Information Security



Automatisierte Generierung von personenbezogenen Passwortlisten

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

eingereicht von

Julian Schrittwieser, BSc

is161528

im Rahmen des

Studienganges Information Security an der Fachhochschule St. Pölten

Betreuer: Dipl.-Ing. Robert Luh, BSc

Zweitbetreuer: Dipl.-Ing. Matthias Schrattenholzer, BSc

St. Pölten, 31. Mai 2018

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Fachhochschule St. Pölten GmbH, Matthias Corvinus-Straße 15, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E:office@fhstp.ac.at, I:www.fhstp.ac.at

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende/Absolvent räumt der FH St. Pölten das Recht ein, die Diplomarbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z.B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

Ort, Datum *Unterschrift*

Danksagung

Zunächst möchte ich mich an dieser Stelle bei meinen beiden Betreuern Dipl.-Ing. Matthias Schrattenholzer und Dipl.-Ing. Robert Luh für die Chance bedanken, an diesem überaus spannenden Thema zu arbeiten. Weiters danke ich Dipl.-Ing. Ernst Langmantel stellvertretend für die Internet Privatstiftung Austria (IPA), die mich im Rahmen der Open Source Internet-Förderaktion »netidee« im Call 2017 mit einem Stipendium unterstützt hat.
Ein besonderer Dank gilt Florian Heissenberger, meinem Kommilitonen Sebastian Eresheim und meinen Arbeitskollegen Ahmad Bassel und David Weichselbaum, durch deren fachliche Anregungen meine Arbeit kontinuierlich verbessert wurde.
Darüber hinaus möchte ich mich bei meinen Eltern bedanken, die mir so vieles ermöglicht und mich stets in meinen Entscheidungen bestätigt haben.
Letztlich danke ich meiner Freundin Anja für ihre wissenschaftliche Expertise und unendliche Unterstützung in allen Belangen meines Lebens.

Kurzfassung

Das in den letzten Jahren stetig wachsende Sicherheitsbewusstsein der Bevölkerung führt dazu, dass eine steigende Anzahl an Menschen von kryptographischen Verfahren Gebrauch macht und ihre lokal gespeicherten Daten verschlüsselt. Während diese Entwicklung aus Datenschutzgründen zu begrüßen ist, birgt sie auch einen negativen Nebeneffekt für die Verbrechensbekämpfung: IT-Forensikerinnen und IT-Forensiker sehen sich nun verstärkt vor das Problem gestellt, Informationen auf in Kriminalfällen sichergestellten Datenträgern nicht auswerten zu können, da diese vom Besitzer verschlüsselt wurden. Um diese für die Aufklärung des Falles wichtigen Beweismittel wieder zugänglich zu machen, besteht oft keine andere Möglichkeit als das unbekannte Passwort durch Ausprobieren aller Kombinationen aus Buchstaben, Zahlen und Zeichen zu erraten, was sich als eine äußerst rechen- und damit zeitintensive Aufgabe herausgestellt hat. Eine Alternative zu diesen sogenannten Brute-Force Attacken stellt die Verwendung von Passwortlisten dar, die viele verschiedene relativ simple und damit unsichere aber dennoch oft genutzte Passwörter beinhalten. Diese generischen Passwortlisten enthalten aber immer seltener das tatsächliche Passwort, da eine gängigere Methode bei der Generierung von Passwörtern das Einbinden von personenbezogenen Begriffen wie Namen, Orten oder Jahreszahlen ist.
In dieser Arbeit wird erforscht, nach welchen Mustern personenbezogene Passwörter aufgebaut sind und ob es möglich ist, automatisiert nach Informationen über eine Person zu suchen, um daraus Passwörter nach diesen Mustern zu erstellen. Zu diesem Zweck wurde ein Tool entwickelt, das nach Eingabe des Benutzernamens der Zielperson in sozialen Netzwerken automatisiert online nach persönlichen Informationen sucht und daraus eine personalisierte Passwortliste generiert. Es konnte gezeigt werden, dass die Erstellung von personenbezogenen Wortlisten und Regeln für den sogenannten Crackingprozess eine schnelle und erfolgsversprechende Alternative zu klassischen Brute-Force Attacken bildet.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Problemstellung 1

1.2 Struktur der Arbeit 2

2 Hintergrund **5**

2.1 Passwörter 5

2.1.1 Geschichte des Passwortes 5

2.1.2 Passwortstrategien 7

2.1.3 Sicherheit 15

2.1.4 Angriffsvektoren 16

2.1.5 Bewertung der Passwortstrategien 21

2.2 Akquise von personenbezogenen Informationen 24

2.2.1 Soziale Netzwerke 24

2.2.2 Suchmaschinen 26

3 State of the Art **29**

3.1 Verwandte Arbeiten 29

3.2 Bestehende Tools 30

3.2.1 Webscraper zur Erstellung von Wortlisten 31

3.2.2 Tools zur Erstellung von Passwörtern 31

3.2.3 Passwortcracker 34

3.2.4 Cracker für Remote Authentication Services 35

4 Design **37**

4.1 Prototyp 38

4.2 Architektur 39

4.2.1 CLI 39

4.2.2 Zwischenspeicherung der Ergebnisse 43

4.2.3 Externe Python Pakete . . . 44
4.3 Datenakquise . . . 47
4.3.1 WebDriver . . . 47
4.3.2 Twitter . . . 48
4.3.3 Facebook . . . 49
4.4 Datenverarbeitung . . . 50
4.4.1 Spracherkennung . . . 51
4.4.2 Textbereinigung . . . 52
4.4.3 Lemmatisierung deutscher Wörter . . . 52
4.4.4 Entfernung von Stoppwörtern . . . 53
4.4.5 Trennung von Textkategorien . . . 56
4.4.6 Bewertung von Begriffen . . . 56
4.4.7 Behandlung von Fauxami . . . 60
4.5 Passwortgenerierung . . . 61
4.6 Herausforderungen . . . 63
4.6.1 Performance . . . 63
4.6.2 Einschränkungen der API . . . 64
4.6.3 Evaluierung . . . 64
4.7 Designentscheidungen . . . 65
4.7.1 Webdriver . . . 65
4.7.2 Erweiterbarkeit . . . 66

5 Evaluierung 67

5.1 Testsystem . . . 67
5.2 Testpersonen . . . 68
5.3 Erkenntnisse . . . 75

6 Diskussion 77

6.1 Beantwortung der Fragestellung . . . 77
6.2 Zukünftige Arbeiten . . . 78
6.2.1 Weitere Quellen . . . 78
6.2.2 Verknüpfung von Profilen . . . 79
6.2.3 Empirische Studie für Gewichtung (Ausgefallenheit - Häufigkeit) . . . 79
6.2.4 Spezialisierung auf weitere Sprachen . . . 80

6.2.5 Weitere Verarbeitungsschritte von Begriffen 80
6.2.6 Machine Learning 81
6.3 Resümee 81

Akronymverzeichnis **83**

Abbildungsverzeichnis **85**

Tabellenverzeichnis **87**

Auflistungsverzeichnis **89**

Literatur **97**

1 Einleitung

»Data is the new oil«

Seit dem viel zitierten Ausspruch des Data Science Experten Clive Humby aus dem Jahr 2006 ist viel Zeit vergangen und die große Bedeutung von Daten mittlerweile unumstritten. Mit dem Akquirieren und Auswerten von persönlichen Daten hat sich ein neuer Markt und damit ein ganzes Berufsfeld gebildet. Soziale Netzwerke, die ihren Mitgliedern im Austausch gegen ihre persönliche Informationen eine Möglichkeit bieten, mit Freunden und Verwandten in Kontakt zu treten, erfreuen sich großer Beliebtheit. Während dieses Teilgebiet der Informatik gerade einen großen Aufschwung erlebt und einem laufenden Wandel unterzogen ist, so ist eine andere Technik nahezu unverändert geblieben: die Authentifizierung mittels Passwörtern.
Passwörter bilden aufgrund ihrer guten Implementierbarkeit und einfachen Verständlichkeit noch immer die meist genutzte Form der Zugangskontrolle in Computersystemen. Gleichzeitig sind sichere Passwörter schwierig zu merken, weshalb bei deren Erstellung oft auf personenbezogene Begriffe wie Namen oder Geburtsdaten zurückgegriffen wird. Durch diesen Umstand entsteht eine Angriffsmöglichkeit auf Passwörter, die es in dieser Arbeit zu erforschen gilt. Es stellt sich die Frage, ob es möglich ist, personenbezogene Informationen durch automatisierte Internetrecherche akquirieren zu können, um daraus in weiterer Folge Passwörter zu erstellen.

1.1 Problemstellung

Kryptographie ist die Wissenschaft, die sich mit dem Verschlüsseln von Informationen beschäftigt. Verschlüsselung ist dabei der Prozess, lesbare Daten unlesbar zu machen, wobei dieser Vorgang nur mit Hilfe eines geheimen Begriffs, dem Schlüssel, wieder rückgängig gemacht werden kann. Ist der Begriff zum Verschlüsseln gleich dem Begriff zum Entschlüsseln, spricht man von symmetrischer Verschlüsselung, andernfalls von asymmetrischer Verschlüsselung. In beiden Fällen hat das Verfahren die Absicht, Informationen vor ungewünschter Einsicht zu schützen und dem Besitzer oder der Besitzerin der Informationen durch Weitergabe des Schlüssels die Entscheidungskraft zu geben, wer die Daten einsehen darf. [1]

Diese Sicherheitsvorkehrung stand jedoch bedingt durch ihre Komplexität lange nur Menschen mit hoher Technikaffinität zur Verfügung. Kryptographie in Form einfach zugänglicher Verschlüsselung von Geräten ist in den letzten Jahren wesentlicher Bestandteil vieler Betriebssystemen geworden, wodurch diese technische Hürde herabgesetzt wurde. IT-Forensikerinnen und IT-Forensiker sehen sich durch die steigende Nutzung von Kryptographie verstärkt vor das Problem gestellt, die Beweise auf sichergestellten Datenträgern nicht auswerten zu können, da diese vom Besitzer verschlüsselt wurden. Rogers und Seigfried konnten bereits 2004 in ihrer Feldstudie *The future of computer forensics: a needs analysis survey* feststellen, dass Verschlüsselung von Forensikerinnen und Forensikern am dritthäufigsten als Problem genannt wird [2]. Aktuellere Arbeiten bestätigen diesen Umstand und zeigen eine Entwicklung zu häufigerer Anwendung von Verschlüsselung in privaten Haushalten. [3, 4]
Um diese für die Aufklärung des Falles wichtigen Informationen wieder zugänglich zu machen, besteht oft keine andere Möglichkeit als das unbekannte Passwort durch Ausprobieren aller Kombinationen aus Buchstaben, Zahlen und Zeichen zu erraten, was eine äußerst rechen- und damit zeitintensive Aufgabe darstellt. Eine Alternative zu diesen Brute-Force Attacken stellen Passwortlisten dar, die viele verschiedene relativ simple und damit unsichere Passwörter einbeziehen. Da aufgrund von steigendem Sicherheitsbewusstsein diese generischen Passwörter immer weniger genutzt werden, hat sich das Einbeziehen von personenbezogenen Begriffen wie Namen, Orten oder Jahreszahlen beim Erstellen von Passwörtern als neue Strategie etabliert.
Auf Basis dieser Bedarfsanalyse stellt sich die Forschungsfrage, die in der vorliegenden Arbeit beantwortet werden soll:

> *»Ist es möglich, personenbezogene Informationen durch Automatisierung sowohl zu sammeln als auch daraus Passwörter zu generieren, die den Kriterien typischer Passwortmuster entsprechen?«*

1.2 Struktur der Arbeit

Um diese Frage beantworten zu können, gilt es theoretisch und anschließend praktische Aspekte des Themas zu bearbeiten.
In Kapitel 2 wird der theoretische Hintergrund von Passwörtern dargelegt. Dabei wird einerseits auf die Geschichte des Passwortes eingegangen und andererseits beliebte Strategien zur Erstellung dieser sowie deren Sicherheit und unterschiedliche Angriffsvektoren beleuchtet. Anschließend folgen die Möglichkeiten zur Akquise von personenbezogenen Informationen sowie

ein Einblick in die theoretischen Hintergründe von sozialen Netzwerken und deren Privatsphäreeinstellungen.
Kapitel 3 befasst sich mit wissenschaftlichen Arbeiten, die sich mit ähnlichen Forschungsfragen beschäftigen. Des Weiteren werden bestehende Tools zur Erstellung von Wortlisten und zum Erraten von Passwörtern vorgestellt.
Es folgt in Kapitel 4 die Präsentation des Scrapingtools *slughorn*, das im Zuge dieser Arbeit entwickelt wurde. Dabei wird auf die Architektur der Software und auf die Methoden der Datenakquise, der Datenverarbeitung und der Passwortgenerierung eingegangen. Letztlich werden Herausforderungen bei der Entwicklung angesprochen, die zu atypischen Entscheidungen bezüglich der Architektur geführt haben.
In Kapitel 5 wird die Software einer Evaluierung unterzogen. Es erfolgt in Kapitel 6 die Diskussion der Arbeit, in der die oben formulierte Fragestellung beantwortet wird. Bevor die Arbeit abschließend rekapituliert wird, werden Möglichkeiten für zukünftige Themen besprochen, die an die Erkenntnisse der vorliegenden Arbeit anknüpfen können.

2 Hintergrund

In diesem Kapitel werden grundlegende Hintergründe zu Techniken erklärt, die für das Verstehen der in den späteren Kapiteln dargelegten Methoden Voraussetzung sind. Einführend wird das Konzept von Passwörtern anhand ihrer Geschichte, der verschiedenen Strategien zur Erstellung und ihrer Sicherheit erläutert. Es folgt eine Erklärung möglicher Angriffsvektoren auf Passwörter. Anschließend geht es um die Akquise von personenbezogenen Informationen am Beispiel der sozialen Netzwerke Facebook und Twitter.

2.1 Passwörter

Trotz laufender Forschung und stetiger Entwicklung neuer Authentifizierungsverfahren basierend auf biometrischen Eigenschaften, wie etwa Fingerabdruckscannern oder Handvenenerkennung, ist das Passwort aufgrund seiner einfachen Implementierbarkeit und Verständlichkeit in modernen Computersystemen allgegenwärtig. In diesem Kapitel gilt es neben der Geschichte des Passwortes verschiedene Methoden aufzuzeigen, die genutzt werden, um diese nach bestimmten Mustern zu generieren. Darauffolgend wird die Frage beantwortet, was ein sicheres Passwort ausmacht und woran dies gemessen werden kann. Abschließend werden verschiedene Angriffsmethoden auf Passwörter beleuchtet und ihre Erfolgschancen in Bezug auf die vorher definierten Passwortstrategien diskutiert.

2.1.1 Geschichte des Passwortes

Geschichtlich kann der Beginn der Nutzung von geheimen Wörtern zur Authentifizierung von Personen nicht genau festgestellt werden. Durch antike Geschichtsschreiber wie Polybius, Erfinder der Polybius-Chiffre, ist bekannt, dass bereits im zweiten beziehungsweise dritten Jahrhundert vor Christus sogenannte Wachwörter militärischen Gebrauch fanden. In seinem Werk *Historíai* [5, S. 487] beschreibt Polybius eine Strategie zur Verteilung von Wörtern, die in der darauffolgenden Nacht zur Authentifizierung bei der Ablöse der Wachen dienten.
Besondere militärische Relevanz hatten Passwörter in den beiden Weltkriegen 1914-1918 be-

ziehungsweise 1938-1945. Ein Beispiel für den Gebrauch von Passwörtern ist das sogenannte *Countersign* Verfahren, ähnlich der aus der Informatik bekannten *Challenge-Response Authentifizierung*. Das unter anderem von US-Amerikanischen Soldaten 1944 in der Schlacht um die Normandie verwendete Verfahren beruht darauf, dass bei einer Annäherung einer Person an eine stationäre Partei, ersterer ein Zeichen (die *Challenge*) präsentiert wird. Diese hat darauf mit dem korrekten Passwort zu antworten, worauf die andere Partei ein Gegenzeichen (Countersign) gibt, um sich ebenfalls zu authentifizieren. Die drei Zeichen (zum Beispiel in Form von Wörtern oder tatsächlichen Handzeichen) wurden vorher in der Einheit bekanntgegeben und nach wenigen Tagen geändert. [6]

In der Informatik fand eine Authentifizierung des Benutzers oder der Benutzerin mithilfe von Benutzername und Passwort möglicherweise ihre erste Verwendung [7, S. 31] in dem vom *MIT Computation Center* entwickelten und 1961 präsentierten *Compatible Time-Sharing System*. Da zu dieser Zeit noch keine Displays vorhanden waren, geschah die Textausgabe mithilfe eines Teleprinters, also einer elektromechanischen Schreibmaschine. Um die Vertraulichkeit des Passwortes zu gewährleisten, wurde daher die Textausgabe nach der Eingabe des Benutzernamens deaktiviert und nach erfolgter Authentifizierung wieder aktiviert [8].

Zu Beginn des UNIX Systems wurden Passwörter aller User in einer einzelnen Textdatei gespeichert, die aufgrund der hohen Vertraulichkeit des Inhalts besonders gegen unbefugten Zugriff gesichert werden musste. Diese Herangehensweise setzt vollständig darauf, dass die Passwortdatei ausreichend geschützt ist. Es wird so ein *Single Point of Failure* gebildet, der bei einem Bruch der Sicherheitsvorkehrungen zu folgenschweren Konsequenzen für das System führen kann. In einer 1979 von Robert Morris und Ken Thompson veröffentlichten Arbeit [9], wird eine Verbesserung dieses Verfahrens vorgestellt, auf der heute noch die Organisation von Passwörtern in UNIX Systemen basiert. Die Autoren schlugen vor, Passwörter mit dem *crypt-Algorithmus* (basierend auf dem *Data Encryption Standard*, kurz DES) zu verschlüsseln, was gesonderte Sicherheitsvorkehrungen für die Datei selbst obsolet machte.

Heute ist das Passwort trotz zahlreicher Alternativen noch immer das meistgenutzte Authentifizierungsverfahren. Dies liegt vor allem daran, dass das Verfahren äußerst simpel zu implementieren ist und keine besonderen Vorkenntnisse oder wie bei biometrischen Verfahren eine spezielle Technik erfordert.

2.1.2 Passwortstrategien

Bezüglich der Wahl der Passwortstrategie konnten Notoatmodjo et al. 2009 feststellen, dass Benutzerinnen und Benutzer abhängig von der Höhe des Schutzbedarfs unterschiedliche Erstellungsmethoden wählen, die ihrer Meinung nach sichere beziehungsweise unsichere Passwörter erzeugen [10]. Dieser Umstand konnte in der Arbeit von Haque et al. 2013 bestätigt werden. In dieser Studie wurden 80 Personen aufgefordert, Passwörter für verschiedene Webseiten aus den Bereichen Finanz (zum Beispiel Onlinebanking), Identität (Social Media) und Inhalt (Nachrichtenseiten) zu erstellen. Nach Auswertung der Passwörter war klar erkennbar, dass bei sicherheitskritischeren Seiten auch höhere Anforderungen an die Passwortsicherheit gestellt wurden. [11]
Bei der Kreation von Passwörtern sind Benutzerinnen und Benutzer vor die Herausforderung gestellt, sowohl Sicherheit als auch leichte Merkbarkeit zu vereinen, was scheinbar einen Widerspruch darstellt: Um es Angreifern zu erschweren, ein Passwort zu erraten, wird empfohlen, die Komplexität durch Einbeziehen unterschiedlicher Zeichengruppen (Kleinbuchstaben, Großbuchstaben, Zahlen, Sonderzeichen) und Steigerung der Zeichenanzahl zu erhöhen (siehe Kapitel 2.1.3). Diese Vorkehrungen können jedoch die Merkbarkeit des Passwortes enorm erschweren.
Im Durchschnitt hat ein Benutzer oder eine Benutzerin 25 Zugänge im Internet, für die Passwörter verwaltet werden müssen [12]. Die kognitiven Fähigkeiten des Menschen sind jedoch nicht darauf ausgelegt, Zeichenketten, die in Kapitel 2.1.3 als *sicher* eingestuft werden, in dieser Menge in dauerhafter Erinnerung zu behalten. Adams et al. gehen sogar davon aus, dass dem durchschnittlichen Menschen nur etwa vier bis fünf Passwörter zugemutet werden können [13]. Diese Diskrepanz zwischen Merkbarkeit und Sicherheit stellt das größte Problem der Authentifizierung mit Passwörtern dar [14]. Benutzer und Benutzerinnen, die mit diesem Konflikt konfrontiert sind, entscheiden sich instinktiv für leichtere Merkbarkeit, da ein vergessenes Passwort ein realistischeres Szenario darstellt als ein Angriff und der Aufwand einer Wiederherstellung des Passwortes als groß betrachtet wird [11]. Seit der Verwendung des Passwortes als Authentifizierungsverfahren, haben sich verschiedene Herangehensweisen bei der Erstellung von Passwörtern entwickelt, die es zu unterschiedlichen Graden vermögen, diese beiden Eigenschaften zu erfüllen.
Im Folgenden werden verschiedene Passwortstrategien vorgestellt, wobei auf die Sicherheit dieser Methoden in Kapitel 2.1.3 eingegangen wird. Bei der Beschreibung des Aufbaus von Passwörtern wird in dieser Arbeit die Notation von Chou et al. verwendet [15]. Dabei werden folgende Zeichengruppen unterschieden:

Zeichengruppe	Abkürzung	Größe	Elemente
Numerisch	N	10	0123456789
Kleinbuchstaben (Lowercase)	L	26	abcdefghijklmnopqrstuvwxyz
Großbuchstaben (Uppercase)	U	26	ABCDEFGHIJKL MNOPQRSTUVWXYZ
Andere (Other)	O	32	~_‘!@#$%^&*()-+=[]\{ }|;’:’,./<>?

Tabelle 2.1: Notation für Zeichengruppen nach Chou et al.

Um Zeichenketten zu beschreiben, kann nun in weiterer Folge die oben definierte Notation mit folgenden Regeln verwendet werden, wobei Z ein beliebiges Zeichen aus der Menge $\{N, L, U, O\}$ sein kann:

1. Z^+ wird als Musterklasse bezeichnet und beschreibt das Vorkommen beliebig vieler Zeichen der Gruppe Z direkt aneinandergereiht.

2. Z_n wird als Muster bezeichnet und beschreibt das Vorkommen von genau *n* Zeichen der Gruppe Z direkt aneinandergereiht.

3. Z_n ist eine detailliertere Beschreibung und daher eine Teilmenge von Z^+.

4. $[ZZ']$ schließt die Zeichen Z und Z' zu einer Gruppe zusammen. Beispiel: $[UL]_3$ beschreibt eine dreistellige Zeichenfolge bestehend aus Groß- und Kleinbuchstaben.

Standardwörter Eine von Florencio et al. 2007 durchgeführte Studie untersucht unter anderem die Art und die Verwendung von verschiedenen Zeichengruppen bei der Erstellung neuer Passwörter. Das Ergebnis zeigt, dass - abhängig von der Anzahl der Zeichen - etwa 65-97% der Testpersonen nur aus Kleinbuchstaben bestehende (L^+) Passwörter, einschließlich kleingeschriebener Eigennamen oder Nomen, wählen [12]. Eine minimale Erweiterung dieser Methode ist durch eine (bei Eigennamen und Nomen) rechtschreibkonforme Großschreibung des ersten Buchstabens gegeben. Dadurch erweitert sich die Musterklasse zu $[UL]_1L^+$. Beispiele für Passwörter dieser Kategorie sind in Tabelle 2.2 angeführt.
Wörter, die in gebräuchlichen Wörterbüchern zu finden sind, bilden also oft äußerst leicht merkbare Passwörter, deren mangelnde Sicherheit jedoch belegt und auch häufig kommuniziert wird. Ein Verfahren, mit dem diese Art von Passwörtern erraten werden kann, wird in Kapitel 2.1.4

vorgestellt.
Aufgrund des Bekanntheitsgrades dieser Schwäche, werden Passwörter dieser niedrigen Sicherheitsstufe bereits von den meisten Online-Plattformen und Firmen abgelehnt. Auch im Falle der oben genannten Studie konnte dem Verhalten der Testpersonen nur entgegengewirkt werden, indem Mindestvoraussetzungen für das zu wählende Passwort - wie etwa Sonderzeichen oder Zahlen - vorgegeben wurden [12].

