Hochschule München
Fakultät für Informatik und Mathematik

Implementierung eines Werkzeugs zur Informationssammlung über Domains zum Zweck der Sicherheitsanalyse

Implementation of a Information Collection Toolset for Domain Security Analysis

BACHELORARBEIT

Alexander Waldeck
Matrikel-Nummer 04718511

Prüfer: Prof. Dr. Hof

Erklärung

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum Unterschrift

Zusammenfassung

In dieser Bachelorarbeit geht es um die Konzeption und Entwicklung eines Werkzeugs zur Informationssammlung über Domains. Der Zweck dieser Informationssammlung ist eine spätere Auswertung der Daten um eine Sicherheitsanalyse und eine Bewertung einer Domain durchzuführen. Hierbei wird auf bereits vorhandene Tools eingegangen und deren, meist kommerzielles Geschäftsmodell untersucht. Außerdem werden die Informationsquellen untersucht, aus denen sich Daten gewinnen lassen. Darauf aufbauend wird mit Standardmethoden der Softwareentwicklung ein Konzept für die Informationsgewinnung erarbeitet und dokumentiert. Anschließend wird auf die Implementierung der Software in Python eingegangen und die Effizienz des Programms bezüglich gefundener Daten und Laufzeit der Datensuche beurteilt.

Inhaltsverzeichnis

1 Einleitung

Im modernen Internet entstehen Tag für Tag neue Gefahren für die Sicherheit in der Informationstechnik [1]. Viren werden geschrieben und verbreiten sich, Angreifer übernehmen Netzwerke und Server, Informationen werden ausgespäht und Webseiten werden lahm gelegt. Sicherheitsabteilungen in Firmen und Organisationen haben daher ein Interesse daran potentiell gefährliche Domains auf denen z.B. Command & Control Server für die Kontrolle über Botnetze laufen zu erkennen und eventuell zu blockieren um das eigene Netzwerk zu schützen und nicht Teil eines Botnetzes zu werden. Der Begriff Domain bedeutet so viel wie *Herrschaftsbereich* und genau das stellt eine Domain in der Informatik dar. Ein zusammenhängender Namensraum in einem Netzwerk wie dem Internet. Der Besitzer oder Verwalter einer solchen Domain kann nun bösartige Absichten haben. Um Benutzer vor Domains mit fragwürdigem Zweck zu schützen, ist es wichtig eine Unterscheidung zu ermöglichen und *Gut* von *Böse* zu trennen.

Im Rahmen dieser Bachelorarbeit wird der Teil eines Domain-Bewertungssystem entstehen welcher Informationen über Domains sammelt, Daten daraus extrahiert und sie für eine Weiterverarbeitung geeignet abspeichert.
Dabei sollen folgende Punkte behandelt werden:

- Analyse von Eigenschaften die über eine Domain in Erfahrung gebracht werden können.
- Identifizierung von Datenquellen, aus denen Informationen bezogen werden können.
- Erstellung eines Tools in der Programmiersprache Python, welches Informationen aus den Datenquellen extrahiert und für eine weitere Auswertung geeignet speichert.
- Der Bewertungsalgorithmus soll nicht Teil dieser Arbeit werden.

Im Folgenden wird ein kurzer Überblick über die Aufteilung der kommenden Kapitel gegeben.

Kapitel 2 beschreibt den Hintergrund der Arbeit und soll einen Einblick in die Grundlagen der Domains und ihrem Zweck geben.

In **Kapitel 3** wird die Einzigartigkeit dieser Arbeit erklärt und auf bereits vorhandene Tools eingegangen.

Kapitel 4 Analysiert die verschiedenen Arten der Informationsquellen und Daten, die daraus extrahiert werden können. Anschließend wird noch eine Anforderungsanalyse an die zu erstellende Software durchgeführt.

In **Kapitel 5** wird das technische Konzept der Software erstellt und mit Diagrammen ein Überblick über die Software und ihre Funktionen gegeben.

Kapitel 6 Beschreibt die Implementierung und das Vorgehen bei der Entwicklung und Programmierung.

Kapitel 7 Prüft die Software hinsichtlich ihrer Leistungsfähigkeit und Qualität der gesammelten Daten.

In **Kapitel 8** wird ein Ausblick gegeben, wie die zukünftige Weiterentwicklung aussehen und was mit den gesammelten Daten passieren könnte.

2 Grundlagen

Dieses Kapitel beschreibt die Grundlagen, die benötigt werden um den Inhalt der Arbeit zu verstehen.

2.1 Domains

Ein DNS-Name ist ein Teilbereich des Domain-Name-Systems (DNS). Einfacher ausgedrückt ist es der einfach zu merkende, eindeutige Name eines Servers im Internet. Eine Domain kann auch in beliebig viele Subdomains aufgeteilt werden um eine feinere Strukturierung beispielsweise innerhalb einer Firma zu realisieren. Das DNS ist dann dafür zuständig den Namen auf eine IP-Adresse zu übersetzen um den gesuchten Server zu finden. Der DNS Namensraum wird in einer Baumstruktur dargestellt. Jeder Knoten im Baum hat eine Bezeichnung, wobei nur Geschwisterknoten nicht die selbe Bezeichnung haben dürfen. Der Domainname eines Knotens setzt sich schließlich aus dem Weg vom Knoten zur Wurzel zusammen, wobei alle Bezeichnungen mit einem Punkt verkettet werden [2]. Siehe Abbildung 2.1.

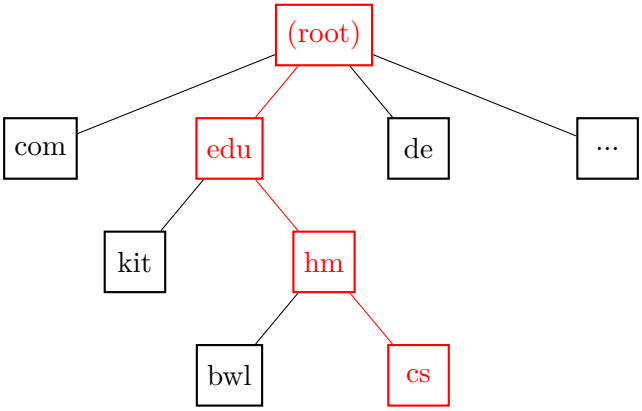

Abbildung 2.1: Domain cs.hm.edu

Jeder Knoten im Namensraum speichert außerdem weitere, relevante Informationen in sogenannten Ressource Records oder RRs. Diese Records bestehen aus:

- **Owner:** Domainname des Eintrags
- **Type:** Art des Eintrags z.B. `A` für IP-Adresse oder `NS` für Nameserver
- **Class:** Verwendete Protokollfamilie, meistens `IN` für Internet
- **TTL:** Time To Live in Sekunden, verwendet wenn RRs zwischengespeichert werden um anzugeben wann sie ablaufen und ungültig werden
- **RData:** Der eigentliche Informationsteil, Typ- und klassenabhängige Daten, z.B. die IP-Adresse zum Type `A` oder der Hostname zum Type `NS` [2]

Bei einer DNS-Anfrage nach einer Domain liefert der DNS-Server nun diese Ressource Records zurück. Ein Webbrowser kann sich nun für den Aufruf einer Seite die `A` Records zum Domainnamen zurückgeben lassen und weiß somit bei welcher IP-Adresse er die Webseite findet. Ein E-Mail Client verwendet hingegen die `MX` Records um den Server zu einer E-Mail Adresse zu finden.

Bei einer Suche nach dem Recordtype `ANY` zur Domain cs.hm.edu liefert der DNS Server die Antwort aus Listing 2.1

```
cs.hm.edu.   1800  IN  NS  ns.e-technik.fh-muenchen.de.
cs.hm.edu.   1800  IN  NS  ns.hm.edu.
cs.hm.edu.   72791 IN  A   129.187.244.60
cs.hm.edu.   14260 IN  MX  10 mailrelay2.lrz.de.
cs.hm.edu.   14260 IN  MX  10 mailrelay1.lrz.de.
```

Listing 2.1: DNS Auflösung von cs.hm.edu

Für diese Arbeit ist wichtig was Domains sind und wofür sie genutzt werden. Erst mit diesem Wissen kann eine Suche nach bösartigen Domains beginnen und erfolgreich werden. In Kapitel 4 werden die für eine Suche relevanten Informationen ermittelt.

2.2 Reverse DNS

Im Domain Name System lässt sich auch ein Domainname zu einer IP-Adresse finden. Eine Reverse-Anfrage liefert meist `PTR` Records zurück, welche auf einen Hostnamen zeigen [3]. Eine Rückwärtssuche ist dann sinnvoll wenn der Hoster zu einer IP-Adresse gefunden werden soll. Oft wird eine Webseite auf einem gemieteten Server betrieben. Eine normale Anfrage liefert die IP-Adresse des Servers. Eine Reverse-Anfrage liefert meist nicht den ursprünglichen Domain Namen, sondern den Namen des Servers auf dem die Webseite betrieben wird. Dieser Name enthält oft hinweise auf den Betreiber beziehungsweise auf den Vermieter. Siehe Listing 2.2.

Um solche Suchen zu ermöglichen wurde im DNS System die Domäne `in-addr.arpa` angelegt. Jede der maximal 255 Subdomains in diesem Ast repräsentiert nun die erste Komponente der gesuchten IP-Adresse. Zum Beispiel `192.in-addr.arpa`. Jede Subdomain hat nun wieder maximal 255 Subdomains welche die zweite Komponente der IP-Adresse repräsentieren. Zur Suche wird dann die IP-Adresse in umgekehrter Reihenfolge vor die Domain `in-addr.arpa` gehängt.

```
google.de.                    242     IN    A      173.194.70.94
94.70.194.173.in-addr.arpa.   84865   IN    PTR    fa-in-f94.1e100.net.
```

Listing 2.2: Auflösung von google.de und revers Auflösung der IP-Adresse

Gibt man nun den Hostnamen `fa-in-f94.1e100.net` in einen Webbrowser ein, wird man auf die Seite www.google.de weitergeleitet.

2.3 Gute und böse Domains

In dieser Arbeit werden immer wieder Begriffe wie *böse Domain* oder *gefährliche Domain* fallen. Hier wird kurz definiert was unter einer bösen Domain zu verstehen ist. Allgemein kann man sagen, dass eine Domain die den Zweck hat Schaden anzurichten als böse eingestuft wird. Da eine Domain an sich nicht böse sein kann, bezieht sich der Begriff immer auf den Betreiber, welcher hinter der Domain steht. Zum Beispiel eine Domain, welche registriert wurde um Kreditkartendaten zu stehlen oder Spam E-Mails zu versenden hat keinen seriösen Zweck. Das Betreiben von Botnetzen und der dazugehörigen Command and Control Server benötigt Domains welche als bösartig eingestuft werden [4]. Eine so eindeutige Unterscheidung ist

jedoch nicht immer so einfach möglich. Eine Webseite, welche einen vermeintlich kostenlosen Dienst anbietet, dem Kunden aber anschließend basierend auf den AGBs in Rechnung stellt, kann auch als bösartig eingestuft werden. Andererseits kann man auch argumentieren, dass der Kunde sich vorhergehend richtig informieren muss. Die Unterscheidung zwischen gut und böse verschwimmt an dieser Stelle. Die Auswertung und Untersuchung der in dieser Arbeit gesammelten Informationen legt schließlich genau fest, ab wann eine Domain als böse zu bezeichnen ist. Dies ist jedoch nicht Teil dieser Arbeit.

3 Related Work

Die meisten vorhandenen Werkzeuge haben einen kommerziellem Hintergrund. Dadurch dass diese Arbeit unter der Open Source Lizenz veröffentlicht werden soll, hebt sie sich von den vorhanden kommerziellen Anbietern ab. Viele private Benutzer oder kleinere Firmen haben oft nicht die finanziellen Mittel, um sich Lizenzen für teure Analysetools zu kaufen. Mit dieser Arbeit wird der Grundstein für ein Domainbewertungssystem gelegt, auf das jeder Zugriff haben kann ohne Geld ausgeben zu müssen. Ein weiterer Vorteil dieses Tools ist die Sammlung von Informationen aus verschiedenen Quellen. Man hat nun die Möglichkeit die selben verschiedener Quellen zu vergleichen und gegebenenfalls Durchschnittswerte zu ermitteln. Die gesammelten Informationen sind teilweise von der Informationsquelle schon vorsortiert oder mit Zusatzinformationen angereichert. Im folgenden werden kurz die bereits vorhandenen Dienste DomainTools, Netcraft und URLmetrics behandelt.

DomainTools ist ein mächtiges Werkzeug zur Domainrecherche, welches auf rein kommerzieller Basis Informationen über Domains zur Verfügung stellt. Auf der Webseite kann man nach freien Domains suchen, Whois Einträge abrufen oder DNS Lookups durchführen. Da das Tool auf einer eigenen Datenbank basiert, ist es sogar möglich eine Reverse Whois suche durchzuführen. Man kann also herausfinden welche Domains auf eine bestimmte Person registriert sind. Um das Angebot abzurunden bieten die Betreiber auch noch Monitoring Dienste, die dem Kunden eine E-Mail schicken, wenn zum Beispiel eine neue Domain mit einem bestimmten Namensbestandteil registriert wird. [5] Die Angebote von DomainTools sind jedoch zum Teil sehr kostspielig. Zur einfachen Benutzung und Integration in andere Systeme bieten die Betreiber eine API an, mit der maschinell Daten ausgelesen und verarbeitet werden können.

Netcraft hingegen legt den Schwerpunkt auf anti Phishing, Security Audits und Internet Datamining. Man kann seine eigenen Internetdienste überwachen, sowie auf Sicherheitslücken hin überprüfen lassen. [6] Im Datamining Sektor bietet Netcraft eine Fülle an Statistiken und Daten über Provider, Hoster und Server. Die Preise für diese kostenpflichtigen Dienste

können per E-Mail nachgefragt werden. Kostenlos kann man sich jedoch eine Toolbar als Browser Plugin herunterladen. Damit wird jede aufgerufene Seite durch Netcraft bewertet und man bekommt einen Hinweis darauf ob die Seite gefährlich sein könnte. Die Toolbar leitet den Benutzer auch auf die Netcraft Seite auf der man sich den Report [7] zu der Bewertung ansehen kann. Dieser Enthält Informationen zum Server, der verwendeten Software, sowie Angaben über den Provider und die Blacklisten, die überprüft wurden. Netcraft bietet ebenso wie DomainTools eine API zur einfachen Abfrage an.

Im Gegensatz zu den beiden anderen Diensten ist **URLMetrics** vollkommen kostenlos. Die Betreiber bieten ein Rankingsystem für US-Amerikanische Seiten an. Die Informationen über eine Seite reichen von Benutzerzahlen über Inhalt der Seite bis hin zu Server IP-Adressen und Ort des Servers. Nachteil des Dienstes ist, dass nicht viele Domains zu finden sind. Hauptsächlich werden Informationen über die größten und populärsten geboten. Für eine Sicherheitsanalyse ist der Dienst daher nur bedingt nutzbar. [8]

Es gibt auch einige Ansätze bösartige Domains oder Botnetze zu finden. Diese basieren aber oftmals auf der Analyse von Datenverkehr zwischen Endbenutzern und DNS Servern [9] [10]. Für eine derartige Analyse muss man entweder Daten von einem Internet Provider zur Verfügung gestellt bekommen, oder man muss direkt auf den Systemen des Providers arbeiten. Diese Arbeit bezieht sich aber nicht auf das finden bösartiger Domains mit Hilfe von Traffic-Analysen, sondern auf die Suche von Informationen über einen gegebenen Domainnamen. Viele der Parameter die bei einer Traffic-Analyse verwendet werden, können jedoch in diese Arbeit mit einfließen.

4 Analyse

In diesem Kapitel wird auf die Art der Informationen, sowie auf die verfügbaren Informationsquellen eingegangen. Im Anschluss daran werden die Anforderungen an die zu entwickelnde Software analysiert.

