z.B. besagt, dass der Zugriff untersagt ist, oder, was öfter der Fall ist, es gibt überhaupt keine Antwort
- ungefiltert - Ergebnis eines TCP ACK-scan
- offen|gefiltert - Ergebnis von Scan-Arten, bei denen offene Ports nicht antworten, und daher nicht zu bestimmen ist, ob der gescannte Port offen oder gefiltert ist. Dies gilt neben UDP-Scans u.a. für TCP FIN, -NULL und -XMAS-Scans
- geschlossen|gefiltert - Ergebnis eines IP ID idle-Scans (siehe unten), bei dem nicht bestimmt werden kann, ob ein Port geschlossen oder gefiltert ist.

[vgl. Lyon, 2016, PSB][1]

Im Folgenden seien die verbreitetsten Arten von TCP-Scans genannt, die Aufschluss über den Zustand der jeweiligen Ziel-Ports geben können. Es sei jedoch erwähnt, dass die Implementation des TCP/IP-Protokollstapels je nach Betriebssystem feine Unterschiede aufweist und dadurch nicht jede Technik für jedes Betriebssystem geeignet ist:

- **TCP Full Connect scan** - This type of scan is the most reliable but also the most detectable. It is easily logged and detected because open ports complete the three-step startup and full connection is established. A final RST closes the open port so that a total of four packets are exchanged. Open ports reply with a SYN/ACK; closed ports respond with an RST/ACK on the second step so that only two packets are exchanged.
- **TCP SYN scan** - This type of scan is known as half-open, because a full TCP connection is not established. TCP SYN scan was originally developed to be stealthy and evade IDS systems, although most now detect it. Open ports reply with a SYN/ACK which is followed by an RST, whereas closed ports respond with an RST/ACK on the second step.
- **TCP FIN scan** - Forget trying to set up a connection; this technique jumps straight to the shutdown. This type of scan sends a FIN packet to the target port. Closed ports should send back an RST. This technique is usually effective only on Unix devices.
- **TCP NULL scan** - Sure, there should be some type of flag in the packet, but a NULL scan sends a packet with no flags set. If the operating system has implemented TCP in accordance with RFC 793, then closed ports will return an RST.
- **TCP ACK scan** - This scan attempts to determine access control list (ACL) rule sets or to identify whether stateless inspection is being used. Wenn mit einem RST geantwortet wird, gilt der Port als ungefiltert. If an ICMP Destination Unreachable, Communication Administrative Prohibited message is returned oder keine Antwort kommt, the port is considered to be filtered.

[1] Es wird im Folgenden vielfach auf die Online-Dokumentation Nmaps verwiesen. Aufgrund der schnellen Veränderungen, die Nmap erfährt, wurde dies als zweckdienlicher angesehen, als auf die stellenweise veraltete gedruckte Dokumentation zu verweisen, die im Literaturverzeichnis unter [Lyon, 2008] zu finden ist.

- **TCP Xmas scan** - [...] this is just a port scan that has toggled on the FIN, URG, and PSH flags. Closed ports should return an RST.

[Gregg, 2015, S. 149ff]

Das Scannen von UDP-Ports ist im Vergleich zum Scannen von TCP-Ports fehlerträchtiger und benötigt mehr Zeit: Da UDP verbindungslos ist und es keinen Handshake wie bei TCP gibt, wird ein eingehendes Datagramm niemals im Rahmen des Protokolls abgelehnt.
Die Möglichkeiten UDP-Ports direkt zu scannen beschränkt sich daher auf zwei Ansätze:
Beim ersten wird in der Regel ein leeres Paket an die Ziel-Ports versendet, wobei geschlossene Ports mit ICMP Type 3 Code 3 "Port Unreachable" reagieren sollten. Andere ICMP-Unreachable-Fehler weisen darauf hin, dass der Port gefiltert ist. Keine Antwort dagegen lassen darauf schließen, dass der Port entweder offen oder gefiltert ist.

Laut Lyon ergeben sich in der Praxis dadurch weitere Probleme: Durch die fehlende Antwort von offenen und gefilterten Ports sei es notwendig das Sondierungspaket nach einem gewissen Timeout noch einmal zu senden für den Fall, dass es verloren gegangen ist. Und die ICMP-Fehlermeldungen, die beim Scannen von geschlossenen UDP-Ports erzeugt werden seien ein noch größeres Problem, denn diese unterlägen, im Gegensatz zu RST-Paketen geschlossener TCP-Ports, oftmals einem Rate-Limiting. Dieses sei in der Regel standardmäßig eingestellt. So seien die ICMP-"Destination Unreachable"-Nachrichten beim Linux-Kernel 2.4.20 auf eine Nachricht pro Sekunde beschränkt.
Passte man den Scan auf das o.g. Rate-Limiting von einem Paket pro Sekunde an, so würde ein vollständiger Scan der 65.536 Ports über 18 Stunden dauern.
Um UDP-Scans zu beschleunigen werden u.a. folgende Ideen genannt: Gleichzeitiges Scannen mehrerer Hosts, sich erst einmal auf die beliebtesten Ports beschränken und das Timeout herabzusetzen.

Bei der zweiten Möglichkeit werden an Portnummern, auf denen für gewöhnlich bestimmte Services lauschen, korrekt formatierte Datagramme für eben jene Services gesandt, um eine Antwort anzuregen. Dies ist natürlich nur für eine begrenzte Anzahl von Ports bzw. Services sinnvoll, da sich bei zu vielen akzeptierbaren Fehlversuchen wieder die Zeitproblematik wie oben beschrieben ergibt.
[vgl. Lyon, 2016, PST]

Insbesondere beim Portscanning können Firewalls und IPS die Ergebnisse verfälschen, indem eingehende oder ausgehende Pakete mit oder ohne Meldung geblockt oder gar verändert werden.
"Keep in mind that while many network scanners do an excellent job in finding and identifying services and ports, the displayed result is always an interpretation. As always, some explanations lie closer to the truth than others." [Svensson, 2016, S. 61]

2.1.3 Service-Erkennung

Sind offene Ports (z.B. durch Portscanning) bekannt, so ist es von Interesse, welche Dienste dahinter stehen. In der Regel laufen Dienste auf bestimmten Standard-Ports. Letztlich kann jedoch vom Administrator entschieden werden, auf welchem Port ein Dienst tatsächlich zur

Verfügung steht. Daher ist es kein zuverlässiger Ansatz, anzunehmen, dass ein bestimmter Dienst auf einem Port lauscht nur weil letzterer der assoziierte Standard-Port ist.

Ein zuverlässigere Möglichkeit ist das *Banner Grabbing*, bei dem eine Verbindung mit einem Service hergestellt und die Information ausgewertet wird, die er über sich selbst preisgibt. Es ist nicht ungewöhnlich, dass man so an detaillierte Informationen über den Dienst und die Version erhält. Allerdings ist dabei zu beachten, dass diese Informationen nicht zwingend korrekt sein müssen. So kann vergessen worden sein, die Informationen mit der neueren Software-Version ebenfalls zu erneuern oder es werden absichtlich falsche Angaben gemacht um mögliche Angreifer in die Irre zu führen. Außerdem kann es sein, dass diese Informationen bewusst zurückgehalten werden. [vgl. Wilhelm, 2013, S.203]

Eine weitere Möglichkeit, einen Service zu ermitteln, ist das *Fingerprinting*. Dabei wird mit dem Service interagiert und das Verhalten mit dem von bekannten Diensten abgeglichen. Demgemäß funktioniert diese Erkennungsmethode nur, wenn entsprechende Eingabedaten und "Fingerabdrücke" vorhanden sind. Diese Methode lässt sich auch mit passiv beobachteten Daten nutzen.

2.1.4 TCP/IP-Stack-Fingerprinting

Was in Abschnitt 2.1.2 als Einschränkung für die Anwendbarkeit bestimmter Portscan-Verfahren erwähnt wurde, kommt uns beim o.g. dritten Ansatz der Betriebssystemerkennung zugute:
"Die Implementierung des TCP/IP-Protokollstacks unterscheidet sich von Betriebssystem zu Betriebssystem geringfügig. Diese geringfügigen Unterschiede zeigen sich in marginalen Differenzen im Verhalten in bestimmten Situationen. Testet und beobachtet man gezielt das Verhalten eines Rechners im Hinblick auf diese Unterschiede, so kann man dadurch relativ präzise feststellen, welches Betriebssystem auf der Maschine läuft. Nimmt man all diese Informationen zusammen, so können sogar in einigen Fällen bestimmte Varianten desselben Betriebssystems unterschieden werden." [Kappes, 2013, S. 243]

Gula merkt an, dass selbst wenn das Stack-Fingerprinting richtig arbeite, könne in der Regel bei entfernten System jedoch nur der Kernel erraten werden, nicht aber die spezifische Version. Tatsächlich beeinflussten aber noch andere Variablen im Netzwerk und auf den Endgeräten das Verhalten des TCP/IP-Stacks, was dazu führen könne, dass Betriebssysteme nicht oder falsch erraten würden. [vgl. Gula, 2009]

Im Folgenden seien ein paar Beispiele für dieses sogenannte Stack Fingerprinting genannt:

- **The FIN probe** - A FIN packet is sent to an open port, and the response is recorded. While RFC 793 states that the required behavior is to not respond, many operating systems, including Windows, will respond with a RESET.
- **Bogus flag probe** - [...] only six valid flags are in the 1-byte TCP header. A bogus flag probe sets one of the used flags along with the SYN flag in an initial packet. Linux responds by setting the same flag in the subsequent packet.
- **Initial Sequence Number (ISN) sampling** - This fingerprinting technique works by looking for patterns in the ISN number. Although some systems use truly random numbers, others, including Windows, increment the number by a small, fixed

amount.

- **IPID sampling** - Many systems increment a system-wide IPID value for each packet they send. Others, including Windows, increment the number by 256 for each packet.
- **TCP initial window** - This fingerprint technique works by tracking the window size in packets returned from the target device. Many operating systems use exact sizes that can be matched against a database to uniquely identify the OS.
- **ACK value** - Here again, vendors differ in the way they have implemented the TCP/IP stack. Some operating systems send back the previous value plus 1; others send back random values.
- **Type of service** - This fingerprinting type tweaks ICMP port unreachable messages and examines the value in the Type of Service (TOS) field. Some use 0; others return different values.
- **TCP options** - Here again, different vendors support TCP options in different ways. By sending packets with different options set, the responses will start to reveal the server's fingerprint.
- **Fragmentation handling** - This fingerprinting technique takes advantage of the fact that different OS vendors handle fragmented packets differently. RFC 1191 specifies that the maximum transmission unit (MTU) is normally set between 68 and 65,535 bytes.

[Gregg, 2015, S. 164ff]

Auch hier lassen sich einige dieser Merkmale passiv Nutzen, wenn man den Netzwerkverkehr des Zielsystems zumindest teilweise mitlesen kann.

2.1.5 SNMP

Bei SNMP (Simple Network Management Protocol) handelt es sich um das wohl bekannteste Protokoll zur Netzwerkkonfiguration und -überwachung.
Es gibt eine oder mehrere Managementstationen, die mit den zu verwaltenden Netzwerkkomponenten kommunizieren. Hierbei gibt es zwei Kommunikationsformen:

1. Frage-Antwort - Eine Managementstation fordert Informationen über den Zustand eines Teils der befragten Komponente oder eine Zustandsänderung an. Die Komponente gibt dann die gewünschten Informationen zurück bzw. quittiert die Zustandsänderung.
2. Nachrichten - Eine überwachte Komponente informiert die Managementstation über bestimmte Ereignisse.

Um den Overhead gering zu halten und so den eigentlichen Netzwerkbetrieb möglichst wenig zu beeinflussen, wird als Transportprotokoll das verbindungslose UDP genutzt. Ansonsten betrüge der Verwaltungsaufwand zur Sicherstellung der korrekten Auslieferung bereits ein Vielfaches der eigentlichen SNMP-Inhalte.
Um die Kommunikation zwischen Managementstationen und Netzwerkkomponenten zu vereinfachen, dienen so genannte Agenten, die in der Regel vom Hersteller der Komponente in diese implementiert sind.
Die zu managenden Objekte und Attribute der Komponenten sind in einer hierarchischen Struktur angeordnet und erhalten eine sich daraus ergebende einzigartige OID (Object

Identifier) über die sie angesprochen werden können. Eine OID besteht aus einer Kette von durch Punkte getrennten Zahlen, die zur besseren Lesbarkeit jeweils über textuelle Pendants verfügen. Eine OID zeigt dabei auf einzelne Objekte oder eine Tabelle, die wiederum mehrere Objekte enthält.
Die OIDs werden nicht einzeln definiert, sondern sondern in MIB (Management Information Base) genannten Gruppen zusammengefasst. Diese werden in der SMI (Structure of Management Information), einer Sprache mit eindeutiger Syntax, definiert.
Der Agent einer Netzwerkkomponente kann mehrere MIBs implementieren um z.B. Zugriff auf bestimmte herstellerspezifische Objekte zu gewähren. Eine Standard-MIB, die jedes SNMP-fähige Gerät implementieren muss ist die MIB-II über die bereits viele grundlegende Systemeigenschaften per SNMP verwaltet werden können. So ist es z.B. möglich Informationen über das Routing oder offene Ports abzurufen.

Die aktuelle Version SNMPv3, verfügt zudem über wichtige Sicherheitsmechanismen und gilt damit als das Standard-Protokoll für sicheres Netzwerkmanagement.
[vgl. Schwenkler 2006, S. 69ff]

2.1.6 DNS

DNS (Domain Name System) ist auf Schicht 7 angesiedelt und dient hauptsächlich dazu, Hostnamen auf Netzwerkadressen abzubilden. Somit ist es möglich, Ressourcen durch intuitiv erfassbare Namen nutzen zu können, ohne sich mit den auf Netzwerkebene genutzten numerischen Adressen zu befassen. Des Weiteren bleiben die Namen von der eigentlich Adresse entkoppelt, wodurch der Umzug einer Ressource zu einer anderen Adresse, bei entsprechender Änderung im DNS, für Nutzer transparent bleibt.
Um dies möglich zu machen wurde mit DNS ein hierarchisches, Domain-basiertes Namensschema sowie ein verteiltes Datenbanksystem, das selbiges umsetzt, eingeführt.

Die Spitze der Namenshierarchie ist in über 250 Top-Level-Domains aufgeteilt und wird von der ICANN (Internet Corporation for Assigned Names and Numbers) verwaltet. Jede Domain kann weiter in Subdomains unterteilt werden, welche wiederum unterteilbar sind, so dass sich eine Baumstruktur ergibt, die DNS-Namensraum genannt wird. Die Blätter repräsentieren dann einen Host oder auch eine Organisation mit einer Menge von Hosts.
Grundsätzlich besteht jede Domain aus dem kompletten Pfad bis zur (unbenannten) Wurzel. Die einzelnen Komponenten sind dabei durch Punkte getrennt.

Jede Domain bestimmt selbst, wie untergeordnete Domains zugewiesen werden. Daher braucht es für das Erstellen einer neuen Domain stets die Erlaubnis der übergeordneten Domain. So hat diese stets volle Kontrolle über all ihre Subdomains und Namenskonflikte werden vermieden. Subdomains können wiederum autonom weitere Subdomains vergeben. Die Namensgebung richtet sich dabei in der Regel nach Organisationsgrenzen statt nach physischen Netzwerken. So können Hosts eines Netzwerks (das beispielsweise von verschiedenen Organisationseinheiten gemeinsam genutzt wird) in verschiedenen Domains liegen. Andersherum ist es ebenso möglich, verschiedene, auch räumlich getrennte Netzwerke unter einem Domainnamen zusammenzufassen.

Die DNS-Datenbank einer Domain besteht aus einer Menge an Resource Records. Dies sind 5-Tupel, die aus dem Namen der betreffenden Domain, einer TTL, der Klasse (andere Werte als IN, was für Internet steht, sind unüblich),dem Typ (siehe Tabelle unten) und einem entsprechenden Wert besteht.

Die wichtigsten DNS-Resource-Record-Typen:

Typ	Bedeutung	Wert
SOA	Start of authority	Parameter für diese Zone
A	IPv4-Adresse eines Hosts	32-Bit Integer
AAAA	IPv6-Adresse eines Hosts	128-Bit Integer
MX	Mail exchange	Priorität und Domain die E-Mails akzeptiert
NS	Nameserver	Name eines Nameservers für diese Domain
CNAME	Canonical name	Domainname
PTR	Pointer	Alias für eine IP-Adresse
SPF	Sender policy framework	Informationen über berechtigte E-Mail-Versender
SRV	Service	Host der einen Service anbietet
TXT	Text	Beschreibender ASCII-Text

Der DNS-Namensraum ist aufgeteilt in sich nicht überlappende Zonen, die als Teilbäume betrachtet werden können. Die Grenzen der Zonen können von den entsprechenden Administratoren festgelegt werden. Jede Zone verfügt über einen oder mehrere Hosts, Nameserver genannt, die die Datenbank für die Zone bereitstellen: Ein primärer Nameserver und ggf. weitere sekundäre Nameserver, die die Informationen von ersterem beziehen.

Stellt man eine Anfrage an einen Nameserver, kann es sich bei den Ergebnissen um authoritative oder gecachte Einträge handeln. Erstere stammen ausschließlich direkt von den verantwortlichen Nameservern der angefragten Domain. Letztere sind, selbst wenn sie gerade erst bezogen wurden, für die Zeit der TTL gespeicherte Einträge. Durch das Caching wird der Vorgang beschleunigt, es besteht jedoch das Risiko, dass Einträge vor Ablauf der TTL veralten. Der Prozess, einen Namen auf eine Adresse abzubilden nennt sich Namensauflösung. Dabei wird eine Anfrage an einen lokalen Nameserver gestellt, der, sofern keine gecachten Einträge vorhanden sind, eine Anfrage an einen Rootserver stellt. Diese Server besitzen Informationen über die Top-Level-Domains, und ihre Adressen werden in der Regel beim Start des DNS-Server aus einer Systemkonfigurationsdatei in den Cache geladen. Vom Rootserver erhält der lokale Nameserver für gewöhnlich nur Informationen über den Nameserver der Top-Level-Domain des angefragten Namens. So arbeitet sich der lokale Nameserver mit weiteren Anfragen die Hierarchie herunter, bis er beim autoritativen Nameserver der angefragten Domain angekommen ist und gibt schließlich die gewünschte Antwort zurück.