Beispiel	**Musterklasse**	**Muster**	**Elemente**
password	L^+	L_8	[password]
Charlie	U_1L^+	U_1L_6	[C, harlie]
motorcycle	L^+	L_{10}	[motorcycle]
football	L^+	L_8	[football]

Tabelle 2.2: Beispiele für Standardwörter

Standardwörter mit Substitution und/oder Anhang Eine Erweiterung der Strategie, die Standardwörter verwendet, die in Wörterbüchern zu finden sind, ist die Methode der Substitution und Anhänge. Als Anhang werden hier Zahlen bezeichnet, die zu einfach zu merkenden Begriffen hinzugefügt werden ($[UL]_1L^+N^+$), um die Komplexität zu erhöhen. Ein häufig verwendetes Beispiel für diese Methode ist das Anhängen eines Datums an einen Namen, wie in Tabelle 2.4 ersichtlich ist.
Substitution ist das Austauschen bestimmter Zeichen mit ähnlich aussehenden Zahlen (N) oder Sonderzeichen (O). Bekanntes Beispiel dafür ist *Leetspeak*, was in der eigenen Schreibweise auch als 1337 dargestellt wird und sich im Internet und der Gamingszene als eine Art *Pseudogeheimsprache* entwickelt hat [16, 17]. Da es keine einheitlichen Regeln für die Transformation von Buchstaben gibt, können an dieser Stelle nur beispielhaft bekannte Substitutionen erwähnt werden. Einige davon sind in Tabelle 2.3 angeführt:

Buchstabe	Substitution
A, a	4, @
B, b	8
E, e	3, €
I, i	1, !, |
L, l	1, !, |
O, o	0
R, r	2
S, s	5
T, t	7, +
Z, z	2

Tabelle 2.3: Beispiele für Buchstabensubstitutionen

In Tabelle 2.4 sind sowohl Beispiele für Substitution und Anhänge als auch für Mischformen angeführt. Bemerkenswert ist die erhöhte Komplexität der Musterklassen beziehungsweise des Musters, die durch Substitution erreicht wird. Wie sich in Kapitel 2.1.3 herausstellen wird, ist diese Komplexität nur für das menschliche Auge sichtbar und für den Computer und damit für Angriffe auf Passwörter nicht vorhanden.

Beispiel	Musterklasse	Muster	Elemente
password123	L^+N^+	L_8N_3	[password, 123]
Charlie1989	$U_1L^+N^+$	$U_1L_6N_4$	[C, harlie, 1989]
m0t0rcycl3	$L^+N^+L^+N^+L^+N^+$	$L_1N_1L_1N_1L_5N_1$	[m, 0, t, 0, rcycl, 3]
f00tb4ll777	$L^+N^+L^+N^+L^+N^+$	$L_1N_2L_2N_1L_2N_4$	[f, 00, tb, 4, ll, 777]

Tabelle 2.4: Beispiele für Standardwörter mit Substitution und/oder Anhang

Mnemonik Eine Methode, die besonders darauf abzielt die Anforderungen Merkbarkeit und Sicherheit zu vereinen, ist das Bilden von Passwörtern auf Basis von Sätzen. Dieses Verfahren, das auch als Mnemonik bekannt ist, kann als *Gedächtnisstütze* bezeichnet werden und funktioniert nach folgendem Prinzip:

1. Wähle einen Satz von ausreichender Länge (mindestens acht Wörter).
2. Füge die Anfangsbuchstaben der Wörter des Satzes zu einer Zeichenkette zusammen.

3. Optional: Ersetze bestimmte Buchstaben durch Zahlen oder Sonderzeichen (siehe Kapitel 2.1.2 Abschnitt *Standardwörter mit Substitution und/oder Anhang*). Außerdem ist es üblich, ganze Wörter durch Buchstaben oder Zahlen zu ersetzen, wobei das Ausmaß der Sinnhaftigkeit vom jeweiligen Menschen und der gesprochenen Sprache abhängt. Bekannte und oft verwendete Wortsubstitutionen sind zum Beispiel
    - *&* statt *und* bzw. *and*
    - *4* statt *for*
    - *u* statt *you*
4. Benutze die entstandene Zeichenkette als Passwort.

Die Methode beruht auf der Tatsache, dass lange Sätze aufgrund des zusätzlichen Kontextes für Menschen leichter zu merken sind als zufällige kurze Zeichenketten ohne Bedeutung [18, 19]. Ein Beispiel für ein so erstelltes Passwort ist *WuptGoT,uw||ud*, das auf dem Satz »*When you play the game of thrones, you win or you die.*« aus der Serie *Game of Thrones* beruht. Das *y* von *you* wurde hierbei durch ein *u* ersetzt, da dies phonetisch betrachtet substituierbar ist. Zusätzlich wurde der Beistrich als Sonderzeichen eingefügt und das *or* durch einen doppelten senkrechten Strich ersetzt, da dies in vielen Programmiersprachen (z.B. Java, C, Perl, ...) als ein logisches *Oder* interpretiert wird. Durch die Wahl einiger Großbuchstaben an geeigneten Stellen (die offizielle Schreibweise des Serientitels lautet *Game of Thrones*) wird zudem die Voraussetzung von vielen Plattformen erfüllt, nach der sowohl Groß- als auch Kleinbuchstaben verwendet werden müssen. Weitere Beispiele für diese Art von Passwörtern sind in Tabelle 2.5 angeführt.
Studien wie etwa jene von Kuo et al. zeigen, dass mnemonische Passwörter - neben der guten Merkbarkeit - aufgrund der verschiedenen Zeichengruppen auch resistenter gegenüber Brute-Force-Attacken (siehe Kapitel 2.1.4) sind. Die große Schwäche der Methode wird durch das oben genannte Beispiel ersichtlich: Anstatt selbst Sätze neu zu bilden, greifen vielen Menschen auf bereits bekannte und etablierte Sätze, wie etwa Redewendungen, Musiktexte oder Werbesprüche zurück. Diese Tatsache kann von Angreifern genutzt werden, indem Wörterbücher auf Basis von diesen bekannten im Internet veröffentlichten Sätzen erstellt werden [19] (siehe Kapitel 2.1.4).
Eine weitere Schwachstelle mnemonischer Passwörter wurde 2017 in einer Arbeit von Kiesel et al. veröffentlicht [20]. Darin wurde versucht, statistische Auffälligkeiten in mnemonischen Passwörtern zu finden. Dies kann zum Beispiel das gehäufte Vorkommen bestimmter Buchstaben an erster Stelle sein, was durch den englischen Artikel *The* aber auch *Der/Die/Das* in der

deutschen Sprache bedingt ist. Ebenso konnten Muster in Substitutionen bestimmter Zeichen oder in der Verwendung von Groß- und Kleinbuchstaben gefunden werden. Diese Erkenntnisse ermöglichen es, den Brute-Force Angriff durch Angabe von Mustern zu optimieren (siehe Kapitel 2.1.4).

Beispiel	Musterklasse	Muster	Elemente
WuptGoT,uw‖ud (»When you play the game of thrones, you win or you die.«)	$[NULO]^+$	$U_1L_3U_1L_1$ $U_1O_1L_2O_2L_2$	[W, upt, G, o, T, uw, ‖, ud]
1gGsmniM (»Einem geschenkten Gaul schaut man nicht ins Maul.«)	$[NULO]^+$	$N_1L_1U_1L_4U_1$	[1, g, G, smni, M]
2bon2b,tit? (»To be or not to be, that is the question«)	$[NULO]^+$	$N_1L_3N_1$ $L_1O_1L_3O_1$	[2, bon, 2, b, „, tit, ?]

Tabelle 2.5: Beispiele für mnemonische Passwörter

Zufallspasswörter Die sicherste Form von Passwörtern (siehe Kapitel 2.1.3) ist eine zufällige Zeichenkette in ausreichender Länge (siehe Kapitel 2.1.4) bestehend aus Klein- und Großbuchstaben, Zahlen und Sonderzeichen. Beispiele dafür sind in Tabelle 2.6 zu sehen. Diese Art von Zeichenkette bildet jedoch auch gleichzeitig die am schwierigsten zu merkende, weshalb diese Herangehensweise nur mit der Verwendung eines zusätzlichen Programms, eines *Passwortmanagers*, praktikabel ist. Diese Software umfasst in einer verschlüsselten Datenbank Benutzernamen, Passwörter und sonstige Informationen über alle Accounts. Um optimale Sicherheit zu gewährleisten, sollte für jedes Service ein eigenes zufällig erstelltes Passwort generiert werden, da so selbst im Falle einer Veröffentlichung eines Passwortes alle anderen Accounts weiterhin geschützt sind. Zugriff zur Datenbank bekommt man nur über ein sogenanntes *Masterpasswort*, welches sogleich den Schwachpunkt dieser Methode darstellt: Da dieses Masterpasswort Zugriff zu allen anderen Passwörtern bietet, stellt es einen *Single Point of Failure* dar und bedarf besonderen Schutzes. Gleichzeitig muss es für den Benutzer oder die Benutzerin aber auch merkbar sein, weshalb es den selben eingeschränkten Methoden und Schwächen unterliegt, die in diesem Kapitel bereits beleuchtet wurden.

Die meisten Generatoren in Passwortmanagern verfügen zusätzlich über die Möglichkeit aussprechbare Zufallspasswörter zu erzeugen. Dieses Verfahren beruht auf einer Arbeit von Morrie Gasser aus dem Jahr 1975 und fügt Silben nach dem Zufallsprinzip so zusammen, dass nicht existierende aber aussprechbare Wörter entstehen und kombiniert diese nach Wunsch auch mit Zahlen und Sonderzeichen [21].

Beispiel	Musterklasse	Muster	Elemente
bin19am_Adel	$[NULO]^+$	$L_3N_2L_2O_1U_1L_3$	[bin, 19, am, _, A, del]
kx8Vb3{S4	$[NULO]^+$	$L_2N_1U_1L_1N_1O_1U_1N_1$	[kx, 8, V, b, 3, {, S, 4]

Tabelle 2.6: Beispiele für Zufallspasswörter

Ausgefallene Methoden

Verwendung von Emojis Aufgrund der steigenden Beliebtheit von Emoticons, also der Repräsentation von kleinen Bildern mit Hilfe von ASCII-Zeichen - zum Beispiel das lächelnde Gesicht *:-)* - wurden diese zu sogenannten Emojis weiterentwickelt. Dabei handelt es sich um farbige ideographische Piktogramme, also um einfache Symbole, die für ganze Wörter oder Begriffe stehen. Das Pendant zum oben erwähnten Emoticon *:-)* bildet etwa das *Emoji Slightly Smiling Face* 🙂. Mit der Version 6.0.0 wurden am 11. Oktober 2010 über 1000 solcher Symbole zum Unicode Standard hinzugefügt, wobei laufend neue Emojis ergänzt werden [22]. Wie bereits anderen Zeichen, wie etwa dem *ß* mit *U+00DF*, wird nun auch jedem Emoji ein sogenannter Codepunkt zugewiesen. Das bereits erwähnte Emoji *Slightly Smiling Face* 🙂 wurde etwa erst 2014 mit Unicode 7.0 standardisiert und erhielt den Codepunkt *U+1F642*. Diese einheitliche Definition der Piktogramme durch Unicode Codepunkte ermöglicht nun auch die Verwendung von Emojis in Passwörtern wie jedes andere Zeichen, da diese unabhängig von betriebssystemspezifischer Darstellung stets gleich encodiert werden.
Eine 2017 erschienene Studie von Golla et al. kam zum Ergebnis, dass Emojis bei der Authentifizierung auf Smartphones aufgrund der leichten Benutzbarkeit enormes Potential aufweisen und eine denkbare Alternative zu etablierten PIN-Codes darstellen: Emojis erfreuen sich großer Beliebtheit und sind ein viel genutztes Medium. Damit sind diese den Benutzern vertraut und sind zudem in der Lage, leicht merkbare Geschichten zu repräsentieren. [23]
Auf Computern zeigt sich jedoch ein elementarer Nachteil dieser Art von Passwörtern: Während Emojis bei der Vergabe des Passwortes noch leicht über Einblendung virtueller Tastaturen verfügbar sind, erweist sich die Eingabe des Passwortes beim Login nach einem Start des Betriebs-

systems oder nach Aufwachen aus dem Ruhezustand oft als schwierig. Diese Einschränkung führt dazu, dass die Verwendung von Emojis in Passwörtern oft zu Problemen führt und daher bisher wenig Anklang gefunden hat.

Kryptonizer Einen Trick, leicht merkbare unsichere Passwörter sicherer zu machen, will die deutsche Firma *Hey!blau Labs e.K.* mit dem Produkt *Kryptonizer* anbieten [24]. Der Kryptonizer ist ein Übersetzer für Passwörter, wobei der Übersetzungsmechanismus individuell für jedes Stück neu erstellt wird und auf ein kleines Plastikplättchen gedruckt wird. Das Produkt kann durch Logos individualisiert und in hohen Stückzahlen bestellt werden. Der beworbene Einsatzzweck des *Kryptonizers* ist die Verteilung als Werbegeschenk. Ein Beispiel für einen Übersetzungsmechanismus ist in Tabelle 2.7 zu sehen:

Start	7rl=		
ABC	**DEF**	**GHI**	**JKL**
6	R	?	k
MNO	**PQRS**	**TUV**	**WXYZ**
n	L	F	J

Tabelle 2.7: Beispiele für einen Übersetzungsmechanismus des Kryptonizers

Der Benutzer oder die Benutzerin wird aufgefordert, ein einfaches Passwort, das unter normalen Umständen als unsicher gelten würde, nach den folgenden Prinzipien zu transformieren:

- Das neue Passwort beginnt mit dem angegebenen Startzeichen (*7rl=*)
- Wähle für jedes Zeichen des Eingabewortes das korrespondierende Zeichen aus der Übersetzungstabelle

Als Beispiel kann das Wort *Dampfschiff* betrachtet werden: Der Startwert ist unabhängig vom Input auf *7rl=* gesetzt. Anschließend wird der Buchstabe *D* betrachtet, der sich in der Gruppe *DEF* befindet und somit zu *R* transformiert wird. Das nächste Zeichen des Ausgangswortes ist *a*. Da bei dieser Variante nicht zwischen Groß- und Kleinbuchstaben unterschieden wird, wird das Zeichen für die Gruppe *ABC* gewählt (welches *6* ist) und dem neuen Passwort hinzugefügt. Nach dieser Methode wird das gesamte Eingangswort abgearbeitet bis letztendlich das Passwort *7rl=R6nLRL6??RR* entsteht.
Dieses Passwort genügt demnach - oberflächlich betrachtet - den Voraussetzungen eines sicheren

Passwortes, indem Zeichen der Mengen O, L, U und N vorkommen. Welche Zeichengruppen unabhängig vom Eingangswort auf jeden Fall vorkommen, hängt vom zufällig generierten Wert im Startsegment ab, wobei die Inspektion verschiedener Kryptonizer die Annahme zulässt, dass zumindest gesichert ein Sonderzeichen gewählt wird.

2.1.3 Sicherheit

Wie bereits in Kapitel 2.1.1 dargelegt wurde, kann das Passwort eine lange Geschichte aufweisen und hat bis heute kaum etwas an seiner Relevanz eingebüßt. Obwohl Benutzern und Benutzerinnen mittlerweile oft bewusst ist, dass die Wahl eines sicheren Passwortes essentiell für dessen Schutzfunktion ist, siegt dennoch oft aus Gründen der Bequemlichkeit aber auch oft aus Überforderung die Simplizität über die Komplexität bei der Wahl eines geeigneten Passwortes [25]. Um diesem Mangel an Informationen entgegenzuwirken, gilt es nun zu definieren, was ein sicheres Passwort ausmacht.

Das Bestimmen der Sicherheit eines Passwortes ist keine triviale Aufgabe, was durch die Menge an verschiedenen Passwortsicherheitsskalen und wissenschaftlichen Arbeiten über diese belegt werden kann [25–27]. De Carnavalet et al. haben in ihrer Arbeit *From Very Weak to Very Strong: Analyzing Password-Strength Meters.* 13 verschiedene Skalen verglichen, die bei verschiedenen Services verwendet werden und dabei festgestellt, dass gleiche Passwörter stark variierende Ergebnisse erzielen. So bekam *password$1* bei *Dropbox* die niedrigste Sicherheitsstufe, während es bei *Twitter* als *perfekt* bezeichnet wurde. Dies ist laut den Autoren durch die verschiedenen verwendeten Methoden und die uneinheitliche Gewichtung der Faktoren begründet. [17]

Unbestritten ist, dass zwei maßgebliche Sicherheitsmerkmale von Passwörtern durch ihre Angriffsmöglichkeiten feststellbar sind:

Der erste Faktor ist die Länge des Passwortes, die beim Versuch das Passwort zu erraten (siehe *Brute-Force-Attacken* in Kapitel 2.1.4), eine wesentliche Rolle spielt. Wie sich zeigen wird, ist das Durchprobieren aller möglichen Zeichenkombinationen nur bis etwa 8 Stellen praktikabel, weshalb viele automatische Überprüfungen von Passwörtern erst ab dieser Anzahl an Zeichen eine positive Rückmeldung geben.

Dass dieser Faktor jedoch nicht uneingeschränkt ein Anzeichen für Sicherheit ist, zeigt das Beispiel *password1*, das mit 9 Zeichen diese Bedingung erfüllt und dennoch klar ersichtlich keine gute Wahl ist. Die Eigenschaft, die in diesem Beispiel verletzt wird, wird in dieser Arbeit als *Einzigartigkeit* beziehungsweise *Unberechenbarkeit* bezeichnet [28]. Während der Schutz durch

Länge nur von der Anzahl der Zeichen und der Größe des Alphabets abhängt, bildet dieses zweite Merkmal eine äußerst komplexe Angriffsfläche, die, wie in Kapitel 2.1.4 näher beschrieben wird, auf verschiedene Arten ausgenutzt werden kann. Das Merkmal der Unberechenbarkeit wird allgemein über das in der Informationstheorie gängige Maß der *Shannon Entropie* festgelegt [29]. Dies ist definiert als der erwartete Wert an Information in einem Text, gemessen in Bits. Diese Beschreibung des Informationsgehaltes kann für den Zweck der Bestimmung der Sicherheit von Passwörtern auch als die Unberechenbarkeit bezeichnet werden. So gibt eine Entropie von 1 an, dass ein Bit Information gespeichert ist, also mit einer Wahrscheinlichkeit von 0,5 erraten werden kann. Eine Erhöhung der Entropie eines Passwortes um 1 führt demnach zu einer Verdoppelung der möglichen Wörter und damit zu einer Verdoppelung der durchschnittlich benötigten Versuche, um das Passwort zu erraten. Die Shannon Entropie einer zufällig gewählten Zeichenfolge lässt sich mit der in Gleichung 2.1 dargestellten Formel berechnen, wobei H für die Entropie in Bits, $|Z|$ für Größe des Alphabets und l für die Länge des Wortes steht.

$$H = log_2(|Z|^l) \tag{2.1}$$

Ein 9-stelliges Passwort bestehend aus zufällig gewählten Kleinbuchstaben besitzt demnach eine *Shannon Entropie* von etwa 42 ($log_2(|26|^9) = 42,304$). Im Vergleich dazu wird das längere Passwort *bizarrgelenk* untersucht, das aus zwei zufällig ausgewählten deutschen Wörtern aus einer Liste von 66.666 Einträgen zusammengesetzt ist [30]. Dieses Passwort besitzt trotz seiner Länge von 12 Zeichen und dem scheinbar gleichen Alphabet aus Kleinbuchstaben eine Shannon Entropie von 32. Diese deutlich niedrigere Entropie ist dadurch begründet, dass das tatsächliche Alphabet für dieses Wort nicht die Menge aller Kleinbuchstaben sondern die Menge aller Wörter aus der Liste ist. Dementsprechend ist die Länge l in diesem Fall nicht 12 sondern 2 ($log_2(|66666|^2) = 32,0493$).
Dieses Beispiel soll zeigen, dass die Länge des Passwortes nicht allein für die Sicherheit ausschlaggebend sein kann und stets beachtet werden muss, welches Alphabet für die Erstellung gewählt wurde und wie sich dies auf die Unberechenbarkeit auswirkt.

2.1.4 Angriffsvektoren

Um die sicherheitsrelevanten Merkmale eines Systems festzustellen, wird oft der Blickwinkel eines Angreifers eingenommen. Mögliche Wege, über welche eine Schutzmaßnahme überwunden werden kann - in diesem Fall in Form des Erratens des Passwortes - werden Angriffsvektoren genannt. Die zwei maßgeblichen Sicherheitsmerkmale von Passwörtern, deren Länge und deren

Einzigartigkeit beziehungsweise Berechenbarkeit, die in Kapitel 2.1.3 erläutert wurden, können auf verschiedene Arten ausgenutzt werden. Bekannte Angriffsvektoren werden im Folgenden näher beleuchtet.

Brute-Force Attacken Ein kryptografisch sicheres Hashverfahren besitzt unter anderem die Eigenschaft, dass es nicht reversibel ist - sprich, dass es nur möglich ist, einen geeigneten Input für den vorliegenden Hash zu finden, indem alle möglichen Kombinationen ausprobiert werden. In dieser Arbeit wird von einem forensischen Szenario ausgegangen, in dem ein aufzudeckendes Passwort in gehashter Form vorliegt und es möglich ist, offline beliebig viele Passwörter auszuprobieren. Im Gegensatz zu Online-Authentifizierungsverfahren, wo ein Account nach wenigen Versuchen gesperrt wird oder eine absichtlich eingeleitete Verzögerung zwischen den Versuchen forciert wird, ist es bei Offlineattacken auf Passwörter möglich, die volle Rechenleistung eines Computers auszunutzen [27]. Als Brute-Force Attacke wird der Versuch bezeichnet, ein Passwort durch automatisiertes Ausprobieren aller möglichen Kombinationen zu erraten. Dabei wird üblicherweise von niedrigen zu hohen Wortlängen hochgearbeitet bis ein weiteres Vorgehen aufgrund der stetig steigenden Kombinationen pro Wortlänge nicht mehr praktikabel ist. Tabelle 2.8 zeigt diesen Anstieg, unter der Annahme, dass Passwörter, wie in Kapitel 2.1.2 beschrieben, aus den 94 verschiedenen Zeichen der Mengen O, L, U und N bestehen können. Beispielhaft werden diese Zahlen auch für eine Einschränkung der möglichen Zeichen auf die Mengen $(L \cup U)$ - Passwörter bestehend aus Buchstaben - beziehungsweise (L) - Passwörter bestehend aus Kleinbuchstaben - angeführt.