4.1 Informationen und Informationsquellen

Im Folgenden werden verschiedene Arten der Informationen und Informationsquellen betrachtet. Die einzelnen Abschnitte erklären wozu die Informationen potentiell genutzt werden können und wie verlässlich die Quellen sind. Da eine Domain auch ohne eine Webseite existieren kann muss auch unterschieden werden ob eine Quelle nutzbar ist, wenn auf der Domain kein Webserver betrieben wird.

4.1.1 DNS Einträge

Die Ressource Records aus einer DNS Abfrage beinhalten Informationen, welche für eine Sicherheitsanalyse der Domain relevant sind. Ein Ansatz gefährliche Domains zu identifizieren ist, den DNS Traffic über längere Zeit zu beobachten und Auffälligkeiten zu erkennen. Plötzliche Änderungen der Domain Namen oder der Anfragefrequenz wären hier als verdächtig zu nennen [9]. Dass das Tool keinen Netzwerkverkehr überwachen, sondern nach einer gegebenen Domain suchen soll, muss bei der Konzeption berücksichtigt werden. Der zeitliche Verlauf von DNS Abfragen muss mit aufgezeichnet werden, um Veränderungen der DNS Daten beobachten zu können. Primär spielen jedoch Informationen eine Rolle, die wir mit einer normalen bzw. einer Reverse Anfrage bekommen. Hier kann man zwischen Zeitlichen, Antwortbasierten und TTL-basierten Merkmalen unterscheiden [9].

Unter **zeitlichen Merkmalen** versteht man den zeitlichen Verlauf der DNS Daten. Dazu müssen über einen gewissen Zeitraum immer wieder Anfragen durchgeführt und miteinander verglichen werden. Als Auffälligkeiten wären hier ständig wechselnde IP-Adressen in einem kurzen Zeitraum zu nennen. Bei der Auswertung der gesammelten Daten kann somit ein zeitliches Profil der Domain erstellt werden. Als weiteres zeitliches Merkmal wären hier noch Domains zu nennen, auf die plötzlich ein hoher Ansturm stattfindet und die nach einer gewissen Zeit nicht mehr genutzt werden. Domain Flux wäre eine Technik zur Verschleierung des Standortes eines C&C Servers [11]. Um dies zu erkennen wäre der Traffic zwischen vielen Hosts und DNS Servern zu betrachten. Das übersteigt allerdings den Rahmen dieser Arbeit und wird nicht weiter behandelt.

Unter **Antwort basierten Merkmalen** versteht man diese Informationen, die man aus DNS Abfragen erhält. Hier spielt die Anzahl unterschiedlicher IP-Adressen eine Rolle. Man muss jedoch darauf achten, dass DNS auch dazu verwendet wird eine Lastverteilung zu realisieren, indem verschiedene Adressen in einem Round Robin verfahren zurückgegeben werden [12]. Da nun Angreifer im Internet nicht an Landesgrenzen gebunden sind, entstehen Auffälligkeiten dann, wenn die zurückbegeben Adressen zusätzlich noch in verschiedenen Ländern liegen. Ein weiterer Punkt ist die Möglichkeit, dass viele Domains auf die selbe IP-Adresse auflösen. So etwas kommt dann zustande wenn die Domain bei einem großen Anbieter gehostet wird. Eine Revers oder Rückwärts Suche nach der IP-Adresse gibt meist den Hostnamen des Servers zurück. Durch eine Suchanfrage bei Google lässt sich dann herausfinden ob der Hoster seriös ist (Google Ranking) [9].

Das Domain Name System ist so ausgelegt, dass ein Server DNS Records zwischenspeichern (cachen) kann. Dafür haben DNS Records immer eine **Time to Live** (TTL). Diese gibt an wie lange ein Eintrag gültig ist und wann er verworfen werden sollte. Angreifern kommt dies zugute, da ihre Netze mit einer kürzeren TTL und dem Round Robin verfahren robuster sind. Erscheint eine Domain auf einer Blacklist oder ein Server wird vom Netz genommen, kann sehr schnell darauf reagiert werden. Da die Alte Domain schon nicht mehr gültig ist, wird einfach eine neue registriert. Bösartige Domains tendieren daher zu niedrigeren TTL Werten [9].

4.1.2 Whois

Jede neue Domain wird von einem Network Information Center (NIC) vergeben. Im Allgemeinen ist ein NIC für eine oder mehrere Top Level Domains (TLDs) zuständig. Für die Deutschen .de Domains wäre dies DENIC [13]. Diese NICs betreiben auch sogenannte WHOIS Server. Bei der Registrierung einer neuen Domain werden Informationen über den Eigentümer, sei es eine Person oder eine Organisation gespeichert. Über einen WHOIS Server lässt sich also herausfinden wem eine bestimmte Domain gehört und wer dafür verantwortlich ist. Die Antworten des Servers sind im Volltext Format und menschenlesbar [14]. Eine zentrale Instanz für Whois Abfragen gibt es nicht. Für jede TLD muss der dazugehörige Server befragt werden. Unter Linux übernimmt das Kommandozeilentool "whois" diese Arbeit. Das Fehlen einer einheitlichen Spezifikation von Kodierung, Form und Inhalt eines Whois Eintrages hat leider zur Folge, dass sich die Einträge schwer maschinell verarbeiten lassen. Jeder Whois Server antwortet auf eine andere Art und Weise. Zur Weiterverarbeitung könnten interessante Daten wie Namen, Adressen, E-Mail Adressen, Telefonnummern oder Zeitstempel extrahiert werden.

4.1.3 Alter der Domain

In Verbindung mit den Whois Informationen erscheint auch das Alter einer Domain als interessant. Gerade Domains die für Phishing oder den Versand von Spam verwendet werden, sind meist recht jung [15]. Das Alter der Domain lässt sich auf verschiedene Arten ermitteln. Entweder man benutzt einen Internet Archiv Dienst wie Archive.org [16] bzw. Netcraft.com [7] oder man versucht in den Whois Einträgen der Domain ein Datum zu finden.

Archiv Dienste laufen in regelmäßigen Abständen über Webseiten und archivieren diese. Ein darüber gefundenes Alter entspricht der ersten Archivierung bzw. dem ersten Snapshot den der Dienst von der Webseite hat. Mit diesen Informationen muss man daher kritisch umgehen. Existiert beispielsweise eine Domain schon mehrere Jahre ungenutzt und wird plötzlich für eine Webseite verwendet, spiegelt das zurückgegebene Datum nicht das Alter der Domain wieder, sondern das Alter der gehosteten Webseite. Außerdem muss man bedenken, dass Domains auch ohne eine Webseite existieren können. Diese Informationsquelle wird dann nutzlos. [17]

Der Dienst von Archive.org stellt eine API also Schnittstelle zum Informationsaustausch zur Verfügung, an die eine Anfrage gestellt werden kann. Man bekommt dann ein JSON (Javascript Object Notation) Objekt mit den gewollten Informationen zurück. Netcraft.com bietet keine solche API. Man müsste also die Webseite parsen um das Alter zu erhalten. Dafür stellt Netcraft noch eine Menge weiterer Daten über eine Domain bereit, die eventuell von Interesse sein könnten. (Dazu später mehr)

Der **Google Cache** ist in diesem Fall nicht nutzbar, da Google nur kurzfristige Schnappschüsse einer Webseite erstellt. In der Regel werden alle sieben Tage die Cache Einträge erneuert und die Alten dabei ersetzt. [18]

Diese Methoden der Altersfindung beziehen sich immer auf Domains mit einer Webseite. Um das Alter einer Domain ohne Webseite herauszufinden gibt es noch die Möglichkeit über eine **Whois Abfrage**. Manche NICs speichern das Datum der Domainregistrierung mit ab und stellen die Daten bei einer Whois Abfrage zur Verfügung. Durch die unzureichende Spezifikation des Whois Dienstes ist das aber keine sichere Informationsquelle. (siehe 3.1.2 Whois) Es kann also durchaus sein, dass man für .com oder .net Domains ein Alter erhält, für .de Domains jedoch nicht.

4.1.4 Blacklisten

Auf Blacklisten werden Webseiten, Domains oder IP-Adressen geführt, die als bösartig eingestuft wurden. Moderne Internetbrowser wie Mozilla Firefox [19] oder Google Chrome [20] verwenden während dem Laden von Webseiten solche Blacklisten um die Seiten zu überprüfen. Wenn die Webseite zum Beispiel eine bekannte Phishing Seite ist wird eine Warnung ausgegeben und der Benutzer informiert. E-Mail Programme verwenden Blacklisten um Spam zu filtern. Dazu werden eingehende E-Mails gecheckt ob sie von einer gelisteten Domain stammen.

Blacklisten werden zum einen maschinell erstellt, indem Algorithmen z.B. die Häufigkeit von Werbung und Schlüsselwörtern überprüfen. Zum anderen bieten E-Mail Programme dem Nutzer meiste einen Button um E-Mails als Spam zu markieren. Diese Information kann auch dazu beitragen dass der entsprechende Sender auf eine Blacklist gesetzt wird. Es gibt auch Ansätze um aus bekannten Einträgen neue abzuleiten. Dazu werden Heuristiken bzw. analytische Verfahren auf der Basis von Erfahrungswerten angewendet um neue Domains zu finden [21].

Zahlreiche Blacklisten werden von Firmen geführt und der Zugriff darauf als Service verkauft. Glücklicherweise gibt es auch freie Listen die als Datenquelle genutzt werden können. Viele Seiten im Netz bieten Tools an um eine Domain über viele Blacklisten hinweg zu prüfen. Im Folgenden werden einige dieser Tools erläutert.

MxToolbox.com: Der Service von MXToolbox ist sehr umfassend was Informationen über Domains angeht. Die Seite bietet Punkte für E-Mail, Netzwerk, Webseiten und DNS Analysen. Unter diesen Punkten verbergen sich knapp 30 Tools um Anfragen zu starten. Unter anderen auch für das Prüfen auf über 100 Blacklisten [22]. Der Service von MxToolbox ist zwar größtenteils frei nutzbar, allerdings nur über die Webseite und man ist auf ein paar Anfragen pro Tag limitiert. Wenn man Zugriff auf die umfassende API haben möchte, muss man sich registrieren und bezahlen. [23] Alternativ kann man die Webseite parsen wenn man mit ein paar Anfragen auskommt. Die Webseite selbst ist relativ instabil und oftmals für längere Zeiträume nicht zu erreichen. Dadurch ist sie als Datenquelle ungeeignet.

IPVoid.com: IPVoid ist ein Tool welches eine angegebene IP-Adresse durch verschiedene Blacklisten laufen lässt und überprüft ob sie gelistet ist. Der Service ist vollkommen kostenlos. Auch eine Anmeldung ist nicht erforderlich. Allerdings gibt es keine API um die Seite zu kontaktieren. Man muss also die Webseite parsen um an Informationen zu gelangen. [24] Auf GitHub wurde ein simples Pythonscript online gestellt, welches die Webseite parsen und die gewünschten Blacklisting Informationen extrahiert. [25] Mit einigen Änderungen kann es für diese Arbeit nützlich sein.

Google Safe Browsing API: Google bietet für Entwickler eine API an, mit deren Hilfe man Webseiten oder Links auf schädlichen Inhalt überprüfen kann. Diese API verwenden auch die Browser Mozilla Firefox und Google Chrome. Google stellt den Entwicklern zwei Varianten der API zur Verfügung. Die “Safe Browsing API v2“ und die “Safe Browsing Lookup API“. Während erstere die zu checkenden Webseiten in gehashter Form überträgt und Caching unterstützt, liefert die “Lookup API“ lediglich ein Keyword, welches den Status der Webseite angibt und agiert vollkommen unverschlüsselt. Um die Google API Nutzen zu können muss man sich als Entwickler bei Google registrieren. [26]

4.1.5 Geotargeting

Unter Geotargeting oder GeoIP versteht man das Lokalisieren eines Hosts anhand seiner IP-Adresse. Diese Information wird dazu verwendet lokalisierte Werbung zu schalten oder Benutzern aus bestimmten Ländern den Zugang zu einem Dienst zu verwehren. Für eine Domainbewertung ist diese Information daher brauchbar, da man anhand einer geografischen Lage die Seriosität eines Dienstes beurteilen kann. Beispielsweise werden die meisten bösartigen Inhalte in Russland gehostet [27].

Der kostenlose Dienst **freegeoip.net** nimmt eine IP-Adresse entgegen und kann diese lokalisieren. Die Genauigkeit hängt dabei vom Zielland ab. In den USA lassen sich Adressen bis auf Städte-Ebene auflösen, wohingegen in Deutschland meist nur Landesebene möglich ist. Für diesen Service bietet freegeoip.net eine einfache API an, welche benutzt werden kann um Anfragen zu stellen. Der Dienst ist auf 10.000 Anfragen pro Stunde beschränkt.

Alle IP-Adressen werden von den fünf Regionalen Internet Registries (RIPE, AfriNIC, ARIN, APNIC und LACNIC) vergeben. Obwohl man weiß an welche Organisationen und in welche Länder sie vergeben wurden, ist eine genaue Lokalisation nicht so einfach möglich. Die IP-Adressen werden meistens Blockweise vergeben. Wenn eine internationale Organisation nun einen Block an Adressen bekommt und auf die global verteilten Standorte aufteilt, kann eine Lokalisation dadurch sehr ungenau werden [28]. In Verbindung mit anderen Analyseverfahren lässt sich die Adresse jedoch mindestens bis auf Landesebene Auflösen, was allerdings für die meisten Anwendungen ausreicht [29]. Einige Datenbanken bieten aber für bestimmte IP-Bereiche auch eine Auflösung bis auf Städte-Ebene an. Solche Datenbaken werden oft von kommerziellen Anbietern zum Kauf bereitgestellt.

Eine Möglichkeit auf die Position einer IP-Adresse zu schließen ist die Route zu der Adresse auszuwerten und von den letzten Routern auf dem Weg den Hostnamen zu betrachten. Oft wird im Hostnamen eines Routers die geografische Lage angegeben (Siehe Listing 4.1). Hierbei ist allerdings darauf zu achten, dass die Form des Namens auch eine Abkürzung oder das Internationale Flughafenkürzel eines großen Flughafens in der Nähe sein kann. Es muss auch nicht immer ein Hinweis zur Lage des Routers enthalten sein. [30] [31]

```
$ traceroute nytimes.com -w 10 -F -q 1
traceroute to nytimes.com (170.149.172.130), 30 hops max,...
1-6 [Provider]
 7  ae52.bar1.Munich1.Level3.net (62.140.24.57)  15.212 ms
 8  ae-19-19.ebr1.Frankfurt1.Level3.net (4.69.153.246)  176.042 ms
 9  ae-47-47.ebr2.Paris1.Level3.net (4.69.143.142)  173.897 ms
10  ae-42-42.ebr2.Washington1.Level3.net (4.69.137.54)  175.315 ms
11  ae-46-46.ebr2.Washington12.Level3.net (4.69.202.54)  174.333 ms
12  ae-6-6.ebr2.Chicago2.Level3.net (4.69.148.146)  178.237 ms
13  ae-1-100.ebr1.Chicago2.Level3.net (4.69.132.113)  179.825 ms
14  ae-3-3.ebr2.Denver1.Level3.net (4.69.132.61)  178.586 ms
15  ae-2-2.ebr2.Seattle1.Level3.net (4.69.132.53)  177.727 ms
16  ae-2-52.edge2.Seattle1.Level3.net (4.69.147.168)  178.273 ms
17  NEW-YORK-TI.edge2.Seattle1.Level3.net (4.53.158.42)  183.018 ms
```

Listing 4.1: Route zu times.com

Als weiterer Anhaltspunkt kann der Whois Record ausgewertet werden. Man geht dabei davon aus, dass die im Record gespeicherte Adresse normalerweise in geografischer Nähe zur Domain liegt [30]. Dass diese Vermutung allerdings sehr fehleranfällig ist, zeigt Listing 4.1. Der Zeitungsverlag New York Times hat den Firmensitz in New York, der Traffic auf die Domain *nytimes.com* wird jedoch nach Seattle geleitet.