Da der lokale Nameserver, um die vollständige Antwort auf die Anfrage zu geben, selbst bei weiteren Nameservern angefragt hat, nennt man seine Arbeitsweise rekursiv. Die anderen Nameserver haben keine weiteren Anfragen an andere Nameserver gestellt und nur einen Teil der Frage beantwortet. Ihre Arbeitsweise wird als iterativ bezeichnet.
[vgl. Tanenbaum, 2011, S. 611ff]

Es gibt u.a. verschiedene Techniken, die sich DNS bedienen, mit denen Geräte gefunden werden und ggf. weitere Informationen ermittelt werden können:

- Forward DNS Lookup - Gibt die IP-Adresse für einen gegebenen Hostnamen zurück. Für eine gegebene Domain können ggf. weitere Informationen wie der authoritative Nameserver, Mailserver oder Services innerhalb der Zone zurückgegeben werden. Außerdem kann über den SOA-Eintrag der primäre Nameserver für die entsprechende Zone ermittelt werden, bei dem der im nachfolgenden erklärte DNS Zone Transfer versucht werden kann.
  Zudem gibt es die Möglichkeit Brute-Force einzusetzen: So können z.B. valide Hostnamen (und deren IP-Adressen) mittels einer Wörterbuch-Attacke ermittelt werden.
- Reverse DNS Lookup - Gibt vor allem den Hostname für eine gegebene IP-Adresse zurück. Für den Fall, dass es einen Eintrag gibt, ist dies als Hinweis auf die Existenz eines entsprechenden Netzwerkteilnehmers anzusehen.
- DNS Zone Transfer - Diese Methode ist eigentlich dazu gedacht, dass weitere, redundante Nameserver aktuelle Daten vom primären Nameserver abfragen können. Sofern keine dies verhindernden Sicherheitsmechanismen konfiguriert wurden, kann ein Zone Transfer von jedem dazu genutzt werden, den vollständigen Datensatz abzufragen. Dieser kann dann Informationen (z.B. über weitere potentielle Ziele) enthalten, die nicht für die Öffentlichkeit bestimmt sind.

[vgl. Baloch, 2015, S. 75ff; Svensson, 2016, S. 71ff]

2.1.7 Service Discovery Protokolle

Service Discovery Protokolle dienen dem Auffinden von Diensten in Netzwerken. Dies geschieht entweder über ein zentrales Verzeichnis registrierter Dienste, das abgefragt werden kann oder die Anfragen erfolgen per Broadcast-Nachricht und die Anbieter von Diensten antworten direkt.
Somit könnten Protokolle wie SSDP als Teil von UPnP und DNS-SD als Teil von Zeroconf-Implementation wie Apple Bonjour oder Avahi durchaus von Interesse für die Dienstsuche in IP-Netzwerken sein. Allerdings ist hier stets fraglich, inwiefern sie in zu untersuchenden Netzen verfügbar sind und falls ja, ob die ermittelten Informationen aktuell und vollständig sind. Daher soll im Folgenden der Fokus auf andere Ansätze gelegt werden.

2.3 Werkzeuge

In diesem Abschnitt soll Software vorgestellt werden, die sich der zuvor beschriebenen technischen Möglichkeiten bedient und bei der Informationsbeschaffung von Nutzen sein kann.
Da ausgereifte Werkzeuge für die gleiche Anwendungsdomäne die technischen Möglichkeiten in der Regel ähnlich nutzen, werden für die nähere Betrachtung im Folgenden je ein oder zwei

repräsentative Werkzeuge aus verschiedenen Gattungen von Werkzeugen ausgewählt. Eine Ausnahme bilden hier die Linux-Systemprogramme, da sie unterschiedlichen Zwecken dienen.

Um Kandidaten für zu betrachtende Werkzeuge zu finden, werden Hinweise aus der Literatur sowie die Top-Listen von http://sectools.org[2] genutzt. Diese werden ggf. weiter mit https://trends.google.de[3] verglichen.
Unter der Annahme, dass Google Trends Daten im Wesentlichen korrekt sind und es einen Zusammenhang zwischen Beliebtheit eines Programmnamens als Suchbegriff und der Bedeutung des Programms für das jeweilige Themengebiet gibt, lassen sich so die relevantesten Werkzeuge für eine Domäne ausmachen.

2.3.1 Linux-Systemprogramme

Mit Zugang zu einem System lassen sich direkt von dort Informationen auslesen. Dies kann z.B. über einen Fernzugang geschehen oder über einen Agenten, der auf dem Zielsystem ausgeführt wird. Mit einem privilegierten Zugang zu einem Gerät lassen sich die meisten und zuverlässigsten Informationen beschaffen. Daher soll an dieser Stelle auf die in der Regel standardmäßig vorhandenen Systemprogramme, die sich zur Informationsbeschaffung eignen, eingegangen werden.

Betriebssystem
Das Betriebssystem kann in der Regel recht einfach ermittelt werden. Auf Linux-Systemen lassen sich Informationen zu Betriebssystem, Kernel-Release, -Version etc. mit dem *uname*-Kommando ausgeben. [vgl. Negus, 2015, S. 72]
Gegebenenfalls lassen sich genauere Informationen zur jeweiligen Distribution über das Kommando *lsb_release* anzeigen. [vgl. Linux Foundation, 2015]

Softwarestand
Der Softwarestand lässt sich über die Paketverwaltung ermitteln. Weit verbreitete Beispiele sind der RPM Package Manager und der Debian Package Manager.
Eine komplette Liste der installierten Pakete erhält man z.B. wie folgt:
rpm -qa bzw. *dpkg-query -l*
Details, wie z.B. die Version, zu Paketen erhält man mit:
rpm -qi <Paketname> bzw. *dpkg -s <Paketname>*
[vgl. Hertzog, 2015, S. 90; Debian, 2015; Bailey, 1997, S.57/61]

Nutzerkonten und Gruppen
In der Datei */etc/nsswitch.conf* ist festgelegt, wo bestimmte wichtige Systeminformationen zu finden sind. Das Kommando *getent* greift darauf zurück.
[vgl. Negus, 2015, S. 187]

[2] laut Gregg der beste Ort um Sicherheitstools zu finden [vgl. Gregg, 2015, S. 31]
[3] Diese Seite visualisiert Beliebtheit und Trends von Suchbegriffen. Bis zu 5 Suchbegriffe können direkt verglichen werden. Um Verzerrung der Ergebnisse zu vermeiden lassen sich weiterhin Themengebiete einschränken: Im Allgemeinen ist der Suchbegriff "satan" aufgrund seiner Mehrdeutigkeit deutlich beliebter als "nessus" (beides Namen von Schwachstellenscannern), schränkt man die Analyse aber auf das Themengebiet "Netzwerksicherheit" ein, so kehrt sich das Ergebnis um.

Mit *getent passwd* erhält man als Ausgabe eine Liste mit einer Zeile pro Nutzer. Wird diese entsprechend geparst, lassen sich u.a. Informationen wie Nutzername, User-ID, Gruppen-ID, Home-Verzeichnis und Login-Shell auslesen.

Mit *last <username>* lassen sich Informationen über die letzten Logins eines Nutzers abrufen.
[vgl. Kerrisk, 2017, last]

Mit *chage -l <username>* lässt sich zudem feststellen, wann das Passwort eines Nutzer zuletzt geändert wurde, wann es abläuft etc.
[vgl. Negus, 2015, S. 594]

Auch Gruppen und deren Mitglieder lassen sich mit *getent group* auslesen.
Einfacherweise können Mitglieder der Gruppe *sudo* ausgelesen werden um Nutzer mit privilegierten Rechten zu finden. Bei einer gründlichen Vorgehensweise müsste hier aber die Konfiguration der Datei */etc/sudoers* analysiert werden.
[vgl. Kerrisk, 2017, getent]

Zudem können wertvolle sicherheitsrelevante Informationen aus der */etc/ssh/sshd_config* gewonnen werden. Beispielhaft seien folgende genannt:

- **AllowUsers <username1> <username2> ...** - bestimmt, dass nur die aufgelisteten Nutzer sich per SSH einloggen können
- **AllowGroups <groupname1> <groupname2> ...** - bestimmt, dass Nutzer, die zu den definierten Gruppen gehören sich per SSH einloggen dürfen, allerdings nur, wenn AllowUsers nicht gesetzt ist
- **PermitRootLogin = without-password/prohibit-password** - verbietet es root sich nur mit einem Passwort einzuloggen.
- **PasswordAuthentication yes** - zeigt an, dass es für normale Nutzer möglich ist, sich nur mit einem Passwort einzuloggen.

[vgl. OpenBSD, 2017, SSHD]

Netzwerk
Zur Netzwerkadministration von Linux-Systemen, werden in der Literatur oftmals Programme wie netstat aus dem net-tools-Paket empfohlen. Dieses Paket ist jedoch auf einigen Linux-Distributionen in den Manpages als deprecated gekennzeichnet und teilweise nicht einmal mehr Teil der Standard-Konfiguration. Auch auf der projekteigenen Manpage wird netstat als obsolet bezeichnet und ss (Teil des Iproute2-Pakets) als Nachfolger genannt. [vgl. Sourceforge, 2013]
Einer der Maintainer begründet, die Entscheidung, net-tools zu ersetzen wie folgt:
"[Net-tools] doesnt [sic] support many of the modern features of the linux kernel, the interface is far from optimal and difficult to use in automatisation, and also, it hasn't got much love in the last years.
On the other side, the iproute suite, introduced around the 2.2 kernel line, has both a much better and consistent interface, is more powerful, and is almost ten years old, so nobody would say it's untested." [Ferrari, 2009]

Bei Iproute2 handelt es sich genau wie bei net-tools um eine Sammlung von Werkzeugen. Für die Informationsbeschaffung im Rahmen dieser Arbeit sind ip und ss interessant:

- ip addr - zeigt die Konfiguration der Netzwerkinterfaces
- ip route show - gibt die Routing-Tabellen aus
- ss - zeigt Informationen zu Netzwerksockets

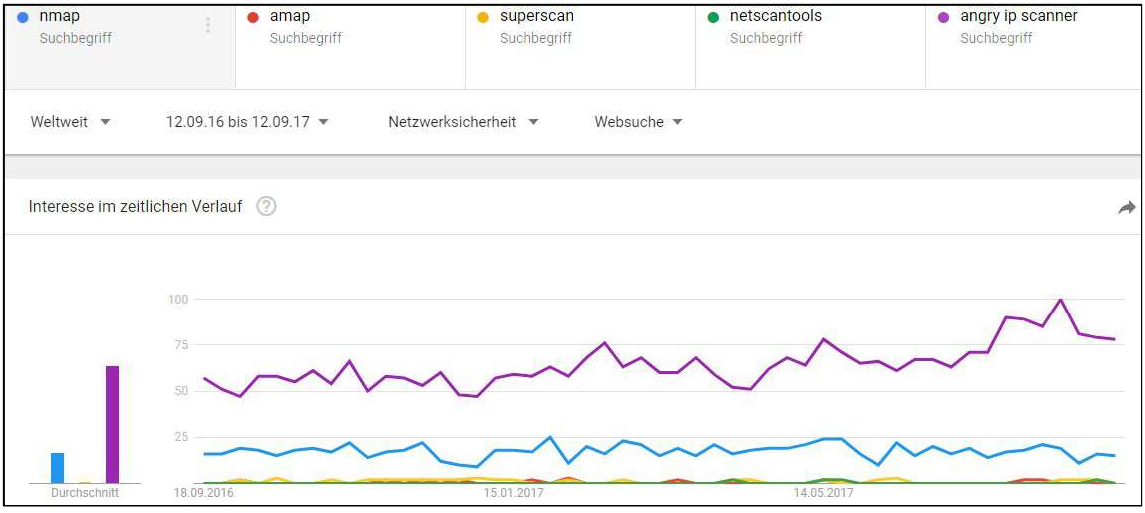
Um nur Ports anzuzeigen, auf denen Dienste bereitstehen, lassen sich in diesem Fall bei ss die gleichen Parameter wie bei netstat nutzen:

-t, --tcp	zeigt TCP-Sockets
-t, --udp	zeigt UDP-Sockets
-l, --listening	zeigt nur lauschende Sockets [Zustand "LISTEN" (TCP) bzw. "UNCONN" (UDP)]
-p, -processes	zeigt den Prozess an, der das Socket nutzt
-n, --numeric	verhindert den Versuch, Portnummern zu Dienstnamen aufzulösen

[Kerrisk, 2017, ss]

2.3.2 Portscanner

Am meisten Erwähnung in der Literatur findet Nmap. Gregg stellt außerdem Super-Scan und THC-Amap vor. [vgl. Gregg, 2015, S.183] Bei der Analyse mit Google Trends unter Einbeziehung der auf in Sectools genannten Scanner zeigt interessanterweise Angry IP Scanner als den beliebtesten. [Sectools, 2017, SCAN]



[Trends, 2017, SCAN]

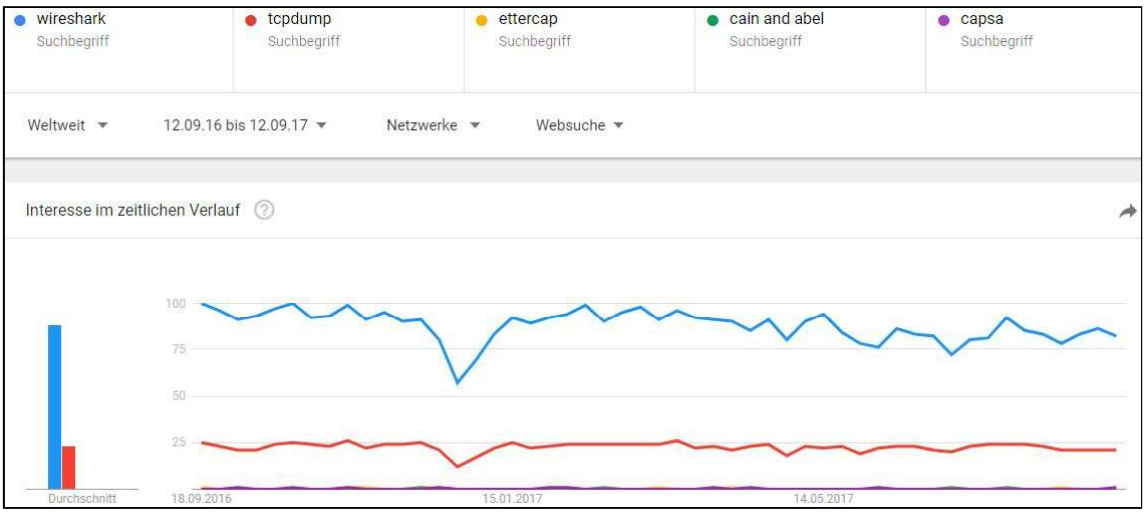
Dies mag einer einfacheren Bedienung geschuldet sein, durch die das Werkzeug einer breiteren Menge an Interessierten zugänglich wird.
In der Literatur wird jedoch oftmals nur Nmap erwähnt und es werden keine Alternativen betrachtet, wenn es um das Thema Portscanning geht. Zudem gibt es einige Bücher, die sich

explizit mit der Benutzung von Nmap für Netzwerkentdeckung und Sicherheitsaudits befassen, was die Beliebtheit von Nmap weiter unterstreicht. Aus diesem Grund soll im folgenden dennoch Nmap betrachtet werden.

Nmap ist ein Kommandozeilen-Tool, das 1997 erstmals veröffentlicht und von einer aktiven Community konsequent weiterentwickelt wird. [Lyon, 2016, HF] Mit Zenmap verfügt es auch über eine GUI, die in der Lage ist Topologien darzustellen und mehrere Scanergebnisse miteinander zu verbinden.
Neben einer Vielzahl von Portscanning-Techniken besitzt Nmap auch Optionen zur Betriebssystemerkennung per TCP/IP-Stack-Fingerprinting sowie zur Service-Erkennung. [vgl. Gregg, 2015, S.183]
Darüber hinaus verfügt Nmap mit der NSE (Nmap Scripting Engine) über ein weiteres Modul, mit dem sich ein zahlreiche Netzwerkaufgabe automatisieren lassen: So kann Nmap in der Programmiersprache Lua geschriebene Scripte in seine Scans einbinden, wodurch Ziele u.a. direkt auf bestimmte Schwachstellen geprüft werden können. [vgl. Lyon, 2016, NSE]
Zudem bietet Nmap mit seinem XML-Output ein gut parsbare, standardisierte Ausgabe, die durch ihre Erweiterbarkeit auch Features neuerer Versionen aufnehmen kann, ohne dass Programme, die auf den Output älterer Versionen bauen, nicht mehr funktionieren.
[vgl. Lyon, 2016, XML]

2.3.3 Network Sniffer

Ist erst einmal der Zugriff auf den fremden Datenverkehr gegeben, gibt es verschiedene Werkzeuge zur Auswertung.
Gregg nennt Wireshark als das beste Werkzeug zur Netzwerkanalyse, das es auf dem Markt gibt. Vergleicht man die weiterhin genannten Tools tcpdump, NetworkMiner, Capsa und andere sowie die Top 10 von Sectools (u.a. Cain and Abel, Ettercap, dsniff) mit Google Trends, so erkennt man, dass Wireshark deutlich an der Spitze ist. [vgl. Gregg, 2015, S.99 und S. 115; Sectools, 2017, SNIFF] tcpdump nimmt hier ebenso deutlich den zweiten Platz ein, während der Rest sich im Vergleich mit den beiden Spitzenreitern untereinander kaum noch abhebt.