Passwortlänge	**Kombinationen**		
n	(L) 26^n	$(L \cup U)$ 52^n	$(O \cup L \cup U \cup N)$ 94^n
0	1	1	1
1	26	52	94
2	676	2.704	8.836
3	17.576	140.608	830.584
4	456.976	7.311.616	78.074.896
5	$\sim 11,8\,Mio.$	$\sim 380,2\,Mio.$	$\sim 7,3\,Mrd.$
6	$\sim 308,9\,Mio.$	$\sim 19,7\,Mrd.$	$\sim 689,9\,Mrd.$
7	$\sim 8,0\,Mrd.$	$\sim 1,03 \times 10^{12}$	$\sim 6,48 \times 10^{13}$
8	$\sim 208,8\,Mrd.$	$\sim 5,35 \times 10^{13}$	$\sim 6,10 \times 10^{15}$
9	$\sim 5,43 \times 10^{12}$	$\sim 2,78 \times 10^{15}$	$\sim 5,73 \times 10^{17}$
10	$\sim 1,41 \times 10^{14}$	$\sim 1,45 \times 10^{17}$	$\sim 5,39 \times 10^{19}$
11	$\sim 3,67 \times 10^{15}$	$\sim 7,52 \times 10^{18}$	$\sim 5,06 \times 10^{21}$
12	$\sim 9,54 \times 10^{16}$	$\sim 3,91 \times 10^{20}$	$\sim 4,76 \times 10^{23}$
...	...	...	...

Tabelle 2.8: Anzahl verschiedener Kombinationen abhängig von Passwortlänge

Es ist ersichtlich, dass dieser exponentielle Anstieg der Anzahl an Kombinationen bereits bei wenigen Stellen zu einer sehr hohen Anzahl an Möglichkeiten führt. Abhängig von der Rechenleistung der vorhandenen Hardware, können so unterschiedlich viele Versuche pro Sekunden gemacht werden. Als Beispiel für eine besonders starke Hardware wird der *Brutalis* [31] der Firma *Sagitta HPC* herangezogen, der ein Setup repräsentiert, das speziell auf Passwortcracking ausgelegt ist. Dieses besteht aus 8 *Nvidida GTX 1080 FE* Grafikkarten und erreicht damit für den Hashalgorithmus MD5 eine Hashrate von 200,3 GH/s und für SHA256256 23.012,1 MH/s. Bei einem verschlüsselten PDF in der Version 1.7 Level 8 (Acrobat 10 - 11) werden 256.200 Versuche pro Sekunde erreicht. [32]

Unter der Annahme, dass das gesuchte Passwort in Form eines Hashes bereitliegt und Zeichen aller Zeichengruppen beinhalten kann, ergeben sich für die zwei beispielhaft angeführten Hashalgorithmen die in Tabelle 2.9 angeführten Zeitdauern, in denen alle möglichen Passwörter durchprobiert werden können. Zusätzlich dazu ist in der letzten Spalte angegeben, wie lange es mit der oben erwähnten Hardware dauern würde, das Passwort eines verschlüsselten PDF zu er-

raten. Dabei ist anzumerken, dass ein erfolgreicher Versuch bereits früher eintreten kann und die angegeben Zeiten die maximale Dauer darstellen. Anhand der Daten in Tabelle 2.9 kann festgestellt werden, dass selbst bei sehr schnellen Hashverfahren wie MD5 eine Brute-Force Attacke ab einer Passwortlänge von 10 Zeichen nicht mehr praktikabel ist.

Passwortlänge	**Dauer**		
n	**MD5**	**SHA256**	**PDF 1.7 Level 8**
0	< 1 Sek.	< 1 Sek.	< 1 Sek.
1	< 1 Sek.	< 1 Sek.	< 1 Sek.
2	< 1 Sek.	< 1 Sek.	< 1 Sek.
3	< 1 Sek.	< 1 Sek.	3,2 Sek.
4	< 1 Sek.	< 1 Sek.	5 Min.
5	< 1 Sek.	< 1 Sek.	8 Stunden
6	3,4 Sek.	10 Sek.	31 Tage
7	5,4 Min.	15 Minuten	∼8 Jahre
8	8,5 Stunden	1 Tag	∼750 Jahre
9	33 Tage	96 Tage	∼70.000 Jahre
10	8 Jahre	25 Jahre	∼6 Mio. Jahre
11	801 Jahre	2.335 Jahre	∼600 Mio. Jahre
12	∼75.000 Jahre	∼220.000 Jahre	∼60 Mrd. Jahre
...	...	...	...

Tabelle 2.9: Dauer um alle Kombinationen auszuprobieren abhängig von Passwortlänge

Diese Aussage der in Tabelle 2.9 dargestellten Daten wird auch durch die graphische Aufbereitung in Abbildung 2.1 ersichtlich. Im Diagramm ist die Dauer, alle Kombinationen abhängig von der maximalen Länge des Passwortes zu probieren, abgebildet. Vertikale Linien markieren die Zeitspannen von einem und zehn Jahren, wobei Brute-Force Attacken mit einer Dauer von über einem Jahr nicht mehr als praktikabel angesehen werden können.

Wörterbuchattacken Wie in Kapitel 2.1.4 Abschnitt *Brute-Force Attacken* beschrieben wurde und in Abbildung 2.1 ersichtlich ist, stellen Brute-Force Attacken bereits bei wenigen Zeichen eine zeitintensive Angriffsmethode dar und sind - je nach Hash- beziehungsweise Verschlüsselungsverfahren - ab einer Länge von etwa acht bis zehn Zeichen nicht mehr praktikabel.
Um längere Passwörter erraten zu können, besteht ein weiterer Ansatz darin, nicht mehr jede

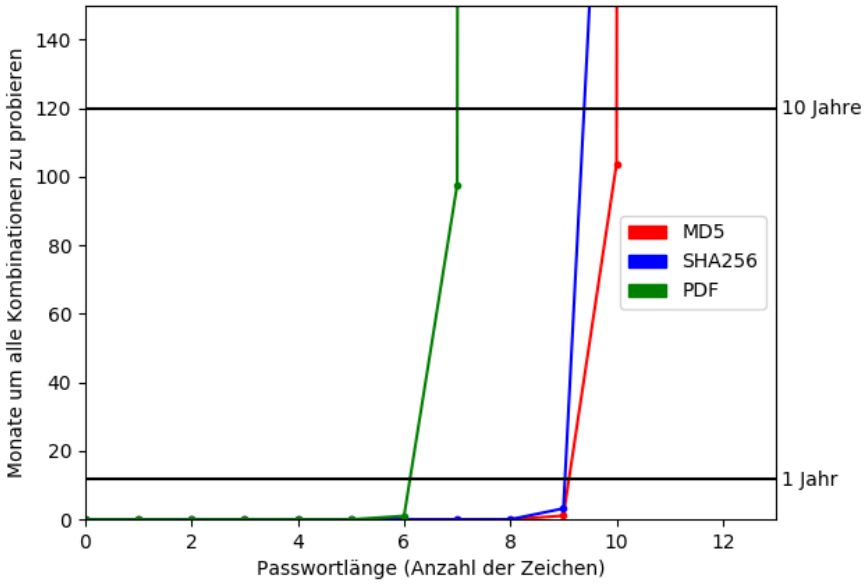


Abbildung 2.1: Dauer, alle Kombinationen zu probieren, abhängig von der Länge des Passwortes

mögliche Kombination von Zeichenketten auszuprobieren, sondern die Ansprüche an eine leichte Merkbarkeit auszunutzen: In Kapitel 2.1.2 wurden Strategien zur Erstellung von Passwörtern auf Basis von Standardwörtern beschrieben. Dabei werden einfach zu merkende Wörter, wie Einträge aus Wörterbüchern oder Namen, entweder ohne weiterer Bearbeitung oder mit Anpassung durch Substitutionen oder Anhänge als Passwörter herangezogen. Während diese Strategie durch eine große Merkbarkeit besticht und daher häufig Anwendung findet, bietet sie Angreifern die Möglichkeit, längere und dadurch scheinbar sicherere Passwörter zu erraten. Listen von möglichen Wörtern werden dabei als Wörterbücher bezeichnet und können als Basis auf tatsächlichen lexikographischen Wörterbüchern verschiedener Sprachen basieren. Um den Fall abzudecken, dass die Zielperson ein Wort gewählt hat, das in keinem Lexikon vorkommt, ist es zu empfehlen, weitere Listen von Wörtern hinzuzufügen. Im Internet ist bereits eine große Anzahl an vorgefertigten Listen zu finden, die für Wörterbuchattacken herangezogen werden können. Beispiele für Wortlisten sind Babynamen, Namen von Sportvereinen, Markennamen oder mnemonische Abkürzungen [20] (siehe Kapitel 2.1.2), wobei individuell abzuwägen ist, welche Listen für eine Zielperson gewählt werden müssen, um ein Wörterbuch mit einer guten Erfolgsaussicht zu generieren.

Letztendlich ist das zusammengestellte Wörterbuch beim Passwortcracking nach bestimmten Regeln anzuwenden. Diese Regeln beschreiben welche Variationen eines einzelnen Wörter-

bucheintrages als Passwort probiert werden sollen. So ist es üblich, zuerst das unveränderte Wort heranzuziehen und anschließend die in Kapitel 2.1.2 beschriebenen Substitutionen und Anhänge vorzunehmen. So besteht die Anzahl der generierten Passwörter aus dem Produkt der Anzahl der Einträge des Wörterbuches und der Anzahl an Regeln. Beispielhaft kann festgestellt werden, dass sich ein Wörterbuch mit 100.000 Einträgen mit 400 Regeln in der selben Größenordnung befindet wie eine Brute-Force Attacke bis zu fünf Zeichen (siehe Tabelle 2.8).

Personenbezogene Passwortlisten Eine mögliche Erweiterung der in Kapitel 2.1.4 Abschnitt *Wörterbuchattacken* beschriebenen Wörterbücher besteht darin, auf die Zielperson zugeschnittene Wortlisten zu erstellen. Darin können Namen von Freunden und Verwandten, Hobbys oder Geburtsdaten enthalten sein. Das Programm CUPP (siehe Kapitel 3.2.2) bietet die Möglichkeit, manuell einen Fragebogen über die Zielperson auszufüllen, woraufhin mögliche Passwörter erstellt werden. Dabei werden bereits die in Kapitel 2.1.2 beschriebenen Substitutionen und Anhänge nach vordefinierten Regeln vorgenommen.
Ziel der vorliegenden Arbeit ist es, automatisiert personenbezogene Wortlisten und Regeln zu erstellen, um diese anschließend im Schritt des Passwortcrackings von einem Programm wie *hashcat* oder *John the Ripper* zu personenbezogenen Passwortlisten kombinieren zu lassen. Als Quelle für die benötigten personenbezogenen Informationen werden beim im Zuge dieser Arbeit entwickelten Tool (siehe Kapitel 4) die Profile von Zielpersonen in sozialen Netzwerken herangezogen.

2.1.5 Bewertung der Passwortstrategien

Im Folgenden wird versucht, die *Shannon Entropie* aus Kapitel 2.1.3 für die in Kapitel 2.1.2 erwähnten Strategien zu berechnen, um festzustellen, welche Herangehensweisen bei der Erstellung von Passwörtern sicher sind und welche vermieden werden sollten. Die zwei Parameter, die für die Berechnung der *Shannon Entropie* von Bedeutung sind, ist einerseits die Größe des Alphabets und andererseits die Anzahl der verwendeten Symbole aus diesem. Als *Alphabet* wird in der theoretischen Informatik eine endliche Menge an Symbolen bezeichnet, die wiederum selbst einzelne Zeichen aber auch Wörter sein können.
Um einen Vergleich der Strategien zu ermöglichen, wird in dieser Bewertung von Passwörtern ausgegangen, die eine Länge von mehr als 10 einzelnen Zeichen besitzen, womit angenommen werden kann, dass diese gegen Brute-Force Attacken resistent sind.

Standardwörter Wird davon ausgegangen, dass nur deutsche Wörter etwa aus der Liste von Benjamin Tenne [30] mit 66.666 Einträgen ausgewählt werden, so besitzt ein einzelnes Wort, wie in Gleichung 2.2 ersichtlich ist, eine sehr niedrige Entropie von 16 Bit.

$$H = log_2(66666^1) = 16,0247 \tag{2.2}$$

Lässt man noch die Option zu, den ersten Buchstaben groß oder klein schreiben zu können, so erhöht sich damit die Entropie um 1 Bit auf 17 Bit.
Bemerkenswert ist, dass die Verwendung von 4 zufällig gewählten Standardwörtern hintereinander bereits zu einer Erhöhung der Entropie auf etwa 64 Bit führt (siehe Gleichung 2.3) und gleichzeitig auch eine hohe Merkbarkeit besitzt. Werden hier ebenfalls Groß- und Kleinschreibung beachtet, so erhöht sich die Entropie pro Wort um 1 Bit auf 68 Bit.

$$H = log_2(66666^4) = 64,0987 \tag{2.3}$$

Standardwörter mit Substitution und/oder Anhang Bei dieser speziellen Form von Standardwörtern wird die Entropie eines einzelnen Wortes von den bestehenden 17 Bit erweitert. Bei angenommenen 3 Substitutionen eines Zeichens im Wort fügt jede 1 Bit an Entropie hinzu. Durch das Hinzufügen eines Zeichens aus der Menge O, die, wie in Tabelle 2.1 ersichtlich ist, aus 32 Zeichen besteht, werden erneut $log_2(32) = 5$ Bit erzeugt. Dies führt bei einem Wort wie *Br4tf!5sh?* zu einer Entropie von 25 Bit.

Mnemonik In der Arbeit von Kuo et al. [19] wurde eine Sammlung von 400.000 Sätzen gebildet, die für mnemonische Passwörter herangezogen werden können. Ein aus so einem Satz gebildetes Passwort besitzt demnach eine Entropie von etwa 19 Bit (Gleichung 2.4), die mit einer Groß- beziehungsweise Kleinschreibung für jedes der durchschnittlichen 8 Wörter und 3 Substitutionen auf etwa 30 Bit erhöht werden kann.

$$H = log_2(400000) = 18,6096 \tag{2.4}$$

Zufallspasswörter Die Entropie von Zufallspasswörtern, die aus 10 zufällig ausgewählten Zeichen der 94 Zeichen großen Mengen N, L, U und O bestehen, ist durch die Formel in Gleichung 2.1 berechenbar. In Gleichung 2.5 wurden die Parameter in diese eingesetzt und eine Entropie von etwa 66 berechnet.

$$H = log_2(94^{10}) = 65,54549 \tag{2.5}$$

Die Sicherheit von 10 zufällig gewählten Zeichen, deren Merkbarkeit für den Menschen eine große Herausforderung darstellt, ist demnach nur wenig größer als 4 zufällig ausgewählte deutsche Wörter.

Verwendung von Emojis In der Unicode Version 11.0 werden 2.784 verschiedene Emojis unterstützt. Eine Verwendung von 10 Emojis führt daher zu einer sehr hohen Entropie von 114 (Gleichung 2.6), wobei in der Arbeit von Golla et al. [23] gezeigt werden konnte, dass in der Praxis nur ein kleiner Bruchteil dieser 2.784 Emojis tatsächlich verwendet wird, weshalb von einer wesentlich kleineren Entropie ausgegangen werden kann. So wurde gezeigt, dass Emojis, die mit positiven Gefühlen verbunden werden, wie Herze oder lachende Gesichter, häufiger gewählt werden, weshalb ein Angreifer diese zuerst probieren wird.

$$H = log_2(2784^{10}) = 114,4294 \tag{2.6}$$

Wird ein Emoji als Anhang an ein Standardwort gewählt, erhöht es die Entropie um 11 Bit, was deutlich höher ist, als ein Zeichen der Menge O, welches nur 5 Bit erreicht.

Kryptonizer Der *Kryptonizer* ist in seiner Sicherheit wie ein Standardwort zu bewerten, da in der Anleitung dazu ermutigt wird, ein solches zu wählen, um es durch den Algorithmus des *Kryptonizers* zu bearbeiten. Die zusätzliche Sicherheit, die hierbei propagiert wird, liegt nur in der Geheimhaltung der Plastikkarte selbst, wobei nicht davon ausgegangen werden kann, dass diese stets gewährleistet ist. Weiters ist dieser Aspekt der Methode nicht in Entropie quantifizierbar.

Fazit Zusammenfassend kann festgehalten werden, dass die Sicherheit der Methoden *Standardwörter* (ein Wort), *Kryptonizer* und *Standardwörter mit Substitution und/oder Anhang* unzulänglich sind. Die Wahl zufälliger Emojis als Passwort wirkt durch die hohe Entropie von 114 vielversprechend, wobei anzunehmen ist, dass eher leicht merkbare und oft gewählte Emojis herangezogen werden, weshalb diese Methode ebenfalls nicht über genug Entropie verfügt. Die Methode der Mnemonik, die einen besonders guten Ruf hinsichtlich der Sicherheit genießt, verfügt mit etwa 30 Bit Entropie über nicht signifikant erhöhte Sicherheit gegenüber den bereits erwähnten Methoden. Dabei wird jedoch davon ausgegangen, dass einem Angreifer die Liste möglicher Basissätze bekannt ist.

Zufallspasswörter und die Verwendung von mindestens 4 Standardwörtern stechen mit einer besonders hohen Entropie hervor. Während Zufallspasswörter dabei nur 10 Zeichen benötigen, um sicher zu sein, ist anzunehmen, dass 4 Standardwörter bereits mindestens doppelt so lange sind. Dies schränkt die Benutzbarkeit aufgrund des hohen Aufwandes bei der Eingabe und der hohen Fehleranfälligkeit ein. Die Merkbarkeit ist jedoch im Vergleich zu Zufallspasswörtern gegeben und macht diese Methode so zu einer guten Alternative.

2.2 Akquise von personenbezogenen Informationen

In diesem Kapitel folgt die Beschreibung verschiedener Methoden, um personenbezogene Informationen über eine bestimmte Person aus dem Internet zu erhalten. Dabei können einerseits soziale Netzwerke als Datenquelle herangezogen werden oder alternativ der Name der Person in Suchmaschinen eingegeben werden. Während die erste Methode den Vorteil bietet, dass es sich garantiert um die gewünschte Person handelt und Namensgleichheiten damit kein Problem darstellen, ist andererseits das Wissen über den Benutzernamen beziehungsweise die eindeutige Nutzer-ID notwendig. Ist dieses nicht vorhanden und auch nicht manuell eruierbar - etwa durch Abgleich der Profilfotos mit vorhandenem Bildmaterial über die Zielperson -, bleibt nur der Weg über Suchmaschinen, die einen breiteren Ansatz wählen. Dabei werden nicht nur soziale Netzwerke als Quellen herangezogen, die bereits im automatisierten Akquisetool berücksichtigt sind, sondern auch Webseiten anderer Natur, wie etwa der persönliche Blog der Zielperson.

2.2.1 Soziale Netzwerke

Twitter Das soziale Netzwerk Twitter besteht seit März 2006 und ist für seine Kurznachrichten - *Tweets* genannt - bekannt. Ein Tweet besteht aus maximal 280 Zeichen (vor September 2017 140 Zeichen) und kann zusätzlich vier Bilder, ein GIF oder ein Video beinhalten [33, 34]. Die Accounts neuer Nutzer des sozialen Netzwerks befinden sich automatisch in der Einstellung, in der alle Tweets öffentlich und daher sichtbar für jeden sind. Diese Postings können gelesen werden, unabhängig davon, ob man selbst einen Account bei Twitter besitzt oder nicht. In den Einstellungen ist es möglich, *Geschützte Tweets* zu aktivieren. Diese sind nur von anderen Nutzern sichtbar, die eine Anfrage zur Erlaubnis der Einsicht vom Besitzer des Accounts bestätigt bekommen haben und in weiterer Folge als *Follower* bezeichnet werden [35]. Wenngleich über den Anteil der *Geschützten Tweets* seitens Twitter keine offiziellen Statistiken bekannt gegeben werden, ist anzunehmen, dass dieser aufgrund des typischen Nutzerverhaltens

bedingt durch die Struktur des Netzwerkes sehr gering ist: Twitter fungiert als Kurznachrichtendienst, wobei ein Kernkonzept sogenannte *Retweets* also Antworten auf Tweets darstellen. Diese ermöglichen eine Diskussion mit fremden Personen, zwischen denen nur über die jeweilige Thematik ein Zusammenhang besteht. Anders als etwa bei Facebook, ist daher anzunehmen, dass Diskussionsgruppen auf Twitter seltener reale Bekanntschaften abbilden und eher spontan zu einem Schlagwort (*Hashtag*) entstehen. Um dies zu ermöglichen, ist es im Interesse des Nutzers oder der Nutzerin, Tweets zu veröffentlichen und nicht auf einen kleinen Personenkreis zu beschränken. Diese These wird auch durch privat durchgeführte Analysen bestätigt, die auf einen sehr geringen Anteil an *geschützten Tweets* von etwa 2 bis 10% kommen [36, 37]. Für eine erleichterte automatisierte Interaktion mit dem Netzwerk stellt Twitter Entwicklern eine Programmierschnittstelle (*Application Programming Interface* kurz API) zur Verfügung. Damit ist es möglich, Tweets zu veröffentlichen, zu lesen oder andere Funktionen wie Werbungen oder Direktnachrichten zu nutzen, ohne die graphische Benutzeroberfläche von Twitter verwenden zu müssen. Für den Zweck der Akquise von Informationen ist ausschließlich die Funktion der API von Interesse, mit der Tweets einer Person gelesen werden können. In der aktuellen Version der API besteht ein dreistufiges Privilegienmodell mit den Kategorien *Standard*, *Premium* und *Enterprise*. Während *Standard* frei zugänglich ist und nur die Tweets der letzten sieben Tage zugänglich macht, sind die höheren Stufen entgeltlich und bieten ein größeres Archiv an. Das Ziel, alle Tweets einer Person auszulesen, kann allerdings nur mit der höchsten Stufe *Enterprise* erreicht werden, welche Zugriff auf alle Tweets seit der Entstehung der Plattform 2006 ermöglicht. Diese Stufe muss individuell von einer Firma beantragt werden, die Bewilligung liegt im Ermessen von Twitter und bei gewährtem Zugriff wird der Preis an die Bedürfnisse des Kunden angepasst. [38]

Für das Tool, das im Zuge dieser Arbeit entstanden ist, ist der Zugriff auf Tweets durch die Programmierschnittstelle aus den oben angeführten Restriktionen nicht praktikabel. Alternativ wurde ein Ansatz gewählt, bei dem der Zugriff auf die graphische Benutzeroberfläche von Twitter simuliert wurde und die Tweets so extrahiert wurden. Dieser Prozess wird in Kapitel 4.3.2 genauer erläutert.

Facebook Das soziale Netzwerk Facebook wurde im Februar 2004 [39] gegründet und besitzt derzeit über 2 Milliarden Nutzerinnen und Nutzer. Auf Facebook können - wie auch bei Twitter - Fotos, Videos, Links und auch Texte geteilt werden, wobei letztere in ihrer Länge nicht beschränkt sind. Die Plattform legt ihren Schwerpunkt auf das Vernetzen von Freunden und Bekannten, wobei Inhalte auch öffentlich zugänglich gemacht werden können.