Versuche in der Vergangenheit die geografische Lage mit in die DNS Records zu stecken wurden von der Community weitestgehend nicht angenommen [32]. Es gibt allerdings wenige Administratoren, die LOC-Records erzeugen.

4.1.6 Lexikalische Analyse und Suchmaschinen

Eine **Lexikalische Analyse** im Zusammenhang mit dieser Arbeit bedeutet, dass Domainnamen auf Auffälligkeiten hin untersucht werden. Dabei kann zum Beispiel überprüft werden ob die Namen echte Wörter eines Wörterbuchs beinhalten oder welchen Prozentanteil der Namen Zahlen ausmachen. Zufalls generierte Domainnamen beinhalten häufig einen verhältnismäßig hohen Anteil an Zahlen und keine sinnvollen Wörter. Betrachtet man nun Domainnamen wie *google.de* oder *Autoscout24.de*, fällt auf dass bei einer Bewertung jedes dieser Merkmale für sich wahrscheinlich zu einer hohen Falsch-Positiv-Rate führt. Wendet man jedoch beide an, lässt sich diese Rate reduzieren [9].

Eine weitere Möglichkeit etwas über den Domainnamen in Erfahrung zu bringen ist das Benutzen von **Suchmaschinen**. Die Anzahl der Treffer liefert eine Aussage über die Popularität der Domain. Bei einer Suche ohne Top Level Domain lässt sich anhand der Treffer

sagen ob der Domainname ein gängiger Begriff ist. Die Suchmaschine Google bietet außerdem weitere Zusatzfunktionen um eine Suche nach bestimmten Kriterien durchzuführen. Mit dem Schlüsselwort *link:* in Verbindung mit einer URL oder einem Domainnamen lässt sich beispielsweise herausfinden welche anderen Seiten auf den Suchbegriff verweisen. Mit dem Schlüsselwort *site:* und einem Domainnamen lassen sich alle indizierten Webseiten einer Domain finden [33].

Unter dem Punkt lexikalische Analyse lässt sich auch die Suche von sogenannten **"Vertipper Domains"** einordnen. Darunter versteht man Domains die das Ziel haben Besucher, welche sich vertippen, auf andere Angebote umzuleiten. Meist werden solche Domains registriert um damit Geld durch Werbeanzeigen zu verdienen. Als Beispiele seien hier *amazon.de* und *amazn.de* genannt. Zweitere Domain führt auf eine Seite auf welcher nur Werbung gehostet wird. Bei Vertipper Domains werden meist Buchstaben vergessen, Buchstaben doppelt getippt, oder es wird auf die phonetische Aussprache geachtet und etwas ähnlich klingendes registriert [34]. Durch Kombination der verschiedenen Vertipper Varianten lässt sich schnell eine große Auswahl an Domainnamen ermitteln, die als Vertipper Domains in Frage kommen. Durch eine Abfrage über das DNS lässt sich herausfinden ob diese dann auch existieren beziehungsweise auf einen Server verweisen.

4.2 Anforderungen

Nachdem die Datenquellen identifiziert und analysiert wurden kann nun mit der Analyse der funktionalen Anforderungen für das zu entwickelnde System begonnen werden. Hierzu werden verschiedene Werkzeuge des Requirement Engineerings eingesetzt.

Das Sogenannte **Kontextdiagramm** (Abbildung 4.1) zeigt die Zusammenarbeit des Systems mit seiner Umgebung. Da die zu entwickelnde Software mit vielen anderen schon bestehenden Komponenten (Datenquellen) zusammenarbeiten muss, bietet sich diese Darstellungsweise an.

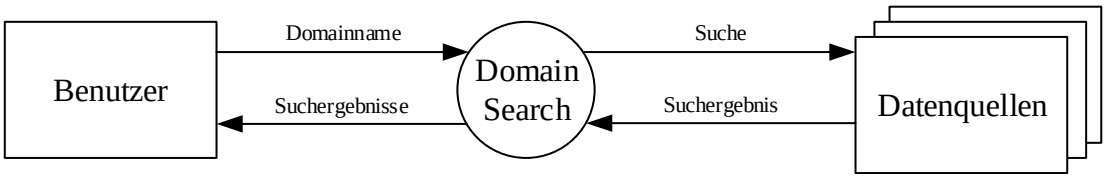

Abbildung 4.1: Kontextdiagramm der Software

User Stories werden verwendet um in der agilen Softwareentwicklung die Anforderungen zu dokumentieren. Durch die einfache und formlose Beschreibung der Anforderung sind sie leicht verständlich und besonders für nicht Technik versierte Kunden einfach zu verfassen beziehungsweise zu lesen. Aus der Aufgabenstellung und der Analyse der Datenquellen lassen sich folgende User Stories ableiten:

- Als Benutzer möchte ich einen Domainnamen eingeben um Informationen über ihn zu suchen.
- Als Benutzer möchte ich eine Liste von Domainnamen abarbeiten lassen. Für jeden Namen soll eine Suche durchgeführt werden.
- Als Benutzer möchte ich alte Suchergebnisse abrufen um die Daten auszuwerten oder einen zeitlichen Verlauf beobachten zu können. Die Daten sollen auch automatisiert abgerufen werden können.
- Als Benutzer möchte ich die Antworten auf einzelne Suchen abspeichern können.
- Als Benutzer möchte ich weitere Informationsquellen hinzufügen können, um die Funktionalität zu erweitern.

- Als Benutzer möchte ich die Software in eventuell bestehende Systeme integrieren können.
- Als Benutzer möchte ich die Software automatisiert suchen lassen.

Als externe Anforderung kommt noch hinzu, dass die Software in der Programmiersprache Python entwickelt werden soll.

Die eben genannten User Stories lassen sich in einem **UML-Anwendungsfalldiagramm** (Abbildung 4.2) darstellen. Dabei kann man die Zusammenhänge zwischen einzelnen Funktionalitäten herausarbeiten. Außerdem wird die Beziehung von Akteuren oder Benutzern zu bestimmten Funktionen in der Software abgebildet [35]. Da in diesem Fall die Software nicht zwischen verschiedenen Benutzern unterscheiden soll, gibt es nur einen Akteur.

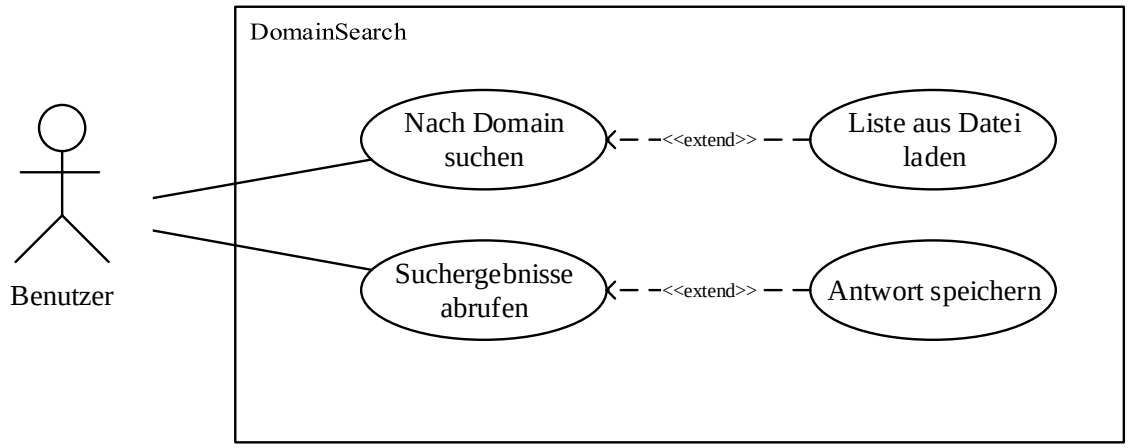

Abbildung 4.2: Anwendungsfalldiagramm

Die letzten drei User Stories werden in diesem Diagramm nicht abgebildet, da es sich um Anforderungen handelt, die vom Benutzer nicht direkt angestoßen werden können. Die automatisierte Suche wird dem Betriebssystem überlassen. Beispielsweise über CRON bei Unix oder den Task Planer bei Windows. Dadurch muss nicht immer ein Prozess im Hintergrund laufen. Um das zu realisieren muss die Software natürlich Kommandozeilenparameter entgegennehmen. Aufgrund der einfachen Interaktion mit dem Benutzer, wird bewusst auf eine GUI verzichtet und das Programm als reines Kommandozeilen Tool entwickelt.

5 Entwurf

Nach der Festlegung der Anforderungen kann nun begonnen werden ein Konzept für die zu entwickelnde Software zu entwerfen. In diesem Kapitel wird der Aufbau der Software erarbeitet. Es wird auch auf eventuell verwendete Bibliotheken oder Software eingegangen und erläutert warum diese verwendet wurden. Es wird im Folgenden der Begriff *Modul* verwendet. Dieser bezieht sich nicht auf ein Python-Modul, sondern auf eine Komponente der Software. Siehe: Kapitel 5.3

Abbildung 5.1 zeigt den Ablauf den die Software durchläuft wenn eine Domain gesucht wird (blauer Datenfluss). Die Software startet Module, diese suchen nach Informationen in den Datenquellen und speichern die gefundenen Daten in geeigneter Form in der Datenbank ab. Der rote Datenfluss zeigt eine Abfrage nach einer bereits getätigten Suche und liefert somit die Daten aus der Datenbank zurück. Dabei wird aus jedem Modul die entsprechende Abfrage geholt und auf der Datenbank ausgeführt.

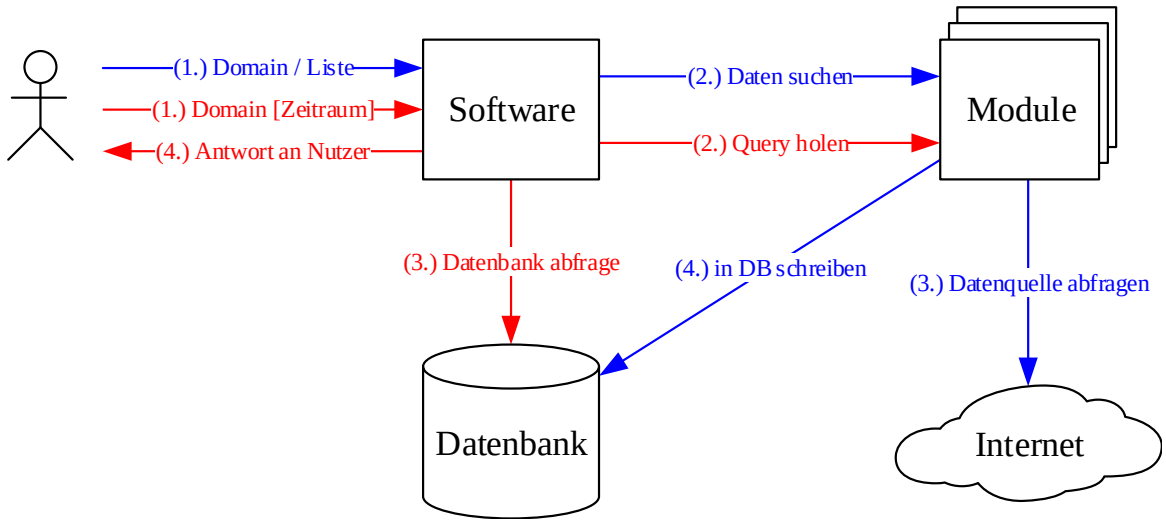

Abbildung 5.1: Ablauf einer Suche

5.1 Schnittstellen

Aus den Anforderungen aus Kapitel 4.2 lassen sich die Schnittstellen des Programms ableiten. Es muss in der Lage sein einen Domainnamen oder eine Liste an Namen entgegenzunehmen und abzuarbeiten. Dies wird mit einem Kommandozeilenparameter realisiert, welcher basierend auf einem Schalter entweder einen Namen oder einen Pfad zu einer Liste entgegennimmt. Als Schalter wird `-l` oder `--list` gewählt. Siehe Listing 5.1

```
root:/# Python3 DomainSearch.py hm.edu
root:/# Python3 DomainSearch.py liste.txt –l
root:/# Python3 DomainSearch.py liste.txt ––list
```

Listing 5.1: Aufrufe mit schalter für Liste

Aus Gründen der Übersichtlichkeit wird zur Abfrage einer Suche ein weiteres Tool entwickelt, welches ebenso Kommandozeilenparameter entgegennimmt, allerdings nur für bereits getätigte Suchen die Datenbank ausliest. Ein zwingendes Argument ist der Domainname nach dem gesucht werden soll. Wird nur dieser angegeben, liefert das Programm die aktuellste Suche nach dieser Domain zurück. Optionale Parameter zur Verfeinerung der Suche werden wie folgt interpretiert:

`-f File` wird dieser Parameter mitgegeben, speichert das Programm sämtlichen output in die mit FILE spezifizierte Datei.

`-a, --all` Liefert **alle** Einträge zu der gegebenen Domain zurück.

`-s SINCE` Liefert aktuellsten Eintrag zu der gegebenen Domain **seit** dem angegebenen Datum zurück.

`-u UNTIL` Liefert aktuellsten Eintrag zu der gegebenen Domain **bis** zu dem angegebenen Datum zurück.

`-i, --info` Liefert nur die Header Informationen (ID, Zeit und Domain) über die Suchergebnisse zurück.

`-d, --domains` Liefert **alle** Domains in der Datenbank zurück. Ignoriert dabei alle anderen Argumente.

Die Suchantwort wird in Textform zeilenweise ausgeben. Dabei werden die Daten der verschiedenen Module durch Leerzeilen separiert.

Um das Tool in bestehende Software zu integrieren wird ein Datenaustauschformat spezifiziert. Abbildung 5.2 zeigt den Aufbau des Formats. Ein Datensatz enthält Header Informationen über die Suche. Also ID oder Nummer der Suche, den Domainnamen und den Zeitstempel. Darüber hinaus werden alle gefundenen Daten in Tabellenform angehängt. Jede Tabelle speichert den Namen des Moduls aus dem die Daten stammen, sowie zweilenweise die Daten. Ein Datensatz bietet außerdem die Möglichkeit mit einem einfachen Aufruf komplett in geeigneter Weise ausgegeben zu werden.

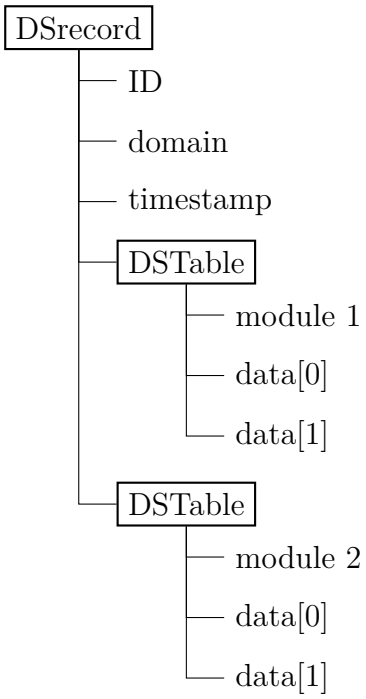

Abbildung 5.2: Beispiel der Struktur eines Datensatzes

Als weitere Integrationsmöglichkeit werden die Klassen des Programms so ausgelegt, dass auch eine Benutzung ohne die Kommandozeilen Tools möglich ist. Dadurch wird der Umweg über das Betriebssystem vermieden, wenn das Programm in eine bereits vorhandene Software integriert werden soll.