[Trends, 2017, SNIFF]

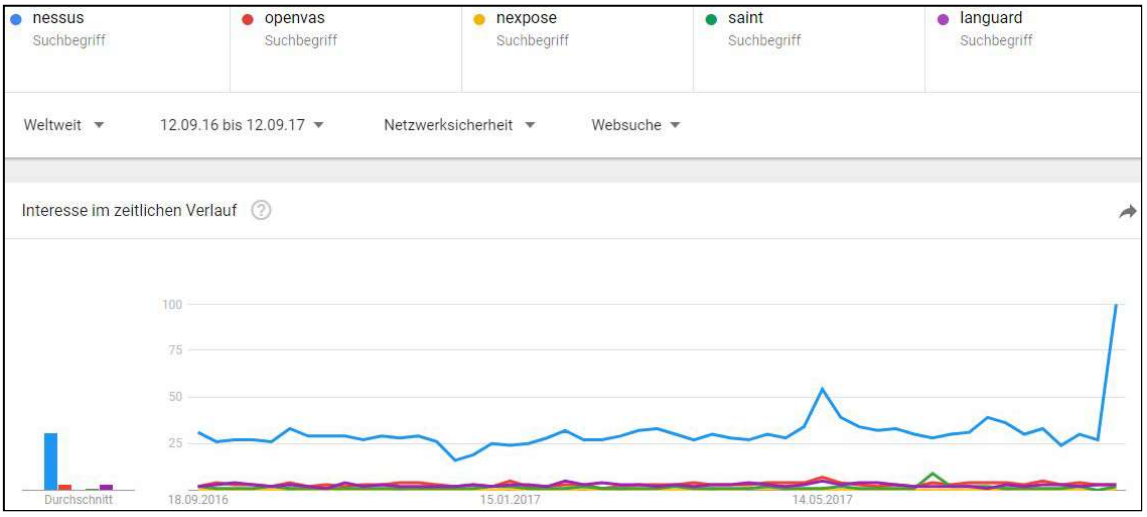
Sowohl Wireshark als auch tcpdump können den kompletten Netzwerkverkehr aufzeichnen. Als GUI-Tool bietet Wireshark u.a. farbliche Hervorhebungen, zahlreiche Filtermöglichkeiten oder das Verfolgen von TCP-Streams an. Für die manuelle Auswertung ist es damit das Werkzeug der Wahl.
Bei tcpdump hingegen handelt es sich um ein leichtgewichtiges Kommandozeilen-Tool und lässt sich damit sehr gut automatisiert nutzen. Das Ausgabeformat ist außerdem kompatibel mit Wireshark, so dass auch eine manuelle Auswertung komfortabel vorgenommen werden kann.

Abschließend sei erwähnt, dass es auch Werkzeuge gibt, die den Netzwerkverkehr auf ganz bestimmte Daten untersuchen. So beinhaltet die Tool-Suite Dsniff verschiedene Werkzeuge z.B. um gezielt nach Mailverkehr, Bildern oder URLs im mitgeschnittenen Verkehr zu suchen.

2.3.4 Schwachstellenscanner

Schwachstellenscanner (engl. Vulnerability Scanners) sind Systeme, die Geräte, Netzwerke oder Anwendungen auf potentielle Schwachstellen, die von einem Angreifer genutzt werden könnten, überprüft.
Zu diesem Zwecke werden Sondierungspakete ausgesandt und die Antworten mit bekannten Daten abgeglichen (Fingerprinting). So können Merkmale wie offene Ports, Dienste, Betriebssysteme und letztlich auch Schwachstellen ermittelt werden. [vgl. Baloch, 2015, S. 121]
Bei diesem aktiven Ansatz kann zum einen die genaue Version verwendeter Software ermittelt und mit Schwachstellendatenbanken abgeglichen werden. Zum anderen kann eine (möglichst) harmlose Variante einer Attacke unter Ausnutzung einer potentiell vorhandenen Schwachstelle ausgeführt werden, um zu beobachten, ob diese gelingt oder nicht.
Manche Schwachstellenscanner können auch passiv arbeiten, indem sie den Netzwerkverkehr beobachten, um anhand der fließenden Pakete Software und deren Versionen zu ermitteln. Zudem gibt es agentenbasierte Ansätze, bei denen ein Programm auf den Netzwerkgeräten läuft, welches Informationen an eine zentrale Stelle übermittelt. Durch den direkten Zugriff auf lokale Software und die Konfiguration ist dieser Ansatz sehr genau. Manche Agenten können auch den Netzwerkverkehr beobachten und so helfen, Geräte zu finden, die noch nicht über einen Agenten verfügen. [vgl. Pompon, 2016, S.171]

Bekannte Vertreter aus dieser Kategorie sind u.a. Nessus, OpenVAS, QualysGuard und Nexpose. Unter Einbeziehung der auf Sectools genannten Werkzeuge (u.a. GFI Languard, SAINT), ergibt sich bei der Analyse mit Google Trends Nessus als die mit Abstand beliebteste Software, gefolgt von OpenVAS. [vgl. Sectools, 2017, VULN]



[Trends, 2017, VULN]

Tatsächlich werden unter dem Namen Nessus mehrere Systeme vertrieben: Zum einen gibt es die kostenfreie Variante "Nessus Home" für Privatanwender und das kostengünstigere "Nessus Professional" für die Verwendung mit nur einem Benutzer. Am umfangreichsten (und teuersten) ist "Nessus Manager". Diese Variante ist für den Mehrbenutzereinsatz in Unternehmen ausgelegt und kann u.a. mehrere Scanner von einer Zentrale managen, bietet die Verwendung von Agenten an und lässt sich mit Patch Management Systemen integrieren. [vgl. Tenable, 2017, Nessus; Tenable, 2017, Obtain][4]
Zudem gibt es den "Nessus Network Monitor", welcher den Netzwerkverkehr kontinuierlich passiv analysiert und damit u.a. dabei hilft Geräte zu bemerken, die zum Zeitpunkt eines sporadischen Scans nicht aktiv waren.
[vgl. Tenable, 2017, Monitor]
Nessus verfügt über eine REST API über die der Service auch ohne Verwendung der Web-GUI genutzt werden kann.

2.3.5 sonstige Werkzeuge

Ping
Es gibt verschiedene Implementationen von Ping-Tools. Das vor allem die Überprüfung der Erreichbarkeit von Hosts zum Anwendungsfall hat. Hierbei wird ein ICMP-"Echo-Request"-Paket an eine IP-Adresse gesendet. Ist ein Host unter dieser Adresse erreichbar sollte er mit einem ICMP-"Echo-Reply"-Paket antworten. Andernfalls sollte der zuständige Router mit einem ICMP-"Destination-Unreachable"-Paket antworten. [vgl. Tanenbaum, 2011, S. 465ff; RFC1812, 1995, S. 55ff]

Traceroute
Traceroute-Algorithmen werden ebenfalls von verschiedenen Programmen implementiert und eignen sich u.a. zum aktiven Sondieren von Routen.

[4] Für Angaben zu den Nessus-Produkten und deren Features wird in Ermangelung anderer Quellen auf die offizielle Webseite des Herstellers verwiesen, wo diese Informationen zum Zeitpunkt des Abrufs bereitgestellt wurden.

Hierbei wird die Route vom Ausgangspunkt zu einem Ziel ermittelt, indem Pakete mit sich fortlaufend erhöhender TTL ausgesandt werden. Router auf dem Weg sollten die TTL vor dem Weiterleiten reduzieren und bei Erhalt eines Pakets mit einer TTL von 0 an den Ausgangspunkt mit einer ICMP TTL-expired-Nachricht antworten. Anhand der erhaltenen Antworten lässt die Vorwärtsroute zu einem Ziel ermitteln. [vgl. Siamwalla, 1998, S. 3]
Das Ergebnis eines Traceroute-Durchlaufs nennt sich Trace und besteht im Wesentlichen aus einer sequentiellen Liste der IP-Adressen der Router auf dem ermittelten Weg (Hops).

Netzplan-Tools
Um Software zu finden, die automatisch einen Netzplan generiert, wurden verschiedene Network Monitoring Systeme betrachtet, die diese Funktionalität anbieten. Viele davon, wie z.B. die von Solarwinds oder Spiceworks, sind nur unter Windows nutzbar und wurden daher nicht weiter betrachtet. Die unter Linux lauffähigen Systeme netTransformer, OpenNMS, The Dude und SmartHawk, bedienen sich SNMP um das Netzwerk zu erkunden.
Andere bei der Recherche gefundene Systeme wie Auvik und Netbrain nutzen SSH um einen direkten Zugang zu den Systemen zu erhalten.
Da sich die relevanten Topologie-Informationen mit einem SNMP- oder SSH-Zugang ohnehin problemlos gewinnen lassen, würde die Verwendung dieser Systeme vor allem großen Ballast für *ANIS* bedeuten, ohne großen Mehrwert zu bieten.
Hier würden viele Voraussetzungen und starke Abhängigkeiten nur für die Erstellung von Netzplänen entstehen. Daher soll die Verwendung dieser komplexen, domänenspezifischen Systeme zu vermieden, und die Erstellung der Netzpläne von der Informationsgewinnung entkoppelt werden.

dig
dig (domain information groper) ist ein Kommandozeilenprogramm mit dem sich DNS-Server abfragen lassen.
[OpenBSD, 2017, DIG]

Net-SNMP
Net-SNMP ist eine Software-Suite zur Nutzung von SNMP und ist für viele Linux-Distributionen bereits Teil der Standardkonfiguration oder steht zumindest zur einfachen Installation über den Paketmanager zur Verfügung.
So gibt es diverse grundlegende Programme wie snmpget, mit dem sich einzelne Werte abfragen lassen oder z.B. snmpwalk, welches mehrere Anfragen ausführt und die Werte ganzer SNMP-Teilbäume zurückgibt.
Des Weiteren gibt es einige "second level"-Programme, die eine komfortablere Nutzung ermöglichen: So lassen sich z.B. mit snmptable SNMP-Tabellen Abfragen und in tabellarischer Form darstellen. Und mit snmpnetstat lassen sich auf einfache Weise netzwerkbezogene Daten wie aktive Sockets oder Routingtabellen abfragen und komfortabel darstellen.
[vgl. Net-SNMP, 2011]

3. Anforderungen

In diesem Abschnitt soll festgelegt werden, welche funktionalen und nicht-funktionalen Anforderungen eine mögliche Lösung für das in der Einleitung beschriebene Ziel der Arbeit

erfüllen sollte. Des Weiteren soll erörtert werden, inwiefern sich die im Grundlagenabschnitt vorgestellten Werkzeuge zu diesem Zweck nutzen lassen. Die Nutzung eines direkten Zugangs bietet dabei oft einfache Möglichkeiten, die bereits in Abschnitt 2.3.1 erwähnt wurden, und werden daher nur erwähnt, wenn es entsprechende ergänzende Informationen gibt.

3.1 Funktionale Anforderungen

3.1.1 FA1: Entdeckung von Geräten

Unter der Annahme, dass keine lückenlose Liste von IP-Adressen der zu inventarisierenden Geräte vorliegt, sind in der Regel nur IP-Adressbereiche bekannt, die es zu erkunden gilt. Dies ist ohnehin ratsam, da sich auch stets Geräte im Netzwerk befinden könnten, die z.B. unerlaubt hinzugefügt oder vergessen worden sind.
Zu beachten ist dabei, dass es sich besonders bei der Entdeckung von Geräten stets um eine Momentaufnahme handelt, denn, egal ob autorisiert oder unautorisiert, Geräte werden zum Netzwerk hinzugefügt und entfernt, ein- und ausgeschaltet.

Nutzung von Ping
Diese Methode allein gilt im Allgemeinen als vergleichsweise unzuverlässig: "Unfortunately for network explorers, many hosts and firewalls now block these packets, rather than responding as required by RFC 1122. For this reason, ICMP-only scans are rarely reliable enough against unknown targets over the Internet." [Lyon, 2016, HD]
Zwar beziehen sich Aussagen wie diese in der Regel auf eine Sondierung von außerhalb des Zielnetzwerks, erfahrungsgemäß ist es jedoch keine Seltenheit, dass Ping-Anfragen auch innerhalb eines Netzwerks nicht beantwortet werden.

Nutzung von Nmap
Um die Zuverlässigkeit der Hostentdeckung zu steigern, wird empfohlen, diese nicht isoliert zu betrachten, sondern in einem Schritt mit einem Portscan erfolgen zu lassen. [vgl. Engebretson, 2013, S. 83]
So werden für Nutzer mit privilegierten Rechten und Zieladressen außerhalb des eigenen Subnetzes bei der Standard-Hostentdeckung Nmaps gleich vier Pakete ausgesandt: Ein ICMP-"Echo-Request", ein TCP-SYN-Paket an Port 443, ein TCP-ACK-Paket an Port 80 und ein ICMP-"Timestamp-Request". An den Antworten auf die TCP-Pakete lässt sich ggf. noch mehr ablesen, als nur die Verfügbarkeit von Zielen. Die angestrebte und entscheidende Information bezüglich der Hostentdeckung ist aber, ob überhaupt eine Antwort vom Gerät kommt. Ist dies der Fall, so kann man in der Regel annehmen, dass das Gerät existiert. [vgl. Lyon, 2016, HD]
Letztlich werden die Möglichkeiten einer Hostentdeckung zuverlässiger, je mehr die Möglichkeiten ausgeschöpft werden: Der ursprüngliche Autor Nmaps, Gordon Lyon, berichtet dazu, dass er bei der Durchführung von Sicherheitsaudits zunächst die üblichsten 1000 TCP-Ports mit weiteren umfassenden Optionen zur Hostentdeckung durchführe (weitere Pings unter Nutzung von TCP-SYN,-ACK auf bestimmten Ports sowie weitere ICMP-Anfragen), um schnell mit der Arbeit beginnen zu können. Anschließend würde ein weiterer Scan aller 65.536 TCP-Ports mit abgeschalteter Hostentdeckung (da Nmap den Portscan ansonsten nur für IP-Adressen durchführt, von denen im Rahmen der Hostentdeckung geantwortet wurde) im Hintergrund gestartet. Dieser Scan kann erst Tage später abgeschlossen sein und wird dann

mit den Ergebnissen des ersten Scans verglichen, um neu entdeckte Geräte und Ports auszumachen. [vgl. Lyon, 2008, S. 71]

Nutzung von Nessus

Nessus führt zur Hostentdeckung außerhalb des eigenen Subnetzes neben ICMP-Pings, sofern nicht anders konfiguriert, TCP-SYN-Scans auf bestimmten Default-Ports durch.
Gegebenenfalls ist auch eine Entdeckung per ARP und UDP möglich. [vgl. Tenable, 2017, S. 131 ff] Auch wenn Nmap noch mehr Optionen zur Hostentdeckung anbietet, sind die wichtigsten davon auch bei Nessus vorhanden. Will man von weiteren Möglichkeiten Gebrauch machen, um diese innerhalb von Nessus weiter zu nutzen, ist es auch möglich, die Ergebnisse von Nmap zu importieren. [Asadoorian, 2009][5]

Nutzung von Netzwerksniffern

Ist Zugriff auf den Verkehr gegeben, können aktive Geräte ausgemacht werden.
Es gibt aber Einschränkungen die auch für die folgenden Bewertungen der Nutzbarkeit für andere Anforderungen gelten:
Auch wenn das passive Belauschen des Netzwerkverkehrs Vorteile bringt wie Schonung der Ressourcen und die Möglichkeit zu jedem Zeitpunkt (im Gegensatz zu einem bestimmten Scan-Zeitpunkt) Aktivität von Geräten zu bemerken, so gibt es doch auch einige Nachteile. So muss zuerst einmal ein geeigneter Knotenpunkt ausgemacht werden, der sich gut eignet den Verkehr zu belauschen, und selbst dann ist die Reichweite dieser Technik beschränkt, sofern kein Broadcast-Medium verwendet wird und nicht weitere Maßnahmen getroffen werden, um den Verkehr dem überwachenden Knoten zugänglich zu machen. Außerdem kann Aktivität nur beobachtet werden, wenn sie stattfindet. Daher werden die Ergebnisse über den Zeitraum der Beobachtung zwar genauer, es gibt aber keinen Hinweis darauf, ob die gewonnen Daten vollständig sind.
Dieser Ansatz erfordert zudem bereits ein größeres Wissen über, und wahrscheinlich weitere Veränderungen im, bestehenden Netzwerk.

Nutzung von DNS

Um Geräte per DNS zu entdecken bietet sich vor allem ein Reverse Lookup aller Adressen im Zielbereich an, da hier die Chance besteht, dass auch Einträge für temporär nicht verfügbare Geräte im Zieladressbereich existieren.
Die auf diese Weise gewonnen Hostnamen lassen sich weiter für DNS-Abfragen nutzen, um z.B. zuständige Name- und Mailserver für die Domain zu finden. Auch ein Zone Transfer könnte versucht werden, benötigt aber wahrscheinlich weitere Konfiguration auf Seiten des primären DNS-Servers.
Die Zuverlässigkeit der per DNS ermittelten Daten hängt immer von der Aktualität der DNS-Datenbank ab. Diese Einschränkung gilt auch für die Nutzung von DNS für andere Anforderungen.

[5] Bestimmte Details zur Arbeitsweise von Nessus, Informationen über dessen Plugins und dergleichen liegen lediglich auf der offiziellen Webseite des Herstellers vor. Daher wird im Folgenden wiederholt auf diese verwiesen.

3.1.2 FA2: Ermittlung offener Ports

Nachdem Geräte im Netzwerk ermittelt worden sind, gilt es herauszufinden, auf welchen Ports diese ansprechbar sind.

Nutzung von Nmap
Als Portscanner ist Nmap hervorragend geeignet um den Zustand von Ports durch aktive Sondierung zu ermitteln. Alle im Grundlagenabschnitt genannten Techniken (und weitere) sind nutzbar. Standardmäßig werden 1000 der als am wichtigsten erachtete Ports gescannt, es lassen sich aber problemlos weitere zu scannende Ports festlegen.

Nutzung von Nessus
Standardmäßig führt Nessus einen TCP-SYN-Scan auf ca 4.790 üblicherweise genutzten Ports aus. Es lässt sich aber auch ein TCP-Connect-Scan einstellen. UDP-Scans sind möglich, hinsichtlich Zeitaufwand und Zuverlässigkeit wird aber zu den folgenden Optionen geraten: Bei gegebenen Zugangsdaten und entsprechender Konfiguration der Zielsysteme kann SNMP oder SSH genutzt werden (für Windows WMI) oder es kann ein Agent installiert werden, der offene Ports durch Ausführen des netstat-Kommandos ermittelt.
[vgl. Tenable, 2017, UG, S. 134ff]

Nutzung von Netzwerksniffern
Trotz der zuvor genannten Einschränkungen kann beobachtet werden, auf welchen Ports Geräte Verbindungen annehmen.

Nutzung von Net-SNMP
Sofern Zugangsdaten für SNMP vorhanden sind und entsprechend konfigurierte Daemons auf den Zielgeräten laufen, lassen sich offene Ports und Adressen über die diese ansprechbar sind mit snmpnetstat komfortabel abfragen.

Nutzung von DNS
Unter den eingangs zu DNS genannten Einschränkungen könnten in der DNS-Datenbank Informationen zu Services nebst IP-Adressen und Ports unter denen diese erreichbar sind enthalten.

Nutzung eines direkten Zugangs
Sofern auf dem Zielsystem installiert, könnte man ebenfalls ein Tool wie Nmap nutzen um den Zustand von Ports zu ermitteln. Gegenüber einem Scan von außen, bietet ein Scan von localhost den Vorteil, dass Pakete nicht von Netzwerkfirewalls gefiltert werden können - die Filterung durch eine entsprechend konfigurierte Hostfirewall bleibt jedoch weiterhin möglich. Es ist allerdings zu beachten, dass Nmap nun ggf. auch Ports (und Services) findet, die tatsächlich nur von localhost aus erreichbar sind.
Ansonsten lassen sich offene Ports auch sehr einfach über das Programm ss (Socket Statistics) aus dem iproute2-Paket nutzen. Neben Portnummer und -zustand ist hier auch direkt ersichtlich, auf und von welchen Adressen Verbindungen angenommen werden und welcher Prozess dahinter steht.