Bei jedem neuen Posting kann die Benutzerin oder der Benutzer entscheiden, mit welcher Personengruppe dieses geteilt werden soll. Dabei kann die Reichweite auf bestimmte Gruppen, Freunde oder Freunden von Freunden begrenzt werden. Außerdem ist es möglich, Inhalte als *öffentlich* zu markieren, sodass jeder andere Nutzer von Facebook, das Posting sehen kann. Eine weitere Einstellungsmöglichkeit erlaubt es, den Inhalt nur sich selbst zugänglich zu machen, womit ihn keine andere Person sehen kann, während dieser trotzdem gespeichert wird und jederzeit für eine größere Personengruppe sichtbar gemacht werden kann. [40]
Alternativ zu diesen privaten Profilen existieren sogenannte *Facebook Seiten*, die die Repräsentation von berühmten Persönlichkeiten und Firmen darstellen. Die Inhalte dieser Seiten sollen ähnlich wie bei Twitter öffentlich zugänglich sein und können auch ohne Facebook Account eingesehen werden.
Facebook bietet zur automatisierten Interaktion mit Inhalten eine API mit dem Namen *Graph API* an. Diese Schnittstelle kann auf den sogenannten *Social Graph* zugreifen, die interne Bezeichnung Facebooks für das Netzwerk aus Profilen und Postings. Zugriffsschlüssel können kostenlos von jeder Person erstellt werden, die einen Facebook-Account besitzt. Zu diesem Zweck muss eine Facebook-App erstellt werden. Danach sind über die API öffentliche Inhalte von *Facebook Seiten* abrufbar. Zusätzlich dazu sind auch Inhalte von privaten Profilen zugänglich, sofern der App von der Benutzerin oder dem Benutzer die notwendigen Zugriffsrechte erteilt wurden. [41]

2.2.2 Suchmaschinen

Mit Hilfe von Suchmaschinen kann bei der Akquise von personenbezogenen Informationen ein alternativer Weg gegangen werden. Durch eine allgemeine Suche wird ein breiteres Ergebnis gewonnen als durch das gezielte Sammeln von Postings in sozialen Netzwerken möglich ist. So werden auch Seiten einbezogen, die in der automatisierten Akquise des Tools nicht berücksichtigt werden können aber besonders wertvolle Informationen beinhalten, wie etwa die persönliche Webseite oder den Blog der Zielperson. So können Seiten oder soziale Netzwerke durchsucht werden, für die noch kein spezialisierter Scraper implementiert wurde. Während eine manuelle Suche nach relevanten Wörtern hier möglich wenn auch mühselig ist, kann bei einer generischen automatisierten Suche auf zusätzlichen sozialen Netzwerken aufgrund des unterschiedlichen Aufbaus der Seiten nicht garantiert werden, dass relevante Informationen extrahiert und alle Postings berücksichtigt werden. Zusätzlich hängt die Qualität der gewonnenen Informationen von der Häufigkeit des Namens der Zielperson ab, da bei besonders typischen Namen die

Gefahr besteht, Webseiten über andere Personen mit dem gleichen Namen einzubeziehen.

3 State of the Art

Zum Thema sichere Passwörter existiert bereits eine breite Forschungsbasis mit Verwendung unterschiedlicher wissenschaftlicher Methoden, die sich jedoch - wie sich in Kapitel 2 gezeigt hat - teilweise in ihren Ergebnissen widersprechen. Die Kombination automatisierter Suche nach personenbezogenen Informationen im Internet mit dem Ziel der Generierung von Passwortlisten stellt jedoch ein weitgehend unbeachtetes Forschungsgebiet dar.
In diesem Kapitel werden Forschungsarbeiten vorgestellt, die ähnliche Herangehensweisen und Themengebiete abdecken wie die vorliegende Arbeit. Anschließend gilt es die Ergebnisse dieser Arbeiten zu vergleichen und Unterschiede festzustellen. Außerdem werden bestehende Tools beleuchtet, die zur Erstellung von Wortlisten sowie zum *Cracken* von Passwörtern verwendet werden können.

3.1 Verwandte Arbeiten

In der Arbeit *Augmenting password recovery with online profiling* von Al-Wehaibi et al. [42] wurde versucht, den Inhalt von 17 Dateien wiederherzustellen, die von verschiedenen Universitätsangehörigen verschlüsselt wurden. Als Basis für den Angriff wurde automatisiert ein personalisiertes Wörterbuch für jede Versuchsperson erstellt, wobei der jeweilige universitäre Webauftritt der Person als Ausgangspunkt herangezogen wurde. Der Crawler folgte anschließend Links in maximal drei Ebenen und extrahierte alle gefundenen Wörter, die anschließend von Stoppwörtern bereinigt wurden. Die verbliebenen Wörter wurden abschließend, ähnlich wie in der vorliegenden Arbeit, auf ihre Häufigkeit im englischen Sprachgebrauch untersucht. Dabei wurden nur jene Wörter als relevant betrachtet, die nicht im englischen Korpus vorkommen oder deren Häufigkeit im Text die Häufigkeit im Sprachgebrauch übersteigt. Die Wörter wurden anschließend ohne weitere Gewichtung dem *Passwort Recovery Tool Kit* (PRTK) der Firma AccessData (siehe Kapitel 3.2.3) übergeben, das den Prozess des Passwortcrackings übernahm. Mit dieser Methode konnten 4 der 17 Dateien wiederhergestellt werden.
Im Vergleich zur vorliegenden Arbeit kann betont werden, dass in der Studie von Al-Wehaibi et

al. mit dem universitären Webauftritt nur der wissenschaftliche Aspekt einer Person abgedeckt wird und Begriffe aus dem Privatleben kaum Einfluss finden werden. Die Wahl von sozialen Netzwerken als Quelle für die Datenakquise kann als umfassender betrachtet werden. Weiters ist zu erwähnen, dass der ausgewählte Personenkreis der Studie durch die hohe Technikaffinität nicht repräsentativ für die Bevölkerung ist. In der Evaluierung der vorliegenden Arbeit in Kapitel 5 wurde daher versucht, eine gleichmäßige Verteilung der Faktoren *Alter*, *Geschlecht*, *Hobbys* und *Beziehungsstand* zu erreichen.

Eine andere Herangehensweise, wie in der Forensik Passwörter von Personen herausgefunden werden können, wird in der Arbeit von Fragkos et al. [43] präsentiert. Dabei wurde die psychologische Seite dieses Themas erforscht, um festzustellen, nach welchen Methoden Personen typischerweise Passwörter erzeugen und ob es möglich ist, diesen Prozess mit Wissen über die Person nachzuahmen. Dabei wurde erkannt, dass bei der Wahl des Passwortes viele psychologische Faktoren eine Rolle spielen. Weiters ist aber auch ausschlaggebend, wofür dieses später verwendet werden soll. So wird an Accounts für den Finanz- oder Gesundheitssektor ein höherer Sicherheitsanspruch gestellt als für anonyme Services oder soziale Netzwerke. Die Arbeit kommt zu dem Ergebnis, dass diese Umkehr der Logik der Passworterstellung eine gute Alternative zu Brute-Force Attacken darstellen kann und vor jeder anderen Methode versucht werden sollte. Ein direkter qualitativer Vergleich mit der vorliegenden Arbeit war aufgrund der fehlenden Evaluierung nicht möglich. Eine Kombination der Methoden kann jedoch in Erwägung gezogen werden.

3.2 Bestehende Tools

Im Internet besteht bereits eine große Auswahl an Software, die den Prozess des Passwortcrackings erleichtern soll. Diese Tools können in drei Kategorien unterteilt werden: Webscraper zur Erstellung von Wortlisten, sowie Tools zur Erstellung von Passwörtern und Passwortcracker. Diese drei Kategorien bilden gleichzeitig auch jene Schritte, nach denen beim Prozess des Passwortcrackings vorgegangen werden kann. Die vorgestellten Tools besitzen teilweise Fähigkeiten aus mehr als einer Kategorie wobei sie stets in jenen Bereich eingeteilt werden, in dem ihre Spezialisierung liegt.

3.2.1 Webscraper zur Erstellung von Wortlisten

Die erste Gruppe besteht aus Webscrapern, deren Ziel es ist, gezielt auf Webseiten nach Texten zu suchen, um diese anschließend zu verarbeiten und eine Wortliste zu generieren.

CeWL CeWL (gesprochen wie das englische Wort *cool*) ist ein Tool zur Erstellung von Wortlisten auf Basis von Webseiten. Es wurde vom Betreiber der Seite DigiNinja [44] Robin Wood in der Programmiersprache Ruby geschrieben und auf GitHub veröffentlicht [45].
Grundsätzlich kann CeWL nach Angabe einer Webseite diese nach Text durchsuchen und die gefundenen Wörter in einer Liste speichern. Zu den Besonderheiten des Tools zählen die vielen Einstellungsmöglichkeiten, um das Ergebnis zu verfeinern. So kann etwa die Suchtiefe eingestellt werden, die angibt, wie oft der Scraper internen (oder wenn dies gewünscht ist auch externen) Links folgen soll. Außerdem ist es möglich, E-Mail-Adressen separat zu speichern oder nach Angabe von Zugangsdaten ein automatisches Login auf der Zielseite durchzuführen.

SmeegeScrape *SmeegeScrape* [46] vom Blog *SmeegeSec* [47] bietet eine Alternative zu CeWL, die in Python geschrieben wurde. Das Tool kann ebenfalls Webseiten nach Texten durchsuchen und daraus Wörter extrahieren. Der Funktionsumfang ist jedoch im Vergleich zum Konkurrenzprodukt deutlich eingeschränkt. So ist es nicht möglich, Links zu folgen oder Einschränkungen bezüglich der Wortlänge oder Kategorie vorzunehmen. Als zusätzliches Feature kann das Tool allerdings auch lokale Textdateien durchsuchen. Diese Funktion bietet CeWL vergleichsweise nicht an.
Das im Jahr 2014 veröffentlichte Tool beruht auf der veralteten Version 2.0.4 der Sprachverarbeitungsbibliothek NLTK. Diese wurde in der neuesten Version auch in der vorliegenden Arbeit verwendet (siehe Kapitel 4.2.3).

3.2.2 Tools zur Erstellung von Passwörtern

Im Folgenden werden Tools vorgestellt, mit deren Hilfe es möglich ist, Passwortlisten nach angegebenen Kriterien zu erstellen.

Crunch *Crunch* [48] ist ein Tool, das in der Linux Distribution *Kali* [49], die vorwiegend für Penetration Tests konzipiert wurde, bereits vorinstalliert ist. Laut den Angaben der Entwickler verwendet *Crunch* Permutations- und Kombinationstechniken, um nach angegebenen Kriterien Passwortlisten zu erstellen. Dabei ist es stets notwendig, eine Minimal- und eine Maximallänge für die zu generierenden Wörter zu definieren. Weiters wird empfohlen, ein Muster anzugeben,

sofern bereits Informationen über das zu erratende Passwort vorliegen. Ein Muster besteht aus sogenannten *Wildcards*, die für Zeichen der Kategorien Klein-, Großbuchstabe, Ziffer oder Symbol stehen können. Ist zum Beispiel bekannt, dass das Passwort aus 4 Kleinbuchstaben und der Jahreszahl *1995* besteht, kann dies mit dem Muster @@@@*1995* ausgedrückt werden. Anstelle der @-Zeichen werden nun alle möglichen Kombination von Kleinbuchstaben gesetzt und die so erstellte Liste ausgegeben.
Eine Stärke von *Crunch* zeichnet sich dadurch aus, dass vor der Erstellung der Passwortliste angezeigt wird, wie viel Speicherplatz die resultierende Liste benötigen wird. Da bei vielen Permutationen eine enorme Größe erreicht werden kann, besteht die Möglichkeit, die Liste auf mehrere Dateien aufzuteilen.

CUPP *CUPP* [50] wurde ursprünglich von Muris Kurgas entwickelt und ist darauf spezialisiert, den Fakt auszunutzen, dass viele Menschen persönliche Informationen in ihre Passwörter integrieren. *CUPP* kann anhand eines ausgefüllten Fragebogens über die Zielperson, der Elemente wie Spitznamen oder Namen und Geburtstage von Kindern und Partner beinhaltet, Passwörter generieren. Außerdem ist es möglich, bestehende Wortlisten aus dem Internet bei der Generierung einzubeziehen. Als Beispiel wird die fiktive Person *Bernhard Müller* genommen, von der das Geburtsdatum (23. Mai 1989), der Spitzname *Müli* und der Name seiner Frau *Konstanze* beziehungsweise seiner Tochter *Anna* (Geburtstag: 09. Oktober 2012) bekannt ist. Außerdem ist die Zielperson großer Fan des Fußballklubs *Rapid Wien* und beschäftigt sich in seiner Freizeit mit dem Anpflanzen von Chilis. Auflistung 1 zeigt Teile des Ablaufes der Fragebogenausfüllung in der Kommandozeile. Das Passwort *Ch!l!2012*, das aus seinem Hobby und dem Geburtsjahr seiner Tochter zusammengesetzt ist und auch Substitutionen (siehe Kapitel 2.1.2) beinhaltet, konnte nach Erstellung einer 23.774 Einträge langen Passwortliste in dieser gefunden werden.
Wenngleich das Tool keine komplexeren Kombinationen der Wörter untereinander durchführt, ist dieser erste Schritt dennoch vielversprechend und macht *CUPP* zu einem wertvollen Hilfsmittel.

```
PC:cupp user$ ./cupp3.py -i

[...]

First Name: Bernhard
Surname: Müller
Nickname: Müli
Birthdate (DDMMYYYY): 23051989

Partner's name: Konstanze
Partner's nickname:
Partner's birthdate (DDMMYYYY):

Child's name: Anna
Child's nickname:
Child's birthdate (DDMMYYYY): 09102012

[...]

Do you want to add some key words about the victim? Y/[N]: Y
Please enter the words, comma-separated. [i.e. hacker,juice,black],
spaces will be removed: rapid,chili
Do you want to add special characters at the end of words? Y/[N]: Y
Do you want to add some random numbers at the end of words? Y/[N]: Y
Leet mode? (i.e. leet = 1337) Y/[N]: Y

[+] Now making a dictionary...
[+] Sorting list and removing duplicates...
[+] Saving dictionary to bernhard.txt, counting 23774 words.
[+] Now load your pistolero with bernhard.txt and shoot! Good luck!
```

Auflistung 1: Fragebogen von CUPP (Ausschnitt)

3.2.3 Passwortcracker

Die folgenden Tools sind die verbreitetsten Vertreter sogenannter Passwortcracker, also Programmen, deren Ziel es ist, den Ausgangsbegriff (das Passwort) eines vorhanden Hashes mit Hilfe gezielter oder geratener Versuche zu finden.

Cain & Abel *Cain & Abel* [51] ist ein Programm zur Wiederherstellung von Passwörtern für Windows Betriebssysteme. Das Tool verfügt über verschiedenste Features wie Wörterbuch Attacken, Brute-Force Attacken, Network-Sniffing, Wiederherstellung von gecachten Passwörtern oder WLAN Kennwörtern. In der neuesten Version, die 2014 veröffentlicht wurde, sind auch Man-in-the-Middle (MITM) Attacken wie ARP-Poisoning durchführbar.

PRTK Die Software *Password Recovery Tool Kit* (PRTK) [52] wurde von der Firma Access-Data entwickelt und stellt ein Hilfsmittel für die IT-Forensik dar. Die aktuelle Version 7.6.0 wurde im Juni 2014 für Windows Betriebssysteme veröffentlicht.
Das Tool verfügt über die Möglichkeit, Textdateien als Input für die Erstellung von Wörterbüchern heranzuziehen. Diese können anschließend nach bestimmten Regeln für einen Angriff auf verschlüsselte Dateien verwendet werden. Weiters können Passwörter aus Windows-spezifischen Systemdateien wie *Windows Registries* oder *SAM Files* extrahiert werden.

John the Ripper *John the Ripper* [53] ist eines der am weitesten verbreiteten Programme für Passwortcracking und für Windows, Linux und macOS Betriebssysteme verfügbar. Mit Hilfe der hauptsächlich in C geschriebenen Software lassen sich aus Passwortfiles, die in der Regel aus Einträgen mit einem Usernamen und einem gehashten Passwort bestehen, die dazugehörigen Passwörter extrahieren, wobei dabei nach verschiedenen Modi vorgegangen werden kann.
Im *Single Crack* Modus wird der Benutzername verwendet, um diesen in verschiedenen verarbeiteten Formen als Passwort zu probieren. Dieser erste Modus zeichnet sich durch seine enorm schnelle Durchführbarkeit und eine verhältnismäßig hohe Erfolgschance aus.
Im zweiten Modus, dem *Wordlist* Modus, muss eine Liste mit Wörtern angegeben werden, deren Einträge nach bestimmten (veränderbaren) Verarbeitungsregeln modifiziert als Passwort probiert werden.
Der dritte Modus bildet die letzte und zeitintensivste Möglichkeit, um das korrekte Passwort zu erraten und wird daher bei fehlender Angabe des Modus zum Schluss herangezogen. Der sogenannte *Incremental* Modus führt eine Brute-Force Attacke aus, wobei angegeben werden kann, welches Zeichenset dafür herangezogen werden soll.

Hashcat Die Software *hashcat* [54] wird von den Entwicklern als der schnellste Passwortcracker bezeichnet und hebt sich durch ihre breite Unterstützung von Grafikkarten sowie seine Spezialisierung auf sogenannte *fast hashes* hervor, also Hashalgorithmen wie MD5, NTLM oder SHA2561, die besonders schnell berechenbar sind.
Für das Cracking von Passwörtern auf Basis von Wortlisten verwendet *hashcat rule files*, also Regeln, nach denen Begriffe in tatsächliche Passwörter transformiert werden sollen. Die Syntax dieser Regeln ist mit der von *John the Ripper* weitgehend identisch, womit diese Dateien für beide Programme verwendet werden können. Regeln, die für die vorliegende Arbeit angefertigt wurden, sind in Kapitel 4.5 angeführt.

3.2.4 Cracker für Remote Authentication Services

Neben Passwortcrackern, die ausschließlich lokal gespeicherte Passwörter erraten können, gibt es eine Reihe von Programmen, die auf ein Brute-Forcing von Authentifizierungsmaßnahmen über das Netzwerk spezialisiert sind. Zu diesen zählen Tools wie THC Hydra [55], Brutus [56] oder Medusa [57], auf die in dieser Arbeit jedoch nicht näher eingegangen wird.

4 Design

Die in Kapitel 2 erarbeiteten Erkenntnisse sollen in weiterer Folge in einem Tool angewandt werden. Aufgabe des Tools wird es sein, auf Basis grundlegender Informationen über eine Person, wie etwa dem realen Namen und Benutzernamen von sozialen Netzwerken wie Facebook oder Twitter, automatisiert Informationen zu sammeln und sie zu möglichen Passwörtern zu verarbeiten. Dieser Prozess ist dreiteilig und gliedert sich in Datenakquise, Datenverarbeitung und Passwortgenerierung.

Der Name des Tools *slughorn* ist einer Figur aus dem 2005 erschienenen Fantasyroman *Harry Potter and the Half-Blood Prince* der britischen Autorin Joanne K. Rowling nachempfunden. Professor Horace Slughorn ist dafür bekannt, vielversprechende Schülerinnen und Schüler um sich zu versammeln und möglichst viele Informationen über diese für sich zu gewinnen. Zusätzlich bildet der erste Teil des Begriffes *slughorn* mit der englischen Bezeichnung für *Schnecke* eine passende Analogie zur Eigenschaft des Kriechens, die auch das Tool im Sinne eines Webcrawlers aufweist. Das Logo des Tools zeigt, wie in Abbildung 4.1 ersichtlich, eine Schnecke mit einem Horn auf der Stirn.

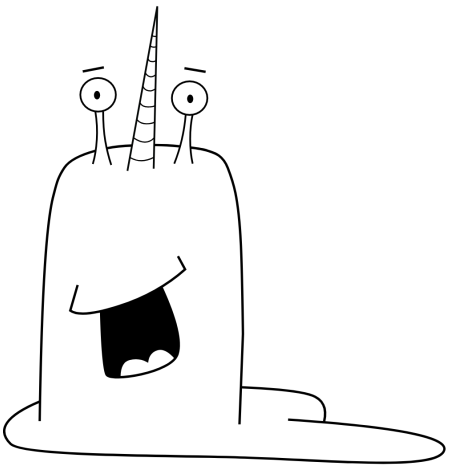


Abbildung 4.1: Logo des Scrapingtools *slughorn*

In diesem Kapitel werden zuerst die Ziele des Prototypen erläutert, bevor näher auf die Archi-

tektur des Programms eingegangen wird. Dabei wird der Aufbau der Module sowohl mit Hilfe von UML-Diagrammen als auch textuell veranschaulicht. Anschließend werden die drei Prozesse - Datenakquise, -verarbeitung und Passwortgenerierung - näher beleuchtet. Abschließend wird auf Herausforderungen eingegangen, die in der Entwicklung des Tools entstanden sind und darauf, wie diese Einfluss auf bestimmte Designentscheidungen genommen haben.

4.1 Prototyp

Das im Zuge dieser Arbeit entstandene Tool *slughorn* verbindet viele jener Erkenntnisse, die im Kapitel 2 gewonnen werden konnten und stellt einen ersten Prototyp eines forensischen Softwareprodukts dar. Die Verfahren, mit denen personenbezogene Informationen gesammelt und zu nach Wahrscheinlichkeit sortierten Wortlisten verarbeitet werden können, wurden in dieser Arbeit erstmalig kombiniert. Die Umsetzung dieser Konzepte zu einem Tool ist daher als *Proof of Concept* zu verstehen und bedarf weiterer Entwicklung, um ein einsatzfähiges Produkt zu erreichen.
Folgende Ziele wurden bei der Planung des Tools berücksichtigt und letztlich auch umgesetzt:

- Datenakquise
  - Postings in sozialen Netzwerken
    - Quelle 1: Facebook
    - Quelle 2: Twitter
- Datenverarbeitung
  - Spracherkennung
  - Textbereinigung
  - Lemmatisierung deutscher Wörter
  - Entfernung von Stoppwörtern
  - Kategorisierung von Begriffen
  - Häufigkeitsanalyse in Postings
  - Häufigkeitsanalyse im Sprachgebrauch
  - Behandlung von Fauxami (*Falschen Freunden*)
  - Berechnung eines Gesamtscores
- Passwortgenerierung
  - Export einer Wortliste

– Generierung einer Regeldatei für *hashcat* oder *John the Ripper*

Der Prototyp wurde in der Programmiersprache Python in der Version 3.6.4 geschrieben. Einschränkungen, die eine Umgestaltung bestimmter Teilaspekte der Software erforderten, sind in Kapitel 4.6 dargelegt.

4.2 Architektur

Das Scrapingtool *slughorn* ist primär über ein *Command Line Interface* (kurz CLI) steuerbar. Von diesem zentralen Eintrittspunkt aus kann ausgewählt werden, welche Quellen für den Scrapingprozess inkludiert werden sollen. Dabei ist für jedes soziale Netzwerk als Quelle der Benutzername der Zielperson anzugeben, den diese auf der Plattform besitzt. Abhängig von den gewählten Quellen werden anschließend Instanzen der entsprechenden Scraperklassen (zum Beispiel *FacebookScraper*) im Pythonpackage *scraper* initialisiert. Diese können wiederum Webdriverklassen (zum Beispiel *FacebookWebdriver*) oder Spiderklassen (zum Beispiel *TwitterSpider*) instanziieren, sofern dies für ihre Aufgabe notwendig ist. Die genaue Funktionsweise der Scraper ist in Kapitel 4.3 erläutert.
Nach erfolgter Akquise der Informationen, werden die extrahierten Postings sozialer Netzwerke einer Instanz der Klasse *ExpressionExtractor* im Pythonpackage *processor* übergeben. Hier wird der in Kapitel 4.4 beschriebene Verarbeitungsprozess in Gang gesetzt.
Abbildung 4.2 zeigt das UML-Klassendiagramm des Prototyps. Es listet die im Programm implementierten Klassen mit deren Attributen und Methoden sowie den Relationen untereinander auf.