Des weiteren wird eine Konfigurationsdatei erstellt, in welcher Daten abgelegt werden die für die Ausführung des Programms wichtig sind, jedoch von dem Benutzer abgeändert oder vorgegeben werden müssen. Als Beispiele wären hier die Verbindungsparameter für die Datenbank oder bestimmte Modulnamen, welche nicht gestartet werden sollen zu nennen. Die Konfigurationsdatei ist auch in der Lage Parameter von Modulen vorzugeben. Dabei ist für die

Übersichtlichkeit darauf zu achten, dass Variablen für Module mit dem Modulnamen als Präfix zu kennzeichnen sind. Durch die `import` Funktion von Python ist diese Konfigurationsdatei einfach einzulesen und zu verwenden.

5.2 Scheduler

In diesem Unterkapitel wird der Scheduler erarbeitet. Dieser soll dazu dienen verschiedene Module des Programms in geeigneter Weise parallel zu starten und zu beenden. Die Parallelität ist wichtig, da die einzelnen Module Daten aus Onlinequellen abrufen und somit mehrere Verbindungen geöffnet werden müssen. Durch ein paralleles öffnen der Verbindungen ist eine Performance Steigerung möglich [36].

`find_modules()`: Der Scheduler ist in der Lage Module zu erkennen, die in einem bestimmten Ordner liegen. Wenn ein Benutzer ein neues Modul hinzufügen will, muss er es nur in den Ordner kopieren und es wird vom Programm automatisch gefunden und gestartet. Dazu lädt der Scheduler alle Dateien im Ordner und prüft sie auf Kompatibilität mit dem Programm. Abbildung 5.3 zeigt den Ablauf einer Suche nach Modulen.

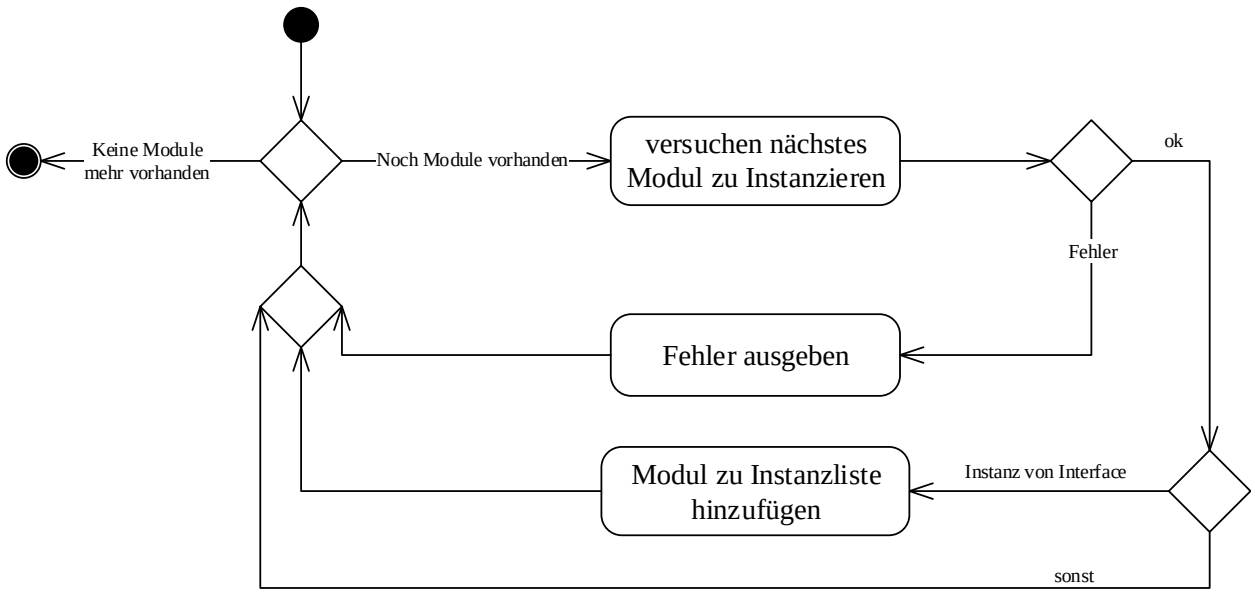

Abbildung 5.3: Laden der Module

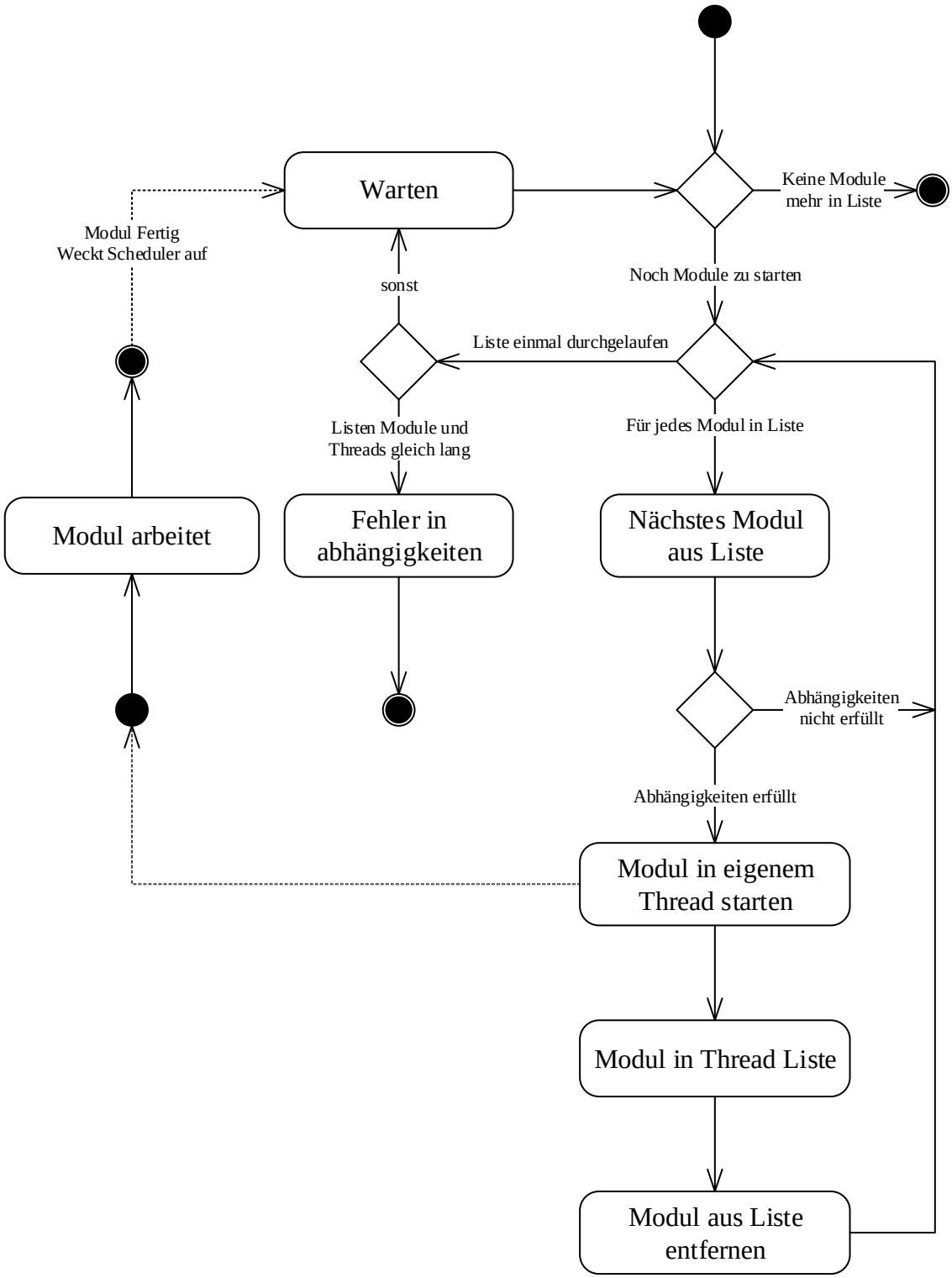

Abbildung 5.4: Starten der Module und überprüfen von Abhängigkeiten

`start_modules(ID, domain)`: Hat der Scheduler alle kompatiblen Module gefunden, startet er die Suche für jedes Modul in einem eigenen Thread. Da nun Abhängigkeiten zwischen einzelnen Modulen bestehen können, achtet der Scheduler darauf ein Modul erst zu starten wenn die Abhängigkeiten erfüllt sind (Abbildung 5.4). Kehrt ein Modul aus der Ausführung zurück, registriert der Scheduler das Modul als abgearbeitet und kann andere Module auf Erfüllung der Abhängigkeiten prüfen. Eine Erkennung von zyklischen Abhängigkeiten ist in diesem Fall ebenso nötig, da es sonst zu Deadlocks führen kann. Eine Schleife in den Abhängigkeiten tritt auf wenn die Menge der abgearbeiteten Module gleich der Menge der gestarteten Module ist, es aber noch Module zum starten gibt. In diesem Fall muss ein Fehler ausgegeben und die Ausführung unterbrochen werden.

`query_modules()`: Wird eine Suche auf der Datenbank ausgeführt, ist der Scheduler dafür zuständig von den Modulen die Suchqueries zu finden. Da er alle Module kennt, ist das eine einfache Funktion welche über die Module iteriert und die Queries holt.

5.3 Module

Die Module der Software sind gedacht um Daten aus verschiedensten Quellen zusammenzutragen und in der Datenbank (siehe Kapitel5.4) zu speichern. Im Folgenden wird die Struktur eines Moduls erarbeitet, sodass es von Scheduler gefunden und eine Suche gestartet werden kann. Jedes Modul wird so konzipiert, dass es in der Datenbank eine oder mehrere Tabellen anlegen kann und die eigenen Daten in diese schreibt. Außerdem muss jedes Modul ein Query bereitstellen mit dem man an die gespeicherten Daten gelangt. Diese wird bei einer Suche über die Datenbank abgerufen und ausgeführt.

Es wird eine abstrakte Basisklasse definiert, die vorgibt welche Methoden ein Modul enthalten muss. Jedes Modul wird von der aufrufenden Instanz mit der Methode `do_run(ID, domain)` gestartet. Diese Methode öffnet die Datenbank Verbindung, erstellt die nötigen Tabellen, führt die Suche durch und schließt anschließend die Datenbank wieder. Siehe Abbildung 5.5. Folgende Methoden werden dazu als abstrakt markiert und müssen bei einem konkreten Modul überschrieben werden:

`__init__()` der Konstruktor muss jedes Modul initialisieren. Hier wird auch die Suchquery angegeben.

`create tables()` wird vor der Suche aufgerufen um in der Datenbank alle Tabellen zu erstellen, die das Modul benötigt. Hierbei muss darauf geachtet werden, dass die Tabellen im Normalfall schon existieren. Das `CREATE TABLE`-Statement sollte also mit der Zusatzoption `IF NOT EXISTS` ausgeführt werden, oder es muss auf andere Art sichergestellt werden, dass die Ausführung nicht fehlschlägt.

`do_search(ID, domain)` führt eine Suche nach der gegebenen Domain durch. Hier werden die Daten abgerufen und in die Datenbank geschrieben. Die ID wird von der aufrufenden Instanz weitergegeben und spezifiziert die genaue Nummer der Suche. Sie ist dadurch eindeutig identifizierbar.

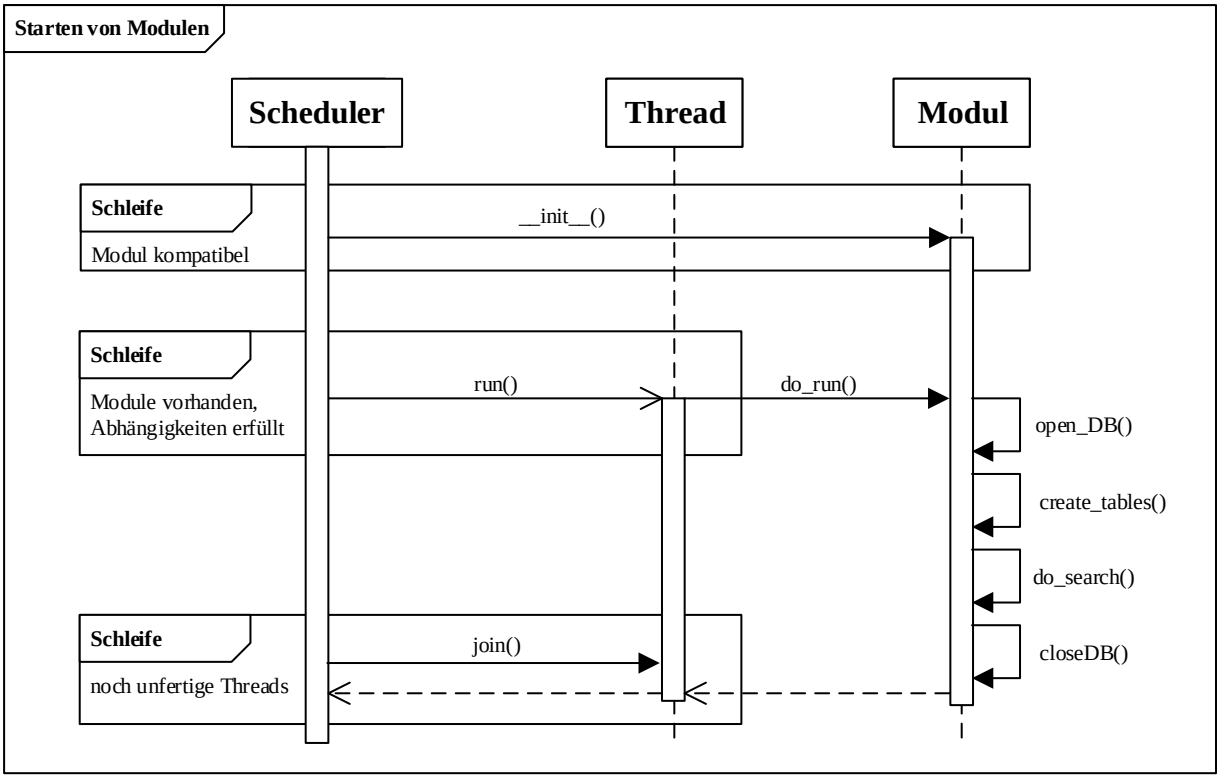

Abbildung 5.5: Ablauf einer Suche

Die Methode `get_queries()` wird vorgegeben um im Falle einer Suche in der Datenbank die Suchquery zurückzugeben. Eine Hilfsmethode zum Debuggen wird auch vorgegeben. Diese nimmt eine Zahl und ein Objekt entgegen, prüft ob die Zahl kleiner als ein in `__init__` gegebenes Debugginglevel ist und druckt das Objekt auf die Konsole.

Zur Abfrage der Datenquellen werden konkrete Module implementiert. Da der Umfang der einzelnen Module relativ klein ist, und der Ablauf einer Suche meist nach einem einfachen Schema abläuft, wird hier nur grob auf die Konzeption aller einzelnen Module eingegangen. Die genaue Umsetzung wird im Kapitel 6 Implementierung behandelt.Meist besteht der Ablauf nur aus Verbindung zu Server herstellen, Daten sammeln und Daten in Datenbank schreiben. Folgende Module werden im Rahmen dieser Arbeit Implementiert:

Blacklist_IPVoid Greift auf die Webseite ipvoid.com zu und sucht nach der Blacklisting Informationen zu der Domain.

Blacklist_MXToolbox Greift auf die Webseite MXToolbox.com zu und sucht in den Blacklisten nach der Domain.

DNSresolver Sucht nach DNS-Records zu der Domain.

DomainAge Greift auf die API von Archive.org zu um den ersten Schnappschuss der Domain zu finden.

GeoIP Fragt auf der Seite freegeoip.org den geografischen Ort der Domain und aller gefundenen IP-Adressen ab.

GoogleSafeBrowsingAPI Sucht in der Google API nach der Domain.