3.1.3 FA3: Ermittlung lauschender Services

Nachdem offene Ports ermittelt worden sind, gilt es herauszufinden, welche Services in welchen Versionen darauf lauschen. Zudem wäre ein Abbildung der Services auf CPE-Namen wünschenswert, um eine möglichst hohe Kompatibilität der Ergebnisse des Werkzeugs mit Schwachstellendatenbanken und dergleichen zu ermöglichen.

Nutzung von Nmap
Nmap verfügt über eine Datenbank, die Portnummern ohne Serviceerkennung auf über 2.200 abbildet. Bei Nutzung Service- und Versionserkennungs-Option, nutzt Nmap zuerst Banner Grabbing und anschließend ggf. Fingerprinting und erkennt damit knapp 2.000 Protokolle anhand von über 11.400 Signaturen. [vgl. Lyon, 2016, VS; Lyon, 2016, VD; Lyon, 2017, CL, Nmap 7.50]
Zudem liefert Nmap im XML-Output einen CPE 2.2-Namen für erkannte Services. [vgl. Lyon, 2016, CPE]

Nutzung von Nessus
Nessus verfügt über 425 Plugins aus der Kategorie Service Detection, mit denen jeweils bestimmte Arten von Services oder konkrete Produkte entdeckt werden können. [vgl. Tenable, 2017, SD]
Außerdem gibt ein Plugin für Nessus, das CPE-Informationen für bestimmte Anwendungen und Betriebssysteme enthält. Falls kein Eintrag vorhanden ist, wird versucht einen zu erzeugen. Dies wird entsprechend gekennzeichnet.
Sofern Nessus Zugang zum System hat, können weitere Anwendungen berücksichtigt werden, die nicht als Dienst nach außen sichtbar sind. [vgl. Asadoorian, 2010]

Nutzung von Netzwerksniffern
Trotz der zuvor genannten Einschränkungen kann der beobachtete Verkehr jederzeit analysiert und kommunizierende Services so ggf. erkannt werden.

Nutzung von DNS
siehe Abschnitt 3.1.2 FA2 - Nutzung von DNS.

Nutzung eines direkten Zugangs
Auch hier ließe sich wieder eine lokale Nmap-Instanz nutzen um Service und Version zu erkennen und auf einen entsprechenden CPE-Namen abzubilden.
Ansonsten wäre es ein Ansatz zu einem Prozess, der einen Port nutzt, die ausführbare Datei zu ermitteln und festzustellen, zu welchem installierten Paket diese gehört. Bei Linux-Systemen erhält man solche Informationen in der Regel von der Paketverwaltung, welche sich wiederum nach Distributionen unterscheiden kann. Von dort könnte man ggf. Anbieter, Produktname, Version und weitere Informationen beziehen, die den Service identifizieren.
Auf Basis dieser Informationen könnte eine Heuristik Quellen von existierenden CPE-Namen wie dem CPE-Dictionary auswerten, um schließlich einen CPE-Namen für den betroffenen Service zu erstellen.
Anwendungen, die nicht in der Paketverwaltung registriert wurden, werden hiervon natürlich nicht erfasst. Dieser Fall soll aus Platzgründen hier nicht weiter betrachtet werden. Da aber

zumindest der offene Port festgestellt worden sein sollte, bietet dieser bereits einen guten Ansatzpunkt für weitere Nachforschung, sollte der dahinterstehende Dienst nicht automatisch ermittelt werden können.

3.1.4 FA4: Ermittlung des Betriebssystems

Rückschlüsse auf das Betriebssystem erlauben Kappes zufolge:

1. die auf dem Gerät verfügbaren Dienste
2. durch Banner Grabbing erhaltene Informationen
3. TCP/IP-Stack-Fingerprinting

[vgl. Kappes, 2013, S. 243/255]

Ein Beispiel zu 1. liefert Manzuik:
"Nmap detected that tcp 139, netbios-ssn, tcp 445, and microsoft-ds were open on the Web and database servers. Considering that netbios-ssn and microsoft-ds are specific to the Windows operating systems we could deduce that both the Web and database servers are running a version of Windows." [Manzuik, 2007, S. 112]
Der zweite Ansatz kann leicht durch folgendes Beispiel veranschaulicht werden: Verbindet man sich z.B. mit Netcat mit einem SSH-Port und erhält als Banner "SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.1", so liegt schon einmal nahe, dass es sich um ein Ubuntu-Linux handelt. Durch Rückgriff auf Datenbanken oder Google, findet man schnell heraus, dass es sich um Ubuntu 16.04 handeln muss. Hier gelten selbstverständlich auch die in Abschnitt 2.1.3 genannten Einschränkungen bzgl. der Verlässlichkeit von Banner Grabbing.
Die dritte Möglichkeit wurde bereits im Grundlagenabschnitt behandelt.

Nutzung von Nmap
Die Betriebssystemerkennung von Nmap nutzt TCP/IP-Stack-Fingerprinting mit einer Datenbank von über 5.300 Fingerprints. [vgl. Lyon, 2016, OD; Lyon, 2017, CL, Nmap 7.40]

Nutzung von Nessus
Neben einem Plugin, das TCP/IP-Stack-Fingerprinting betreibt und bestimmte Dienste analysiert, gibt es weitere Scripte, die aufgrund von Diensten das Betriebssystem identifizieren oder raten. Zudem gibt es auch Scripte, die einen evtl. bereitgestellten Zugang nutzen und das Betriebssystem durch bestimmte Kommandos oder Auslesen von Dateien ermitteln.
[vgl. Gula, 2009]

Nutzung von Netzwerksniffern
Auch hier gelten wieder die genannten Einschränkungen und Möglichkeiten, die bei der Ermittlung von Services gelten, nur dass das Fingerprinting sich auf die Besonderheiten im TCP/IP-Stack der kommunizierenden System konzentriert.

Nutzung von Net-SNMP
Das Objekt *sysDescr* mit der OID 1.3.6.1.2.1.1.1.0 sollte laut Spezifikation folgenden Wert liefern: "A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters." [RFC1213, 1991]

Bei Linux Systemen entspricht der zurückgegebene Wert eigenen Beobachtungen zufolge in im Wesentlichen der Ausgabe von *uname -a*.

Nutzung von DNS
DNS erlaubt es Angaben zum Betriebssystem eines Geräts zu machen, dies ist jedoch nicht gebräuchlich.

2.1.5 FA5: Ermittlung des allgemeinen Softwarestands

Sofern dies nicht über spezielle Dienste angeboten wird, gibt es ohne Zugang zum System in der Regel keinen praktikablen Ansatz, Informationen über Softwarestand (abgesehen von den zuvor behandelten, nach außen sichtbaren Diensten) und zu erhalten.

Nutzung von Nessus
Sofern bestimmte Patch Management Systeme für den zu inventarisierenden Bereich genutzt werden (Symantec Altiris, IBM Tivoli Endpoint Manager (BigFix), Red Hat Satellite Server, Microsoft SCCM, Dell KACE K1000, and Microsoft WSUS), kann Nessus bei Bekanntgabe der Zugangsdaten darauf zurückgreifen. [vgl. Tenable, 2017, S. 182; Asadoorian, 2013]

Nutzung von Net-SNMP
Wenn das Zielsystem die *Host Resources MIB* implementiert, lässt sich über die Tabelle *hrSWInstalledTable* unter der OID 1.3.6.1.2.1.25.6.3 die installierte Software z.B. mit snmpwalk abfragen. Laut Spezifikation sollte für jedes installierte Stück Software eine Zeile in der Tabelle existieren. Neben einer ID, einem Typ und dem Installationsdatum gibt es in dieser auch einen Wert, der u.a. Hersteller, Revision und den gebräuchlichen Namen enthalten soll.
[vgl. RFC2790, 2000]

3.1.6 FA6: Ermittlung von Nutzerkonten

Für die Ermittlung von Nutzerkonten gelten die gleichen Einschränkungen wie für die Ermittlung des allgemeinen Softwarestands.

Nutzung von Nessus
Es ist z.B. bei Windows-Systemen möglich, Nutzerkonten über bestimmte Dienste zu ermitteln. Dies ist jedoch als Schwachstellentest anzusehen.
Besteht Zugang zum Zielsystem, lassen sich Compliance Checks zweckentfremden, um Nutzerkonten zu ermitteln. Eigentlich werden diese genutzt um festzustellen, ob gewünschte Anforderungen erfüllt sind und lassen sich dafür beliebig konfigurieren. [vgl. Tenable, 2017, CC]

3.1.7 FA7: Ermittlung der Topologie

Topologien können in verschiedenen Auflösungen dargestellt werden. Für die angestrebte Darstellung ist eine Auflösung der Topologie auf Router-Ebene am sinnvollsten.
"Router level: The topology at this level is often the result of grouping interfaces that belong to the same router [...]. At this level, a node represents an IP-compliant network device (e.g., a

host or a router with multiple interfaces). Two nodes are connected by an edge if the corresponding devices have interfaces that are on the same IP broadcast domain."
[Motamedi, 2015, S. 2]
Zudem wäre es wünschenswert Subnetze zu ermitteln.
Um eine möglichst aussagekräftige Router-Level-Topologie mit Subnetzen zu erhalten, muss im Wesentlichen Folgendes erreicht werden:

1. Identifizierung individueller Knoten
2. Ermittlung der Verbindungen zwischen den Knoten
3. Zuordnung der Knoten zu Subnetzen

Nutzung von Netzwerksniffern
Zur Erkennung der Topologie müsste die Beobachtung des Verkehrs an mehreren geeigneten Knotenpunkten erfolgen. Durch Zusammenführen der Daten, Identifizierung einzelner Pakete und Analyse der TTL an den verschiedenen Beobachtungspunkten könnten Rückschlüsse auf die Topologie erlauben.

Nutzung von Net-SNMP
Mit snmpnetstat kann man sich unter Verwendung der Option "-Cr" die Routen des befragten Systems ausgeben lassen.
Sofern SNMP-Zugänge für alle Zielsysteme vorhanden ist, lässt sich so mit zuverlässigen Werten die Topologie rekonstruieren.

Nutzung von Traceroute/Nmap
Da Traceroute (eine solche Funktion ist auch in Nmap enthalten) ein sehr simples Werkzeug ist, keinen Zugang zu und nur minimale Kooperation von beteiligten Geräten fordert, ist es ein beliebtes Mittel zur Erfassung von Topologien.
Acharya und Gouda beschreiben in ihren Publikationen zum Thema die Einschränkungen und Grenzen von traceroute zur Ermittlung von Topologien, da sich dieser Anwendungsfall auf das Network Tracing-Problem zurückführen ließe:
"The *network tracing problem* is to design an algorithm that takes as input a trace set T that is generable from a network, and produces a network N such that T is generable from N and not from any other network." [Acharya, 2009, S. 64]

A trace set T is generable from a network N iff the following five conditions are satisfied:

1. Every trace in T is generable from N.
2. For every pair of terminal nodes x, y in N, T has at least one trace between x and y.
3. Every edge in N occurs in at least one trace in T.
4. The unique identifier [wovon es exakt einen pro Knoten geben muss, Anmerkung des Verfassers] of every node in N occurs in at least one trace in T.
5. T is consistent: for every two distinct nodes x and y, exactly the same nodes must occur between x and y in every trace in T where both x and y occur.

[Acharya, 2010, a, S.186]
Die Autoren erklären in ihrem Paper zum Schwachen Network Tracing-Problem (hier ist als Lösung eine kleine Menge von Netzwerken erlaubt, aus der sich die Menge von Traces T erzeugen ließe), dass die o.g. fünf Bedingungen zu stark erscheinen könnten, dass aber, wenn auch nur eine von ihnen nicht erfüllt wäre, die Anzahl der Netzwerke exponentiell anstiege, von denen die Menge von Traces T generierbar wäre.

Schließlich kommen sie zu dem Ergebnis, dass selbst mit all diesen Einschränkungen es nur für bestimmte Arten von Netzwerken, wie Bäume und ungerade Ringe, möglich ist, Lösungen zu berechnen. [vgl. Acharya, 2010, a, S.186/193]
Für den konkreten Anwendungsfall im Rahmen dieser Arbeit könnte man die zu ermittelnde Topologie z.B. als Baum annehmen, für den das Problem lösbar ist. Da laut der o.g. zweiten Bedingung allerdings für jedes Paar von Endknoten Traces vorliegen müssten, ist dieser Ansatz schon allein aus diesem Grunde hier nicht praktikabel, da dies Zugang zu jedem Gerät im Netzwerk voraussetzen würde.
Auch die vierte Bedingung ist kaum zu erfüllen: Aliasing liegt fast immer vor, da Router über verschiedene IP-Adressen identifiziert werden. In einem unveröffentlichten Paper legen die Autoren daher die Entwicklung von Heuristiken nahe:
"[...]even in the absence of anonymous nodes, network tracing may be an intractable problem owing to node aliasing. Our hardness results not only bound the power of current probe-based as well as graph theoretic approaches for alias resolution, but also present the first theoretical justification for employing heuristics to attack the problem." [Acharya, 2010, b]

Nutzung von Route Analytics
Dinger und Hartenstein beschreiben Route Analytics wie folgt:
"Hierzu werden die Nachrichten des internen Routing-Protokolls wie OSPF [...] überwacht und aufgezeichnet. Somit kann zu jeder Zeit die aktuelle Routing-Topologie innerhalb der Organisation nachvollzogen werden und eine Analyse der aktuellen Pfade vorgenommen werden." [Dinger, 2008, S. 173]
Da diese Technik es erfordern würde Geräte zu installieren, die das jeweilig verwendete Routingprotokoll beherrschen und bei statischem Routing nutzlos wäre, soll sie hier nicht weiter behandelt werden.

Nutzung von DNS
DNS-Namen können Rückschlüsse auf auf die Topologie erlauben. Dies ist jedoch schwer oder nur unter bestimmten Voraussetzungen automatisierbar.

3.1.8 FA8: Visualisierung der Topologie

Die Topologie eines Netzwerks wird im Allgemeinen durch einen sogenannten Netzplan dargestellt. Dabei handelt es sich um ein Diagramm, in dem Geräte, oder auf höherer Abstraktionsebene auch ganze Netzsegmente, als Knoten und die dazwischen liegenden Verbindungen als Kanten dargestellt werden.
Durch unterschiedliche Icons können spezielle Komponenten wie Router oder Datenbanken kenntlich gemacht werden. Für weitere Übersichtlichkeit können logische Einheiten, wie z.B. Subnetze, durch grafische Elemente zusammengefasst werden. Auch treffende Beschriftungen sollten zur besseren Verständlichkeit nicht fehlen.
Hier soll im Rahmen der Implementierung ein geeignetes Werkzeug gefunden werden, das sich nutzen lässt, aus den gewonnenen Daten einen Netzplan zu erzeugen.

3.2 Nicht funktionale Anforderungen

3.2.1 NFA1: Geringe Voraussetzungen

Es sollen möglichst wenig Gegebenheiten im zu inventarisierenden Netzwerk vorausgesetzt werden, die nötig sind damit *ANIS* korrekt arbeitet.
Ebenso soll der Einrichtungsaufwand möglichst gering gehalten werden und am bestehenden Netzwerk sollen möglichst wenig Veränderungen vor Einsatz des Systems vorgenommen werden müssen. Eventuell notwendige Eingriffe sollen dabei möglichst wenig invasiv sein.

3.2.2 NFA2: Wenig Neuentwicklung

Wo möglich sollen existierende Werkzeuge genutzt werden, um die selbst entwickelte Codebasis möglichst gering und damit übersichtlich zu halten. Gleichzeitig soll die Auswahl verwendeter Software von Dritten möglichst gering gehalten werden, um nicht unnötig viele Abhängigkeiten zu erzeugen.

3.2.3 NFA3: Erweiterbarkeit

Das Werkzeug soll so aufgebaut sein, dass es einfach zu erweitern ist und erzeugte Dateien automatisiert weiter nutzbar sind.

3.3. Bewertung nutzbarer Tools

An dieser Stelle sollen die wichtigsten Werkzeuge hinsichtlich ihrer Eignung für eine mögliche Lösung zur Erfüllung ausgewählter Anforderungen tabellarisch bewertet werden.

Tools → **Anforderungen ↓**	Direkt-zugang	Nmap	tcpdump	Nessus	ping	Trace-route	dig	Net-SNMP
Geräte entdecken		++	o	++	+	+	+	--
offene Ports ermitteln	++	++	o	+	--	--	o	++
Services ermitteln	++	++	o	+	--	--	o	o
Betriebssystem ermitteln	++	+	o	+	--	--	-	++
allg. Software-stand ermitteln	++	--	--	o	--	--	--	+
Nutzerkonten ermitteln	++	--	--	o	--	--	--	-
Topologie ermitteln	++	+	--	--	--	o	o	+
Geringe Voraussetzungen	--	+	--	o	++	++	o	o

Legende:

(++) sehr gut geeignet
(+) gut geeignet
(o) bedingt geeignet
(-) wenig geeignet
(--) nicht geeignet

4. Konzept

Im Folgenden gilt es, mögliche Lösungsansätze zu evaluieren und auszuwählen, eine Lösungsstrategie und schließlich eine Architektur, die diese umsetzt, zu entwickeln.