4.2.1 CLI

Das Scrapingtool *slughorn* besitzt mit dem CLI einen zentralen Eintrittspunkt in das Programm, von dem ein Ablauf mit den gewünschten Optionen gestartet werden kann. Ein Ablauf ist ein sequentieller Prozess, der mit den übergebenen Parametern gestartet wird und dann selbstständig abläuft, ohne erneut auf eine Eingabe des Users zu warten. Eine Ausnahme bildet hier das erneute Starten eines bereits begonnenen Ablaufs, bei dem der User zu entscheiden hat, ob ein bereits gespeichertes Zwischenergebnis zur Fortsetzung verwendet werden soll (siehe Kapitel 4.2.2). Bei Fehlern wird der Ablauf unterbrochen und dem User eine Rückmeldung über das Problem gegeben. Bei erfolgreicher Extraktion und Verarbeitung ist eine Passwortliste das erzielte Ergebnis eines Ablaufes. Abbildung 4.3 zeigt ein UML-Sequenzdiagramm eines Ablaufs

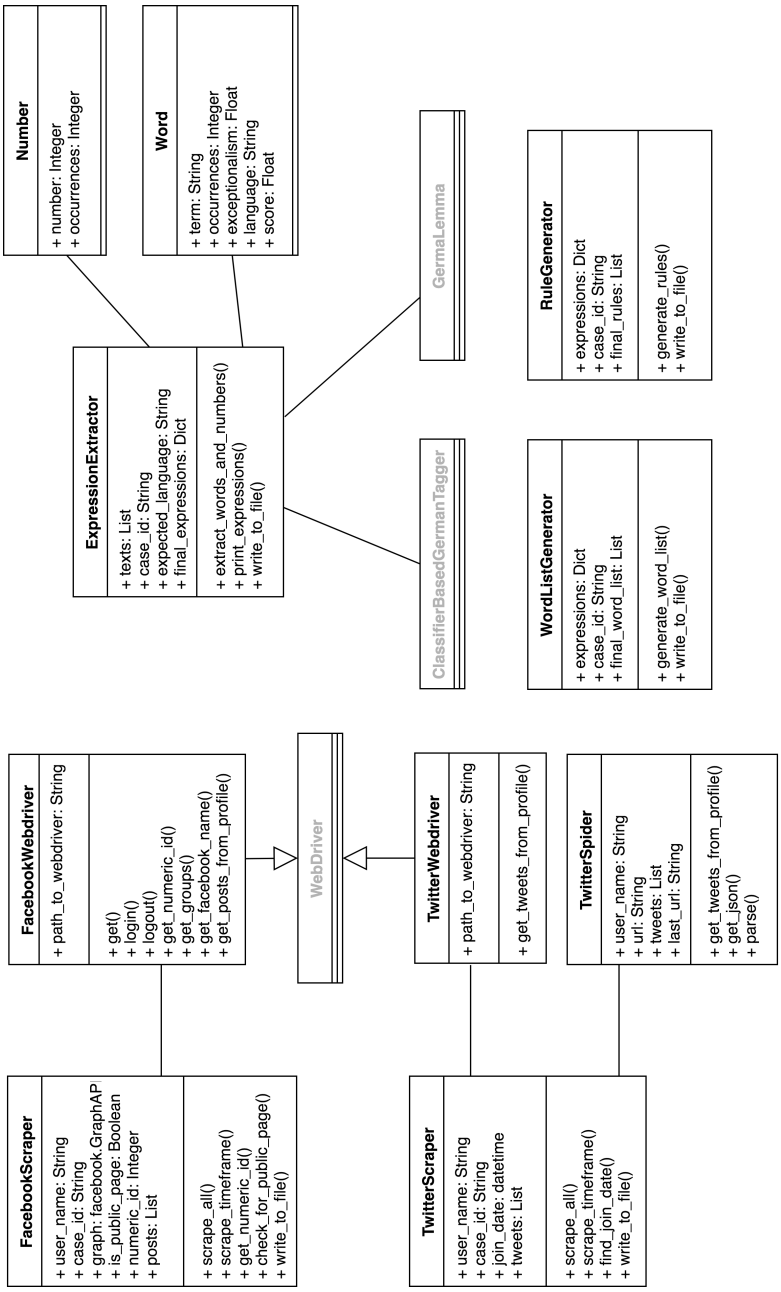


Abbildung 4.2: UML-Klassendiagramm

und den Informationsfluss durch das Programm. Angestoßen wird dieser Prozess stets durch den User über das CLI. Die einzelnen Schritte des Extraktionsprozesses werden an dieser Stelle zu einer Aktion zusammengefasst und detailliert in Kapitel 4.4 erläutert.

Das CLI ist mit dem Python Paket *Click* aufgebaut, welches auf einfache Art und Weise erlaubt, Command-Line Tools zu erstellen. Eine Methode wird dabei mit *@click.command()* annotiert

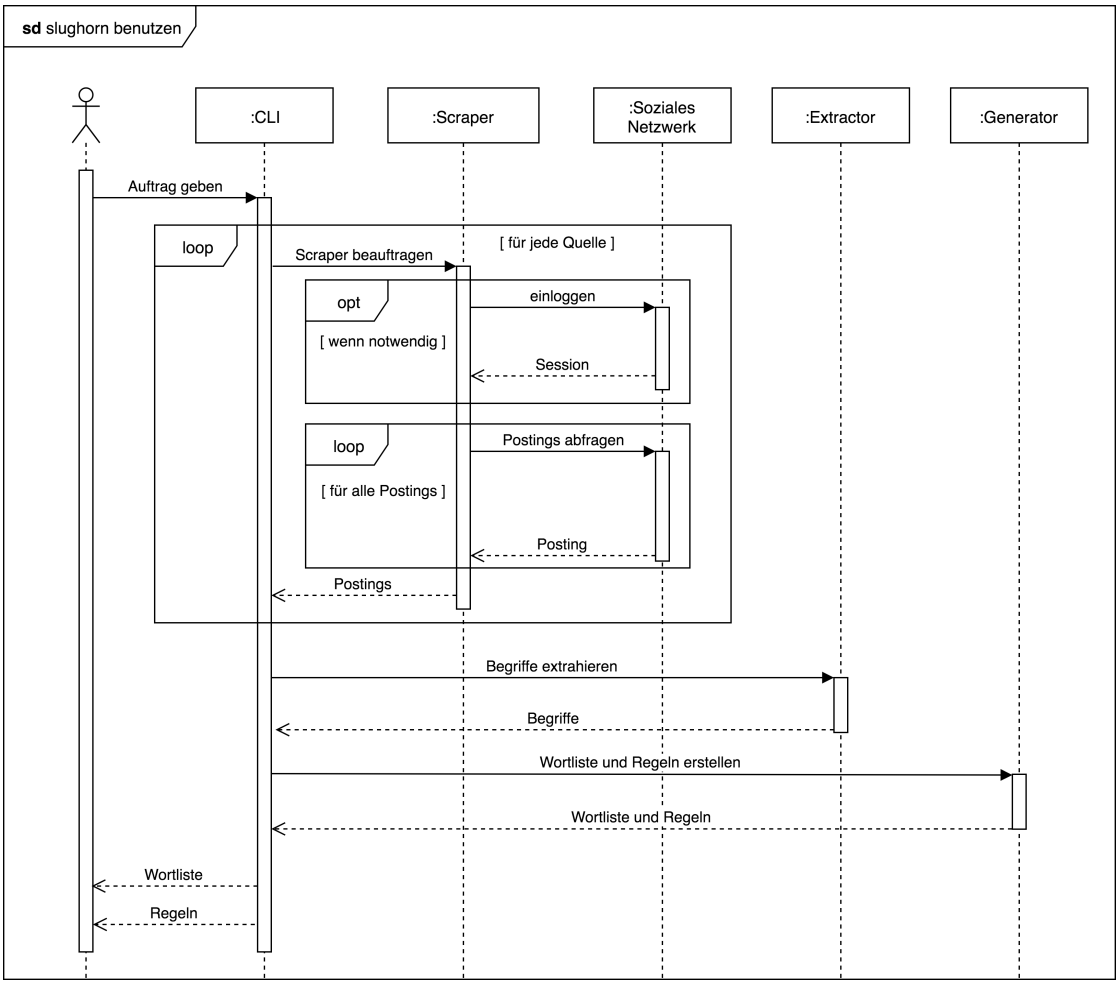


Abbildung 4.3: UML-Sequenzdiagramm

und besitzt sieben Optionen, über die das Programm gesteuert werden kann. Diese sind in Tabelle 4.1 aufgelistet. Während die Angabe der einzelnen Benutzernamen nicht verpflichtend ist, ist es doch nötig, zumindest einen Benutzernamen anzugegeben, da sonst kein Ablauf gestartet werden kann.

Option	Kurzform	Verpflichtend	Beschreibung
--case_id	-c	Ja	Die Fallbezeichnung wird für eine einheitliche Benennung der Ergebnisse verwendet
--facebook_username	-f	Nein	Der Benutzername der Zielperson auf Facebook
--twitter_username	-t	Nein	Der Benutzername der Zielperson auf Twitter
--language	-l	Nein	Die erwartete Sprache der Postings. Wird herangezogen, wenn Spracherkennung fehlschlägt
--output	-o	Nein	Der optionale Pfad, an dem die Ergebnisse gespeichert werden sollen
--weight	-w	Nein	Die Gewichtung für die Ausgefallenheit der Wörter zwischen 0.0 und 1.0 (Standard: 0.5)
--txt		Nein	Speichert das Zwischenergebnis der Begriffsextraktion als .txt statt .pkl, damit es einsehbar ist
--delete_constants		Nein	Löscht, falls vorhanden, die bestehende Constants-Datei

Tabelle 4.1: Optionen des CLI

Zusammen mit dem CLI wurde ein Python Setupskript (*setup.py*) erstellt, das eine einfache Installation des gesamten Tools ermöglicht und das CLI als Eintrittspunkt angibt. So ist nach erfolgter Installation ein Ausführen des Tools über die Kommandozeile möglich. Ein Befehl für einen exemplarischen Ablauf unter Verwendung aller Optionen ist in Auflistung 2 ersichtlich.

```
1 slughorn -c FALL_08 -f ola.n -t onordmann -o /pfad/zu/ordner/
```

Auflistung 2: Beispielbefehl mit Optionen

4.2.2 Zwischenspeicherung der Ergebnisse

Ein Ablauf setzt sich aus mehreren Subprozessen zusammen, die sequentiell abgearbeitet werden. Das Ergebnis jedes Teilprozesses wird in der Kette dem nächsten Subprozess als Input übergeben. Zusätzlich wird das Ergebnis auch zwischengespeichert und steht bei einer erneuten Durchführung zur Verfügung. Dieser Ablauf ist in Abbildung 4.4 dargestellt.

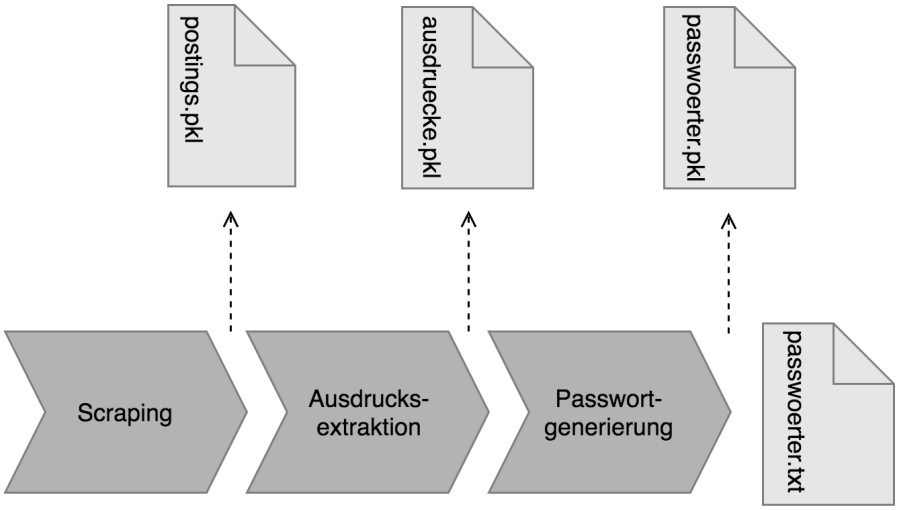


Abbildung 4.4: Abfolge der Subprozesse mit Zwischenspeicherungen

Der erste Subprozess ist das Scraping, das je nach Anzahl der beauftragten Quellen mehrmals durchgeführt werden kann. Die gesammelten Informationen werden für jede Quelle zwischengespeichert. Es folgt die Ausdrucksextraktion, bei der die Ergebnisse aller Quellen zusammengefasst und bearbeitet werden. Das Ergebnis dieses Schrittes wird ebenfalls gespeichert. Zuletzt werden aus den Ausdrücken Passwörter generiert und diese anschließend sowohl zwischengespeichert als auch als Endergebnis exportiert.

Eine Zwischenspeicherung zeichnet sich dadurch aus, dass ein Pythonobjekt mit den erarbeiteten Informationen als sogenanntes *pickle* gespeichert wird. *pickle* ist ein Pythonmodul, das es ermöglicht, Pythonobjekte zu serialisieren, also in einen Bytestream zu konvertieren und als Datei zu speichern. Dieser Prozess des *picklens* kann durch das *unpickling* wieder rückgängig gemacht werden, wodurch der gespeicherte Bytestream wieder in ein Pythonobjekt rückgeführt wird [58]. Zwischengespeicherte Ergebnisse liegen daher nicht in einer menschenlesbaren Form vor, können aber vom Programm erneut eingelesen und verwendet werden. Dadurch ist es möglich, nach einem Abbruch des Ablaufes rechenintensive Abschnitte zu überspringen und auf zwischengespeicherte Ergebnisse aufzubauen.

4.2.3 Externe Python Pakete

Für die Umsetzung des Scrapingtools *slughorn* ist eine Vielzahl von Funktionalitäten aus unterschiedlichen Themengebieten wie Spracherkennung, Webscraping oder Textverarbeitung von Nöten. Die selbstständige Umsetzung dieser Anforderungen würde den Rahmen dieser Arbeit sprengen, weshalb auf externe Python Pakete zurückgegriffen wird. Diese stellen zum Teil umfangreiche Softwareprojekte dar, die von einem großen Entwicklerteam betreut werden. Aufgrund von Lizenzen, die eine Verwendung und Weiterentwicklung (unter anderem nur für wissenschaftliche Zwecke) erlauben, ist es möglich, die hochwertige Arbeit und Expertise von Kolleginnen und Kollegen für dieses Tool zu verwenden.
Im Folgenden werden diese Python Pakete vorgestellt und ihr Zweck in der vorliegenden Arbeit erklärt. Diese Auflistung enthält nur Pakete, die nicht in der Python Standard Library [59] enthalten sind und über Paketmanager installiert werden müssen.

Click (Version 6.7) *Click* [60] ermöglicht es in wenigen Zeilen Code Command-Line Tools aufzubauen. Im Scrapingtool *slughorn* wird *Click* verwendet, um das CLI zu bilden, welches die zentrale Eintrittsstelle des Programms bildet (siehe Kapitel 4.2.1).

click-spinner (Version 0.1.7) *click-spinner* [61] ist eine optische Erweiterung für *Click* und ermöglicht es, eine Ladeanimation anzuzeigen. Dies signalisiert den Benutzern, dass das Tool arbeitet und verhindert den Abbruch des Prozesses.

Facebook SDK for Python (Version 2.0.0) *Facebook SDK for Python* [62] ist ein Paket, das es ermöglicht mittels Python mit der *Facebook Graph API* zu interagieren. Diese Schnittstelle wird von Facebook zur Verfügung gestellt, um Daten in den sogenannten *Social Graph* zu schreiben und davon zu lesen. Der *Social Graph* ist die Bezeichnung für das Netzwerk aus Profilen, Fotos, Postings und deren Beziehungen untereinander. [63]
In *slughorn* wird das Paket dazu verwendet, mit öffentlichen Profilen zu kommunizieren und Informationen von diesen zu erlangen. Informationen von privaten Profilen können hingegen auf diese Art nicht akquiriert werden (siehe Kapitel 4.6.2).

Beautiful Soup (Version 4.6.0) *Beautiful Soup* [64] ist ein Paket, mit dessen Hilfe Inhalte von Webseiten extrahiert werden können. Es umfasst zahlreiche Methoden, um schnell durch den Quelltext navigieren zu können und wird im vorliegenden Projekt verwendet, um das Beitrittsdatum zu Twitter von einem gezielten Profil zu finden. Da der von *Beautiful Soup*

abgerufene Quelltext der Webseite keine dynamisch nachgeladenen Inhalte, wie etwa durch *JavaScript*, enthält, ist diese Herangehensweise für das Extrahieren von Postings nicht geeignet (siehe Kapitel 4.6.2).

Requests (Version 2.18.4) *Requests* [65] ist eine HTTP Bibliothek für Python, die es ermöglicht, auf einfachem Weg HTTP-Requests abzusenden und die Ergebnisse zu verarbeiten. In der vorliegenden Arbeit wird das Paket im *TwitterScraper* verwendet um das Beitrittsdatum herauszufinden. Die Klasse *TwitterSpider* nutzt *Requests* um den HTTP-Request abzuschicken und die Antwort in Form einer JSON-Datei zu verarbeiten (siehe Kapitel 4.3.2).

fake_useragent (Version 0.1.10) Mit Hilfe von *fake-useragent* [66] wird der User-Agent Header für die HTTP-Requests in der Klasse *TwitterSpider* generiert, um einen normalen Abruf mittels Browser zu simulieren. Das Paket nutzt User-Agents der Seite *useragentstring.com* [67] und wählt jene nach Statistiken von *w3schools* [68] aus, um eine realitätsnahe Verteilung zu erreichen.

lxml (Version 4.1.1) *lxml* [69] ist eine Python Bibliothek, mit der eine einfache Suche und Bearbeitung von Elementen in XML-Dokumenten möglich ist. Im entwickelten Tool *slughorn* wird das Paket dazu verwendet, den Text von Tweets aus dem HTML-Bereich der heruntergeladenen JSON-Datei zu extrahieren (siehe Kapitel 4.3.2).

Selenium (Version 3.8.0) *Selenium* [70] ist eine Browserautomationssoftware, die vorwiegend für Integrationstests von Software verwendet wird. Sogenannte *WebDriver* werden dazu verwendet, einen Browser zu simulieren und Interaktionen mit Webseiten nachzuahmen. Dabei ist es auch möglich, *JavaScript* Code auszulösen und zu scrollen. Dies ist für die Umsetzung des *FacebookScrapers* von *slughorn* von elementarer Bedeutung.

fastText (Version 0.1.0) *fastText* [71] ist eine Textklassifizierungssoftware von Facebook, die unter anderem auch eine Spracherkennung ermöglicht. Ein bereits trainiertes Modell mit 176 Sprachen kann auf der Webseite des Pakets heruntergeladen werden und basiert auf zwei Forschungsarbeiten von Joulin et al. [72, 73].

pycountry (Version 17.9.23) Mithilfe des Pakets *pycountry* [74] ist es möglich, Länder, Sprachen und Währungen von standardisierten ISO Abkürzungen in ihre Langform umzuwandeln beziehungsweise diese in verschiedene Sprachen zu übersetzen. Das Ergebnis der Sprach-

detektion von *fastText* ist die zwei- beziehungsweise drei-stellige ISO 639 Abkürzung (zum Beispiel *en* für *Englisch*), welche in diesem Projekt von *pycountry* in den englischen Langnamen übersetzt wird, um von NLTK weiterverarbeitet werden zu können.

NLTK (Version 3.2.5) NLTK [75] ist eine Open-Source Bibliothek an Programmen und Textkorpora, die eine umfassende Arbeit mit der menschlichen Sprache erleichtern soll. Das Tool wird einerseits verwendet, um aus den extrahierten Postings Stoppwörter, also häufige und unwichtige Wörter wie Artikel oder Konjunktionen, zu entfernen. Andererseits wurde ein sogenannter *Classifier Based Tagger* für die Bibliothek NLTK trainiert, um in weiterer Folge eine Lemmatisierung von deutschen Wörtern zu ermöglichen. Dieses Verfahren ist in Kapitel 4.4.3 beschrieben.

ClassifierBasedGermanTagger Der *ClassifierBasedGermanTagger* [76] ist eine Implementierung eines deutschen *ClassifierBasedTagger* für die Sprachbibliothek NLTK von Philipp Nolte. Diese Klasse ist über keinen Paketmanager erhältlich und wird daher dem Projekt *slughorn* direkt unverändert hinzugefügt. Der Tagger wurde im Zuge dieser Arbeit mithilfe des TIGER-Corpus darauf trainiert, POS (*part-of-speech*) Bestandteile - wie etwa Artikel, Nomen oder Verben - von Sätzen zu erkennen. Die trainierte Instanz der Klasse ist in serialisierter Form (*pickle*) dem Projekt beigefügt und wird bei der Lemmatisierung verwendet (siehe Kapitel 4.4.3).

GermaLemma *GermaLemma* [77] ist ein Paket, das von Markus Konrad vom *WZB Berlin Social Science Center* entwickelt wurde. Es ermöglicht eine Lemmatisierung deutscher Wörter. Dabei wird ein dekliniertes Wort in seine Stammform rückgeführt, zum Beispiel wird so aus *Häusern* das Wort *Haus* oder aus *ging* der Infinitiv *gehen*. Für diese Funktionalität ist es zuerst notwendig, den korrekten POS-Tag für das Wort zu bestimmen, wozu ein trainierter Tagger notwendig ist. In diesem Projekt wird eine Instanz der Klasse *ClassifierBasedGermanTagger* verwendet (siehe 4.2.3).

pyphen (Version 0.9.4) *pyphen* [78] ist ein Paket, mit dem eine Silbentrennung bei Wörtern vollzogen werden kann. Diese Funktionalität ist eine Voraussetzung für die Lemmatisierung durch *GermaLemma* (siehe 4.2.3).

wordfreq (Version 1.6.1) *wordfreq* [79] von Robert Speer ermöglicht es, für Wörter anhand ihrer Sprache eine Häufigkeit im Sprachgebrauch zu bestimmen. Diese Funktionalität wird in der Extraktion der Begriffe dazu verwendet, herauszufinden, wie ausgefallen ein Begriff ist, um

diesen in weiterer Folge als einen wahrscheinlicheren Kandidaten für die Passworterstellung zu klassifizieren (siehe Kapitel 4.4.6).

4.3 Datenakquise

Das Herzstück der Datenakquise von *slughorn* ist der Scrapingprozess. Dabei wird für jede angegebene Quelle ein dedizierter Scraper initialisiert, der auf verschiedene Arten Informationen extrahiert. Die Art der Extraktion ist von den Eigenschaften, aber vor allem von den Privatsphäre-bedingten Einschränkungen der Quellen abhängig. So können über die offizielle Facebook Graph API nur Postings von öffentlichen Profilen ausgelesen werden. In diesem Fall ist es notwendig, einen Zugriff auf das Profil mittels Browser zu simulieren. Dies ist mithilfe eines *Selenium* WebDrivers möglich.

4.3.1 WebDriver

WebDriver ist ein Interface der Browserautomatisierungssoftware *Selenium*, die in Kapitel 4.2.3 bereits beschrieben wurde. Abgeleitet von dieser Schnittstelle bestehen diverse Implementierungen für gängige Browser, wie etwa Chrome, Firefox oder InternetExplorer, die jeweils einen Browser simulieren und über eine Programmierschnittstelle automatisiert Befehle und Aktionen ausführen können. So ist es möglich durch Webseiten zu navigieren, zu scrollen oder *JavaScript*-Funktionen auszulösen. Letzteres ist beim Scrapingprozess von Postings aus sozialen Netzwerken besonders wichtig, da etwa Tweets dynamisch mittels *JavaScript* nachgeladen werden, sobald der User das Ende der Seite erreicht hat. Nach einer Verbesserung des *TwitterScrapers* konnte auf die Verwendung eine WebDrivers verzichtet werden (siehe Kapitel 4.3.2). Im entwickelten Tool *slughorn* benötigt daher ausschließlich der *FacebookScraper* einen WebDriver, den *FacebookWebdriver*. Dies ist durch die Notwendigkeit eines Logins bei Facebook bedingt, um Postings extrahieren zu können. Alle alternativ evaluierten Methoden führten nicht zur erfolgreichen Akquise von Postings, weshalb der WebDriver mit all seinen Nachteilen eingesetzt werden musste.
Der in der Entwicklung getestete WebDriver *PhantomJSDriver* stellt einen Browser ohne graphische Benutzeroberfläche dar und soll besonders gut dafür geeignet sein, durch Webseiten zu navigieren, die eine hohe Integration an *JavaScript* Elementen besitzen. Tests zeigten jedoch, dass das Auslesen von Facebook-Postings mit *PhantomJS* nicht möglich war, woraus zu schließen ist, dass dies von Seiten Facebooks zu verhindern versucht wird. Eine Umsetzung mit dem

ChromeDriver führte schließlich zum erwünschten Erfolg und wird daher in weiterer Folge für den *FacebookScraper* genutzt.