GoogleSearch Führt eine Google-Suche nach der Domain durch und speichert die Anzahl der Treffer.

Nmap Führt einen Portscan auf der Zieldomain durch.

RobotsTxt Sucht auf der Domain nach der **robots.txt** Datei.

SpellChecker Prüft den Domainnamen auf Übereinstimmung mit einem Wörterbuch.

Traceroute Sucht die Route zu der Domain und speichert diese ab.

Typo Generiert viele Vertipper-Domain-Namen und prüft diese auf Existenz.

Whois Sucht den Whois-Record der Domain.

5.4 Datenbank

Die Datenbank in der die gefunden Informationen abgelegt werden, sollte möglichst dynamisch aufgebaut sein. Da jedes Modul eine andere Art an Daten für die Speicherung liefert, wird die Datenbank von jedem Modul um die benötigten Tabellen erweitert. Das Führt allerdings dazu, dass die Suche aufwändig wird. Die Tabellenstruktur steht nicht von vornherein fest, sondern kann sich mit jedem neuen Modul ändern. Dafür wurde festgelegt, dass jedes Modul eine Suchquery zur Verfügung stellen muss, mit der man die Daten des Moduls abrufen kann.

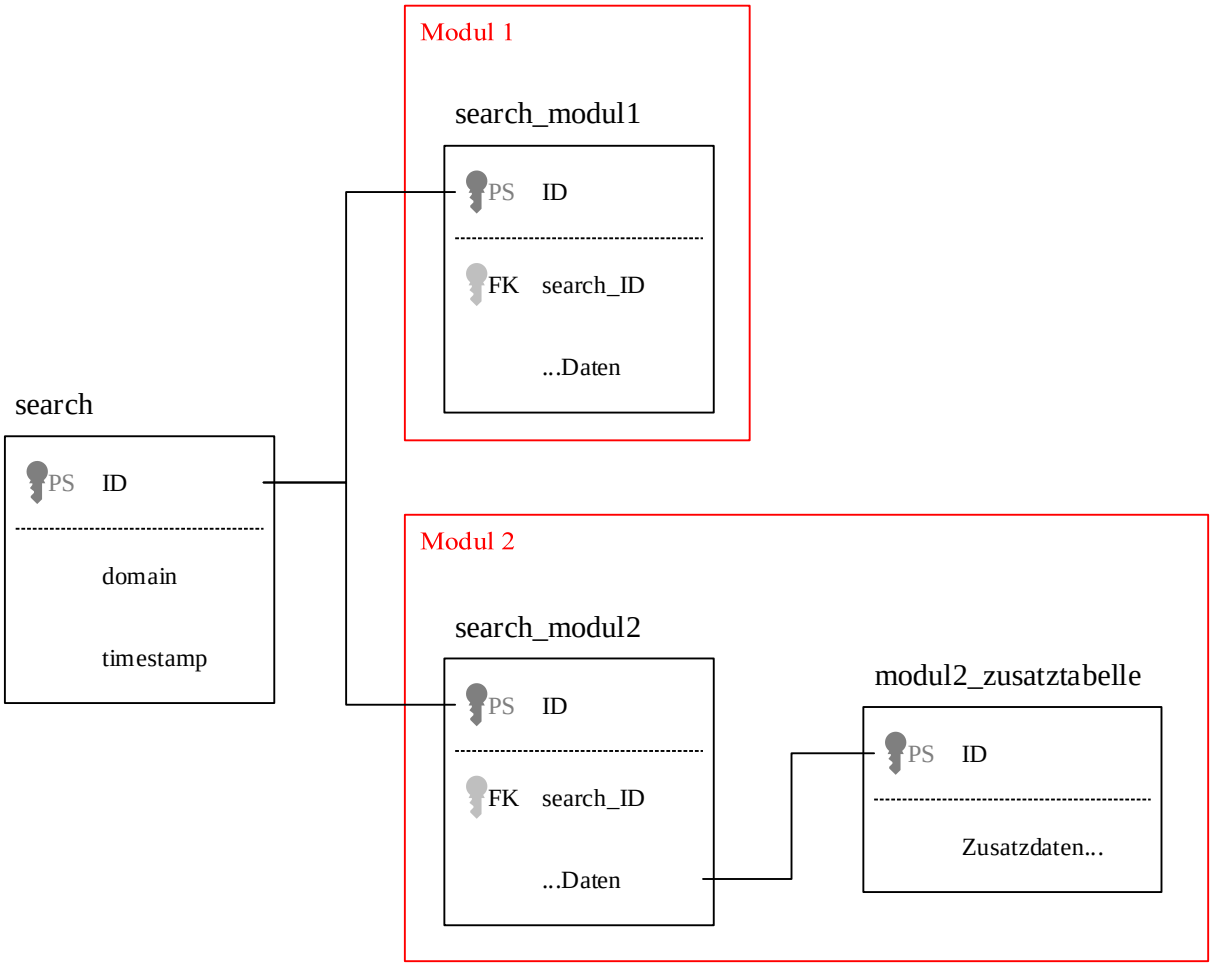

Abbildung 5.6: Schema der Datenbank

Die grundsätzliche Struktur der Datenbank sieht vor, dass eine zentrale Tabelle `search` einen Suchvorgang mit den unter Kapitel 5.1 beschriebenen Header Informationen speichert und eine ID besitzt, mit der die Suche identifiziert werden kann. Diese ID dient auch dazu die Informationen in den Modultabellen zu referenzieren. Wenn ein Modul zusätzliche Tabellen benötigt, könne diese ohne weiteres angelegt und benutzt werden. Abbildung 5.6 zeigt den Aufbau der Datenbank. Es muss dabei darauf geachtet werden, dass keine Konflikte bei Tabellennamen entstehen. Dafür werden folgende Namenskonventionen festgelegt:

- Der Tabellenname der zentralen Tabelle ist `search`.
- Der Name von Tabellen die die ID in der zentralen Tabelle mit einem Fremdschlüssel referenzieren sollte aus `serach_` und dem Modulnamen zusammengesetzt sein. Z.B.: `search_dns`
- Der Name von zusätzlichen Tabellen, die ein Modul benötigt, sollte aus Modulname und dem Tabellennamen bestehen. Zum Beispiel: `dns_zusatzinfos`

Zum Zugriff auf die Datenbank wird eine Klasse erstellt, welche eine Verbindung zur Datenbank herstellt, die zentrale Tabelle `search` anlegt und folgende Methoden anbietet:

`__init__()` initialisiert die Datenbankverbindung. Holt sich aus der Konfigurationsdatei die Verbindungsparameter, baut eine Verbindung auf und versucht die zentrale Tabelle `search` anzulegen.

`insert_search(domain)` fügt eine neue Suche in die Datenbank ein. Dazu wird der Domainname und der Zeitstempel in der zentralen Tabelle abgelegt. Die ID der Tabellenzeile wird zurückgegeben und für alle folgenden Module als Referenz benötigt.

`create_table()` legt die zentrale Tabelle an.

`close_connection()` schließt die Datenbankverbindung.

`get_connection()` gibt das Verbindungsobjekt zurück. Wird für alle Module benötigt um in die Datenbank schreiben zu können.

5.5 Kommandozeilentools

Im Folgenden wird der Zusammenhang zwischen den in Kapitel 5.1 bis 5.4 genannten Komponenten erarbeitet. Dazu entstehen zwei Kommandozeilentools, welche eine Suche in die Datenbank schreiben und eine Suche aus der Datenbank abfragen können.

DomainSearch bietet mit Hilfe der unter Kapitel 5.1 genannten Kommandozeilenparameter die Möglichkeit eine Suche durchzuführen. Um das Programm auch in andere Software einzubinden, werden folgende Methoden angeboten: `perform_search(name)` verbindet zur Datenbank, fügt eine neue Suche ein, lädt alle Module über den Scheduler und startet schließlich alle Module mit dem zu suchenden Domainnamen und der ID der Suche. `perform_listsearch(list)` lädt die als Parameter mitgegebene Datei und führt für jede Zeile eine Suche durch. Zeilen die mit `#` beginnen werden ignoriert. Für eine schnellere Suche ist die Funktion `perform_parallel_listsearch(list)` vorgesehen. Diese lädt ebenso eine Liste, startet jedoch eine in der Konfiguration angegebene Anzahl Subprozesse zum suchen. Damit beim Importieren des Python-Moduls die Kommandozeilenparameter nicht erstellt und ausgewertet werden, muss dieser Teil in einen Block, welcher nur ausgeführt wird, wenn das Programm direkt gestartet wird.

DomainSearchDB bietet die Möglichkeit eine vergangene Suche aus der Datenbank abzurufen. Auch hier werden die Kommandozeilenparameter von Kapitel 5.1 verwendet um eine Abfrage durchzuführen. Das Programm wertet die gegebenen Parameter aus und formatiert die Ausgabe für die Konsole. Dieses Python-Modul sollte jedoch nicht direkt importiert werden. Zur Integration in andere Software wird die Klasse DBReader angelegt, welche alle nötigen Methoden zur Suche in der Datenbank bietet. DomainSearchDB greift auch auf diese Methoden zurück.

DBReader dient zur Suche in der Datenbank. In diesem Python-Modul wird auch das Datenaustauschformat implementiert. Der DBReader gibt Daten ausschließlich in diesem Format zurück. Um die Daten aus der Datenbank auszulesen muss der Reader von jedem Modul die entsprechende Suchquery bekommen. Dazu wird im Scheduler die Methode `query_modules()` aufgerufen. Folgende Methoden bietet der DBReader:

`__init__()` öffnet nur eine Datenbankverbindung.

`get_tables()` gibt eine Liste aller Tabellen der Datenbank zurück.

`get_domains()` Gibt eine Liste aller Domains zurück, für die Daten in der Datenbank vorhanden sind.

`get_IDs_by_domain(domain, beginTime, endTime)` Gibt die IDs aller Suchen nach einer bestimmten Domain zurück, die zwischen den beiden Zeitangaben liegen. Die Zeitangaben sind beide optional. Wenn diese weggelassen werden, wird das Minimum bzw. Maximum zur Suche verwendet.

`get_by_ID(ID)` Liefert genau einen Datensatz mit der gegebenen ID zurück. Der Datensatz wird in dem unter Kapitel 5.1 genannten Datenformat zurückgegeben.

`get_by_domain(domain, beginTime, endTime)` Gibt wie `get_IDs_by_domain` alle Datensätze zurück, die zu der gegebenen Domain gehören und zwischen den beiden Zeitangaben liegen. Die Datensätze werden als Liste Zurückgegeben.

5.6 Sonstiges

Zur besseren Übersicht und Wartbarkeit wird ein Python-Modul angelegt, welches nur die Ausgaben auf die Konsole übernimmt. Bis auf Debugging-Ausgaben sollten alle Anderen über die Methoden von `out` getätigt werden.

Des weiteren gibt es eine Klasse **command**. Diese führt einen externen Prozess aus. Für verschiedene Module wird diese Funktionalität benötigt um Linux Systembefehle ausführen zu können. Zum Beispiel `whois` oder `traceroute`. Damit die komplette Programmausführung bei einer langen Antwortzeit nicht stockt, wird ein timeout eingebaut um den Prozess nach Ablauf einer gegeben Zeit zu terminieren. Der timeout kann individuell für jedes Modul in der Konfigurationsdatei abgelegt werden.

6 Implementierung

Nach der Konzeption der Software folgt in diesem Kapitel das Vorgehen bei der Implementierung. Dabei wird auf das Programm selbst, sowie auf die Module eingegangen und verwendete Bibliotheken erklärt.

6.1 Projektstruktur

Um eine möglichst übersichtliche Struktur zu schaffen, wird das Projekt in drei Packages unterteilt. Packages repräsentieren bei Python Ordner im Dateisystem. Ein Einfügen von neuen Modulen in den Ordner *modules* entspricht also auch das Einfügen in das dazugehörige Package.

main Dieses Package beinhaltet die ausführbaren Komponenten des Programms, sowie die Konfigurationsdatei. Die Klasse DBReader, welche für die Suche in der Datenbank zuständig ist, wird ebenfalls in diesem Package abgelegt, da sie von externer Software direkt eingebunden werden kann. Die Komponenten in den anderen Packages müssen für eine externe Verwendung der Software nicht direkt eingebunden werden.

modules In diesem Package werden alle Module abgelegt, die von dem Programm gefunden und ausgeführt werden sollen. Die `__init__.py` Datei des Packages definiert die abstrakte Basisklasse für Module.

additional Hier liegen alle zusätzlichen Klassen und Python-Module, welche benötigt werden um das Programm auszuführen. Für externe Nutzung der Software muss dieses Package nicht direkt eingebunden werden.

Die verwendete Datenbank ist eine MySQL Datenbank in Standardkonfiguration. Es wurde lediglich der Benutzer **program** mit dem Passwort **blubb** erstellt, welcher Zugriff auf die

Datenbank **domain** hat. Zur Verbindung mit der Datenbank werden die Verbindungsdaten in der Konfigurationsdatei abgelegt.

Als Entwicklungsumgebung wurde die kostenlos vertriebene Eclipse Plattform mit PyDev verwendet. PyDev ist ein Open Source Plugin für die Python-Entwicklung. Als Betriebssystem wurde ein Debian Linux gewählt, da sich dort sehr einfach in der Konsole mit MySQL und Python arbeiten lässt. Diese Wahl unterliegt jedoch der persönlichen Präferenz. Es hätte genauso gut ein Windows oder Mac Betriebssystem verwendet werden können.

6.2 Rahmenprogramm

In diesem Teil wird das Rahmenprogramm beschrieben, welches die Koordination zwischen den einzelnen Modulen und einer vom Benutzer eingegebenen Suche übernimmt. Der Umfang der Software erlaubt es nicht jede Zeile Code zu beschreiben deswegen werden hier nur die interessanten Stellen erklärt.