4.1 Mögliche Lösungsansätze

Eine wesentliche Hürde für die automatisierte Inventarisierung bringt die gängige Praxis der Kompartmentalisierung in Organisationsnetzwerken Probleme mit sich. Hierbei werden verschiedene Bereiche, durch Netzwerkfirewalls voneinander getrennt. So kann es z.B. sein, dass bestimmte Teile des Netzwerks von anderen Teilen aus komplett unsichtbar bleiben. Dies betrifft beide Arten von Ansätzen zur Informationsgewinnung, die jeweils unter der Prämisse entsprechender Firewallregeln anwendbar sind:

1. Solche, die einen Zugang zum System bzw. einem Dienst, der gewissermaßen eine Innensicht des Systems bietet, voraussetzen (zugangsbasiert).
2. Solche, die keinen Zugang erfordern und sich auf eine Außensicht des Systems beschränken (zugangslos)

zugangsbasierte Lösungen
Sollten Möglichkeiten für zugangsbasierte Techniken wie SNMP oder ein Direktzugang (z.B. per SSH) bestehen, so werden wahrscheinlich auch bereits entsprechende Firewallregeln gesetzt sein, was die Voraussetzungen an dieser Stelle gering hält.
Zudem sind Ergebnisse, die man mit Zugang erhält, vergleichsweise zuverlässig.
Bei einem Konsolenzugang per SSH sind die Möglichkeiten der Informationsgewinnung über das System lediglich durch die Rechte des verwendeten Nutzerkontos beschränkt. Bei SNMP ist es zudem von Bedeutung, welche MIBs vom zu untersuchenden System unterstützt werden und welche Informationen diese preisgeben.
Auf der anderen Seite sind die Voraussetzungen hier recht hoch, da man nicht ohne weiteres davon ausgehen kann, dass derartige Zugänge bereits zur Verfügung stehen, sondern, dass diese erst auf jedem zu untersuchenden System geschaffen werden müssen. Neben dem Konfigurations- und anschließendem Wartungsaufwand kann es auch andere Gründe geben, die dagegen sprechen, einen solchen Zugang zu schaffen. So birgt z.B. jedes zusätzliche Nutzerkonto auch immer die Gefahr, missbraucht zu werden. Ein weiteres Beispiel sind Systeme, wie Log-Server, für die die Verwaltung von Zugängen besonders restriktiv gehandhabt werden muss.

zugangslose Lösungen
Zugangslose Techniken bieten in der Regel keine Möglichkeiten, den allgemeinen Softwarestand und Nutzerkonten zu ermitteln.
Außerdem ist es bei zugangslosen Techniken wahrscheinlich, dass Firewallregeln, die die korrekte Funktion *ANIS'* ermöglichen, noch gesetzt werden müssen. Auch Intrusion Prevention Systeme müssen so konfiguriert werden, dass die Aktivitäten des Inventarisierungssystems keine Flut an Ereignissen auslösen und sie sich nicht gegenseitig stören.

Die nötige Konfiguration beschränkt sich jedoch im Vergleich mit zugangsbasierten Techniken nur auf relativ wenige Geräte. Da ansonsten keine weitere Konfiguration an bestehenden Geräten vorgenommen werden muss, bewerte ich die Voraussetzungen der zugangslosen Techniken als geringer.

hybride Lösungen
Denkbar wäre auch eine hybride Lösung, die z.B. innerhalb von Subnetzen im wesentlichen mit zugangslosen Techniken arbeitet. Für erkannte Gateways könnte dann ein Zugang genutzt werden, um Informationen über die Netzwerkinterfaces auszulesen. Die erkannten Verbindungen zwischen den Subnetzen würden dann genutzt werden um die Topologie zu schlussfolgern.
Im Vergleich zu rein zugangsbasierten Techniken fällt der Einrichtungsaufwand hier wesentlich geringer aus. Letztlich bleiben hier aber die Schwächen von zugangslosen Techniken bestehen und der Konfigurationsaufwand wurde lediglich etwas verschoben.

Bewertung und Auswahl
Aufgrund der Heterogenität von Netzwerken ist es kaum möglich ein universell einsetzbares und dennoch schlankes System zu entwickeln. Entweder ist das System für eine hohe Zahl verschiedener Netzwerke geeignet und es ist selbst dementsprechend groß. Oder es wird auf einer eingeschränkten Anzahl an Gegebenheiten aufgebaut, die mit hoher Wahrscheinlichkeit in jedem Netzwerk zu finden sind, und es muss dann ggf. weiter angepasst werden oder ist ansonsten weniger mächtig.
Besonders die Anforderung der geringen Voraussetzungen ist in Hinblick auf eine möglichst universelle Anwendbarkeit als besonders wichtig zu bewerten. Da hybride Ansätze keinen nennenswerten Vorteil gegenüber zugangslosen Möglichkeiten bieten und zugangsbasierte Strategien einen sehr hohen Einrichtungsaufwand bedeuten, ist der zugangslose Ansatz als am geeignetsten für *ANIS* anzusehen. Selbst wenn die Anforderungen der Ermittlung von Nutzerkonten und des allgemeinen Softwarestands auf diesem Wege nicht direkt erfüllt werden können, soll die folgende Lösungsstrategie auf zugangslosen Verfahren aufbauen. Das System soll letztlich aber auch so ausgelegt werden, dass es um Module, die die Nutzung zugangsbasierter Techniken ermöglichen, erweiterbar ist.

4.2 Lösungsstrategie

4.2.1 Topologieerkennung mit Traceroute

Da das Problem unter den in Abschnitt 3.1.7 genannten Bedingungen nicht lösbar ist, gilt es, eine Heuristik zu entwickeln.

Problematik
Im Folgenden sollen noch einmal die sich in der Praxis ergebenden möglichen Schwierigkeiten festgehalten werden:

1. Es liegen keine vollständigen Traces vor
2. Aliasing - Die identifizierenden Bezeichner in IP-Netzwerken sind die IP-Adressen. Router verfügen aber pro Interface mit dem sie Teil eines Subnetzes sind über

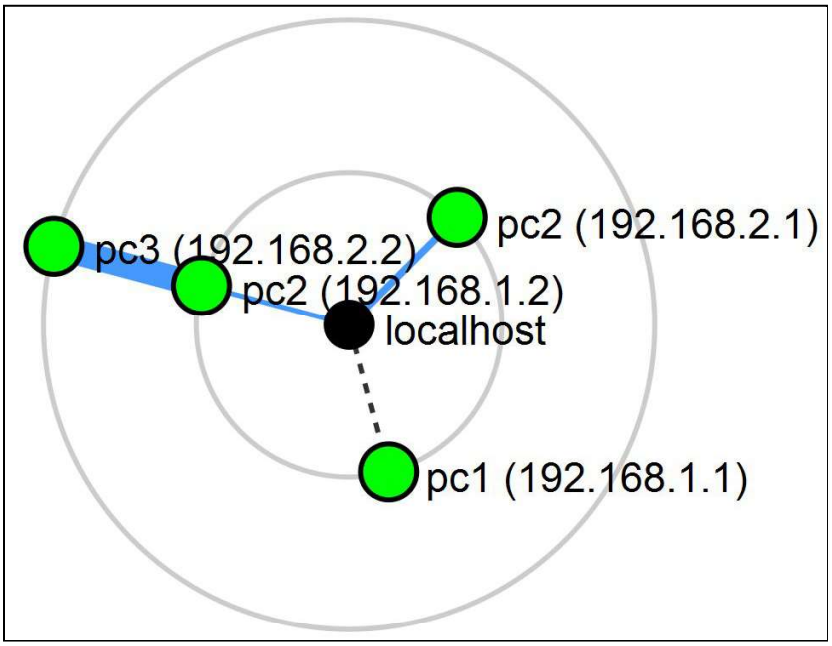
mindestens eine solche Adresse.
In der Praxis führt dies dazu, dass in den durch Traceroute gewonnenen Daten ein Router als zwei separate Knoten erscheint. Dadurch kann es u.a. sein, dass existierende Routen nicht erkannt werden und die erkannte Topologie größer als die tatsächliche ist. [vgl. Haddadi, 2008]

3. blockierende Router - bezeichnet Router, die auf Sondierungspakete weder reagieren, noch diese weiterleiten und damit verhindern, weitere Informationen über den Pfad zu sammeln
4. anonyme Router - anonyme Router leiten die Sondierungspakete zwar weiter, schicken selbst aber keine Antworten an die Quelle zurück. [vgl. Holbert, 2014, S. 13ff]. Es ist auch möglich, dass Router nur bei manchen Traces reagieren und bei anderen nicht, also anonym bleiben.
"An anonymous node in a trace may or may not be identical to any other anonymous or named node. Consequently, there may be multiple topologies for the computed network, all of which can generate the observed trace set." [Acharya, 2010, a, S. 185]
5. asymmetrisches Routing - liegt vor, wenn sich der Pfad von einem Knoten x zu einem Knoten y und der Pfad von y zu x unterscheiden.
6. Load balancing - Traceroute findet genau einen Pfad vom Start zum Ziel. Durch Load balancing können daher Anomalien auftreten, durch die z.B. Knoten übersehen werden. [vgl. Augustin, 2006, S. 154]
7. evtl. Veränderungen während des Ermittelns der Topologie

Aus Platzgründen soll an dieser Stelle nicht weiter auf mögliche Lösungsansätze für die alle einzelnen Probleme eingegangen werden. Stattdessen sollen 4 bis 7 direkt als Annahme für die Heuristik als ausgeschlossen vorausgesetzt werden. Dies ist, dem besten Wissens des Autors entsprechend, auch praktikabel. Schwierigkeit 3 ließe sich abschwächen, indem das für die Sondierungspakete genutzte Protokoll und ggf. die angegebenen Ziel-Ports sich daran orientieren, auf welche Pakete die Ziel-Hosts in früheren Schritten der Informationssammlung bereits reagiert haben.

Beschränkung auf das Problem des Aliasing

Auch bei unvollständigen Traces (Schwierigkeit 1) lassen sich eigenen Beobachtungen zufolge mit Zenmap, der GUI Nmaps, unter Nutzung von Traces bereits recht aussagekräftige Karten erzeugen. Daher soll die Heuristik von der Frage ausgehend, wie man diese verbessern kann, entwickelt werden.



Die obige Abbildung zeigt eine solche, mit Zenmap erzeugte, Karte. Das hervorstechende Problem ist hier das Aliasing (Schwierigkeit 2): pc2 wird doppelt erkannt,da als identifizierendes Merkmal jeweils die IP-Adressen seiner beiden Interfaces genutzt werden. Es gibt verschiedene Ansätze Alias gleicher Knoten zusammenzuführen (Alias-Auflösung):

- Es wird ein Paket an einen ungenutzten Port gesendet. Als Antwort sollte mit einer ICMP-Port unreachable-Nachricht geantwortet werden, die gemäß RFC1812 als Absenderadresse die des ausgehenden Interfaces trägt. Dieses sollte für die Route zurück zum Sender stets gleich sein, auch wenn ein anderes Interface angesprochen wurde. [vgl. RFC1812, 1995; Huffaker, 2002, S. 50]
  Eigenen Beobachtungen und auch der Literatur zufolge, trägt das Antwort-Paket als Absender jedoch oft eine andere IP-Adresse, als wie oben spezifiziert. U.a. die IP-Adresse des Interfaces, das die Nachricht empfangen hat. [vgl. Motamedi, 2015, S.1049 und 1054]
- Viele Router sollen einen Zähler für IP-IDs haben, der von allen Interfaces geteilt wird. Bilden bei der abwechselnden Sondierung zweier IP-Adressen die IDs der Antwortpakete eine sequentielle Folge, so liegt es nahe, dass sie von verschiedenen Interfaces des selben Router stammen. [vgl. Keys, 2010, S. 51]
  Im eigenen Versuch wurden die IP-IDs jedoch pro Interface mit zufälligem Zuwachs separat hochgezählt.
- Es können andere identifizierende Merkmale herangezogen werden, die die Maschine von sich preisgibt. So gibt es z.B. Nmap-Scripts, die doppelte SSH-Hostkeys, SSL-Zertifikate oder NetBIOS-Namen erkennen. [vgl. Karlsson]
  In der Praxis stehen diese Merkmale jedoch oft nicht zur Verfügung oder sind nicht aussagekräftig, wenn z.B. SSH-Hostkeys aus Gründen der Einfachheit mehrfach verwendet werden.
- Bestimmte Namenskonventionen bei der Vergabe von DNS-Hostnamen könnten bei der Aliasauflösung helfen. [vgl. Keys, 2010, S. 51]

- Von bestimmten Endpunkten kann die Rückrichtung bereits vorhandener Traces ermittelt wird. Hier sollten dann weitere Aliases der verbindenden Router (die in der Topologie bisher selbst wie Endpunkte erschienen) auftauchen, welche dann unter der Annahme des symmetrischen Routings aufgelöst werden können.
  Beispiel: Bei einem Trace (A,B,C) und der Rückrichtung (C,D,A) kann angenommen werden, dass B und D zwei Interfaces des gleichen Routers sind, obwohl D von A aus zuvor wie ein Endpunkt wirkte.

Zudem gibt es neuere Ansätze, zur Erfassung der Topologie des Internets, bei denen zuerst durch Heuristiken Subnetze geschlussfolgert werden, um schließlich aufgrund der ermittelten Subnetze und der Hop-Entfernung vom Sondierungsstandpunkt auch Alias aufzulösen.

Als State of the Art gilt hier ExploreNET. [vgl. Grailet, 2016, S. 1] Motamedi schreibt dazu: "Subnet level discovery tools such as XNET [52] aim to reveal all ping-able IP addresses on a subnet. XNET identifies boundaries associated with the IP prefix of a subnet with a series of tests on IPs that can potentially be in one subnet. The methodology is developed based on the fact that all IP addresses in one subnet share a prefix and have at most one-hop distance difference from a vantage point. The problem is that the size of the subnet is in general unknown. Given IP address t that is n hops away from a vantage point, XNET probes IPs in the prefix that includes t starting from the smallest /31 prefix (mate-31). If the probes to all IPs in this prefix travel through the same route and their hop distances to the vantage point are within the boundaries that support their existence in the same subnet as t, then the target prefix is expanded and IPs in this expanded prefix are subjected to the same tests. XNET incrementally expands the prefix until at least one IP fails the tests. At this point the last successfully tested prefix identifies the subnet that includes t." [Motamedi, 2015]

Grailet et al. sehen bei ExploreNET vor allem das Problem, dass es weder eine Garantie gäbe, dass die geschlussfolgerten Subnetze korrekt seien, noch eine Metrik, die es ermögliche sie zu bewerten. Zudem tendiere ExploreNET dazu, große Subnetze in mehrer kleinere, und somit unvollständige zu zerlegen. [vgl. Grailet, 2016, S. 1]
TreeNET baut auf ExploreNET auf und erweitert es um eine Verfeinerungsphase und Klassifikationen für die erkannten Subnetze um eine qualitative Bewertung zu ermöglichen. Letztlich wird die Glaubwürdigkeit der ermittelten Subnetze mit 80 - 90 % angegeben.
[vgl. Grailet, 2016]
Aufgrund der relativ guten Ergebnisse bei der Subnetzerkennung und der darauf aufbauenden Aliasauflösung sind diese Ansätze durchaus wertvoll für die Internet-Topologie-Erkennung. Für eine Inventariserung des Netzwerks, gerade unter Sicherheitsaspekten, mangelt es den Methoden jedoch an Zuverlässigkeit.

Lösungsansatz
Um Angesichts dieser Schwierigkeiten, die Topologie zu ermitteln, dennoch zuverlässige Ergebnisse zu erzielen, kann die Zielsetzung des möglichst geringen Einrichtungsaufwandes nicht in gewünschtem Maße eingehalten werden.
Stattdessen muss für diese Aufgabe der Zugang zu jeweils einem Gerät mit einem Agenten in jedem Subnetz vorausgesetzt werden, um dann unter der Annahme des symmetrischen Routings Alias aufzulösen. Hier bietet es sich an, den Subnetzen des zu inventarisierenden

Bereichs je ein dediziertes Gerät hinzuzufügen. So kann zum einen vermieden werden, sensible Knoten wie Router freizugeben und zum anderen wird der Konfigurationsaufwand gering gehalten, der auch bei weniger sensiblen Geräten des Bestands entstehen würde. Zusätzlich zu dem Vorteil, dass ein dediziertes Gerät ohne weitere Konfiguration alle Voraussetzungen mitbringt, die es zum arbeiten benötigt (wie z.B. den *ANIS*-Agenten oder Scan-Tools), entsteht durch die Standpunkte in jedem Subnetz der weitere Vorteil, dass *ANIS* innerhalb der Subnetze ungestört von Netzwerkfirewalls arbeiten kann.
Leider ist hierdurch eine weitere, nicht unwesentliche, Voraussetzung entstanden: Die Subnetze müssen zunächst bekannt sein. Hier könnten Tools wie ExploreNET als erster Einstiegspunkt dienen, allerdings muss klar sein, deren Funktionsfähigkeit u.a. von Netzwerkfirewalls weiterhin massiv eingeschränkt werden kann: Standardmäßig werden sowohl bei ExploreNET als auch bei TreeNET zur Sondierung ICMP-Echo-Requests ausgesendet. Es gibt zwar Modi für UDP und TCP, allerdings ist es nicht möglich Zielports festzulegen. Zudem ist eine komplette Abdeckung nicht gewährleistet.
Die Ergebnisse könnten jedoch durch die hinzugefügten Geräte verifiziert bzw. verbessert werden, indem geprüft wird, ob die initial angegebenen Adressbereiche durch die erschlossenen Subnetze abgedeckt werden.

Heuristik
Unter den genannten Voraussetzungen ergibt sich folgende Heuristik zur Alias-Auflösung und Topologieermittlung mittels Traceroute:

- Die Agenten in den Subnetzen machen Traces zu allen anderen Agenten-Maschinen in anderen Subnetzen.
- Bei der zentralen Verarbeitung werden alle IP-Adressen den bekannten Subnetzen zugeordnet.
- Es wird je ein Trace aus jedem *Subnetz A* zu jedem anderen *Subnetz B* ausgewählt.
- Durch Vergleich der Routen *A nach B* und *B nach A* werden Aliase eliminiert. (wird im folgenden Abschnitt näher beschrieben)
- Geräte, die über mehrere IP-Adressen (Alias) aus verschiedenen Subnetzen verfügen, stellen die die Subnetze verbindenden Router dar.

Die Netzwerkfirewalls müssen somit lediglich für die Kommunikation der *ANIS*-Komponenten geöffnet werden.