4.3.2 Twitter

Das Scrapingmodul für Twitter (siehe Kapitel 2.2.1) ist mit der Klasse *TwitterScraper* umgesetzt. Der Scrapingprozess liest alle Tweets des gewünschten Users aus, wobei dafür zuerst das Beitrittsdatum aus dem Quelltext des Profils extrahiert wird. Dieser kann durch einen simplen HTTPS GET-Request angefordert und mithilfe von *Beautiful Soup* verarbeitet werden (siehe 4.2.3).

In einem ersten Versuch wurde der Scrapingprozess so umgesetzt, dass die sogenannte *Timeline* bestehend aus allen Tweets der Zielperson mit Hilfe eines Chrome WebDrivers mit dem Tool *Selenium* (siehe 4.2.3) aufgerufen und ausgelesen wird. Überschreitet die Anzahl der Tweets die Zahl 20, so werden die restlichen Tweets erst dynamisch mit *JavaScript* nachgeladen, wenn im Browser an das Ende der Seite gescrollt wird. Dies kann ebenfalls mit Hilfe von *Selenium* simuliert werden. Da bei dieser Methode eine tatsächliche Interaktion eines Benutzers mit Twitter simuliert wird, muss dem Browser nach dem Scrollaufruf etwas Zeit gegeben werden, um den neuen Inhalt nachzuladen, bevor mit Sicherheit gesagt werden kann, dass keine weiteren Tweets vorhanden sind. Diese Verzögerungen führen dazu, dass der Scrapingprozess deutlich länger dauert, als eine vergleichbare Umsetzung mit einer Progammierschnittstelle, die im Fall von Twitter jedoch nicht praktikabel wäre (siehe Kapitel 2.2.1). Bei einem Test mit einem sehr aktiven Twitternutzer konnte festgestellt werden, dass das Scrapingmodul für Twitter in etwa 378 Tweets pro Minute lesen kann (insgesamt 32.892 Tweets in 87 Minuten).
In der Verbesserungsphase dieser Methode, konnte festgestellt werden, dass beim soeben erwähnten dynamischen Ladeprozess weiterer Tweets über eine leicht modifizierte URL eine JSON-Datei mit den nächsten 20 Tweets abgerufen wird. Die in der Timeline angezeigten Tweets sind von oben nach unten absteigend nach Aktualität sortiert, sodass neu geladene Tweets eine niedrigere ID besitzen als jene an der Spitze der Timeline. Die ID des letzten angezeigten Tweets wird daher als Parameter *max_position* übergeben, um sicherzustellen, dass die nächsten 20 geladenen Tweets älter sind. Diese Funktionalität kann im verbesserten Scrapingprozess für Twitter angewandt werden. Es wird eine Instanz der Klasse *TwitterSpider* beauftragt, die Tweets des gewünschten Zeitraums abzufragen. Dies erfolgt mit dem Paket *Requests* (siehe 4.2.3) mit der URL zur JSON-Datei, wobei zu Beginn keine *max_position* angegeben wird. Aus dem JSON

kann anschließend ein HTML-Text extrahiert werden, welcher mit Hilfe des Pakets *lxml* (siehe 4.2.3) geparst wird, um die ersten 20 Tweets zu extrahieren. Ebenfalls im JSON enthalten ist ein Wert mit der Bezeichnung *min_position*, welcher die ID des letzten Tweets angibt und im nächsten Aufruf als neuer Wert für *max_position* gewählt werden kann. Ein Pseudocode, der die beschriebene Funktionalität abbildet, ist in Auflistung 3 zu sehen.

```
1  funktion finde_tweets_von_profil(username, zeitraum) {
2          url = erstelle_erste_url(username, zeitraum)
3          solange url ≠ letzte_url wiederhole
4                  json = rufe_json_ab(url)
5                  min_position = json['min_position']
6                  html = json['html']
7                  neue_tweets = extrahiere_tweets(html)
8                  gesamt_tweets += neue_tweets
9                  letzte_url = url
10                 url = erstelle_neue_url(username, zeitraum,
                   ↪ min_position)
11         return gesamt_tweets
12 }
```

Auflistung 3: Algorithmus für das Auslesen von Tweets

Die so angepasste Methode ist vor unvorhersehbaren Seiteneffekten, die eine Browsersimulation mit sich bringt, geschützt und extrahiert die oben erwähnten 32.892 Tweets des Testusers in etwa 22 Minuten. Dies stellt mit 1.495 Tweets pro Minute eine Vervierfachung der Geschwindigkeit dar.

4.3.3 Facebook

Die Klasse *FacebookScraper* ist für das Scraping von Facebook (siehe Kapitel 2.2.1) verantwortlich. Ähnlich zum *TwitterScraper* werden hier alle Postings einer Zielperson ausgelesen. Hierfür wird ein Zugriff auf Facebook über die minimalistische Version von Facebook auf *mbasic.facebook.com* gewählt, da diese Seite besonders wenige ablenkende Elemente enthält und ein Finden der gesuchten Postings vereinfacht.
Um Postings auf Facebook zu erhalten, ist es zuerst notwendig mit einem Account angemel-

det zu sein. Es kann daher vorkommen, dass mit verschiedenen Accounts mehr oder weniger Postings extrahiert werden können, da dies von den Privatsphäreeinstellungen der Zielperson abhängig ist. So ist es möglich, ein Posting öffentlich verfügbar zu machen oder auf Freunde oder *Freunde von Freunden* zu beschränken. In der Entwicklung von *slughorn* wurde ein eigener leerer Account angelegt, weshalb nur öffentliche Postings extrahiert werden konnten.
Im Scrapingprozess von Facebook wird die zeitliche Einteilung der *mbasic*-Version der Plattform ausgenutzt, bei der die Postings nach Jahren sortiert sind. Wenn innerhalb eines Jahres mehr als fünf Postings verfügbar sind, erscheint ein Link mit der Bezeichnung *Mehr anzeigen*, wobei dies von der Spracheinstellung des angemeldeten Profils abhängig ist und gegebenenfalls angepasst werden muss. Diesem Link wird so lange gefolgt, bis keine weiteren Postings mehr verfügbar sind und das nächste Jahr bearbeitet werden kann. Die Geschwindigkeit des Scrapingprozesses ist mit etwa 275 Postings pro Minute deutlich langsamer als die Facebook Graph API (etwa 5.600 Postings pro Minute) und der *TwitterScraper* mittels *TwitterSpider* (1.495 Tweets pro Minute, siehe Kapitel 4.3.2).

4.4 Datenverarbeitung

Nach erfolgter Extraktion aller Postings sozialer Netzwerke, werden diese zusammengeführt und dem nächsten Schritt, der Datenverarbeitung, übergeben. Hierbei ist es von zentraler Bedeutung, aus den Sätzen jene Wörter und Zeichen zu entfernen, die für eine Passwortgenerierung nicht in Frage kommen. Weiters gilt es, die verbliebenen Begriffe nach Relevanz zu sortieren.
Abbildung 4.5 zeigt ein UML-Aktivitätsdiagramm, welches den Prozess der Wortextraktion beschreibt. Dabei wird über alle extrahierten Postings iteriert und seriell die folgenden Schritte durchgeführt: Spracherkennung, Textbereinigung, Lemmatisierung (falls das Posting in deutscher Sprache geschrieben ist), Entfernung von Stoppwörtern, Trennung von Ausdrücken in Kategorien und Zählen der Vorkommnisse des Wortes. Das Ergebnis dieser Kette von Prozessen ist eine Liste an Ausdrücken, die in die Kategorien *Wort* und *Zahl* fallen können. Die Wörter werden anschließend in einem zweiten iterativen Verfahren weiterverarbeitet. Die Schritte umfassen dabei das Berechnen der Ausgefallenheit von Wörtern, das Kombinieren von sogenannten Fauxami und das Ermitteln eines Gesamtscores.
Die oben erwähnten Prozesse werden im Folgenden genauer beleuchtet.

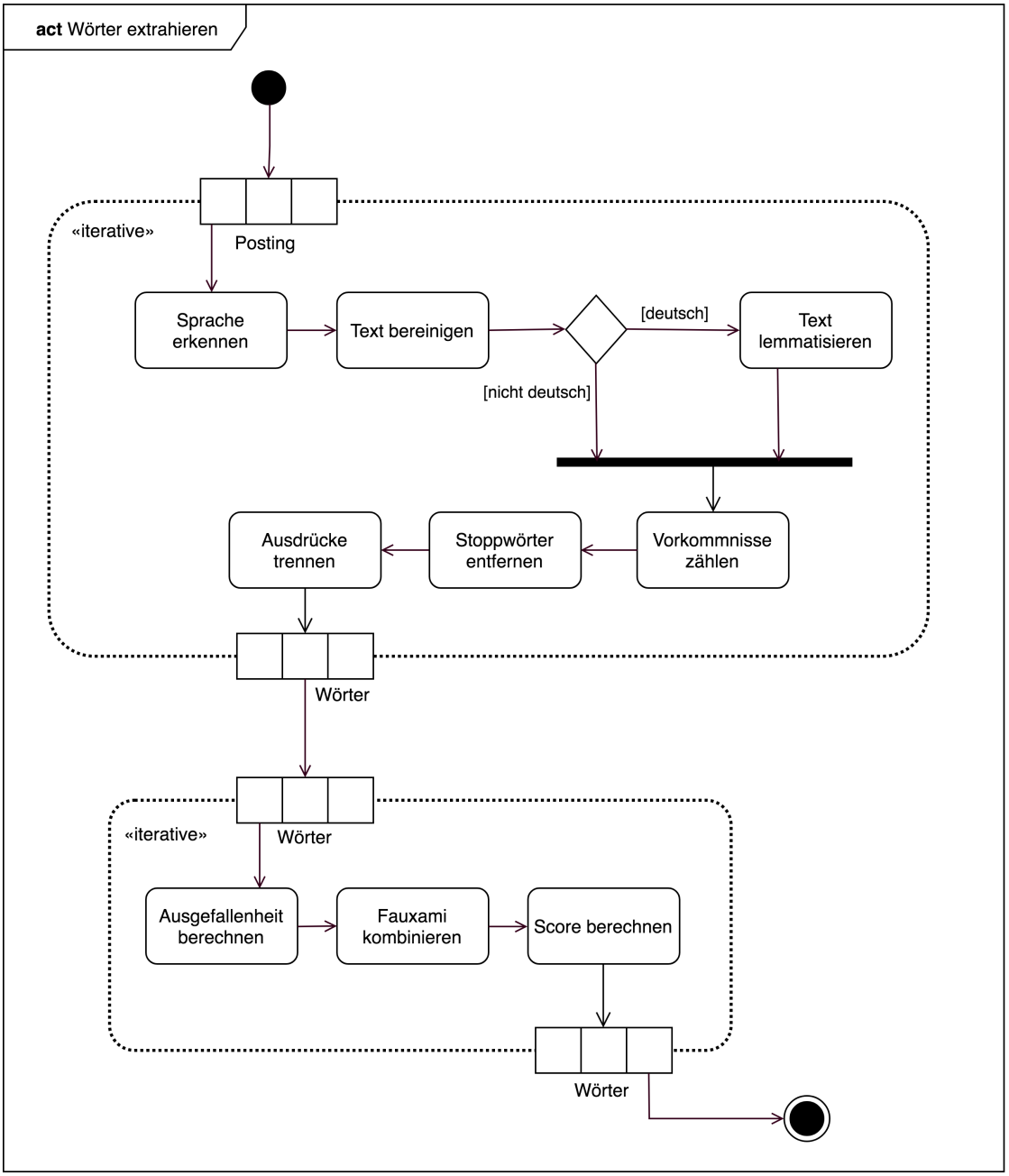


Abbildung 4.5: UML-Aktivitätsdiagramm über den Prozess der Wortextraktion

4.4.1 Spracherkennung

Der Prozess der Spracherkennung bildet den ersten Schritt für jedes extrahierte Posting und stellt eine essentielle Vorarbeit für die kommende Bereinigung der Wörter dar. Das geeignetste Python Paket zur Durchführung dieser Aufgabe ist *fastText*, das in Kapitel 4.2.3 bereits vorgestellt wurde. Die Textklassifizierungssoftware wurde von Facebook entwickelt und stellt auch ein Modell zur Erkennung von 176 Sprachen zur Verfügung. Dieses Modell wurde dem Projekt hinzugefügt

und klassifiziert die Postings anschließend mit einer zweistelligen ISO 639-3 Abkürzung, welche darauffolgend in den englischen Langnamen der Sprache transformiert wird (siehe Kapitel 4.2.3).

4.4.2 Textbereinigung

Um eine klare Identifikation von Stoppwörtern garantieren zu können, müssen Anführungszeichen sowie Beistriche und andere Sonderzeichen aus dem Text entfernt werden. Dabei wird zwischen Zeichen unterschieden, die einfach entfernt werden können, ohne den Bedeutungsgehalt von Wörtern zu gefährden und jenen, die durch Leerzeichen ersetzt werden müssen. Ein Beispiel für die zweite Kategorie ist das Zeichen -, das oft zwischen zusammengesetzten Wörtern für Trennung sorgt (*UKW-Sender*). Im Fall einer unüberlegten Entfernung des Zeichens würde das so erzeugte neue Wort inkorrekt werden (*UKWSender*). Durch eine Ersetzung mit einem Leerzeichen werden aus einem Wort zwar zwei, die Bedeutung beider Bestandteile bleibt jedoch zumindest erhalten (*UKW* und *Sender*).
Die vollständige Entfernung von Zeichen wurde anhand einer sogenannten *Whitelist* durchgeführt. Dabei wird eine Liste derer definiert, die im Text erwünscht sind, während alle anderen Zeichen entfernt werden. Diese Liste umfasst das Leerzeichen, alle Ziffern und lateinische Buchstaben, sowie deren Erweiterungen wie die deutschen Umlaute. Konkret beinhaltet die Liste jene Unicode Codepoints in Dezimalschreibweise, die in Tabelle 4.2 einsehbar sind.

Unicode Bereich (dezimal)	Beschreibung	Beispiel
32	Leerzeichen	
[48, 57]	Ziffern	*4*
[65, 90]	Großbuchstaben	*A*
[97, 122]	Kleinbuchstaben	*a*
[192, 214], [216, 246], [248, 687]	Lateinische Erweiterungen	*ö*

Tabelle 4.2: Unicode Whitelist

4.4.3 Lemmatisierung deutscher Wörter

Das Lemma ist jene Form eines Wortes, die in einem Wörterbuch eingetragen wird [80]. Bei Verben ist dies der Infinitiv Präsens Aktiv (*gehen*, *spielen*, ...), bei Nomen der Nominativ Singular (*Haus*, *Ball*, ...) und bei Adjektiven die ungebeugte, ungesteigerte (*Positiv*) Form (*heiß*, *langsam*, ...) [81]. Für die Erstellung von Passwörten ist es notwendig, aus Sätzen extrahierte

Wörter, die in verschiedenen grammatikalischen Formen auftreten können, in diese Grundform rückzuführen. Diese Prozedur wird als *Lemmatisierung* bezeichnet.
Zu diesem Zweck wird das von Markus Konrad vom *WZB Berlin Social Science Center* entwickelte Paket *GermaLemma* verwendet (siehe Kapitel 4.2.3). Die Software verwendet eine große Sammlung an deutschen Texten, den *TIGER Corpus* vom Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart [82, 83]. Der Textkorpus besteht aus etwa 50.000 deutschen Sätzen, die Zeitungen entnommen sind und zu deren Wörtern die jeweiligen Lemmata hinzugefügt wurden. Zusätzlich dazu ist jedes Wort im Korpus mit einem POS-Tag versehen, der angibt, um welchen Teil eines Satzes es sich handelt (*Artikel*, *Nomen*, *Konjunktion*, ..."). Diese Eigenschaft führt dazu, dass der *TIGER Corpus* auch ausgezeichnet geeignet ist, einen für die korrekte Funktionalität von *GermaLemma* notwendigen POS-Tagger zu trainieren. Im Korpus werden 58 Tags unterschieden, wobei für die Lemmatisierung nur wesentlich ist, ob es sich um ein Nomen, Verb, Adjektiv oder etwas anderes handelt, aus dem kein Lemma erstellt werden kann. Der notwendige POS-Tagger wurde mit der Klasse *ClassifierBasedGermanTagger* (siehe Kapitel 4.2.3) von Phillipp Nolte umgesetzt, trainiert und anschließend in serialisierter Form (als *pickle* Datei) *GermaLemma* zur Verfügung gestellt.
GermaLemma erreicht mit diesen Modellen eine Genauigkeit von 84,17%.

4.4.4 Entfernung von Stoppwörtern

Im nächsten Schritt werden sogenannte *Stoppwörter*, also häufig auftretende Wörter von geringer Relevanz, entfernt. Dafür wird der Textkorpus *stopwords* der Sprachverarbeitungsbibliothek NLTK verwendet, die in Kapitel 4.2.3 beschrieben ist. Beispielhaft werden in Auflistung 4 die deutschen Stoppwörter angeführt, die im NLTK Textkorpus *stopwords* vorhanden sind.

```
['aber', 'alle', 'allem', 'allen', 'aller', 'alles', 'als',
↪ 'also', 'am', 'an', 'ander', 'andere', 'anderem', 'anderen',
↪ 'anderer', 'anderes', 'anderm', 'andern', 'anderr',
↪ 'anders', 'auch', 'auf', 'aus', 'bei', 'bin', 'bis', 'bist',
↪ 'da', 'damit', 'dann', 'der', 'den', 'des', 'dem', 'die',
↪ 'das', 'daß', 'derselbe', 'derselben', 'denselben',
↪ 'desselben', 'demselben', 'dieselbe', 'dieselben',
↪ 'dasselbe', 'dazu', 'dein', 'deine', 'deinem', 'deinen',
↪ 'deiner', 'deines', 'denn', 'derer', 'dessen', 'dich',
↪ 'dir', 'du', 'dies', 'diese', 'diesem', 'diesen', 'dieser',
↪ 'dieses', 'doch', 'dort', 'durch', 'ein', 'eine', 'einem',
↪ 'einen', 'einer', 'eines', 'einig', 'einige', 'einigem',
↪ 'einigen', 'einiger', 'einiges', 'einmal', 'er', 'ihn',
↪ 'ihm', 'es', 'etwas', 'euer', 'eure', 'eurem', 'euren',
↪ 'eurer', 'eures', 'für', 'gegen', 'gewesen', 'hab', 'habe',
↪ 'haben', 'hat', 'hatte', 'hatten', 'hier', 'hin', 'hinter',
↪ 'ich', 'mich', 'mir', 'ihr', 'ihre', 'ihrem', 'ihren',
↪ 'ihrer', 'ihres', 'euch', 'im', 'in', 'indem', 'ins', 'ist',
↪ 'jede', 'jedem', 'jeden', 'jeder', 'jedes', 'jene', 'jenem',
↪ 'jenen', 'jener', 'jenes', 'jetzt', 'kann', 'kein', 'keine',
↪ 'keinem', 'keinen', 'keiner', 'keines', 'können', 'könnte',
↪ 'machen', 'man', 'manche', 'manchem', 'manchen', 'mancher',
↪ 'manches', 'mein', 'meine', 'meinem', 'meinen', 'meiner',
↪ 'meines', 'mit', 'muss', 'musste', 'nach', 'nicht',
↪ 'nichts', 'noch', 'nun', 'nur', 'ob', 'oder', 'ohne',
↪ 'sehr', 'sein', 'seine', 'seinem', 'seinen', 'seiner',
↪ 'seines', 'selbst', 'sich', 'sie', 'ihnen', 'sind', 'so',
↪ 'solche', 'solchem', 'solchen', 'solcher', 'solches',
↪ 'soll', 'sollte', 'sondern', 'sonst', 'über', 'um', 'und',
↪ 'uns', 'unsere', 'unserem', 'unseren', 'unser', 'unseres',
↪ 'unter', 'viel', 'vom', 'von', 'vor', 'während', 'war',
↪ 'waren', 'warst', 'was', 'weg', 'weil', 'weiter',
```

```
'welche', 'welchem', 'welchen', 'welcher', 'welches', 'wenn',
↪ 'werde', 'werden', 'wie', 'wieder', 'will', 'wir', 'wird',
↪ 'wirst', 'wo', 'wollen', 'wollte', 'würde', 'würden', 'zu',
↪ 'zum', 'zur', 'zwar', 'zwischen']
```

Auflistung 4: Deutsche Stoppwörter

In Tests hat sich gezeigt, dass diese Liste der Stoppwörter noch nicht umfassend genug ist, weshalb diese erweitert wurde. Dabei wurden die n häufigsten deutschen und englischen Wörter mit dem Python Paket *wordfreq*, das in Kapitel 4.2.3 vorgestellt wurde, ermittelt. n steht dabei für die Anzahl der Stoppwörter von NLTK in der jeweiligen Sprache. Anschließend wurde verglichen, welche dieser Wörter nicht in den Stoppwörtern enthalten sind. Bevor diese der Stoppwörterliste hinzugefügt wurden, wurde manuell überprüft, ob darin häufige Wörter vorkommen, die dennoch für die Erstellung von Passwörtern von Bedeutung sind. Dabei fiel etwa das Wort *berlin* auf, das anschließend wieder entfernt wurde.
Somit wurden etwa 80 Wörter zur Liste der deutschen Stoppwörter hinzugefügt (siehe Auflistung 5).

```
1 ['etwa', 'wer', 'gerade', 'bitte', 'nie', 'steht', 'gehen',
  ↪ 'sagen', 'jahren', 'beim', 'sehen', 'ersten', 'stadt',
  ↪ 'geht', 'heute', 'kommen', 'erst', 'recht', 'bereits',
  ↪ 'tag', 'sei', 'frage', 'wirklich', 'zeit', 'wegen', 'ja',
  ↪ 'the', 'gar', 'ab', 'wäre', 'lassen', 'warum', 'drei',
  ↪ 'neue', 'müssen', 'kommt', 'viele', 'sowie', 'macht', 'mal',
  ↪ 'seit', 'wurden', 'wurde', 'weiss', 'jedoch', 'dass',
  ↪ 'konnte', 'dafür', 'genau', 'immer', 'hätte', 'einfach',
  ↪ 'jahre', 'menschen', 'neuen', 'leben', 'uhr', 'gute',
  ↪ 'ganz', 'teil', 'gut', 'hast', 'wohl', 'gemacht', 'dabei',
  ↪ 'leute', 'gibt', 'besser', 'mehr', 'jahr', 'geben', 'of',
  ↪ 'schon', 'welt', 'zwei', 'vielleicht', 'ende', 'denen',
  ↪ 'ziemlich']
```

Auflistung 5: Zusätzliche deutsche Stoppwörter

4.4.5 Trennung von Textkategorien

Nach der Entfernung der Stoppwörter, werden die verbleibenden Begriffe in Kategorien eingeteilt. Dabei wird zwischen *Wort* und *Zahl* unterschieden. Diese werden in separate Listen sortiert und unterschiedlich weiterverarbeitet. Wörter bedürfen der meisten zusätzlichen Schritte, während Zahlen nur weiter behandelt werden, wenn dies explizit erwähnt ist.

4.4.6 Bewertung von Begriffen

In diesem Kapitel werden jene Relevanzkriterien erläutert, nach denen die nun verbleibenden Wörter sortiert werden. Dazu wird jedem Wort und jeder Zahl ein *Score* (eine Bewertung) zugewiesen, der sich bei Wörtern sowohl aus der Ausgefallenheit als auch der Häufigkeit zusammensetzt. Bei Zahlen besteht dieser nur aus der Häufigkeit.