Um die Schnittstelle zwischen der Kommandozeile und dem Programm zu schaffen, werden die Python-internen Funktionen von `argparse` benutzt. Mit wenigen Befehlen lassen sich die mitgegebenen Kommandozeilenargumente parsen und im Programm verwenden. Das Hilfemenü der Applikation wird ebenfalls vollautomatisch von dem Parser erstellt. Listing 6.1 zeigt den dafür notwendigen Code am Beispiel der DomainSearchDB.py.

```
# Create the help menu and the command line options.
parser = argparse.ArgumentParser(description=programDescr)

parser.add_argument('domain', nargs='?', help=domainHelptxt)
parser.add_argument('-f', '--file', help=fHelptxt)
parser.add_argument('-s', '--since', help=sHelptxt)
parser.add_argument('-u', '--until', help=uHelptxt)
parser.add_argument('-a', '--all', help=aHelptxt, action='store_true')
parser.add_argument('-i', '--info', help=iHelptxt, action='store_true')
parser.add_argument('-d', '--domains', help=dHelptxt, action='store_true')
```

Listing 6.1: Erstellen der Kommandozeilenargumente in DomainSearchDB.py

6.2.1 Suche nach einer Domain

In der Anwendung **DomainSearch.py** wird eine neue Suche in die Datenbank eingetragen und anschließend werden mithilfe des Schedulers alle Module gestartet. Listing 6.2 zeigt die dafür notwendigen Aufrufe.

```
def perform_search(name):
    with threading.Lock():
        # Perform one search for the given domain.
        name = name.strip()
        out.print_search_started(name)

        db = database.db()
        searchID = db.insert_search(name)
        db.close_connection()

        sched = scheduler.Scheduler()
        sched.find_modules()
        sched.start_modules(searchID, name)
```

Listing 6.2: Starten einer Suche in DomainSearch.py

Der Scheduler benutzt zum finden der Module das Python eigene `import`. Um ein Modul Instanziieren zu können, muss es sich im Namensraum der aufrufenden Funktion befinden. Mit `from modules import *` können alle Module in den Namensraum geladen werden. Allerdings greift der Stern-Operator nur auf Python-Module zu, welche in der `__all__` Variablen des Packages genannt werden. Diese Schwierigkeit wurde dadurch umgangen, dass alle *.py Dateien, welche in dem Packageordner liegen in die `__all__` Variable geschrieben werden. Siehe Listing 6.3.

```
# import all files from the modules directory
listOfModules = []
for f in glob.glob(os.path.dirname(modules.__file__) + '/*.py'):
    listOfModules.append(os.path.basename(f)[:-3])

modules.__all__ = listOfModules
from modules import *
```

Listing 6.3: Importieren der Module in scheduler.py

Zum Instanziieren wird anschließend der Namensraum durchsucht und die abstrakte Basisklasse, alle Namen die mit einem Unterstrich beginnen sowie das Package mit dem Namen abc übersprungen. Aus abc stammt die Elternklasse ABCMeta, welche zur Definition der Basisklasse benötigt wird. Es wir außerdem darauf geachtet, dass nur Module instanziiert

werden, welche nicht in der `norun` Variablen der Konfigurationsdatei stehen. Anschließend wird noch überprüft ob die instanziierten Module von der Basisklasse abgeleitet wurden. Siehe Listing 6.4

```
for moduleName in modules.__dict__.keys():  # find all imported names
    if \
    not moduleName.startswith('_') and      \
    moduleName is not 'abc' and             \
    moduleName is not 'DatasourceBase' and  \
    moduleName not in norun:  # Drop all 'unwanted' names
        try:
            mod = globals()[moduleName].__getattribute__(moduleName)()
            if isinstance(mod, modules.DatasourceBase):
                self.moduleList.append(mod)

        except Exception as err:
            if develop == 1: raise err
            else: out.print_module_error(moduleName, err)
```

Listing 6.4: Instanziieren der Module in scheduler.py

Listing 6.5 zeigt Das Starten der Module durch den Scheduler. Dazu wird die Liste der Modulinstanzen durchlaufen und überprüft ob die Abhängigkeiten, welche ein Modul haben kann erfüllt sind. Anschließend wird die `do_run(ID, domain)` Methode des Moduls in einem eigenen Thread aufgerufen und das Modul beginnt zu arbeiten. Nach Abschluss der Suche kehrt der Thread zurück, trägt sich selbst in die Liste der abgearbeiteten Module ein und ist somit als erfüllte Abhängigkeit registriert. Um ein ständiges Prüfen der Abhängigkeiten zu vermeiden wurde eine Synchronisation über ein Monitor-Objekt (in Python: Condition Variable) benutzt. Jedes fertige Modul weckt den Scheduler auf. Dieser prüft alle noch vorhandenen Module ob ihre Abhängigkeiten nun erfüllt sind. Um Schleifen oder Konflikte in den Abhängigkeiten zu erkennen, wird überprüft ob die Liste der abgearbeiteten Module genauso lang ist wie die Anzahl der gestarteten Modul-Threads. In diesem Fall gibt es kein Modul mehr, welches den Scheduler aufwecken könnte. Ein Konflikt mit den Abhängigkeiten tritt auf und die Schleife wird verlassen.

```
while self.moduleList:
    with cv: # condition variable
        for module in self.moduleList[:]:
            if module.dependencies.issubset(done):
                self.moduleList.remove(module)
                thread = Worker(module, searchID, domain)
                thread.start()
                threads.append(thread)

        if len(threads) is len(done):  #  loop in the dependencies?
            out.print_dependency_loop(self.moduleList)
            break;
        cv.wait()
```

Listing 6.5: Thread zum Module starten in scheduler.py

Im Falle einer Suche durch eine Liste, wird im einfachen Fall die Funktion von Listing 6.2 für jede Zeile der Liste wiederholt. Für lange Listen kann der Suchvorgang sehr zeitaufwändig werden. Zum Starten einer parallelen Suche wurde der Anwendung der Parameter `--parallel` oder `-p` hinzugefügt. Hierbei muss eine gegebene Anzahl an Worker Threads erzeugt werden, welche diese Liste abarbeiten. Durch eine Limitierung des Pythoninterpreters muss für echte Parallelität ein Programm in Subprozesse und nicht Threads aufgeteilt werden. [37] Also wird für jede zu suchende Domain das Programm `DomainSearch.py [domain]` für eine einzelne Domain in einem Subprozess gestartet und die Parameter des ursprünglichen Aufrufs werden weitergegeben. Die zu suchenden Domains werden von den Worker Threads aus einer Queue geholt und an die Subprozesse weitergegeben. Listing 6.6 zeigt das Erstellen der Subprozesse und das Pipen der Ausgabe auf die Konsole des Hauptprogramms.

```
def worker():
    while True:
        domain = domainQueue.get()
        command = baseCommand[:]
        command.append(domain)
        with subprocess.Popen(command, stdout=PIPE, stderr=PIPE) as proc:
            ans = proc.communicate()
            with lock:
                nonlocal counter
                out.print_listsearch(counter, ans[0], ans[1])
                counter += 1
        domainQueue.task_done()

for _ in range(parallelProc):
    thread = Thread(target=worker)
    thread.setDaemon(True)
    thread.start()
```

Listing 6.6: Parallele Suche in DomainSearch.py

6.2.2 Suche in der Datenbank

Um bereits gefundene Daten aus der Datenbank abzurufen wird das Programm **DomainSearchDB.py** gestartet. Die Suche beginnt damit, die Parameter aus Kapitel 5.1 auszuwerten. Da das Programm als Parameter auch ein Datums- und Zeitformat annimmt, ist es wichtig mehre Möglichkeiten der Eingabe zu erkennen und zu tolerieren. Listing 6.7 zeigt die Funktion `_dateformat(date)`, welche die Eingabe entgegennimmt und versucht in ein `datetime`-format zu parsen. Stellt die Funktion fest, dass die gegebenen Datumsformate nicht mit der Eingabe übereinstimmen, wird ein `ValueError` geworfen.

Für die anschließende Suche verwendet DomainSearchDB.py das Python-Modul **DBReader.py**. Darin wird das Datenformat zur Darstellung von gefundenen Datensätzen definiert, und ebenso die Klasse DBReader, mit deren Hilfe die Datenbank ausgelesen wird.

```
formats = {'%Y-%m-%d',
           '%d-%m-%Y',
           '%Y-%m-%d %H:%M',
           '%d-%m-%Y %H:%M',
           '%Y-%m-%d %H:%M:%S',
           '%d-%m-%Y %H:%M:%S'}

for line in formats.copy():
    formats.add(line.replace('-', '.'))
    formats.add(line.replace('-', '/'))

for line in formats:
    try:
        return datetime.strptime(date, line)
    except ValueError:
        pass
raise ValueError("'%s' is not a valid date/time" % date)
```

Listing 6.7: Erkennen und Parsen von Datumseingaben in DomainSearchDB.py

Die Klasse **DSRecord** wird von dem Python eigenen `list` abgeleitet und enthält neben den oben genannten Header Informationen eine Liste mit DSTable Datensätzen. **DSTable** wird ebenso von `list` abgeleitet und enthält den Namen des Moduls von welchem die Daten stammen. Das Erben von der Basisklasse ermöglicht es die `__str__` Methode zu überschreiben um eine sinnvolle Ausgabe zu erzeugen. Abbildung 6.1 zeigt die Zusammenhänge der Einzelkomponenten.

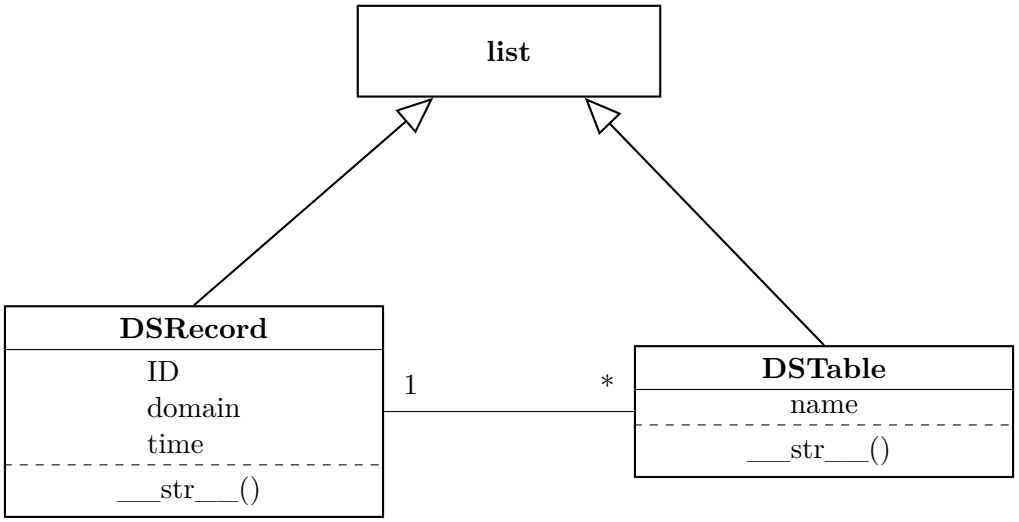

Abbildung 6.1: Datenformat zur Darstellung eines Datensatzes

Wie im Kapitel 5.5 unter dem Punkt DBReader genannt verfügt die Klasse über Funktionen zum auslesen der Daten. Listing 6.8 zeigt anhand der Funktion `get_by_id(ID)` wie ein Datensatz (DSRecord) mit Daten aus der Datenbank gefüllt wird. Der Cursor entspricht dem Zeiger auf die Datenbank und kann somit Queries ausführen. Er enthält ebenso die zurückgegebenen Daten. Als Erstes werden aus der zentralen Tabelle die Headerdaten der Suche geladen. Anschließend wird aus dem Scheduler eine Liste der Modulnamen mit zugehöriger Modul-Query geholt und für jedes Modul diese Query ausgeführt. Die zurückgegebenen Daten werden in einen neuen DSTable eintrag geschrieben, welcher dem DSRecord angehängt wird.

```
query = 'SELECT * FROM search where search.ID = %s'
cursor.execute(query, (ID,))
if cursor.arraysize > 0:
    for line in cursor: # header informations
        domain = line[1]
        time = line[2]

        record = DSRecord(ID, domain, time)

        sched = scheduler.Scheduler()
        sched.find_modules()
        queries = sched.query_modules()

        for module in queries:
            modName = module[0]
            query = module[1]
            cursor.execute(query, (ID,))
            tableEntry = DSTable(modName) # new table
            for line in cursor: # append data line by line
                data = ''
                for item in line[2:]: data += str(item) + ', '
                tableEntry.append(data[:-2])
            record.append(tableEntry)

        record.sort(key=lambda table: table.name)
        return record
```

Listing 6.8: Holen der Daten anhand einer Such-ID in DBReader.py

6.3 Module

In den bisherigen Kapiteln wurden Module nur abstrakt behandelt und ihr Konzept beschrieben (Kapitel 5.3). In diesem Teil geht es um konkrete Module und ihre Implementierung. Alle Module folgen dem selben Ablauf: Tabellen erzeugen, Daten suchen und Daten eintragen. Die grundlegenden Unterschiede finden sich in den einzelnen Schritten und werden hier erklärt.

Blacklist_IPVoid

Dieses Modul greift auf die Webseite von IPVoid.com zu um Blacklisting Informationen zu sammeln. Da die Webseite nur nach IP-Adressen jedoch nicht nach Domains suchen kann, müssen schon IP-Adressen zur Verfügung stehen. Als Abhängigkeit wird also das Modul

DNSresolver eingetragen. Das Modul greift nun vor einer Suche auf die Datenbanktabelle des DNSresolvers zu um die gefundenen IP-Adressen zu holen. Durch eine Analyse der Webseite konnte herausgefunden werden, dass für eine Suche ein HTTP GET-Request auf den Pfad `/scan/[IP]` abgesetzt werden muss. Dies führt jedoch nicht immer zu einer passenden Antwort. Der Dienst hinter IPVoid scheint Suchergebnisse zu cachen und bei bekannten IP-Adressen das gecachte Ergebnis zu liefern. Wenn die Adresse unbekannt ist, kann man mit einem Button eine Suche ausführen lassen. Hinter diesem Button steckt ein HTTP POST-Request welcher als Parameter die IP-Adresse mitsendet. Damit der Webserver von IPVoid antwortet, ist noch das Headerfeld `Content-Type` nötig. Um das Modul einfach zu halten wird als erstes der POST und danach der GET Request ausgeführt (Listing 6.9). Die Suche liefert somit immer ein Ergebnis. Da die Antwort des Servers als HTML-Seite geliefert wird, müssen anschließend mit passenden regulären Ausdrücken die benötigten Informationen herausgefiltert werden.

```
params = 'ip=%s' % ip
headers = {'Content-Type' : 'application/x-www-form-urlencoded'}
conn = client.HTTPConnection(self.serverName, 80)

conn.request('POST' , '/', params, headers)
response = conn.getresponse()
response.read()

conn.request('GET' , '/scan/%s' % (ip))
response = conn.getresponse().read().decode('UTF-8')
```

Listing 6.9: HTTP Requests in Blacklist_IPVoid.py

Blacklist_MXToolbox

Dieses Modul funktioniert ähnlich wie das IPVoid Modul. Die Webseite MXToolbox.com kann jedoch auch Domainnamen entgegennehmen um in Blacklisten zu suchen. Es werden also keine Abhängigkeiten angegeben. Durch analysieren der Seite wurde herausgefunden, dass die dynamischen Inhalte über ASP.Net nachgeladen werden. Nach einem HTTP POST-Request auf die entsprechende Datei antwortet der Server mit einer Art JSON. Dieses ist jedoch falsch formatiert und kann nicht gewöhnlich verarbeitet werden. Es wird also wieder mit regulären Ausdrücken nach den gewünschten Daten gesucht.

DNS-Resolver

Das DNSresolver Modul ist eines der wichtigsten in der Software. Einige andere Module greifen auf die gefundenen DNS Daten zu, um zum Beispiel mit der IP-Adresse weiter zu suchen. Dieses Modul benutzt das dnsPython Package [38] um Informationen in dem Domain Name System zu finden. Dazu wird als erstes die IP-Adresse des zuständigen Nameservers gesucht. Danach wird auf dieser IP eine rekursive Suche nach dem Recordtype `ANY` durchgeführt. Rekursiv bedeutet hier, dass für jeden gefundenen `CNAME`-Eintrag ein weitere Suche gestartet wird. Um endlose Rekursionen zu vermeiden, kann in der Konfigurationsdatei ein Rekursionslimit festgelegt werden. Manchmal enthält eine Antwort auf den Typ `ANY` keinen `A`- oder `AAAA`-Record. Ist dies der Fall wird nach diesen nochmal explizit gesucht. Doppelte Einträge werden durch das Zwischenspeichern im Python eigenen Datentyp `set` vermeiden. Jedes Element kann nur einmal vorkommen und wir bei erneutem einfügen ignoriert. Am Ende der Suche wird das vollständige Set in die Datenbank geschrieben. Wie Listing 6.10 zeigt macht eine komplizierte Datenstruktur des dnspython-Packages es relativ aufwändig die gesuchten Daten aus den Antworten herauszusuchen.

```
hasA = hasAAAA = False
cnames = set()
if answer:
    for section in answer:
        for block in section:
            name = str(block.name)
            ttl = block.ttl
            rdclass = str(dns.rdataclass.to_text(block.rdclass))
            rdtype = str(dns.rdatatype.to_text(block.rdtype))
            for entry in block:
                data = (searchID, name, ttl, rdclass, rdtype, str(entry))
                self._completeSet.add(data)
                if rdtype is 'CNAME': cnames.add(entry.target)
            if rdtype is 'A': hasA = True
            if rdtype is 'AAAA' : hasAAAA = True
return (hasA, hasAAAA, cnames)
```

Listing 6.10: Zerteilen einer Antwort in nutzbare Daten in DNSresolver.py

DomainAge

Das DomainAge Modul versucht das Alter der Domain herauszufinden. Das Modul verbindet sich mit dem Webservice von Archive.org und holt sich den Zeitstempel der ersten Archivierung. Dazu wird eine HTTP Verbindung aufgebaut und eine GET-Anfrage an `/wayback/available?url=[domain]&timestamp=` gesendet. Listing 6.11 zeigt die Verarbeitung der JSON Antwort.

```
try:
    data = json.loads(answer)
    snapshots = data.get('archived_snapshots')
    closest = snapshots.get('closest')
    age = closest.get('timestamp')

except AttributeError: raise DomainAgeError('Kein Alter gefunden')
```

Listing 6.11: Parsen der JSON Antwort in DomainAge.py

GeoIP

GeoIP.py versucht mithilfe des Webservices von freegeoip.net eine Ortung der Domain durchzuführen. Der Webservice bietet ebenfalls wie von Archive.org eine Antwort im JSON Format an. Dazu muss nur eine GET-Anfrage an `\json\[domain]` gesendet werden. Wenn das Parsen in ein JSON Objekt erfolgreich war, können mit Befehlen wie `data.get('country_name')` alle Informationen daraus gelesen werden. Das Modul überprüft nicht nur den Domainnamen, sondern auch noch alle IP-Adressen die schon gefunden wurden. Dazu wird das DNSresolver-Modul als Voraussetzung benötigt.