Algorithmus zur Aliasauflösung
An dieser Stelle soll der grundsätzliche Algorithmus zur Aliasauflösung in Pseudocode dargestellt werden. Die genaue Implementierung und eventuelle Maßnahmen zur Performanceverbesserung bleiben hiervon unberührt.
Es gelten folgende Voraussetzungen: Traces werden von Agenten-Geräten erstellt, die sich in genau einem Subnetz befinden und somit selbst keine Router sind. Es wird ein Traceroute zu allen anderen Agenten ausgeführt, so dass es für jeden Trace *A nach B* einen gleich langen Trace *B nach A* gibt.
Als Eingabe dient eine Liste *Subnets*, die alle bekannten Subnetze enthält (z.B. in CIDR-Notation), sowie eine Map *Devices*, die bereits für alle entdeckten IP-Adressen ein Schlüssel-Wert-Paar enthält. Dabei sind die Schlüssel IP-Adressen, und die Werte jeweils ein Zeiger auf ein Objekt *Device*. Letzteres verfügt über ein Attribut *ips* hinter dem sich eine

Menge verbirgt, in der die IP-Adressen des Objekts gespeichert sind. Der untenstehende Algorithmus vereint diese *Device*-Objekte, falls zwei Einträge als dem selben Gerät zugehörig erkannt wurden, und aktualisiert die *Devices*-Map entsprechend, so dass hinter jedem Schlüssel schließlich ein einzigartiges Objekt mit allen bekannten IP-Adressen (Alias) des Geräts gespeichert sind (inkl. der Adresse, die als Schlüssel dient).
Zur Vereinfachung wird eine Hilfsfunktion TRACE(*subnetA*, *subnetB*) genutzt, die für zwei Subnetze ein Array von IP-Adressen zurückgibt, welches den Trace zwischen den beiden repräsentiert. Das erste Element stellt dabei die Quelle des Trace in *subnetA*, und das letzte Element das Ziel in *subnetB*, dar.

```
1  RESOLVE-ALIAS(Subnets,Devices)
2      for each subnetA in Subnets do
3          for each subnetB in Subnets do
4              if subnetA ≠ subnetB then
5                  traceA := TRACE(subnetA,subnetB)
6                  traceB := TRACE(subnetB,subnetA)
7                  traceB := traceB.reverse  //traceB-Reihenfolge umkehren
8                  for i := 1 to traceA.length
9                      hopA := traceA[i]
10                     hopB := traceB[i]
11                     if hopA ≠ hopB then
12                         Devices := INTEGRATE-ALIAS(Devices,hopA,hopB)
13     return Devices
```

```
1  INTEGRATE-ALIAS(Devices,hopA,hopB)
2      deviceA := Devices.get(hopA)
3      deviceB := Devices.get(hopB)
4      deviceA.ips.addAll(deviceB.ips)
5      for each address in deviceB.ips
6          Devices.put(address,deviceA)
7      return Devices
```

Beispiel für einen Durchlauf des Algorithmus

Angenommen es existiert ein Agenten-Gerät mit der Adresse *a* im *Subnetz A* und ein Agenten-Gerät mit der Adresse *d* im *Subnetz D*. Außerdem gibt es einen Router, der mit der Adresse *b* in *Subnetz A* und mit der Adresse *c* in *Subnetz B* vertreten ist (siehe obige Abbildung).
Weiterhin existieren Traces, so dass TRACE die folgenden Werte liefert:
TRACE(*Subnetz A*, *Subnetz B*) = [a,b,d]
TRACE(*Subnetz B*, *Subnetz A*) = [d,c,a]
Wenn in Zeile 4 von RESOLVE-ALIAS *subnetA* = *Subnetz A* und *subnetB* = *Subnetz B* ist, werden in den folgenden Zeilen den Variablen *traceA* und *traceB* die o.g. Traces zugewiesen.
In Zeile 7 wird *traceB* umgedreht, so dass aus [d,c,a] nun [a,c,d] wird.
In der folgenden Schleife werden die Adressen der beiden Traces ([a,b,d] und [a,c,d]) indexweise miteinander verglichen.

Beim Index 2 wird in Zeile 11 die Ungleichheit der beiden Adressen *b* und *c* festgestellt. Es wurde also ein Alias erkannt und die Adressen werden durch die Funktion INTEGRATE-ALIAS in die *Devices*-Map integriert:
In Zeile 4 werden alle Adressen von *deviceB* den Adressen von *deviceA* hinzugefügt. Anschließend wird die *Devices*-Map aktualisiert, so dass alle Adressen von *deviceB* auf das vereinte Objekt zeigen.
Am Ende des Durchlaufs ist *deviceA.ips* = [b,c] und die *Devices*-Map hat die Einträge [b : *deviceA*, c : *deviceA*].

4.2.2 Perspektiven

Die zur Erkennung der Topologie getroffenen Voraussetzungen bieten auch neue Möglichkeiten:
Geräte können von verschiedenen Standpunkten im Netzwerk unterschiedlich wahrgenommen werden. Dies kann bei der Inventarisierung nun einfach berücksichtigt werden. Daher soll jedes entdeckte Gerät über eine Menge von Perspektiven dargestellt werden, die darüber Aufschluss geben, wie es von bestimmten Quellen aus wahrgenommen wird.

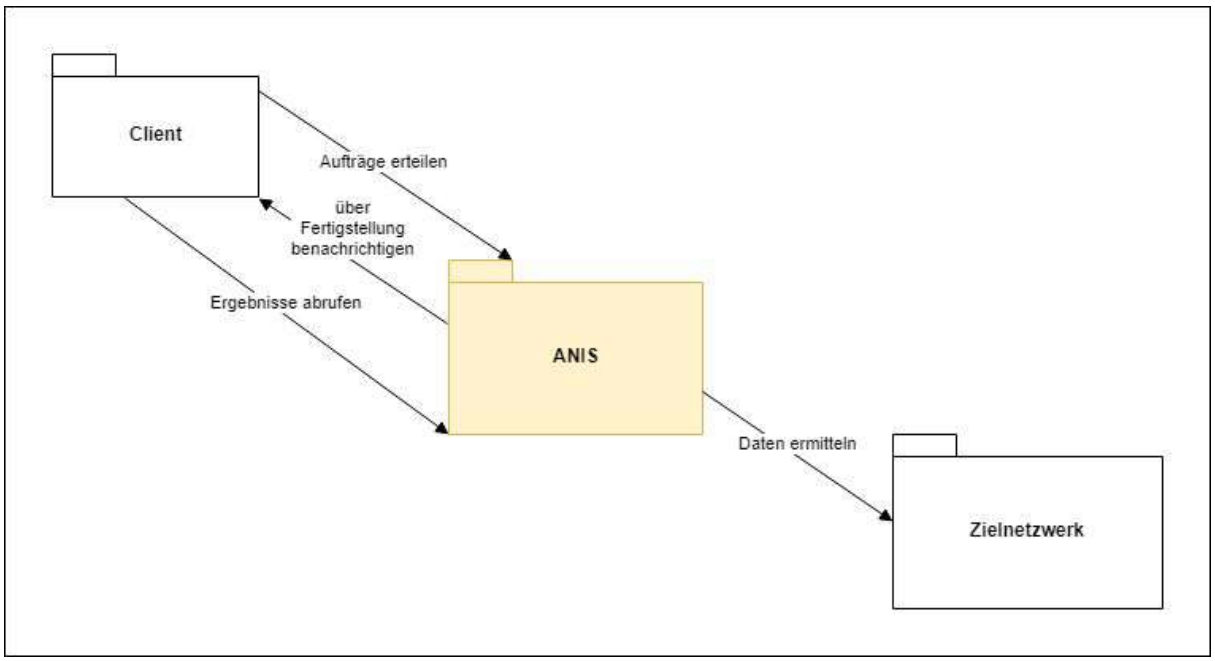
4.2.3 Nutzung von Nmap

Wie im vorigen Abschnitt erörtert und an dessen Ende tabellarisch zusammenfassend dargestellt, ist Nmap gut geeignet, um die Anforderungen bezüglich zu sammelnder Informationen, soweit es zugangslose Techniken erlauben, zu erfüllen.
Für die meisten Anforderungen lassen sich die von nmap gewonnenen Informationen direkt weiterverarbeiten. Auch die Traceroutes können von Nmap ausgeführt werden.

4.3 Architektur

An dieser Stelle soll die für *ANIS* gewählte Architektur veranschaulicht und erläutert werden.

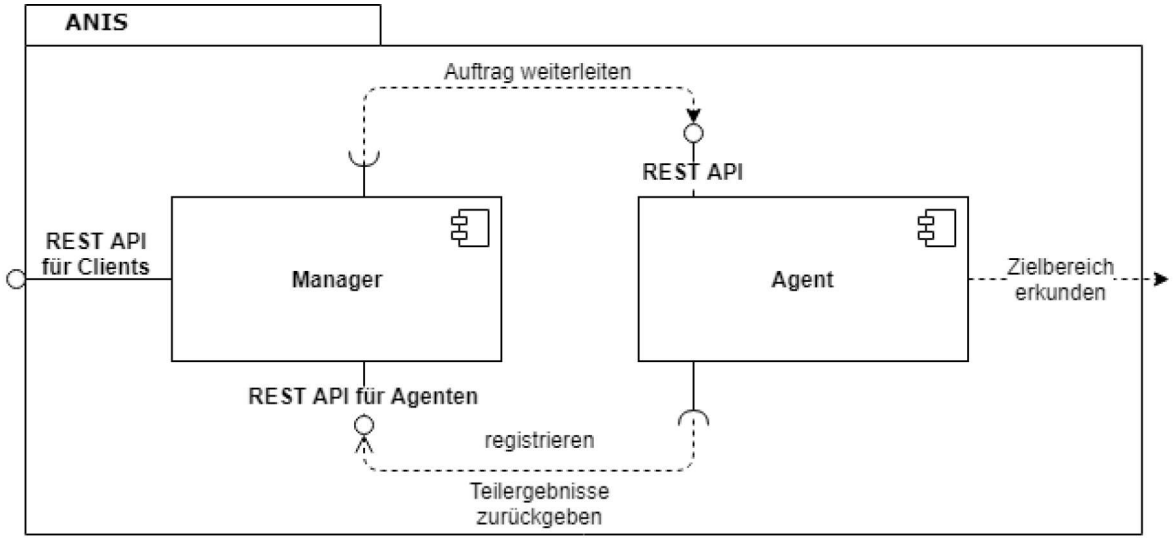
4.3.1 Kontextsicht



Die obenstehende Abbildung zeigt *ANIS* im Kontext der angrenzenden Systeme:
Clients können *ANIS* nutzen um Aufträge zur Inventarisierung zu erteilen und Ergebnisse abzurufen. Mit dem Zielnetzwerk interagiert *ANIS* um entsprechend der Aufträge Informationen zu gewinnen.

4.3.2 Bausteinsicht Gesamtsystem

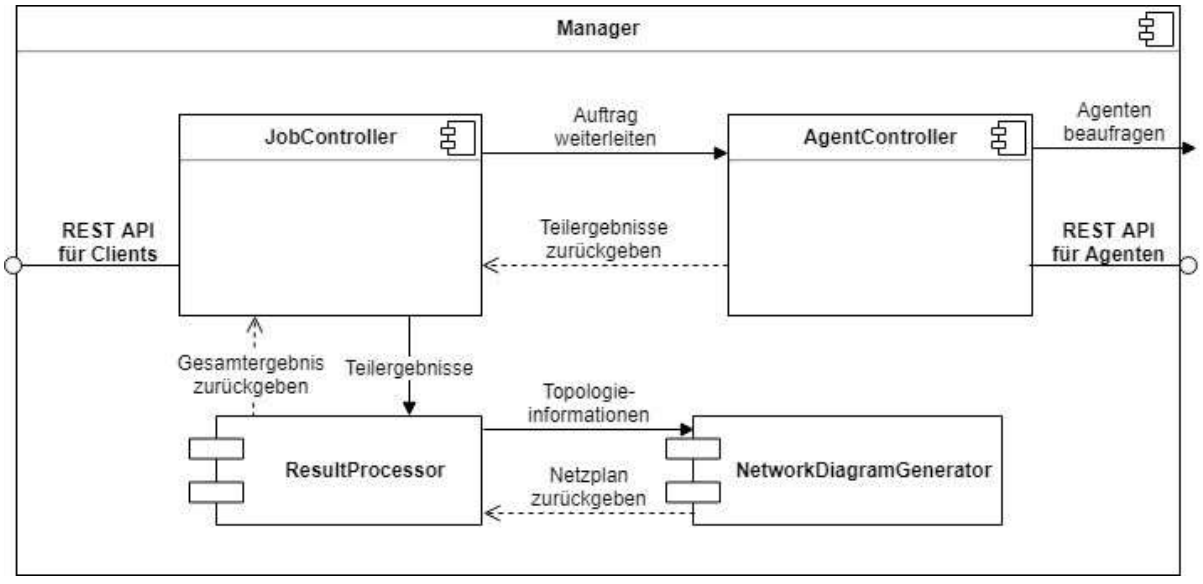
Es wurde bereits zuvor festgestellt, dass die Daten von mehreren Punkten aus gesammelt werden müssen und die Ergebnisse von einer Stelle aus abrufbar sein sollen. Daher bietet sich die Verwendung einer Master-Slave-Architektur an.



Es gibt eine Managerkomponente (den Master), die Aufträge annimmt und über die Ergebnisse abgerufen werden können. Sie wird auf einer physischen Maschine ausgeführt wird und umfasst mehrere Subkomponenten zur Verarbeitung.

Zudem gibt es Agenten (Slaves), die an verschiedenen Stellen im Netzwerk platziert werden. Sie führen die eigentliche Erkundung, des im Auftrag festgelegten Adressbereichs, im für sie sichtbaren Netzsegment, durch.
Auch das Gerät, auf dem die Managerkomponente ausgeführt wird, kann Agenten beheimaten.

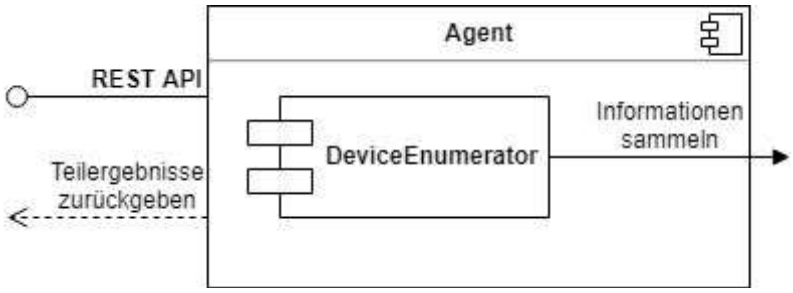
4.3.3 Bausteinsicht Managerkomponente



Die Verarbeitung der Aufträge innerhalb der Managerkomponente erfolgt über ihre Subkomponenten:
Der JobController stellt über eine REST API das Starten von Inventarisierungsaufträgen durch Clients sowie den Abruf der entsprechenden Ergebnisse zur Verfügung.

Der AgentController hält eine Menge verfügbarer Agenten, die sich über eine weitere REST API registrieren. Er leitet Inventarisierungsaufträge an die Agenten weiter und empfängt deren (Teil-)Ergebnisse ebenfalls über die REST API.
Die Auswertung der von den Agenten erhaltenen Teilergebnisse innerhalb des JobControllers erfolgt durch eine ResultProcessor-Komponente. Sie verarbeitet die Teilergebnisse eines Auftrags und integriert diese letzlich zu Gesamtergebnissen (inkl. Topologie), die dann persistiert werden. Der ResultProcessor soll austauschbar sein, so dass unterschiedliche Arten von Teilergebnissen verarbeitet werden können.
Zur Erstellung der Netzpläne wird eine weitere Komponente verwendet: Der NetworkDiagramGenerator erhält Informationen zur Topologie und erstellt daraus einen Netzplan, der Teil des Gesamtergebnisses wird, aber auch separat von Clients über die REST API abgerufen werden kann.

4.3.4 Bausteinsicht Agent



Agenten verfügen ebenfalls über eine REST API über die Inventarisierungsaufträge angenommen werden. Der Agent nutzt einen DeviceEnumerator, der die die Methodik der Erkundung bestimmt und bei Bedarf ausgetauscht bzw. angepasst werden kann. Die Rückgabe von Agenten zu einem Auftrag enthält eine Liste der entdeckten Geräte und zu diesen ermittelten Informationen. Außerdem gibt der Agent Informationen zum Subnetz, in dem er sich befindet, zurück, da diese von Bedeutung für die Ermittlung der Topologie durch den Manager sind.

4.4 Clientschnittstelle

4.4.1 Erteilung von Aufträgen

Die Erteilung von Aufträgen erfolgt über die REST-Schnittstelle des JobControllers. Dazu wird ein JSON-Objekt im Body einen HTTP-Post-Requests an *http://<manager_ip>:<client_api_port>/jobs* geschickt:

```
{
  "parameters": {
    "targetRange": "1.2.3-5.6,7",
    "callbackUri": "http://1.2.3.4:8080"
  }
}
```

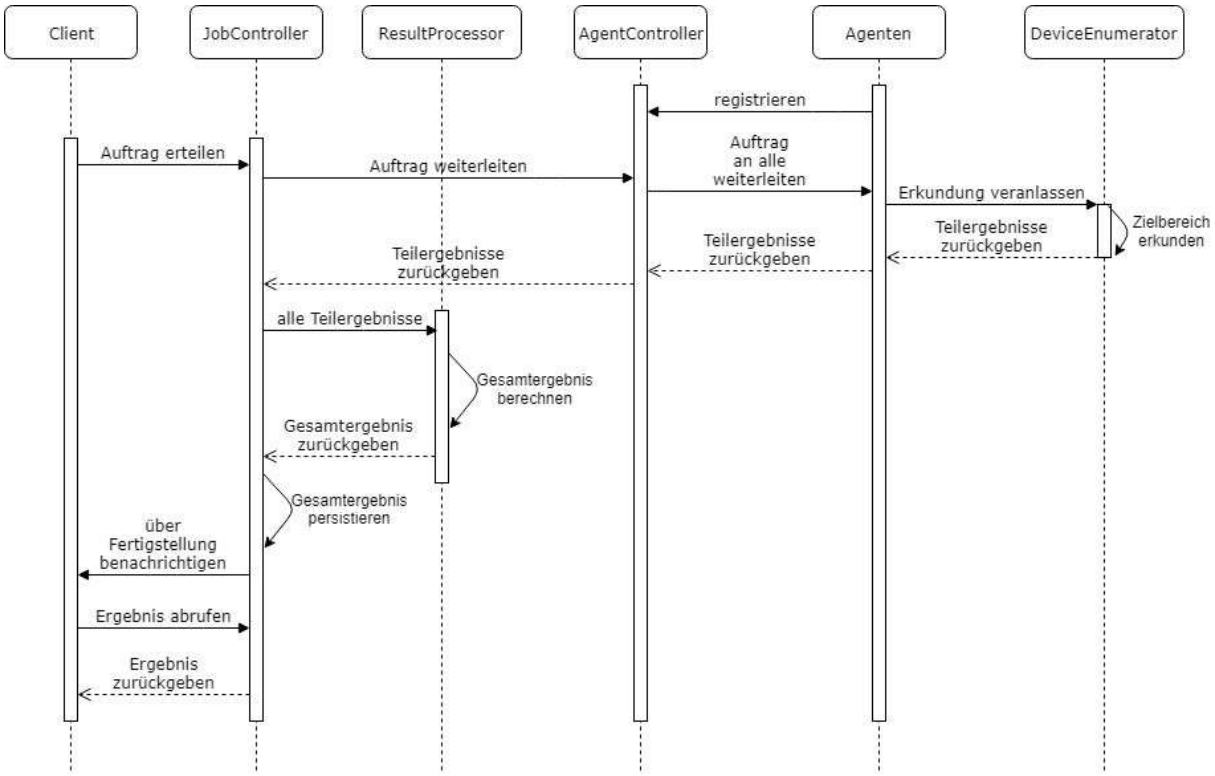
Das zu sendende Objekt benötigt nur ein Attribut mit dem Namen "parameters", welches selbst ein Objekt ist, das eine Menge von Schlüssel-Wert-Paaren enthält. Auf diesem Wege bleibt das Objekt zum Erteilen von Aufträgen flexibel und es können beliebige Schlüssel-Wert-Paare hinzugefügt werden, über die für den Agenten (bzw. dem jeweilig genutzten DeviceEnumerator) Vorgaben für den Auftrag übergeben werden können. Verpflichtend ist lediglich "targetRange", welches den Zieladressbereich angibt. Die Angabe erfolgt auf die gleiche flexible Weise, wie es auch bei Nmap der Fall ist. Im obigen Beispiel würden also folgende (fiktive) Adressen untersucht werden: 1.2.3.6, 1.2.3.7, 1.2.4.6, 1.2.4.7, 1.2.5.6 und 1.2.5.7. [vgl. Lyon, 2016, TS]
Um bei abgeschlossenem Auftrag informiert zu werden kann optional kann eine "callbackUri" angegeben werden, auf die ein HTTP-GET-Request ausgeführt werden soll.
Bei erfolgreicher Erstellung des Auftrags erhält man eine Antwort mit dem Statuscode 201. Zusätzlich wird im Location-Header der Antwort angegeben, wo die Ergebnisse des Auftrags abzurufen sind.