Ausgefallenheit

wordfreq Die Häufigkeit eines Wortes im allgemeinen Sprachgebrauch wird durch das Paket *wordfreq* von Robert Speer bestimmt [79] (siehe Kapitel 4.2.3). *wordfreq* verlangt zur Bestimmung der Häufigkeit einerseits das Wort und die Sprache, andererseits die gewünschte Wortliste, die für die Analyse verwendet werden soll. Die Wortliste *large* vereint dabei verschiedenste Quellen, wie etwa Twitter, Wikipedia, Zeitungsartikel oder Bücher, und beinhaltet Begriffe, die zumindest einmal pro 100 Millionen Wörter vorkommen. Die Methode *word_frequency* hat als Ergebnis eine Zahl zwischen 0 und 1, wobei diese die relative Häufigkeit angibt. Der Wert 0 bedeutet daher, dass das gesuchte Wort kein einziges Mal im Textkorpus vorkommt, während 1 dafür steht, dass alle Wörter der gesuchte Begriff sind. Zweiteres betrachtet macht nicht nur rasch ersichtlich, dass dies ein unrealistisches Szenario darstellt, sondern führt auch vor Augen, dass die relativen Häufigkeiten der Wörter zwischen 0 und 1 nicht gleichmäßig verteilt sind. So zeigt das Histogramm in Abbildung 4.6, dass bei 14.586 eines österreichischen Nachrichtenmoderator extrahierten deutschen Wörtern, der Großteil von ihnen eine relative Häufigkeit von unter 0.01 aufweist, also weniger als 1% des deutschen Sprachgebrauchs ausmacht. Das deutsche Wort mit der höchsten relativen Häufigkeit ist laut *wordfreq* der Artikel *die*, der einen Wert von 0.03236 erreicht, jedoch nach der Entfernung von Stoppwörtern nicht mehr enthalten ist.

Zipf Skalierung Wie von Van Heuven et al. vorgeschlagen, ist eine Transformation der Häufigkeitswerte auf die Zipfskala eine Möglichkeit, diese ungleichmäßige Verteilung zu korrigieren. Dabei wird der Logarithmus zur Basis 10 der Anzahl der Vorkommnisse eines Wortes pro

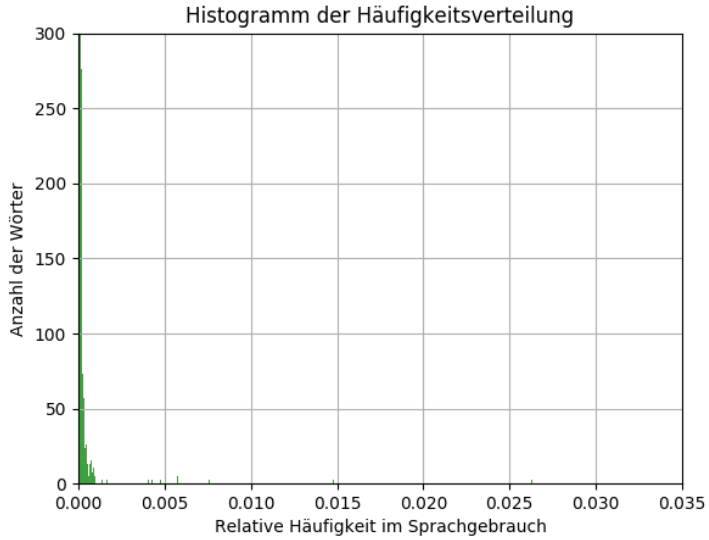


Abbildung 4.6: Histogramm der Häufigkeit von extrahierten Wörtern im Sprachgebrauch

Milliarde Wörter berechnet. Dies bietet den Vorteil einer gleichmäßigeren Verteilung der Werte zwischen 0 und 9 und damit der Realisierung einer Normalverteilung mit einem μ von etwa 3,5. [84, 85]

Das Histogramm der oben genannten Daten ist nach der Transformation in Abbildung 4.7 ersichtlich.

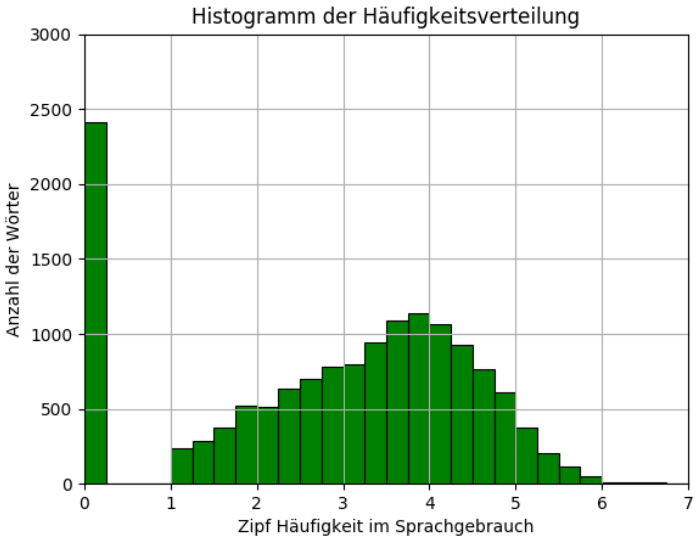


Abbildung 4.7: Histogramm der Zipfhäufigkeit von extrahierten Wörtern im Sprachgebrauch

Tabelle 4.3 zeigt beispielhaft die Beziehung der relativen Häufigkeit zur Zipfhäufigkeit. Wörter mit einem Zipfwert von 0 kommen maximal einmal vor, wobei festzuhalten ist, dass die gesteigerte Häufigkeit dieses Wertes in den oben genannten Beispieldaten dadurch begründet ist, dass viele Wörter in den Postings geschrieben wurden, die im Textkorpus gar nicht vorkommen.
Wörter mit einer Zipfhäufigkeit von 8 oder 9 würden etwa 10% beziehungsweise sogar 100% des Textkorpus ausmachen. Dies entspricht nicht dem praktischen Vorkommen.

Zipf	Vorkommnisse pro Milliarde Wörter	relative Häufigkeit	Beispiele ($\pm 0, 1$)
0	1	10^{-9}	*Siegertext*
1	10	10^{-8}	*Medienbetrieb*, *hartgesotten*, *Drogendelikt*
2	100	10^{-7}	*Wahlkarte*, *Sonderklasse*, *abtrünnig*
3	1.000	10^{-6}	*interviewen*, *Prominenz*, *miserabel*
4	10.000	10^{-5}	*Journalismus*, *Bundespräsident*
5	100.000	10^{-4}	*Ergebnis*, *Opfer*
6	1 Mio.	0,001	*weil*, *heute*, *seit*
7	10 Mio.	0,01	*von*, *den*
8	100 Mio.	0,1	-
9	1 Mrd.	1,0	-

Tabelle 4.3: Beziehung der relativen Häufigkeit zur Zipfhäufigkeit

Diese Darstellungsform wurde von Van Heuven et al. vor allem aus Gründen des besseren Verständnisses vorgeschlagen. In der vorliegenden Form des Tools *slughorn* wird der Wert der Ausgefallenheit jedoch nicht an den Benutzer oder die Benutzerin rückgemeldet, sondern ausschließlich für die Berechnung des Scores (siehe Kapitel 4.4.6 Abschnitt *Score*) und somit für die Reihung der Wörter nach Relevanz herangezogen. Aus diesen Gründen ist die relative Häufigkeit trotz ungleichmäßiger Verteilung vorzuziehen, da sich dieser Wert bereits in einem Bereich von $[0, 1]$ bewegt und somit leichter mit dem Wert der Häufigkeit, der in Kapitel 4.4.6 Abschnitt *Häufigkeit* ähnlich berechnet wird, kombiniert werden kann.
Für eine potentielle Erweiterung des Tools ist die Anzeige der Zipfhäufigkeit für den User vorstellbar.

Lineare Skalierung Um der ungleichmäßigen Verteilung entgegenzuwirken und die Unterschiede zwischen den Werten hervorzuheben, wird die relative Häufigkeit der Wörter im Sprachgebrauch einer linearen Skalierung auf den Bereich $[0, 1]$ unterzogen. Dafür wird der maximal auftretende Wert als neues Maximum u verwendet.
Die allgemeine Formel zur Berechnung des neuen Wertes im Bereich $[0, 1]$ aus einem alten Wert im Bereich $[l, u]$ ist in Gleichung 4.1 dargestellt [86].

$$\tilde{x} = \frac{x - l}{u - l} \tag{4.1}$$

Der mögliche Bereich des relativen Anteils x ist $[0, u]$, wobei u den maximal auftretenden relativen Anteil darstellt. So reduziert sich die Formel für die Berechnung des neuen Wertes, wie in Gleichung 4.2 zu sehen ist.

$$\tilde{x} = \frac{x}{u} \tag{4.2}$$

Komplement Abschließend wird die Ausgefallenheit eines Wortes (A) als das Komplement der linear skalierten relativen Häufigkeit bezeichnet. Dies ist jener Wert A zur relativen Häufigkeit x, der mit diesem addiert 1 ergibt (siehe Gleichung 4.3).

$$A = 1 - x \tag{4.3}$$

Häufigkeit Die Vorkommnisse jedes Wortes werden bei der Bearbeitung der Postings laufend mitgezählt und letztendlich durch die Anzahl der gesamten Wörter dividiert. Dies ergibt den Anteil eines Wortes an der gesamten Wörterliste. Dieser Anteil wird anschließend linear erneut auf den Bereich $[0, 1]$ skaliert, wobei der maximal auftretende Anteil als neues Maximum gewählt wird. Für diese Berechnung wird erneut die Formel aus Auflistung 4.2 herangezogen, wobei der so berechnete Wert als H für Häufigkeit bezeichnet wird.
Als Beispiel kann das Wort *Nationalrat* mit 6 Vorkommnissen in einer fiktiven Wortextraktion von 55 verschiedenen Wörtern mit insgesamt 390 Vorkommnissen herangezogen werden. Der Anteil des Wortes ist somit $\frac{6}{390} = 0,015385$. Dieser Wert wird anschließend linear skaliert, indem durch den Wert des am häufigsten aufgetretenen Wortes *Österreich* (28 Vorkommnisse, also ein Anteil von $0,071795$) dividiert wird. Dies ergibt einen Wert H von $0,214286$ als Endergebnis.

Score Der Score S gibt die Relevanz eines Wortes an und wird für die Reihung der Begriffe herangezogen. Er setzt sich aus den beiden Werten A (*Ausgefallenheit*) und H (*Häufigkeit*) zusammen, die, wie in den Kapiteln 4.4.6 Abschnitt *Ausgefallenheit* beziehungsweise 4.4.6 Abschnitt *Häufigkeit* dargelegt wurde, Werte aus dem Bereich $[0, 1]$ sind. Gleichung 4.4 zeigt die Berechnung des Scores aus diesen beiden Werten mit den Gewichtungen w_A beziehungsweise w_H.

$$S = w_A \times A + w_H \times H \tag{4.4}$$

Die Werte für die Gewichtungen w_A und w_H ergänzen einander auf 1 (Gleichung 4.5) und geben den Einflussfaktor der Variablen A und H an. Die optimale Gewichtung kann mit Hilfe einer empirischen Untersuchung festgestellt werden, bei der eine große Menge an realen Passwörtern bestehend aus personenbezogenen Informationen mit Wortextraktionen aus sozialen Netzwerken verglichen werden, um so die Größe der Einflussfaktoren Häufigkeit und Ausgefallenheit der Wörter festzustellen. Dieses Vorhaben übersteigt den Rahmen der vorliegenden Arbeit und bildet einen Anknüpfungspunkt für zukünftige Forschungen in diesem Gebiet (siehe Kapitel 6.2.3).

$$w_A + w_H = 1 \tag{4.5}$$

Im Zuge der Entwicklung des Tools *slughorn* wurde eine Gewichtung der Werte A und H mit 0.5 und 0.5 approximiert und als Standardwert gewählt. Abbildung 4.8 zeigt, dass bei einer gleichen Gewichtung der Parameter eine annähernde Normalverteilung entsteht. Dies ist dadurch begründet, dass die Häufigkeitsverteilung des Parameters Häufigkeit rechtsschief und die des Parameters Ausgefallenheit linksschief ist. Links- und Rechtsschiefe bedeutet dabei, dass der überwiegende Teil der Daten rechtsseitig beziehungsweise linksseitig konzentriert ist.
Die Umstellung der Gewichtung ist über eine Option des CLI möglich und in Kapitel 4.2.1 erläutert.

Die Reihung in der Begriffskategorie *Zahl* erfolgt ausschließlich über die Anzahl der Vorkommnisse, weshalb an dieser Stelle keine Berechnung eines Scores vorgenommen werden muss.

4.4.7 Behandlung von Fauxami

Sogenannte *Fauxami* oder *Falsche Freunde* sind Wörter, die in verschiedenen Sprachen vorkommen, aber unterschiedliche Bedeutung haben. Diese Paare können auch in leicht unterschiedlich

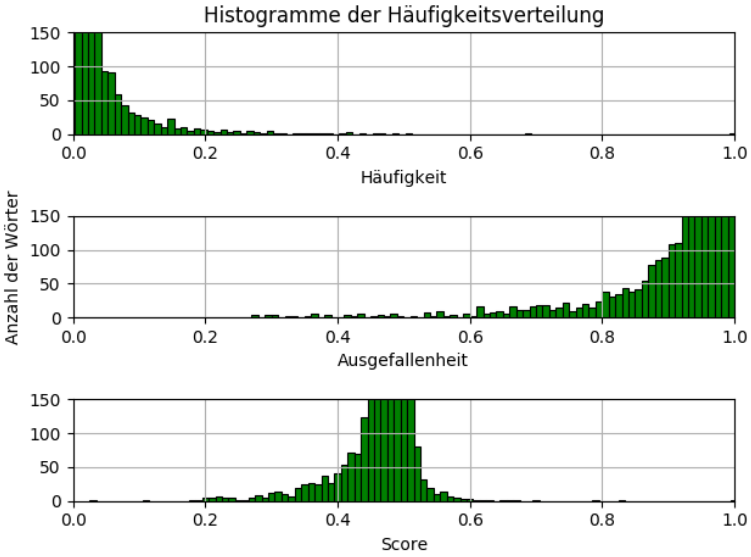


Abbildung 4.8: Kombinierter Score bei gleicher Gewichtung der Parameter

geschriebener Form auftreten, wobei für den Zweck dieser Arbeit ausschließlich identische (abgesehen von der Großschreibung) Fauxami von Interesse sind. Ein Beispiel für so einen *Falschen Freund* ist etwa das deutsche Wort *Gift*, das im Englischen ein Geschenk bezeichnet.
Bei der Behandlung von Fauxami werden die Werte der beiden Ausgefallenheiten verglichen (siehe Kapitel 4.4.6) und die Vorkommnisse des niedrigeren zum höheren addiert bevor ersteres dann verworfen wird. So wird sichergestellt, dass kein Wort doppelt vorkommen kann und wichtigere Wörter weiter oben in der Liste stehen. Am Beispiel des oben genannten Wortes *Gift*, das im Deutschen eine höhere Ausgefallenheit besitzt als im Englischen, bedeutet dies, dass die Vorkommnisse des Wortes in der englischen Liste zu der der deutschen Liste hinzugefügt werden. Anschließend wird der englische Eintrag gelöscht, da seine Ausgefallenheit niedriger ist.

4.5 Passwortgenerierung

Nach erfolgter Akquise von personenbezogenen Informationen und Extraktion relevanter Ausdrücke sowie Sortierung dieser, gilt es, die eigentliche Generierung von Passwörtern nach definierten Regeln durchzuführen. Die Funktion des Ausprobierens von Passwörtern (Passwortcracking) wurde bereits von etablierten Programmen wie den in den Kapiteln 3.2.3 Abschnitt *John the Ripper* sowie 3.2.3 Abschnitt *Hashcat* vorgestellten Toolkits umgesetzt und im Zuge dieser Arbeit verwendet.

Um die extrahierte und sortierte Liste von Wörtern für die Toolkits verwendbar zu machen, ist es notwendig, diese ohne zusätzliche Informationen wie Häufigkeit, Ausgefallenheit oder Score durch Zeilenumbrüche getrennt in einer Textdatei zu speichern. Dieser Vorgang wird in *slughorn* von der Klasse *WordListGenerator* übernommen.
Anschließend wird eine Instanz der Klasse *RuleGenerator* initiiert, deren Aufgabe es ist, auf Basis der extrahierten Zahlen dedizierte Regeln zu erstellen, bei denen diese mit den Elementen der zuvor erwähnten Wortliste kombiniert werden. Dabei werden jene Regeln angewandt, die in Tabelle 4.4 mit dem Wort *beispiel* und der Zahl *1986* angeführt sind. Die Funktion *l* gibt dabei das jeweilige Wort in Kleinbuchstaben zurück, *u* in Großbuchstaben, *c* schreibt den ersten Buchstaben groß und den Rest klein, während *C* den umgekehrten Fall abdeckt. *r* dreht die Reihenfolge der Buchstaben um, sodass das Wort von hinten nach vorne geschrieben wird. Eine Einfügung eines Zeichens ans Ende des Wortes erfolgt mit der Funktion *$*. ^ stellt ein Zeichen zu Beginn des Wortes an. Die in diesen Regeln verwendeten Funktionen sind sowohl in *hashcat* als auch in *John the Ripper* verfügbar, weshalb die vom *RuleGenerator* erstellte Datei in beiden Programmen verwendet werden kann.

Regel	**Beschreibung**	**Transformation**
l$1$9$8$6	Kleinbuchstaben, Anhängung der Zahl	beispiel1986
l^6^8^9^1	Kleinbuchstaben, Voranstellung der Zahl	1986beispiel
u$1$9$8$6	Großbuchstaben, Anhängung der Zahl	BEISPIEL1986
u^6^8^9^1	Großbuchstaben, Voranstellung der Zahl	1986BEISPIEL
c$1$9$8$6	erster Bst. groß und Rest klein, Anhängung der Zahl	Beispiel1986
c^6^8^9^1	erster Bst. groß und Rest klein, Voranstellung der Zahl	1986Beispiel
C$1$9$8$6	erster Bst. klein und Rest groß, Anhängung der Zahl	bEISPIEL1986
C^6^8^9^1	erster Bst. klein und Rest groß, Voranstellung der Zahl	1986bEISPIEL
r$1$9$8$6	umgekehrte Reihenfolge, Anhängung der Zahl	leipsieb1986
r^6^8^9^1	umgekehrte Reihenfolge, Voranstellung der Zahl	1986leipsieb

Tabelle 4.4: Regeln nach denen Zahlen mit Wörtern kombiniert werden

Zusätzlich werden diese Regeln mit jenen von *hashcat* in der Datei *best64.rule* bereitgestellten ergänzt. Dieses aus nur 77 Regeln bestehende Werk wurde so gewählt, dass damit eine breite Masse an Passwörtern erraten werden kann und die Regeln nicht zu spezialisiert auf ein Set von öffentlich gewordenen Passwörtern angepasst sind. Kann mit dem vom *RuleGenerator* erstell-

ten Regelwerk kein Erfolg erzielt werden, ist zu empfehlen eine breitere Herangehensweise zu wählen, die mehr Regeln enthält und damit eine längere Zeitdauer beansprucht aber höhere Erfolgsaussichten bietet. Ein derartiges Regelwerk ist in der Datei *dive.rule* enthalten, das ebenfalls Bestandteil des Toolkits *hashcat* ist.

4.6 Herausforderungen

In diesem Kapitel geht es um Herausforderungen, die es bei der Entwicklung des Tools *slughorn* zu bewältigen galt und die Änderungen an der geplanten Architektur erforderten. Zum einen wird das Thema *Performance* besprochen, das in der Softwareentwicklung stets von großer Relevanz ist. Weiters wird auf die Einschränkungen der API eingegangen, die die Nutzung für diese Arbeit unmöglich machen und den Bedarf einer Alternative geweckt haben. Zuletzt wird auf die Schwierigkeiten bei der Evaluierung des Tools eingegangen.

4.6.1 Performance

Eine große Herausforderung bei der Entwicklung des Toolkits war die Geschwindigkeit des Akquise- und Verarbeitungsprozesses. Eine Abfrage der Postings mit Hilfe einer API war aufgrund der in Kapitel 4.6.2 beschriebenen Einschränkungen in den meisten Fällen nicht möglich, weshalb eine Alternative gefunden werden musste, die jedoch mit Abstrichen in der Performance einhergeht. Postings von privaten Facebookprofilen können demnach nur durch Simulation eines Browsers automatisiert abgefragt werden. Dies ist ein deutlich langsamerer Prozess als die Abfrage der öffentlichen Facebookseiten mit der API. Im Fall von Twitter konnte eine Methode erarbeitet werden, die auf die Verwendung von Browsersimulation verzichtet und dadurch einen Verlust der Performance größtenteils verhindert.
Bei der Verarbeitung der akquirierten Postings bestand aufgrund der großen Zahl an Verarbeitungsschritten die Gefahr, dass durch die notwendigen Schleifen im Programm ein hoher Zeitverlust entsteht. Aus diesem Grund wurde bereits in der Planungsphase des Tools darauf geachtet, dass rechenintensive Prozesse, deren Ergebnisse mehrmals benötigt werden, einmalig durchgeführt und späteren Schritten zur Verfügung gestellt werden. Zusätzlich dazu werden die extrahierten Informationen in einem Python Dictionary gespeichert, was einer Hashtabelle entspricht. Diese hat den Vorteil, dass darin Elemente äußerst schnell gesucht und Transformationen wie das Sortieren einer Liste von Wörtern sehr schnell durchgeführt werden können.
Die weiteren Verarbeitungsschritte wie die Generierung der finalen Wortliste und das Erstellen

der Crackingregeln sind mit keinen rechenintensiven Operationen verbunden und können daher in der Performanceanalyse vernachlässigt werden.
Tabelle 4.5 zeigt einen Überblick über die Performance der einzelnen Schritte.

Schritt	Elemente pro Minute	Beispiel
Akquise (Facebook, privates Profil)	∼275 Postings	1.500 Postings in 5:27 Min.
Akquise (Facebook, öffentl. Seite)	∼5.600 Postings	1.500 Postings in 0:16 Min.
Akquise (Twitter)	∼1.500 Tweets	1.500 Tweets in 1:00 Min.
Extraktion von Ausdrücken	∼500 Ausdrücke	1500 Wörter in 3:00 Min.
Erstellung der Wortliste	-	<1 Sek.
Erstellung der Regeln	-	<1 Sek.

Tabelle 4.5: Performanceanalyse der Akquise- und Verarbeitungsschritte

4.6.2 Einschränkungen der API

Die in Kapitel 2.2.1 vorgestellte API von Twitter und deren eingeschränkte kostenlose Benutzbarkeit durch das Stufenmodell führt dazu, dass ein vollständig anderes Akquiseverfahren verwendet werden musste. Im Gegensatz zu Twitter sind die meisten Profile auf Facebook privat, wobei diese über die Graph API grundsätzlich aus Datenschutzgründen nicht ansprechbar sind. Aufgrund dieser Einschränkungen musste nach Alternativen zu einer API gesucht werden.
Die Simulation eines Browsers stellt einen aufwendigeren Prozess dar, der auch eine schlechte Wartbarkeit aufweist. Durch kleine Änderung des Seitenaufbaus von Twitter oder Facebook können Elemente nicht mehr gefunden werden, die zur automatisierten Navigation durch die Seitenstruktur notwendig sind. Daher ist laufend zu überprüfen, ob Änderungen durchgeführt wurden und diese im Tool angepasst werden müssen.