GoogleSafeBrowsingAPI

Dieses Modul benutzt die Safebrowsing API von Google. Um einen Zugriff auf die API zu bekommen, muss man sich als Entwickler bei Google registrieren und bekommt einen API-Key zugewiesen, welcher bei jeder Anfrage mitgesendet werden muss. Dieser wird in der Konfigurationsdatei abgelegt und in diesem Modul importiert. Die Abfrage selbst ist einfach gehalten. Eine HTTP GET-Anfrage an den Google Server genügt. Wenn der Server mit `204 No Content` antwortet, dann ist die Domain In Ordnung. Andernfalls wird ein einfaches Schlüsselwort zurückgegeben, welches direkt in die Datenbank eingetragen werden kann.

GoogleSearch

Das Modul GoogleSearch.py führt eine Google Suche nach dem Domainnamen durch. Dazu werden die verschiedenen Suchparameter *link:* und *site:* benutzt und die Anzahl der jeweiligen Ergebnisse gespeichert. Zur Suche muss nach einer GET-Anfrage auf `/search?q=[suche]` die Antwort mit Regulären Ausdrücken durchsucht werden. Im Anschluss wird noch der Service von prapi.net genutzt um den Pagerank der Domain herauszufinden. Dazu reicht es eine GET-Anfrage auf `/pr.php?url=[domain]&f=text` abzusetzen. Die Antwort erfolgt im Textformat und kann direkt gespeichert werden.

Nmap

In dem Modul Nmap.py wird eine Suche nach offenen Ports auf der Zieldomain durchgeführt. Dazu wird das Linux Tool *nmap* verwendet. Die Suchparameter für *nmap* sind `-p 0-1024 -sS` und führen zu einer Suche im Portbereich von 0 bis 1024. Der Portscan wird als Stealth- bzw. SYN-Scan durchgeführt. Das bedeutet es wird ein Paket zum TCP Verbindungsaufbau gesendet und entsprechend der Antwort des Servers bewertet. Bei einer SYN/ACK Antwort wäre der Port offen und der Server will eine Verbindung aufbauen. Bei einem geschlossenen Port wird der Server mit RST antworten. Kommt keine Antwort, wird der Port gefiltert und alle Pakete werden verworfen. Listing 6.12 zeigt einen Portscan auf der Domain cs.hm.edu.

```
Nmap scan report for cs.hm.edu (129.187.244.60)
Host is up (0.0023s latency).
rDNS record for 129.187.244.60: w3-1.rz.fh-muenchen.de
Not shown: 1020 closed ports
PORT    STATE    SERVICE
0/tcp   filtered unknown
22/tcp  open     ssh
80/tcp  open     http
111/tcp open     rpcbind
443/tcp open     https
```

Listing 6.12: Portscan auf cs.hm.edu

Die externe Anwendung wird in einem eigenen Prozess ausgeführt. Hierzu wir die Hilfsklasse **Command** verwendet. Ein timeout kann ebenso in der Konfiguration angegeben werden. Um das Nmap Modul ausführen zu können werden root-Rechte auf dem System benötigt. Die Antwort wird mit hilfe von regulären Ausdrücken zerlegt und in die Datenbank geschrieben.

RobotsTxt

```
while doAgain and counter < robotsTxt_maxDepth:
    doAgain = False
    counter += 1
    # open connection
    self._print(2, domain, request)
    if https:
        conn = client.HTTPSConnection(domain, 443, timeout=robotsTxt_timout)
    else:
        conn = client.HTTPConnection(domain, 80, timeout=robotsTxt_timout)

    conn.request('GET' , request)
    response = conn.getresponse()

    # read response
    code = response.code
    headers = response.getheaders()
    content = response.read().decode('utf-8', 'ignore')
    self._print(1, code)
    self._print(3, content)

    # look for Location in header fields if code redirects and look again
    if code >= 300 and code <= 303:
        for title, loc in headers:
            self._print(3, title, loc)
            if title.lower() == 'location':
                match = re.search(r'http(s?)://(.*?)(\$|/.*)', loc)
                if match:
                    # print(match.group(1), match.group(2), match.group(3))
                    if match.group(1) == 's':
                        https = True

                    newRequest = match.group(3)
                    newDomain = match.group(2)
                    if domain == newDomain and request == newRequest:
                        return None
                    if newRequest:
                        request = newRequest
                    domain = newDomain
                    break
        doAgain = True
        conn.close()
        continue

    # return if robots.txt found
    if code is 200:  # Ok
        return content
```

Listing 6.13: Suchen nach robots.txt und folgen der Weiterleitungen in RobotsTxt.py

Hier wird versucht die Datei robots.txt im Basisverzeichnis der Domain zu finden und zu laden. Das Modul muss nun darauf achten, dass je nach Konfiguration des Webservers die Datei eventuell nicht direkt im Basisverzeichnis liegt, sondern zum Beispiel unter der Subdomain www zu finden ist. Dazu wird überprüft ob die Anfrage eine Weiterleitung meldet. Durch einen regulären Ausdruck wird das Ziel der Weiterleitung bestimmt und herausgefunden ob es sich um eine HTTPS Anfrage handeln soll. Um zirkuläre Weiterleitungen zu erkennen, kann in der Konfiguration eine maximale Tiefe angegeben werden, nach der das Modul keiner Weiterleitung mehr folgt und keine Datei zurückliefert. Listing 6.13 zeigt die Suche und Weiterleitung. Wird die Datei gefunden, wird sie zurückgegeben und direkt in die Datenbank geschrieben.

SpellChecker

Das Modul Spellchecker.py benutzt das Python Paket Enchant [39] um den Domainnamen mit einem Wörterbuch zu vergleichen. Außerdem wird das Verhältnis zwischen Anzahl der Buchstaben und Zahlen im Namen ermittelt. Für die Wörterbuchprüfung werden alle Substrings des Domainnamens ab einer in der Konfiguration gegebenen Länge generiert und auf Übereinstimmung mit verschiedenen Wörterbüchern getestet. Listing 6.14 zeigt wie ein Dictionary mit einem Wörterbuch initialisiert und anschließend die Methode `check(word)` aufgerufen wird. Existiert das Wort, wird es zusammen mit der Sprache abgespeichert.

```
substrings = self.substrings(domain, lenOfWords)

for sub in substrings[:]:
    substrings.append(sub[0].upper() + sub[1:])

# check every substring in every language
for lang in dicts:
    checker = enchant.Dict(lang)
    for word in substrings:
        if checker.check(word):
            matches.add((word.lower(), lang))

return matches
```

Listing 6.14: Prüfen aller Substrings auf Übereinstimmung mit Wörterbüchern

Traceroute

Das Traceroute.py Modul führt eine Verfolgung der Route zur Zieldomain durch. Dazu wird ähnlich wie im *nmap* Modul ein Linux Systemtool verwendet. Das Tool *traceroute* nimmt neben der Zieldomain auch Parameter entgegen, welche in der Konfiguration angegeben werden können. In diesem Fall wird nur der Parameter `-q 1` mitgegeben. Das führt dazu, dass nur ein Paket gesendet wird. In der Standardkonfiguration werden drei Pakete gesendet, diese können jedoch verschiedene Routen nehmen und machen die Ausgabe dann unübersichtlich. Das Resultat der Routenverfolgung wird direkt in der Datenbank abgelegt.

Typo

Das Typo.py Modul generiert für den Domainnamen alle möglichen Vetipper-Domain-Namen und Prüft diese auf Existenz. Dazu werden in einer Liste alle generierten Namen gespeichert und anschließend wird versucht mit dem Linux-Tool *dig* eine IP-Adresse zu ermitteln. Weicht diese Adresse von der Adresse der ursprünglichen Domain ab, wird diese Vertipperdomain in der Datenbank gespeichert. Damit die Top Level Domain (TLD) nicht in die Generierung mit einbezogen wird, wir sie vom Domainnamen abgetrennt. Die TLD wird an jeden generierten Namen wieder angehängt. Zum generieren der Namen wird dann die Domain ohne Punkt und Bindestriche gespeichert. Das Führt zum Beispiel zu `cshm.edu` statt `cs.hm.edu` Listing 6.15 zeigt wie für jede Stelle des Namens der Buchstabe entfernt, verdoppelt und mit dem Nebenbuchstaben vertauscht wird. Punkte für Subdomains werden in allen Kombinationen gespeichert. Das bedeutet `fs.cs.hm.edu` wird zu `fscs.hm.edu` und `fs.cshm.edu`. Das Selbe wird mit Bindestrichen durchgeführt. An den Domainnamen wird noch `www` ohne einen trennenden Punkt angehängt. In der Konfigurationsdatei kann man noch angeben welche Buchstaben typischerweise vertauscht werden. Zum Beispiel *s* und *z* oder *e* und *3*. Diese Namen werden anschließend noch in allen Kombinationen erzeugt. Der letzte Schritt der Erzeugung ist das Anhängen von Standard TLDs. Dafür wird jeder erzeugte Name nochmal mit jeder TLD gespeichert. Diese TLDs können ebenso in der Konfiguration angegeben werden.

```
for i in range(0, len(domain)):
    if domain[i].isalnum():
        typos.add(domain[:i + 1] + domain[i:] + tld)
        typos.add(domain[:i] + domain[i + 1:] + tld)
        if i < len(domain) - 1:
            typos.add(domain[:i] + domain[i + 1]
                      + domain[i] + domain[i + 2:] + tld)
```

Listing 6.15: Verdoppeln, Entfernen und Vertauschen der Buchstaben

Zum Prüfen ob die Domains existieren, wird versucht eine IP-Adresse zu ermitteln. Das Linux Tool *dig* führt eine DNS-Auflösung nach einer angegebenen Domain durch. Mit dem Parameter `+short` wird die Ausgabe auf die gefundenen Adressen beschränkt und man kann die Daten direkt weiterverarbeiten. *Dig* wird wie *traceroute* oder *nmap* ebenfalls in einem eigenen Prozess gestartet. Da die Zahl der generierten Namen sehr hoch sein kann, kann in der Konfiguration eine Maximalanzahl an parallelen Prozessen angegeben werden. Das Starten eines neuen Prozesses wird dann verzögert bis ein Alter zurückkehrt. Siehe Listing 6.16.

```
threadLimiter = threading.BoundedSemaphore(typo_maxThreads)
def run(typo):
    threadLimiter.acquire()
    try:
        proc = subprocess.Popen(['dig', typo, '+short'], stdout=PIPE)
        ans = proc.communicate(input)[0]
        for line in ans.decode('utf-8').splitlines():
            if line not in foundIPs:
                foundTypos.add(typo)
    finally:
        threadLimiter.release()


for typo in listOftypos:
    thr = threading.Thread(target=run, args=(typo,))
    threads.append(thr)
    thr.start()
```

Listing 6.16: Starten des dig Kommandos in externen Prozessen

Whois

Das Whois.py Modul arbeitet wie das Traceroute Modul. In einem externen Prozess wird das Linux Tool *whois* gestartet und sucht nach der Domain. Die Ausgabe wird direkt in die Datenbank geschrieben.

7 Evaluation

Im folgenden Kapitel Evaluation geht es darum das entwickelte Programm zu testen und die Qualität der gefundenen Daten sowie die Effizienz der Software zu bewerten.

7.1 Performanz der Software

Um eine Suche durchzuführen wird immer Zeit benötigt. Hier wird bewertet, wie effizient die Software arbeitet und wie lange es dauert bis die Software ein Ergebnis liefern kann.

Zum Testen der Modullaufzeiten wird die Domain *Amazon.de* und alle Module bis auf Blacklist_MXToolbox verwendet. Dieses wird ausgeschlossen, da die Seite eine hohe Ausfallquote hat und oft nicht erreichbar ist. Nach zehn Suchen wird ein Mittelwert gebildet und die Laufzeiten der einzelnen Module miteinander verglichen (Tabelle 7.1). Dabei kann man sehr gut erkennen, dass das Modul Traceroute am meisten Zeit benötigt und den gesamten Programmablauf ausbremst. Durch die Konfigurationsdatei ist der Timeout von traceroute auf 10 Sekunden festgesetzt. Traceroute wird also in der Ausführung unterbrochen und liefert meist nur einen Teil der Routenverfolgung zurück. Wird eine Route nicht zwingend benötigt, oder ist die Laufzeit wichtig, wäre es zu erwägen das Traceroute Modul von der Suche auszuschließen. Ein weiteres zeitaufwändiges Modul ist Nmap. Wenn ein Portscan nicht zwingend notwendig ist, kann das Modul zur Steigerung der Performace ebenfalls deaktiviert werden. Bei der parallelen Suche muss außerdem auf die Anzahl der Verbindungen geachtet werden. Durch das öffnen vieler paralleler Verbindungen werden die Systemressourcen stark ausgelastet. Das Typo Modul beispielsweise öffnet mit *dig* viele parallele Verbindungen zum checken der Vertipperdomains. *Nmap* öffnet ebenfalls sehr viele gleichzeitige Verbindungen.

Modul	Durchschnittslaufzeit in Sekunden
Blacklist_IPVoid	1,091
Blacklist_MXToolbox	—
DNSresolver	0,314
DomainAge	0,495
GeoIP	2,195
GoogleSafeBrowsingAPI	0,145
GoogleSearch	1,349
Nmap	4,836
RobotsTxt	0,284
SpellChecker	0,095
Traceroute	10,026
Typo	0,583
Whois	0,166
Alle Module	**10,043**

Tabelle 7.1: Laufzeiten der einzelnen Module für die Domain Amazon.de

Will man wie im folgenden Kapitel eine Liste mit 10.000 Domains abarbeiten, dauert das in sequenzieller Form $10.000 * 10s$ also etwa 28 Stunden. Zum Testen der Laufzeit wird eine viel kleinere Liste verwendet. Fünfzig Einträge sollten ausreichen um die Performanz zu beurteilen. Außerdem muss für eine parallele Suche das Modul Nmap ausgeschlossen werden. Das Öffnen derart vieler Verbindungen führt zu einer vollständigen Auslastung der Bandbreite und der Systemressourcen. Dies hat zur Folge, dass viele Module Fehler melden weil sie keine Verbindungen mehr bekommen. Es wird eine sequenzielle Suche und jeweils Eine mit 2, 5, 10, 15, 20 und 50 Threads parallel gestartet. Nach jeweils 10 Durchgängen erhält man die Mittelwerte aus Tabelle 7.1. Dabei kann man gut erkennen, dass die Laufzeit indirekt proportional zur Anzahl der Threads sinkt(Tabelle 7.1). Ab einem gewissen Punkt bringt ein Erhöhen der Parallelität keine Verbesserung mehr. Der Schwellenwert hängt von der Leistungsfähigkeit des Systems ab. Bei einer zu hohen Anzahl an Threads kann außerdem das System unbenutzbar werden und sich gegebenenfalls komplett aufhängen. Eine weitere Limitierung ist die Datenbank. Die verwendete MySQL Datenbank unterstützt in Standardkonfiguration bis zu 151 simultane Verbindungen. Versuchen nun 50 Threads jeweils zehn Module gleichzeitig zu starten, kann die Datenbank dies nicht mehr verarbeiten. Durch Umkonfigurieren kann eine MySQL Datenbank je nach System 500 bis 1000 simultane Verbindungen verkraften [40].