4.4.2 Abruf von Ergebnissen

Die Ergebnisse sind abzurufen per HTTP-GET-Request an die URI, die bei Erstellung des Auftrags im Location-Header zurückgegeben wurde. Ist der Auftrag noch nicht abgeschlossen, erhält man eine Antwort mit dem Statuscode 409. Ansonsten erhält man im Body der Antwort ein JSON-Objekt mit dem Ergebnis. Details dazu werden in Abschnitt 5 erläutert.
Unter anderem erhält das Ergebnis-Objekt auch ein Attribut *netzplanUri*, welches die Adresse enthält, unter der der Netzplan abrufbar ist.

4.5 Programmlauf

Das folgende Sequenzdiagramm zeigt einen beispielhaften Programmlauf.



5. Implementierung

Die Implementierung erfolgt in Java 8. Die Zielplattform ist Ubuntu Server 16.04 LTS mit folgenden Abweichungen von der Standardkonfiguration:

- Paket *default-jre* installiert

Für Maschinen auf denen Agenten mit dem beispielhaft implementierten DeviceEnumerator (NmapDeviceEnumerator) ausgeführt werden, gelten weiterhin folgende Voraussetzungen:

- Die neueste Nmap Version muss installiert sein. (Da sich in den offiziellen Paketquellen eine ältere Version befindet, bietet es sich an, den Quelltext aus dem offiziellen Repository[6] selbst zu kompilieren.)
- Die Umgebungsvariable $PATH muss um das Installationsverzeichnis von Nmap erweitert werden, so dass Nmap über den Befehl *nmap* möglich ist.
- Es muss sichergestellt werden, dass Nmap mit erhöhten Rechten arbeiten kann. (z.B. indem der Agent mit erhöhten Rechten ausgeführt wird)
- Bei der Maschine handelt es sich um ein Endgerät mit nur einem Netzwerkinterface (außer dem Loopback-Interface).
- Der lokale Hostname sollte die Maschine als Agenten-Maschine identifizieren, da sie ebenfalls im erzeugten Netzplan erscheint

5.1 Verwendete Entwicklungswerkzeuge

Als Entwicklungsumgebung wurde die Eclipse IDE for Java Developers[7] verwendet.
Zur Versionsverwaltung wurde Git[8] mit einem Repository von Github[9] benutzt.
Als Build-Management-Tool wurde Apache Maven[10] verwendet, welches es ermöglicht, Abhängigkeiten zentral zu deklarieren. Dank der Unterstützung durch Eclipse werden Abhängigkeiten beim Build, sofern nicht bereits vorhanden, automatisch heruntergeladen.

Zusätzlich wurde Project Lombok[11] verwendet, eine Java-Bibliothek und IDE-Plugin, das durch Annotationen im Java-Code dazu angewiesen wird, im Hintergrund oft verwendeten Boilerplate-Code zu generieren. So können u.a. Getter, Setter, Konstruktoren oder sinnvolle Hashcode- und Equals-Überschreibungen generiert und genutzt werden, ohne dass diese im Quellcode zu sehen sind.

[6] siehe https://svn.nmap.org/
[7] siehe https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygenr
[8] siehe https://git-scm.com/
[9] siehe https://github.com
[10] siehe https://maven.apache.org/
[11] siehe https://projectlombok.org/

5.2 Verwendete Bibliotheken und Frameworks

Spark
Bei Spark[12] handelt es sich um ein Framework zur Erstellung von Web-Anwendungen in Java und Kotlin. Der Fokus wird hier auf Einfachheit und die Minimierung von Boilerplate-Code gelegt.
Spark verfügt über einen eingebetteten Jetty-Webserver. Durch HTTP-Verben, Pfade und Callback-Methoden lassen sich hier auf einfache Weise Routen definieren.
In der vorliegenden Arbeit wurde Spark genutzt, um die verwendeten REST APIs zu realisieren.

Unirest
Unirest[13] ist eine Sammlung von Bibliotheken, die für zahlreiche Programmiersprachen existiert und dem einfachen Erstellen von HTTP-Requests dient.
Hier wurde Unirest für die Kommunikation der Komponenten untereinander, über Ihre REST APIs, genutzt.

nmap4j
nmap4j[14] ist eine Java-Bibliothek zur Nutzung von Nmap. Es ist möglich Nmap auszuführen, sowie dessen Output zu parsen und zu persistieren.
Der ursprüngliche Code ist unter o.g. Quelle zu finden. Für diese Arbeit wurde jedoch der Fork unter https://github.com/narkisr/nmap4j genutzt, da das ursprüngliche Repository nicht mehr gepflegt wird.
In der vorliegenden Arbeit wurde nmap4j genutzt, um den XML-Output von Nmap parsen. Aus dem Output ließen sich auf diese Weise Objekte erzeugen, so dass eine weitere Verarbeitung direkt möglich war.

Gson
Gson[15] ist eine Java-Bibliothek mit der JSON-Objekte in Java-Objekte und umgekehrt umgewandelt werden können.
Hier wurde Gson genutzt um die Übertragung von Objekten per HTTP zu ermöglichen.

Apache Commons Net
Apache Commons Net[16] ist eine Bibliothek, die eigentlich dem Zugriff auf zahlreiche grundlegende Internetprotokolle dient.
In der vorliegenden Arbeit wurde lediglich eine Klasse genutzt, die es ermöglicht zu überprüfen, ob sich eine gegebene IP-Adresse innerhalb eines in CIDR-Notation gegebenen Subnetzes befindet. Zwar wäre es auch möglich gewesen, eine solche Funktion selbst zu entwickeln, um die Abhängigkeiten gering zu halten, doch gerade durch die wenige Verwendung entstünde auch keine großes Problem, wenn der Code in Zukunft nicht mehr verwendet werden könnte.

[12] siehe http://sparkjava.com/
[13] siehe http://unirest.io/java.html
[14] siehe https://sourceforge.net/projects/nmap4j/
[15] siehe https://github.com/google/gson
[16] siehe https://commons.apache.org/proper/commons-net/

vis.js
Bei vis.js[17] handelt es sich um eine Visualisierungsbibliothek, die verschiedene Diagrammtypen mit Hilfe von Javascript im Browser darstellen kann.
Die "Network"-Komponente bietet eine breite Palette von Möglichkeiten um Netzwerke darzustellen. Die zugrundeliegenden Daten sind dabei im Wesentlichen in Javascript definierte Knoten und Kanten, welche automatisch angeordnet werden.
Die minimalisierte Bibliotheksdatei ist nur gut 600 kB groß und ein mit vis.js erstellter Netzplan kann einfach in einer HTML-Datei definiert werden. Damit ist diese Möglichkeit nicht nur leichtgewichtig, sondern auch sehr portabel.

5.3 Austauschbare Module

Generell gilt, dass ein DeviceEnumerator und ein ResultProcessor aufeinander abgestimmt sein müssen. Da letzterer die gesammelten Ergebnisse des ersten verarbeitet. Dies betrifft vor allem das vom DeviceEnumerator erzeugte AgentResult, welches an den Manager übermittelt und schließlich mit den anderen Agentenergebnissen vom ResultProcessor verarbeitet werden.
Die konkreten Implementationen werden per Dependency Injection in den jeweiligen Startup-Klassen übergeben (ManagerStartup bzw. AgentStartup), welche die Main-Methode enthalten, durch die die jeweilige Komponente gestartet wird.
Im Folgenden sollen die Beispielimplementationen erläutert werden:

NmapDeviceEnumerator
Der von den Agenten genutzte DeviceEnumerator basiert auf Nmap: Die Agenten scannen den Zieladressbereich mit den Nmap-Optionen *-O* und *-sV*. Standardmäßig wird ein DNS-Reverse-Lookup der Ziel-Adressen versucht, die 1000 wichtigsten Ports gescannt und es erfolgt eine Betriebssystem- und Serviceerkennung.
Über eine weiteres Schlüssel-Wert-Paar mit dem Schlüssel "nmapParameters" innerhalb des "parameters"-Objekts bei der Auftragserteilung können weitere Parameter für Nmap festgelegt werden. Der Wert "-sS -sU -p-" würde z.B. dafür sorgen, dass alle TCP- und UDP-Ports gescannt werden.
Zusätzlich werden auch alle Maschinen, auf denen sich Agenten befinden, mit der Option *--traceroute* gescannt, um die nötigen Informationen zum Ermitteln der Topologie zu gewinnen.
Die so gewonnenen Informationen bilden den wesentlichen Teil der Informationen, die zurück an den Manager geschickt werden.

TraceResultProcessor
Der vom Manager genutzte ResultProcessor integriert die von den Agenten übermittelten Informationen:
Zuerst werden die IP-Adressen der entdeckten Geräte den bekannten Subnetzen zugeordnet. Dann werden alle Traces um einen "nullten" Hop erweitert, so dass der Trace A - B mit dem Trace B - A vergleichbar wird. Die Alias-Auflösung funktioniert dann, wie am Ende von Abschnitt 4.2.1 (ab "Heuristik") beschrieben.

[17] siehe http://visjs.org/

Dann werden die Informationen der verschiedenen Agenten verglichen und es werden Perspektiven gebildet, die Aufschluss darüber geben, wie die entdeckten Geräte von verschiedenen Standpunkten aus wahrgenommen werden.
Schließlich wird der Netzplan generiert und das Gesamtergebnis an den Manager zurückgegeben.

5.4 Ergebnis als JSON-Objekt

Im Folgenden sollen die Ergebnisse, die als JSON-Objekt zurückgegeben werden, näher erläutert werden. Dazu will ich auf die einzelnen Attribute des Objekts eingehen:

jobId: eine ID für den jeweiligen Auftrag, der auf Basis der Systemzeit vom Manager zugewiesen wird

subnetsInfo: Beinhaltet ein Objekt, das aus zwei Arrays besteht:

1. *subnets*: Die durchsuchten Subnetze welche wiederum Objekte sind:
    a. *CIDRAddress*: beschreibt das Subnetz in CIDR-Notation
    b. *usedIps*: ein Array mit entdeckten, genutzten IP-Adressen innerhalb des Subnetzes
2. *discoveredIpsWithoutSubnetMapping*: ein Array, das entdeckte IP-Adressen beinhaltet, die keinem bekannten Subnetz zugeordnet werden können. (Dies kann zum einen daran liegen, dass es ein Subnetz gibt, in dem sich noch kein Agent befindet. Zum anderen kann es sich um Geräte handeln, die z.B. bei Traceroutes beiläufig entdeckt wurden und außerhalb der bekannten Subnetze liegen.)

devices: Ein Array, das für jedes gefundene Gerät aus dem Zielbereich ein Objekt enthält. Diese sind wie folgt aufgebaut:

1. *ips*: ein Array mit den IP-Adressen, die diesem Gerät zugeordnet sind
2. *subnets*: ein Array mit Subnetzadressen in CIDR-Notation für jedes Subnetz, zu dem das Gerät gehört
3. *perspectives*: ein Array von Objekten, die Aufschluss darüber geben, wie das jeweilige Gerät von den Agenten wahrgenommen wird. (Verfügt ein Gerät über keine Perspektive, so liegt es außerhalb des Zielbereichs und wurde beiläufig entdeckt.) Wird ein Gerät von verschiedenen Agenten gleich wahrgenommen, so werden die Informationen zusammengefasst:
    a. *perspectiveSourceIps*: ein Array von IP-Adressen, das die Adresse jedes Agenten enthält, der das Gerät wie folgt wahrnimmt.
    b. *targetIP*: Zieladresse des Gerät unter der es wie folgt wahrgenommen wird.
    c. *deviceInfo*: die eigentliche Information über das Gerät mit folgenden Attributen:
        i. *ipV4Addresses*: ein Array mit IP-Adressen für dieses Gerät (enthält in der Regel nur die targetIP)
        ii. *hostname*: per DNS ermittelter Hostname bzw. lokaler Hostname bei Agenten-Maschinen
        iii. *ports*: ein Array mit Objekten, die Informationen über nicht geschlossene Ports beinhalten. Um an dieser Stelle zu verkürzen sei

nur gesagt, dass hier u.a. Informationen über Portnummer, Protokoll, Dienst, Produkt, Version und CPE-Namen enthalten sind.

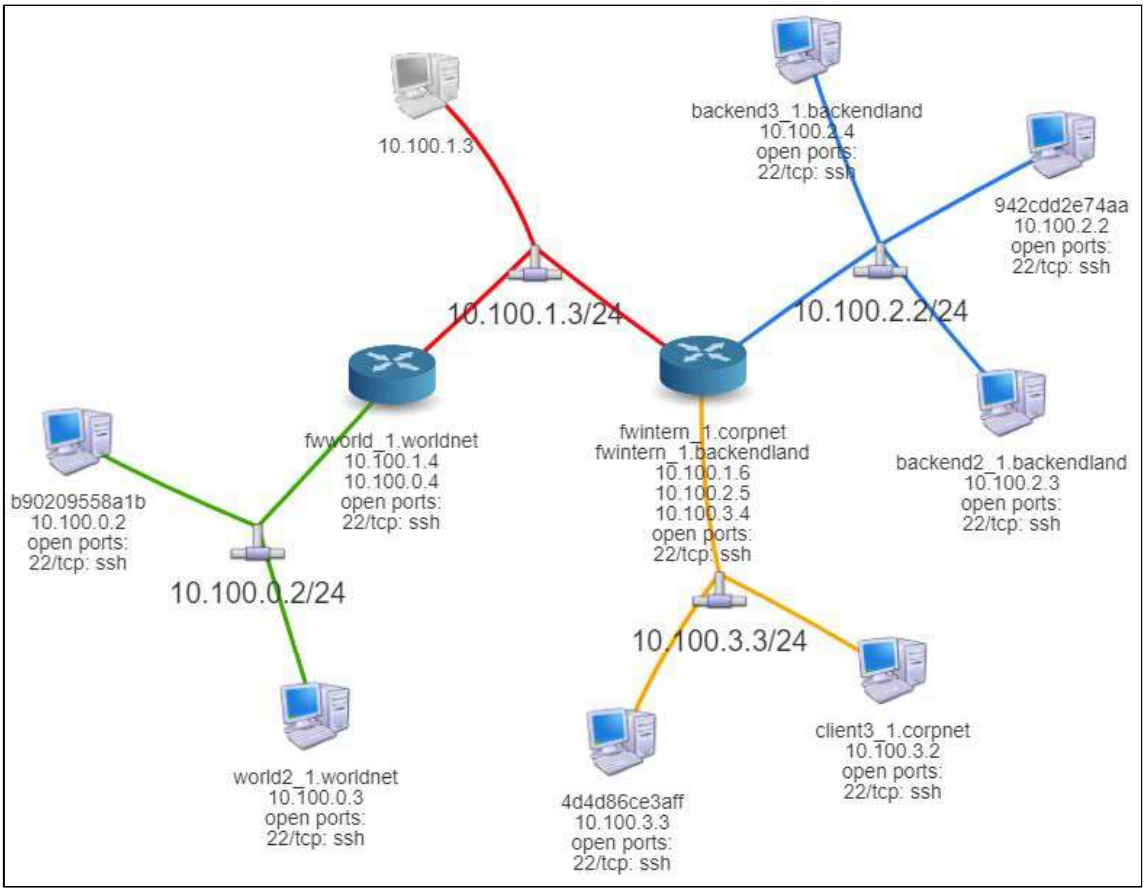
iv. *osCpes*: ein Array mit möglichen CPE-Namen, die Ergebnis der Betriebssystemerkennung waren

traces: ein 2-dimensionales Array, das Traces enthält, die statt aus einfachen Hop-Adressen aus *Device*-Objekten besteht. (Für andere DeviceEnumerator- und ResultProcessor-Implementierungen könnte an dieser Stelle ein anderes Attribut sinnvoll sein. Um die Implementierung einfach zu halten und dennoch die wichtigsten ermittelten Daten im Ergebnis mitzuliefern, wurde dieser Weg gewählt.)

netzplanUri: URI unter der der generierte Netzplan als HTML-Datei abrufbar ist

netzplan: Die Netzplan-Datei selbst als String eingebettet.

5.5 Visualisierung



Die oben dargestellte Abbildung zeigt einen beispielhaften, von *ANIS* erstellten Netzplan. Um eine optimale Übersicht zu erlauben, wird sich auf die wichtigsten Informationen über Geräte und deren Verbindungen beschränkt:

- Geräte werden als Knoten dargestellt und mit ggf. entdeckten Hostnamen, IP-Adressen sowie Informationen zu Port, Protokoll und Dienst versehen. (ein Port wird dargestellt, sobald er aus irgendeiner Perspektive als nicht geschlossen wahrgenommen wird)

    - Gehört ein Gerät zu mehreren Subnetzen wird als Symbol ein Router-Icon statt eines PC-Icons verwendet
    - Ist ein Gerät nicht im Zieladressbereich, wird es mit einem grauen PC-Icon dargestellt.
- Subnetze werden ebenfalls als Knoten mit einem Pipe-Icon dargestellt. Diese sind mit Ihrer Subnetzadresse in CIDR-Notation versehen und haben Verbindungen zu allen Mitgliedern des Subnetzes

5.6 Probleme bei der Implementierung

5.6.1 Fehlerhafte Ausgaben von Nmap

Teilweise sind die Traces zu einzelnen Hosts im Nmap-XML-Output unvollständig: Sie enthalten nicht den ersten Hop, der mit einer TTL von 1 erreicht wurde. Der zweite Hop hingegen ist enthalten. Das Verhalten scheint nicht ohne weiteres reproduzierbar.
Um trotz dieses Problems kurzfristig mit richtigen Ergebnissen arbeiten zu können, wurde ein Workaround entwickelt: Die Traces werden vor der Weiterverarbeitung auf ihre Plausibilität geprüft und bei Bedarf wird ein erneuter Scan für den betroffenen Host ausgeführt. Der fehlerhafte Trace wird dann durch den neuen ausgetauscht. Eine Weiterverarbeitung erfolgt erst, wenn alle Traces fehlerfrei sind.

5.6.2 Lücken in der Implementation von nmap4j

Es gibt einen Fehler in der Implementierung in NMapRunHandlerImpl.java. Beim parsen der Traces innerhalb des Nmap-XML-Outputs wird hier versucht die Portnummer zu einem Long zu parsen. Wird als Methode aber ICMP verwendet, gibt es keine Portnummer, so dass an dieser Stelle mit null gearbeitet wird, weshalb es zu einer NumberFormatException kommt. Ich habe daher eine Prüfung eingebaut, die sicherstellt, dass der Wert nicht null ist.
Zudem musste das Auslesen von CPE-Namen für Services aus dem Nmap-XML-Output nachimplementiert werden.

6. Fazit und Ausblick

Im Folgenden soll kurz über die Ergebnisse der Arbeit und das Erreichen der zu Beginn der Arbeit festgelegten Zielsetzung resümiert werden. Zudem sollen mögliche Ansatzpunkte für aufbauende Arbeiten gegeben werden.

6.1 Was wurde erarbeitet?

Es wurden verschiedene Werkzeuge auf Ihre Nutzbarkeit für die Netzwerkinventarisierung hin analysiert und bewertet. Schließlich wurde mit *ANIS* (Automated Network Inventory System) ein prototypisches System zur Netzwerkinventarisierung entwickelt, mit dem von verschiedenen Standpunkten Daten gewonnen und schließlich zentral gesammelt und verarbeitet werden können. Die vorliegenden Module zur Datengewinnung und -verarbeitung sind dabei als Beispielimplementation anzusehen, da sie durch verschiedene Rahmenbedingungen in unterschiedlichen Netzwerken nicht universell geeignet sind. *ANIS* ist jedoch so ausgelegt, dass entsprechende Module leicht nachimplementiert und integriert werden können.
Zudem sind die Ergebnisse durch ihre strukturierte Darstellung im JSON-Format gut automatisiert weiter nutzbar.
Die Visualisierung der Topologie in Form eines Netzplans, der auch die wichtigsten Kenndaten der entdeckten Geräte beinhaltet, erleichtert es, die gewonnenen Daten zu erfassen und sich darüber auszutauschen.

6.2 Wurde das Ziel erreicht?

Vor allem, was den geringen Installationsaufwand und die universelle Einsetzbarkeit angeht, mussten Abstriche gegenüber den ursprünglichen Erwartungen gemacht werden. Dies ist vor allem der Annahme der Kompartmentalisierung und dem Bestreben, eine korrekte Topologie ohne Zugang zu ermitteln, geschuldet.
Auch ist die Ermittlung von Nutzerkonten und des allgemeinen Softwarestands mit der vorliegenden Beispielimplementierung nicht abgedeckt. Auch dies wäre durch einen Direktzugang mit Leichtigkeit zu lösen gewesen. Hierauf wurde aber bewusst verzichtet, da es als wichtiger bewertet wurde, zu zeigen, was ohne Zugänge und Rückgriff auf spezielle Systeme möglich ist.

6.3 Ausblick

Grundsätzlich könnten für die Zukunft empirische Arbeiten von Interesse sein, die sich damit beschäftigen, welche nutzbaren Mechanismen und Hürden für eine Inventarisierung in der Regel in Organisationsnetzwerken vorhanden sind. Darauf aufbauend könnte *ANIS* ggf. um Module erweitert werden, die für eine größere Zahl von Netzwerken direkt oder zumindest mit geringer Konfiguration einsetzbar ist.
Weiterhin wäre es interessant, die gewonnenen Daten auf Veränderungen zu untersuchen. So könnte der Vergleich von Ergebnissen unterschiedlicher Zeitpunkte z.B. Aufschluss darüber geben, welche Geräte oder Dienste hinzugekommen sind.

7. Literaturverzeichnis

[Acharya, 2009] H. B. Acharya/M. G. Gouda. A Theory of Network Tracing. In: R. Guerraoui/F. Petit (Hg.). Stabilization, Safety, and Security of Distributed Systems. Springer-Verlag, 2009. (S. 62 - 74)

[Acharya, 2010, a] H. B. Acharya/M. G. Gouda. The Weak Network Tracing Problem. In: K. Kant et al. (Hg.). Distributed Computing and Networking. Springer-Verlag, 2010. (S. 184 - 194)

[Acharya, 2010, b] H. B. Acharya/M. G. Gouda. On Topology Inference in the Presence of Aliasing. 2010.
http://www.cs.utexas.edu/~acharya/Outputs/Tracing%20and%20Monitoring/3/3.pdf (abgerufen am 31.8.2017)

[Asadoorian, 2009] Paul Asadoorian. Using Nmap Results With Nessus Batch Scanning. 2009. https://www.tenable.com/blog/using-nmap-results-with-nessus-batch-scanning (abgerufen am 31.8.2017)

[Asadoorian, 2010] Paul Asadoorian. Common Platform Enumeration (CPE) with Nessus. 2010. https://www.tenable.com/blog/common-platform-enumeration-cpe-with-nessus (abgerufen am 31.8.2017)

[Asadoorian, 2013] Paul Asadoorian. Linux/UNIX Patch Auditing Using Nessus. 2013. https://www.tenable.com/blog/linuxunix-patch-auditing-using-nessus (abgerufen am 31.8.2017)

[Augustin, 2006] Augustin et al.. Avoiding traceroute anomalies with Paris traceroute. Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. ACM, 2006.

[Bailey, 1997] Edward C. Bailey. Maximum RPM. Red Hat Software, Inc., 1997.

[Baloch, 2015] Rafay Baloch. Ethical Hacking and Penetration Testing Guide. CRC Press, 2015.

[Damron, 2016] Jason Damron. Why A Complete Network Inventory is Critical for Security. 2016.
https://blog.greatbaysoftware.com/why-a-complete-network-inventory-is-critical-for-security (abgerufen am 31.8.2017)

[Debian, 2015] Debian Project. ListInstalledPackages. 2015.
https://wiki.debian.org/ListInstalledPackages (abgerufen am 31.8.2017)

[Engebretson, 2013] Patrick Engebretson. The Basics of Hacking and Penetration Testing, Second Edition. Elsevier, 2013.

[Ferrari, 2009] Martin Ferrari. net-tools future. 2009. https://lists.debian.org/debian-devel/2009/03/msg00780.html (abgerufen am 31.8.2017)

[Grailet, 2016] Jean-Francois Grailet et al.. TreeNET: Discovering and Connecting Subnets. In: 8th International Workshop on Traffic Monitoring and Analysis (TMA). University of Liege, 2016. http://hdl.handle.net/2268/194693

[Gregg, 2015] Michael Gregg. The Network Security Test Lab: A Step-by-Step Guide. John Wiley & Sons, Inc., 2015.

[Gula, 2009] Ron Gula. Enhanced Operating System Identification with Nessus. 2009. https://www.tenable.com/blog/enhanced-operating-system-identification-with-nessus (abgerufen am 31.8.2017)

[Haddadi, 2008] Haddadi et al.. Network Topologies: Inference, Modeling, and Generation. In: Nelson L.S. Da Fonseca (Hg.). IEEE Communications Surveys & Tutorials, Volume: 10, Issue: 2, Second Quarter 2008. IEEE, 2008. (S. 48 - 69)

[Dinger, 2008] J. Dinger/H. Hartenstein. Netzwerk- und IT-Sicherheitsmanagement. Eine Einführung. Universitätsverlag Karlsruhe, 2008.

[Hertzog, 2015] R. Hertzog/R. Mas. The Debian Administrator's Handbook. R. Hertzog, R. Mas, Freexian SARL, 2015. https://debian-handbook.info/download/stable/debian-handbook.pdf

[Holbert, 2014] Brett D. Holbert. Network Topology Inference with Partial Path Information and Probabilistic Cascading Failures through Interdependent Networks. PennState University Libraries, 2014. https://etda.libraries.psu.edu/files/final_submissions/10168

[Huffaker, 2002] Bradley Huffaker et. al. Topology discovery by active probing. In: Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops. IEEE, 2002. (S. 90 - 96)

[Jorgensen, 2015] Robert Jorgensen. Network Inventory, Configuration Management, and Security. 2015. http://blog.securitymetrics.com/2015/08/network-inventory-configuration.html (abgerufen am 31.8.2017)

[Kappes, 2013] Martin Kappes. Netzwerk- und Datensicherheit, Eine praktische Einführung. 2. Auflage. Springer Vieweg, 2013.

[Karlsson] Patrik Karlsson. File duplicates. https://nmap.org/nsedoc/scripts/duplicates.html (abgerufen am 31.8.2017)

[Kerrisk, 2017, getent] Michael Kerrisk. getent(1). 2017. http://man7.org/linux/man-pages/man1/getent.1.html (abgerufen am 31.8.2017)

[Kerrisk, 2017, last] Michael Kerrisk. last(1). 2017.

http://man7.org/linux/man-pages/man1/last.1.html (abgerufen am 12.9.2017)

[Kerrisk, 2017, ss] Michael Kerrisk. ss(8). 2017.
http://man7.org/linux/man-pages/man8/ss.8.html (abgerufen am 31.8.2017)

[Keys, 2010] Ken Keys. Internet-Scale IP Alias Resolution Techniques. ACM SIGCOMM Computer Communication Review 40, no. 1. ACM New York, 2010. (S. 50 - 55)

[Limoncelli, 2007] Limoncelli et al..The Practice of System and Network Administration, Second Edition. Addison-Wesley, 2007.

[Linux Foundation, 2015] Linux Foundation. lsb_release. 2015.
https://refspecs.linuxfoundation.org/LSB_5.0.0/LSB-Core-generic/LSB-Core-generic/lsbrelease.html (abgerufen am 12.9.2017)

[Lyon, 2008] Gordon "Fyodor" Lyon. Nmap Network Scanning: Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Com LLC, 2008.

[Lyon, 2017, CL] Gordon "Fyodor" Lyon. Changelog. 2017.
https://nmap.org/changelog.html (abgerufen am 1.9.2017)

[Lyon, 2016, CPE] Gordon "Fyodor" Lyon. Common Platform Enumeration (CPE). 2016.
https://nmap.org/book/output-formats-cpe.html (abgerufen am 31.8.2017)

[Lyon, 2016, HD] Gordon "Fyodor" Lyon. Host Discovery. 2016.
https://nmap.org/book/man-host-discovery.html (abgerufen am 31.8.2017)

[Lyon, 2016, HF] Gordon "Fyodor" Lyon. The History and Future of Nmap. 2016.
https://nmap.org/book/history-future.html (abgerufen am 31.8.2017)

[Lyon, 2016, NSE] Gordon "Fyodor" Lyon. Chapter 9. Nmap Scripting Engine. 2016.
https://nmap.org/book/nse.html (abgerufen am 31.8.2017)

[Lyon, 2016, OD] Gordon "Fyodor" Lyon. Chapter 9. OS Detection. 2016.
https://nmap.org/book/man-os-detection.html (abgerufen am 2.9.2017)

[Lyon, 2016, PSB] Gordon "Fyodor" Lyon. Port Scanning Basics. 2016.
https://nmap.org/book/man-port-scanning-basics.html (abgerufen am 31.8.2017)

[Lyon, 2016, PST] Gordon "Fyodor" Lyon. Port Scanning Techniques. 2016.
https://nmap.org/book/man-port-scanning-techniques.html (abgerufen am 31.8.2017)

[Lyon, 2016, VD] Gordon "Fyodor" Lyon. Service and Version Detection. 2016.
https://nmap.org/book/man-version-detection.html (abgerufen am 1.9.2017)

[Lyon, 2016, VS] Gordon "Fyodor" Lyon. Technique Described. 2016. https://nmap.org/book/vscan-technique.html (abgerufen am 31.8.2017)

[Lyon, 2016, TS] Gordon "Fyodor" Lyon. Target Specification. 2016. https://nmap.org/book/nping-man-target-specification.html (abgerufen am 31.8.2017)

[Lyon, 2016, XML] Gordon "Fyodor" Lyon. XML-Output. 2016. https://nmap.org/book/output-formats-xml-output.html (abgerufen am 31.8.2017)

[Manzuik, 2007] Steve Manzuik. Network Security Assessment: From Vulnerability to Patch. Syngress Publishing, 2007.

[Motamedi, 2015] Reza Motamedi et al.. A Survey of Techniques for Internet Topology Discovery. In: IEEE Communications Surveys & Tutorials, Issue 2, Secondquarter 2015. IEEE, 2015. (S. 1044 - 1065)

[Nazir, 2007] Fawad Nazir et al.. Constella: A Complete IP Network Topology Discovery Solution. In: S. Ata/C. S. Hong (Hg.). Managing Next Generation Networks and Services. Springer-Verlag, 2007. (S. 425 - 436)

[Negus, 2015] Christopher Negus. Linux Bible. The Comprehensive, Tutorial Resource. Ninth Edition. John Wiley & Sons, 2015.

[Net-SNMP, 2011] Net-SNMP. Man pages. 2011. http://www.net-snmp.org/docs/man/ (abgerufen am 31.8.2017)

[OpenBSD, 2017, SSHD] OpenBSD. SSHD_CONFIG(5). 2017. http://man.openbsd.org/sshd_config (abgerufen am 31.8.2017)

[OpenBSD, 2017, DIG] OpenBSD. dig(1). 2017. http://man.openbsd.org/dig.1 (abgerufen am 31.8.2017)

[Pompon, 2016] Raymond Pompon. IT Security Risk Control Management: An Audit Preparation Plan. Apress, 2016.

[RFC1213, 1991] K. McCloghrie/M. Rose. RFC1213, Management Information Base for Network Management of TCP/IP-based internets: MIB-II. IETF, 1991. https://www.ietf.org/rfc/rfc1213.txt

[RFC1812, 1995] F. Baker (Hg.). RFC1812, Requirements for IP Version 4 Routers. IETF, 1995. https://www.ietf.org/rfc/rfc1812.txt

[RFC2790, 2000] S. Waldbusser/P. Grillo. RFC2790, Host Resources MIB. IETF, 2000. https://tools.ietf.org/html/rfc2790

[Schwenkler, 2006] Thomas Schwenkler. Sicheres Netzwerkmanagement. Springer-Verlag, 2006.

[Sectools, 2017, SCAN] SecTools.Org: Top 125 Network Security Tools. 2017. http://sectools.org/tag/port-scanners/ (abgerufen am 31.8.2017)

[Sectools, 2017, SNIFF] SecTools.Org: Top 125 Network Security Tools. 2017. http://sectools.org/tag/sniffers/ (abgerufen am 31.8.2017)

[Sectools, 2017, VULN] SecTools.Org: Top 125 Network Security Tools. 2017. http://sectools.org/tag/vuln-scanners/ (abgerufen am 31.8.2017)

[Siamwalla, 1998] R. Siamwalla et al. Discovering Internet Topology. Unveröffentlichtes Manuskript, 1998. http://www.cs.cornell.edu/skeshav/papers/discovery.pdf (abgerufen am 31.8.2017)

[Sourceforge, 2013] Sourceforge. NETSTAT. 2013. http://net-tools.sourceforge.net/man/netstat.8.html (abgerufen am 31.8.2017)

[Svensson, 2016] Robert Svensson. From Hacking to Report Writing: An Introduction to Security and Penetration Testing. Apress, 2016.

[Tanenbaum, 2011] A.S. Tanenbaum/D. J. Wetherall. Computer Networks. Fifth Edition. Pearson Education, 2011.

[Tenable, 2017, CC] Tenable Network Security, Inc. Enabling the Compliance Checks With Nessus. 2017 https://www.tenable.com/tips/enabling-the-compliance-checks-with-nessus (abgerufen am 31.8.2017)

[Tenable, 2017, SD] Tenable Network Security, Inc. Plugins: Service detection. 2017 https://www.tenable.com/plugins/index.php?view=all&family=Service+detection (abgerufen am 2.9.2017)

[Tenable, 2017, UG] Tenable Network Security, Inc. Nessus 6.10 User Guide. Tenable Network Security, Inc., 2017. https://docs.tenable.com/nessus/Content/Resources/PDF/Nessus_6_10.pdf

[Tenable, 2017, Nessus] Tenable Network Security, Inc. Nessus Vulnerability Scanner. 2017 https://www.tenable.com/products/nessus-vulnerability-scanner (abgerufen am 31.8.2017)

[Tenable, 2017, Obtain] Tenable Network Security, Inc. Obtain An Activation Code. 2017 https://www.tenable.com/products/nessus/nessus-plugins/obtain-an-activation-code (abgerufen am 31.8.2017)

[Tenable, 2017, Monitor] Tenable Network Security, Inc. Nessus Network Monitor. 2017 https://www.tenable.com/products/nessus-network-monitor (abgerufen am 31.8.2017)

[Trends, 2017, SCAN]
https://trends.google.de/trends/explore?cat=344&date=2016-09-12%202017-09-12&q=nmap,amap,superscan,netscantools,angry%20ip%20scanner (abgerufen am 12.9.2017)

[Trends, 2017, SNIFF]
https://trends.google.de/trends/explore?cat=311&date=2016-09-12%202017-09-12&q=wireshark,tcpdump,ettercap,cain%20and%20abel,capsa (abgerufen am 12.9.2017)

[Trends, 2017, VULN]
https://trends.google.de/trends/explore?cat=344&date=2016-09-12%202017-09-12&q=nessus,openvas,nexpose,saint,languard (abgerufen am 12.9.2017)

[Wilhelm, 2013] Thomas Wilhelm. Professional Penetration Testing, Second Edition. Elsevier, 2013.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. September 2017 Lutz Jankowski