Anzahl Threads	Laufzeit in Sekunden
Sequenziell	531,27
2	260,67
5	106,77
10	69,02
15	70,53
20	66,48
50	62,94

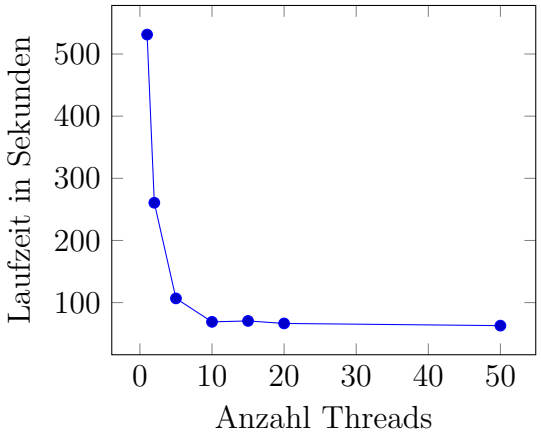

Abbildung 7.1: Anzahl der Threads im Verhältnis zu Laufzeit

7.2 Qualität der Daten

In diesem Teil wird die Qualität der Daten bewertet. Dazu werden die von Alexa.com [41] bestimmten 10.000 meistbesuchten Domains weltweit als Testdatensatz verwendet. Nach einer vierstündigen Suche wird in der Datenbank geprüft für wie viele Domains überhaupt Daten vorliegen. In der Suche wurden die Module Blacklist_MXToolbox und Nmap aus den in Kapitel 7.1 genannten Gründen ausgeschlossen. Diese Auswertung gibt noch keine Auskunft darüber ob die gefunden Daten verwendbar sind. Hier wird nur getestet wie viele Datensätze nach einer Suche überhaupt vorhanden sind.

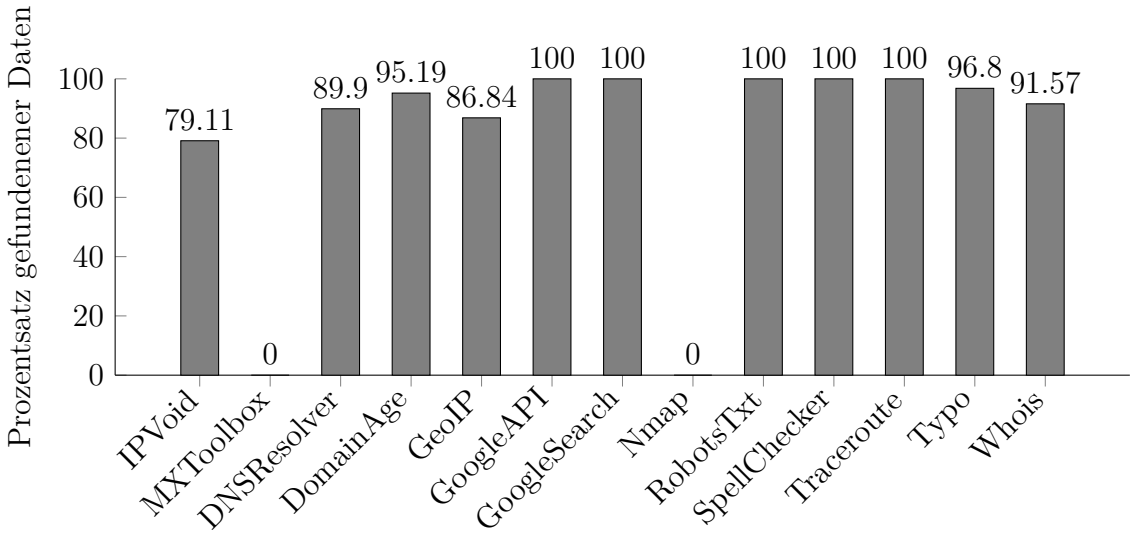

Abbildung 7.2: Gefundene Datensätze pro Modul

In Abbildung 7.2 ist gut zu erkennen, dass Module wie Blacklist_IPVoid oder GeoIP tendenziell weniger Daten geliefert haben. Das könnte daran liegen, dass diese Module externe Seiten benutzen und nicht immer eine Antwort bekommen oder von dem zugrundeliegenden Dienst zeitweise geblockt werden. Zu erkennen ist auch die Zuverlässigkeit der Google Dienste. Diese Module liefern immer ein Ergebnis. Ebenso das Modul Spellchecker, welches keine Netzwerkverbindungen benötigt. Ein hoher Anteil der fehlenden Daten entsteht natürlich auch durch die Timeouts der Module. Falls ein Dienst im Internet nicht antwortet wird das Modul nach einer gegebenen Zeit übersprungen.

Bei genauerer Untersuchung der Datensätze fällt auf das die Module Traceroute und GoogleSearch nicht verwendbare Datensätze ablegen. Stößt das Google Modul z.B. auf einen Fehler wird als Standardwert None eingetragen. In den 10.000 Datensätzen hat das Google Modul nur 566 mal brauchbare Daten geliefert. Das liegt daran, dass viele parallele Verbindungen auf den Google Server als Angriff interpretiert werden und der Server daraufhin blockiert. Abhilfe schafft nur eine langsamere Abfragerate durch eine Reduzierung der Threadzahl [42]. Das Traceroute Modul benötigt viel Zeit um Ergebnisse zu liefern. Läuft das Modul nun in einen Timeout ohne überhaupt etwas gefunden zu haben, wird ein leerer Datensatz angelegt. Mit 94,8% ist der Anteil nicht leerer Datensätze etwa auf dem Niveau der anderen Module. Das bedeutet allerdings nicht, dass das Modul vollständige Routen gefunden hat. Im Talle eines Timeouts wird die Routenverfolgung bis zu dem Abbruchzeitpunkt zurückgegeben und gespeichert.

Interessant sind auch die Daten des GeoIP Moduls. In Tabelle 7.2 kann man sehen welcher Anteil der Datensätze bestimmte geografische Daten enthält. Alle gefundenen Datensätze enthalten mindestens eine Landesangabe und relativ ungenaue Koordinaten. Die Stadt kann jedoch nur bei 71% der Domains bestimmt werden.

Daten	**Anteil**
Land	100,00%
Region	71,06%
Stadt	71,68%
Postleitzahl	47,05%
Koordinaten	100,00%

Tabelle 7.2: Anteile der gefundenen Geografischen Daten

Das RobotsTxt Modul liefert für etwa 6,4% aller überprüften Domains eine HTML Seite statt einer robots.txt Datei zurück und ungefähr 5,5% der Einträge besagen, dass kein Webserver hinter der Domain existiert. Das Modul schreibt gewollt Fehlermeldungen mit in die Datenbank, da “No Webserver on this Domain“ auch eine nutzbare Information ist. Dadurch, dass die robots.txt Datei nur eine Konvention ist, muss ein Server die Datei nicht anbieten und kann eine Anfrage darauf auch auf die Hauptseite der Domain weiterleiten. Die Unterschiede in der Struktur der gefundenen Daten können somit eine automatisierte Weiterverarbeitung erschweren.

Alle anderen Module schreiben ausschließlich nutzbare Informationen in die Datensätze oder erstellen gar keinen Datensatz. Im Schnitt liefert ein Modul somit zu etwa 86% der Domains nutzbare Daten.

8 Ausblick

Durch diese Arbeit wurde ein Grundstein zu einem Domainbewertungssystem gelegt. Durch einen modularen Aufbau der Software und ein dynamisches Verwalten der Module wird eine automatisierte Suche nach einzelnen Domains oder ganzen Listen ermöglicht. Durch eine automatisierte Auswertung der Daten können anschließend Statistiken und Bewertungen über Domains erstellt werden. Zum Beispiel könnten über längere Zeitspannen hinweg die Whois oder DNS Einträge einer Domain beobachtet werden und auf auftretende Auffälligkeiten angemessen reagiert werden. Die Parameter für eine Bewertung müssten in einer zukünftigen Arbeit noch erfasst und bestimmt werden.

Durch die Analyse verschiedenster Datenquellen wurde es ermöglicht geeignete zu wählen, entsprechende Module zu planen und zu implementieren. Die Entwicklung kann als Erfolg angesehen werden, da die gegebenen und gewünschten Anforderungen umgesetzt wurden und die gefundenen Daten verwertbar sind.

Durch die Modularität der Software lassen sich schnell, neue Datenquellen hinzufügen und bestehende anpassen. Zum Beispiel könnte ein Modul entwickelt werden, welches Daten aus der kostenpflichtigen API von Domaintools lädt oder eine eventuelle Webseite hinter einer Domain auf bestimmte Schlüsselwörter durchsucht. Die bereits vorhandenen Module könnten durch eine Überarbeitung noch optimiert werden, um eine höhere Ausbeute an nutzbaren Daten zu erzielen. Die umfassenden Einstellungsmöglichkeiten durch die Konfigurationsdatei ermöglichen es die Software auf verschiedene Anwendungszwecke und auf die Leistungsfähigkeit des eigenen Systems anzupassen.

Eine OpenSource Lizenz auf der Software sorgt dafür, dass sie sich von bestehenden, kommerziellen Lösungen abhebt und für jedermann zugänglich ist. Es wurde dafür die BSD-Lizenz gewählt. Diese besagt, dass die Software verändert und weitergegeben werden darf, es muss jedoch der ursprüngliche Copyright vermerk beibehalten werden.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Literaturverzeichnis

[1] C. Funk and M. Garnaeva, "Kaspersky Security Bulletin 2013/2014 – Statistik für das Jahr 2013," Dezember 2013, http://www.viruslist.com/de/analysis?pubid=200883839 (Stand 11.09.2014).

[2] P. Mockapetris, "RFC 1034 Domain Names - Concepts and Facilities," IETF, RFC, 1987.

[3] M. Lottor, "RFC 1033 Domain Administrators Operations Guide," IETF, RFC, 1987.

[4] T. Niedl, "Erkennung „bösartiger "domains," *Network*, vol. 79, 2011.

[5] "Domaintools," http://www.domaintools.com/ (Stand 11.09.2014).

[6] "Netcraft," http://www.netcraft.com/ (Stand 11.09.2014).

[7] "Netcraft - site report," http://toolbar.netcraft.com/site_report/ (Stand 11.09.2014).

[8] "Urlmetrics," http://urlm.co/ (Stand 11.09.2014).

[9] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *NDSS*. The Internet Society, 2011.

[10] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces," in *ACSAC*. IEEE Computer Society, 2009, pp. 311–320, ISBN: 978-0-7695-3919-5.

[11] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. A. Kemmerer, C. Kruegel, and G. Vigna, "Your Botnet is My Botnet: Analysis of a Botnet Takeover," in *ACM Conference on Computer and Communications Security*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 635–647, ISBN: 978-1-60558-894-0.

[12] T. Brisco, "RFC 1794 DNS Support for Load Balancing," IETF, RFC, April 1995.

[13] "Denic," http://www.denic.de/ (Stand 11.09.2014).

[14] L. Daigle, "Rfc 3912 whois protocol specification," IETF, RFC, 2004.

[15] N. Witte, "Rating The Authenticity Of Websites," in *16th Twente Student Conference on IT*. University of Twente, 2012.

[16] "Archive.org - Internet Wayback Machine," https://archive.org/ (Stand 11.09.2014).

[17] "Zweinull-blog.de - Alter einer Domain ermitteln," http://www.zweinull-blog.de/2012/06/alter-einer-domain-ermitteln/ (Stand 11.09.2014).

[18] "Google Cache," http://www.google-cache.de/ (Stand 11.09.2014).

[19] "Mozilla - Phishing and malware protection," https://support.mozilla.org/en-US/kb/how-does-phishing-and-malware-protection-work (Stand 11.09.2014).

[20] "Google - Phishing- und Malware-Warnungen," https://support.google.com/chrome/answer/99020?hl=de (Stand 11.09.2014).

[21] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, "Phishnet: Predictive blacklisting to detect phishing attacks." in *INFOCOM*. IEEE, 2010, pp. 346–350, ISBN: 978-1-4244-5838-7.

[22] "Mxtoolbox - Blacklist Check," http://mxtoolbox.com/blacklists.aspx (Stand 11.09.2014).

[23] "Mxtoolbox - RESTful API," http://mxtoolbox.com/restapi.aspx (Stand 11.09.2014).

[24] "Ipvoid - What is IPVoid?" http://www.ipvoid.com/about-us/ (Stand 11.09.2014).

[25] "Github - ipvoid-blacklist-checker," https://github.com/josemanimala/ipvoid-blacklist-checker (Stand 11.09.2014).

[26] "Google - Safe Browsing API," https://developers.google.com/safe-browsing/ (Stand 11.09.2014).

[27] Y. Namestnikov, "IT Threat Evolution: Q3 2012," http://newsroom.kaspersky.eu/fileadmin/user_upload/it/Presskits/Q3_2012_REPORT.pdf (Stand 11.09.2014).

[28] B. Gueye, S. Uhlig, and S. Fdida, "Investigating the imprecision of IP block-based geolocation," in *Passive and Active Network Measurement, 8th Internatinoal Conference*, ser. Lecture Notes in Computer Science, vol. 4427. Springer, 2007, pp. 237–240.

[29] S. Zander, "On the Accuracy of IP Geolocation Based on IP Allocation Data," Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 120524A, 2012.

[30] S. E. Cruz, "An Investigation into Better Techniques for Geo-Targeting," Mastersthesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2002.

[31] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts." in *SIGCOMM*, 2001, pp. 173–185.

[32] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson, "Rfc 1876 a means for expressing location information in the domain name system," IETF, RFC, 1996.

[33] "Google - Search Operators," https://support.google.com/websearch/answer/136861?hl=en (Stand 11.09.2014).

[34] "United domains - vertipperdomains," http://blog.united-domains.de/2013/11/vertipperdomains-von-vergessenen-buchstaben-und-fetten-fingern/ (Stand 11.09.2014).

[35] M. Guist, "UML Anwendungsfalldiagramm," https://www.fbi.h-da.de/labore/case/uml/anwendungsfalldiagramm.html (Stand 11.09.2014).

[36] E. M. Nahum, D. J. Yates, J. F. Kurose, and D. F. Towsley, "Performance issues in parallelized network protocols." in *OSDI*. USENIX Association, 1994, pp. 125–137.

[37] Python Software Foundation, "Python library FAQ," https://docs.python.org/3.2/faq/library.html#can-t-we-get-rid-of-the-global-interpreter-lock (Stand 11.09.2014).

[38] "dnspython - A DNS Toolkit for Python," http://www.dnspython.org/ (Stand 11.09.2014).

[39] "pyEnchant - A spellchecking library for Python," https://pythonhosted.org/pyenchant/api/enchant.html (Stand 11.09.2014).

[40] "MySQL Referenzhandbuch," http://dev.mysql.com/doc/refman/5.1/en/too-many-connections.html (Stand 11.09.2014).

[41] "Alexa.com," http://www.alexa.com/ (Stand 11.09.2014).

[42] "Google Support - Captcha Abfragen," https://support.google.com/websearch/answer/86640?hl=de (Stand 11.09.2014).