4.6.3 Evaluierung

Um die Funktionsweise des im Zuge der Arbeit entstandenen Tools zu evaluieren, ist es notwendig, mit empirischen Daten zu arbeiten. Ein korrektes Testszenario besteht dabei aus realen Personen, ihren Profilen in den sozialen Netzwerken Twitter und/oder Facebook und tatsächlichen personenbezogenen Passwörtern. Während es trivial ist, eine repräsentative Menge an Profilen zu finden, so fehlen bei dieser Herangehensweise die dazugehörigen Passwörter. Eine andere Möglichkeit, das Testszenario aufzubauen, besteht darin, durch Hacks öffentlich gewordene Passwörter als Basis zu nehmen. Dabei besteht jedoch das Problem, dass von diesen Passwör-

tern nicht zweifelsfrei auf die dazugehörigen Profile der Besitzer zu schließen ist. Wenngleich bei vielen Leaks E-Mailadressen vorhanden sind, so ist es dennoch nicht möglich, damit eine Verbindung zu den Accounts herzustellen, da die Privatsphäreeinstellungen von Facebook und Twitter keine Suche nach E-Mailadressen zulassen.
Dieser Mangel an realen Testdaten stellt eine große Herausforderung bei der Evaluierung des Tools dar und führt dazu, dass dabei Testaccounts und erfundene Passwörter herangezogen werden müssen (siehe Kapitel 5).

4.7 Designentscheidungen

Aufgrund der in Kapitel 4.6 dargelegten Herausforderungen, die es bei der Entwicklung des Tools *slughorn* zu bewältigen gab, und auch durch Eigenschaften, die das Projekt einzigartig machen, waren bei der Architektur des Programms und der Wahl der Techniken Entscheidungen zu treffen, die zusätzlicher Erklärung bedürfen. Diese gilt es in diesem Kapitel darzulegen und zu begründen.

4.7.1 Webdriver

Die restriktiven Eigenschaften der API von Twitter und Facebook wurden bereits in Kapitel 4.6.2 beschrieben. Diese führen dazu, dass eine aufwendigere Akquisemethode gewählt werden musste, die mit Nachteilen, wie schlechter Wartbarkeit, verbunden ist. Die Wahl einer Browsersimulation ist durch die Notwendigkeit, dynamisch durch *JavaScript* ausgelöste Inhalte nachzuladen, bedingt. Die dadurch entstehenden Performance-Einschränkungen sind im Vergleich zu Lösungen mit einer API enorm und wurden in Kapitel 4.6.1 erläutert.
Die dafür implementierten Klassen *FacebookWebdriver* und *TwitterWebdriver* sind Subklassen von *webdriver.Chrome* des Python Pakets *Selenium* (siehe Kapitel 4.2.3). Dieses Design der Vererbung bietet den Vorteil, dass die Methoden der Superklasse, wie etwa *get*, um eine Webseite zu laden, auch in den implementierten Klassen verfügbar sind. Weiters können diese auch mit neuen Methoden erweitert werden, die für die Quelle von besonderer Bedeutung sind. Eine derartige Methode ist *get_numeric_id*, die für die Erlangung der numerischen ID aus einem Facebookprofilnamen notwendig ist.
Die Klasse *FacebookWebdriver* basiert lose auf dem GitHub Projekt *FacebookBot* [87] von User *hikaruAi*, wobei Methoden für den Zweck der vorliegenden Arbeit ergänzt und Fehler korrigiert wurden.

Eine Alternative zur Verwendung des *TwitterWebdrivers* wurde im Zuge der Arbeit mit der Implementierung der Klasse *TwitterSpider* geschaffen. Diese verzichtet vollständig auf die Verwendung von Browsersimulation und beseitigt damit die oben erwähnten Nachteile. Bei der Entwicklung weiterer Scraper sind daher Alternativen zur Akquise mit *Selenium* zu suchen und wenn möglich dieser vorzuziehen.

4.7.2 Erweiterbarkeit

Eine besonders wichtige Designentscheidung war eine Architektur, die es ermöglicht, auf einfachem Weg neue Quellen für die Datenakquise zu ergänzen. Im Zuge dieser Arbeit wurden beispielhaft zwei soziale Netzwerke - Facebook und Twitter - ausgewählt und Scraper implementiert. Für eine Weiterentwicklung des Tools, ist es nur notwendig, weitere Scraper hinzuzufügen und diese im CLI (siehe Kapitel 4.2.1) zu ergänzen. Somit ist keine Anpassung der Prozesse *Datenverarbeitung* und *Passwortgenerierung* notwendig und garantiert damit eine hohe Erweiterbarkeit.

5 Evaluierung

In diesem Kapitel soll nun die Arbeit in Form des entstandenen Tools *slughorn* evaluiert werden. Dabei wird zuerst auf den Aufbau des Testsystems eingegangen, bevor anschließend die Testpersonen vorgestellt und die Ergebnisse präsentiert werden. Abschließend wird dargelegt, welche Erkenntnisse daraus geschlossen werden können und welche Aussagen dies über die Qualität des Tools *slughorn* zulässt.

5.1 Testsystem

In Kapitel 4.6.3 wurde bereits angesprochen, dass die Evaluierung der vorliegenden Arbeit aufgrund der fehlenden Daten eine große Herausforderung darstellt. Da durch Hacks öffentlich gewordene Passwörter nicht den Profilen in sozialen Netzwerken zugeordnet werden können, ist es notwendig, eine Evaluierung auf Basis von bestehenden Profilen und zugeordneten Passwörtern durchzuführen. Dabei wurden fünf Personen ausgewählt, wobei darauf geachtet wurde, eine möglichst gleichmäßige Verteilung der Faktoren *Alter*, *Geschlecht*, *Hobbys* und *Beziehungsstand* zu erreichen. Die Social Media Profile dieser fünf Testpersonen wurden anschließend vier unabhängigen Personen mit der Aufgabe vorgelegt, für jede Testperson fünf personenbezogene Passwörter zu erstellen. Dabei sollte darauf geachtet werden, dass die fünf Passwörter in ihrer Sicherheit von Stufe 1 bis Stufe 5 zunehmen und verschiedene Passwortstrategien bei der Erstellung verwendet werden. Die Definition der Sicherheitsstufen wurde bewusst offen gelassen, um den in Kapitel 2.1.3 gewonnenen Erkenntnissen gerecht zu werden, nach denen die wahrgenommene und tatsächliche Sicherheit von Passwörtern häufig divergieren und letztere mit Hilfe von unterschiedlichen Methoden bestimmt werden kann. Indem den vier unabhängigen Personen diese Freiheiten bei der Passwortgenerierung gegeben wurden, konnte sichergestellt werden, dass die zugewiesenen Passwörter realistisch sind und deren Muster unbeeinflusst gewählt wurden.

Anschließend wurde mit Hilfe von *slughorn* für jede Testperson eine Wortliste und eine personalisierte Regeldatei erstellt. Für die zugewiesenen Passwörter wurden die MD5-Hashes berechnet

und in einer Datei gespeichert. Die Wahl für den Hashalgorithmus fiel auf MD5, da dieser sehr schnell durchgeführt werden kann und so sofort sichtbar ist, welche Passwörter erfolgreich wiederhergestellt werden können. Zuletzt wurde der Crackingprozess mit *hashcat* gestartet, wobei zuerst die personalisierte Regeldatei und anschließend die von *hashcat* zur Verfügung gestellte Datei *dive.rule* herangezogen wurde. Auflistung 6 zeigt beispielhaft die Befehle, mit denen der Prozess für Testperson 1 gestartet wurde.

```
1 hashcat -m 0 passwords.hash wordlist_testperson_1.txt -r
  ↪ testperson_1.rule --debug-mode=4 --debug-file=matched.rule
2 hashcat -m 0 passwords.hash wordlist_testperson_1.txt -r
  ↪ dive.rule --debug-mode=4 --debug-file=matched.rule
```

Auflistung 6: Befehle für *hashcat*

5.2 Testpersonen

Die fünf Testpersonen werden im Folgenden anonymisiert anhand von Steckbriefen vorgestellt und die zugewiesenen Passwörter sowie die Ergebnisse des Crackingversuchs in den dazugehörigen Tabellen 5.2, 5.3, 5.4, 5.5 und 5.6 dargestellt.
Bei einer erfolgreichen Wiederherstellung des Passwortes wird unterschieden, mit welcher Regeldatei das gesuchte Passwort erstellt wurde: *slughorn* steht dabei für die in Kapitel 4.5 beschriebene, mit Hilfe der akquirierten Zahlen generierte Regeldatei, die bei allen Testpersonen zuerst probiert wurde, während *Dive* die Datei *dive.rule* angibt. In der letzten Spalte ist die erfolgreiche Kombination aus Ausgangswort der Wortliste und angewandter Regel angeführt, die eine erfolgreiche Wiederherstellung ermöglichte.
Konnte das gesuchte Passwort nicht wiederhergestellt werden, ist eine von fünf in Tabelle 5.1 beschriebenen Mängelkategorien angeführt.

Abkürzung	Kurzfassung	Beschreibung
M1	Wortliste	Ein für das Passwort benötigtes Wort befindet sich nicht in der Wortliste.
M1a	Profilverknüpfung	Um ein für das Passwort benötigtes Wort zu akquirieren, ist es notwendig, Verknüpfungen zu anderen Profilen herzustellen. Ein Beispiel dafür ist eine Zielperson, die personenbezogene Informationen eines oder einer Verwandten verwendet.
M2	Regelkomplexität	Das für das Passwort benötigte Wort konnte zwar akquiriert werden, es wurde jedoch eine Transformation (etwa durch Subsitutionen oder Anhänge, siehe Kapitel 2.1.2) vorgenommen, die nicht durch die Regeln abgedeckt ist.
M3	Grundformfehler	Das für das Passwort benötigte Wort konnte zwar akquiriert werden, wurde jedoch inkorrekt in seine Grundform zurückgeführt (siehe Kapitel 4.4.3).
M4	Trennung von Begriffen	In der Datenverarbeitung (siehe Kapitel 4.4) wurden zusammenhängende Begriffe, die mit zwei oder mehreren Bestandteilen aufgebaut sind, in separate Begriffe getrennt und so deren Bedeutung zerstört. Ein Beispiel dafür ist der Name der britischen Komikergruppe *Monty Python*, der zusammengefasst bearbeitet werden muss.

Tabelle 5.1: Beschreibung der Mängelkategorien

Testperson 1

- Alter: 25
- Geschlecht: männlich
- Beruf: Student (Elektrotechnik)

- Hobbys: Klettern, Mountainbike, Rockmusik (spielt E-Bass und E-Gitarre in einer Band)
- Beziehungsstand: ledig, keine Kinder
- Sprache: Deutsch
- Quellen: Facebook
- Länge der Wortliste: 3.205
- Erfolgsquote: 5/20 (25%)

Stufe	Passwort	Wiederherstellung	Ausgangswort:Regel
1	klettern123	M1	
1	klettern	dive.rule	kletterhelm:’7 $n
1	playstation3	dive.rule	playstationnetwork:’B $3
1	oblivion666	dive.rule	oblivion:$6 $6 $6
2	klettermax	M1	
2	kraxln1992	M1	
2	hobbiton22	M1	
2	?fitnesscenter	dive.rule	fitnesscenter:^?
3	m0unt@1nb1ke	dive.rule	mountainbike:sa@ si1 so0
3	1Day2remember	M4	
3	777Arduino	M1	
3	roteskreuz144	M4	
4	psnetworker!	M1	
4	pl@yst@ti0n	M2	
4	rotes+144	M4	
4	rettungscola777	M1	
5	cROSSFIT999	M2	
5	rockapal00za!	M2	
5	sdrohc$$$	M2	
5	saniTÄTER;	M1	

Tabelle 5.2: Ergebnisse für Testperson 1

Testperson 2

- Alter: 52
- Geschlecht: männlich

- Beruf: Journalist
- Hobbys: Politik, Journalismus, Reisen (Italien)
- Beziehungsstand: verheiratet, zwei Kinder
- Sprache: Deutsch
- Quellen: Facebook, Twitter
- Länge der Wortliste: 61.233
- Erfolgsquote: 14/20 (70%)

Stufe	Passwort	Wiederherstellung	Ausgangswort:Regel
1	österreich	slughorn.rule	$HEX[c3b6737465 727265696368]::
1	legislaturperiode	slughorn.rule	legislaturperiode::
1	politik123	slughorn.rule	politik:$1 $2 $3
1	orf1234	slughorn.rule	orf1:l $2 $3 $4
2	problembär56	slughorn.rule	$HEX[70726f626c65 6d62c3a472]:l $5 $6
2	August1966	slughorn.rule	august:c $1 $9 $6 $6
2	Mexiko2000	slughorn.rule	mexiko:c $2 $0 $0 $0
2	falc0falc0	dive.rule	falco:so0 d
3	meinungsfreiheit!	dive.rule	meinungsfreiheit:$!
3	journalistdesjahres2004	M1	
3	presserat_2005	M2	
3	stra$$enbahn12	M2	
4	nATIONALRAT97	slughorn.rule	nationalrat:C $9 $7
4	1t@113n	dive.rule	italien:sa@ sc< se3 si1 so0 ss$
4	italien&2001	M2	
4	ingrid1996	slughorn.rule	ingrid:l $1 $9 $9 $6
5	$Fl0renz1992	M2	
5	annahoJ2004	M1a	
5	sumsilanruoj100	slughorn.rule	journalismus:r $1 $0 $0
5	sCHNECKENPOST1	slughorn.rule	schneckenpost:C $1

Tabelle 5.3: Ergebnisse für Testperson 2

Testperson 3

- Alter: 36
- Geschlecht: männlich
- Beruf: Filmemacher
- Hobbys: Filme, Filmfestivals, Kultur der 1980er Jahre
- Beziehungsstand: verheiratet, keine Kinder
- Quellen: Facebook
- Länge der Wortliste: 1.960
- Erfolgsquote: 4/20 (20%)

Stufe	Passwort	Wiederherstellung	Ausgangswort:Regel
1	pearljam	M1	
1	Frontale2015	slughorn.rule	frontale:c $2 $0 $1 $5
1	topgun1986	M4	
1	handyfilm66	slughorn.rule	handyfilm:l $6 $6
2	r3inhard&	M2	
2	haydncinema99	M4	
2	tomcruise55	M1	
2	PREMIERE_	dive.rule	premiere:u $_
3	handyfilm§	M2	
3	_haydn123	M2	
3	abrakadabra78	M1	
3	hollywood_LA	M1	
4	back2thefuture	M1	
4	mcfly1983	M1	
4	adlihtam2003	M1a	
4	hIGHLANDER	dive.rule	highlander:C
5	F1lmf3$t1v@l	M2	
5	_shortfilms123	M1	
5	bibibloxberg%5%	M1	
5	who_IS_who!!	M1	

Tabelle 5.4: Ergebnisse für Testperson 3

Testperson 4

- Alter: 45
- Geschlecht: weiblich
- Beruf: Lehrerin
- Hobbys: Kochen, Backen, Bloggen
- Beziehungsstand: verheiratet, zwei Kinder
- Sprache: Englisch
- Quellen: Facebook, Twitter
- Länge der Wortliste: 658
- Erfolgsquote: 8/20 (40%)

Stufe	Passwort	Wiederherstellung	Ausgangswort:Regel
1	blueberry123	slughorn.rule	blueberry:$1 $2 $3
1	Backware99	M3	
1	payerbach	slughorn.rule	payerbach::
1	avocadoavocado	dive.rule	avocado:p1
2	Blueberry1991	dive.rule	blueberry:c $1 $9 $9 $1
2	f@ceb00k	M1	
2	pastalovers3	M2	
2	amazingitalyy	M1	
3	SWEETpotatoes	dive.rule	sweetpotatoes:T0 T4 T3 T2 T1
3	chee$ec@ke	dive.rule	cheesecake:sa@ ss$
3	baking!love	M2	
3	1999VERENA	dive.rule	verena:u ^9 ^9 ^9 ^1
4	cake_in_my_belly	M1	
4	lueberryb777	M2	
4	deliciouscandy_	M1	
4	1000cinnamonbuns	M1	
5	samohtsamoht34	M1a	
5	ihccong17	slughorn.rule	gnocchi:r $1 $7
5	t0m@t0es0up	M1	
5	tea>coffee!	M1	

Tabelle 5.5: Ergebnisse für Testperson 4

Testperson 5

- Alter: 18
- Geschlecht: weiblich
- Beruf: Schülerin
- Hobbys: Schminken, Mode, Veganismus, Reisen, Bloggen
- Beziehungsstand: ledig
- Sprache: Deutsch
- Quellen: Facebook, Twitter
- Länge der Wortliste: 9.655
- Erfolgsquote: 10/20 (50%)

Stufe	Passwort	Wiederherstellung	Ausgangswort:Regel
1	donauinselfest	slughorn.rule	donauinselfest::
1	eisenstadt4	slughorn.rule	eisenstadt:$4
1	instagirl	slughorn.rule	instagirl::
1	yogayoga	slughorn.rule	yoga:Z5 '6 O31] p1
2	travelbl0gger	M1	
2	sydney2012	slughorn.rule	sydney:l $2 $0 $1 $2
2	SNAPCHAT17	slughorn.rule	snapchat:u $1 $7
2	omomom123	M1	
3	veganlove3	M1	
3	i<3newyork	M2	
3	333jakob	M1a	
3	bloglovin666	dive.rule	bloglovin:$6 $6 $6
4	newyork<3me	M2	
4	nIKEFREE!	dive.rule	nikefree:$! C
4	viennacalling!	M1	
4	GANZWIEN16	M1	
5	ytuaeb98	slughorn.rule	beauty:r $9 $8
5	g1rlb0$$	dive.rule	girlboss:sa@ sc< se3 si1 so0 ss$
5	animalright$	M1	
5	friendsn0tf00d	M1	

Tabelle 5.6: Ergebnisse für Testperson 5

5.3 Erkenntnisse

Als erste Beobachtung kann festgestellt werden, dass die Erfolgsquote höher zu sein scheint, je mehr Einträge die extrahierte Wortliste enthält. So konnte bei *Testperson 2* mit einer Liste von 60.000 Einträgen das beste Ergebnis erzielt werden. Dieses zeigt die erwartete direkte Proportionalität der Werte *Größe der Wortliste* und *Erfolgsquote*. Einen Ausreißer bildet einzig *Testperson 4*, bei der die wenigsten Wörter extrahiert wurden, deren Erfolgsquote jedoch dennoch im Mittelfeld liegt.
Bezüglich der fünf Sicherheitsstufen ist ein Trend zu erkennen, demzufolge Passwörter mit einer niedrigeren Stufe eher erraten werden können. Bemerkenswert ist jedoch, dass etwa ein Viertel der erfolgreich wiederhergestellten Passwörter der höheren Stufe 4 oder sogar 5 sind. Nachdem die fünf Sicherheitsstufen nicht definiert und daher der persönlichen Einschätzung der Personen überlassen wurden, die die Passwörter erstellt haben, zeigt dies, dass die wahrgenommene Sicherheit oft nicht mit der tatsächlichen übereinstimmt (siehe Kapitel 5.1 und 2.1.3).
Weiters ist zu sehen, dass die von *slughorn* erstellte Regeldatei in 23% der Fälle zum Erfolg führte und in weiteren 18% ein weiterer Crackingprozess mit der von *hashcat* bereitgestellten Datei *dive.rule* das Passwort wiederherstellen konnte. Die Verteilung der Erfolgsquote von 41% bestätigt die in Kapitel 4.5 gefällte Entscheidung zuerst das kleine, schnelle und personalisierte Regelwerk anzuwenden und erst dann auf das große Regelwerk *dive.rule* zurückzugreifen.
Hinsichtlich der Mängel kann beobachtet werden, dass in über 50% der Fälle einer erfolglosen Wiederherstellung das Fehlen des verwendeten Wortes in der Wortliste (M1) als Grund festgestellt werden kann. Bei etwa einem Drittel dieser Fälle mangelte es an der Komplexität der verwendeten Regeln (M2). Jeweils unter 10% der negativen Ergebnisse waren durch die irrtümliche Trennung von Begriffen (M4) und die fehlende Verknüpfung von Profilen (M1a) bedingt. Ein Fehler in der Bestimmung der Grundform (M3) konnte nur in einem einzigen Fall festgestellt werden und ist daher als Mangel zu vernachlässigen.
Letztlich ist ein Vergleich zu Brute-Force Attacken zu ziehen, zu denen das vorgestellte Tool *slughorn* durch die automatisierte Generierung von personenbezogenen Passwortlisten eine schnellere Alternative bieten soll. Als Vergleichswert wurde die Dauer des Akquise- und Extraktionsprozesses mit durchschnittlich zehn Minuten und des Crackingprozesses mit durchschnittlich fünf Minuten festgehalten. Wie in Kapitel 2.1.4 dargelegt wurde, ist eine Brute-Force Attacke beim Hashalgorithmus *MD5* nur bis zu einer Länge von neun Zeichen praktikabel. Von den 100 zugewiesenen Passwörtern fallen 20 unter diese Länge (zwei der Länge sieben, sieben der Länge acht und elf der Länge neun), wobei nur die zwei Passwörter der Länge sieben mit etwa 5

Minuten für die Brute-Force Attacken schneller wiederhergestellt werden könnten als mit der personalisierten Passwortliste. Dabei ist zu betonen, dass die genannten Zeiten für eine Brute-Force Attacke der Tabelle 2.9 entnommen sind und mit einer deutlich höheren Rechenleistung bestimmt wurden als für die vorliegende Evaluierung zur Verfügung stand. Es kann daher festgestellt werden, dass die Verwendung von personenbezogenen Passwortlisten einen deutlichen zeitlichen Vorteil gegenüber Brute-Force Attacken ermöglicht und damit eine gute Alternative zu diesen bildet.

6 Diskussion

In diesem Kapitel gilt es, die Forschungsfrage aus Kapitel 1.1 zu beantworten. Weiters wird auf Aspekte eingegangen, die die Arbeit eingeschränkt haben und wie diese und weitere Themen in zukünftigen Arbeiten behandelt werden können, bevor die Arbeit abschließend rekapituliert wird.

6.1 Beantwortung der Fragestellung

Kapitel 1.1 stellt die Forschungsfrage der vorliegenden Arbeit, ob es möglich ist, personenbezogene Informationen durch Automatisierung sowohl zu sammeln als auch daraus Passwörter zu generieren, die den Kriterien typischer Passwortmuster entsprechen. Bei der Beantwortung gilt es darauf zu achten, die einzelnen Teilaspekte dieser Frage zu berücksichtigen: Einerseits geht es um die automatisierte Akquise personenbezogener Information, andererseits um eine automatisierte Weiterverarbeitung dieser zu Passwörtern. Im letzten Teil der Frage wird die Anforderung an diese Passwörter in Form typischer Passwortmuster definiert.
Wie im Zuge der theoretischen Vorarbeit in Kapitel 2.2 und der praktischen Umsetzung in Kapitel 4.3 zum Thema Informationsakquise gezeigt wurde, kann der erste Teil der Frage positiv beantwortet werden. In Kapitel 4.4 wurde beschrieben, wie Informationen in Form von Postings aus sozialen Netzwerken verarbeitet werden können und dies im Tool *slughorn* umgesetzt wurde. Auch der letzte Teil der Frage nach dem Entsprechen typischer Passwortmuster, die in Kapitel 2.1.2 erläutert wurden, kann bejaht werden.
Die positive Evaluierung aus Kapitel 5 bestätigt, dass alle drei Teilaspekte und damit die gesamte Forschungsfrage in der vorliegenden Arbeit behandelt und praktisch umgesetzt wurden. Es ist daher festzuhalten, dass es möglich ist, personenbezogene Informationen durch Automatisierung sowohl zu sammeln als auch daraus Passwörter zu generieren, die den Kriterien typischer Passwortmuster entsprechen.

6.2 Zukünftige Arbeiten

Die vorliegende Arbeit konzentriert sich aufgrund der zeitlichen Einschränkung auf einen kleinen Teil des Themengebiets der Passworterstellung mit Hilfe von personenbezogenen Informationen. Das entstandene Tool *slughorn* bildet, wie in Kapitel 4.1 erläutert wurde, einen Prototyp in Form einer *Proof of Concept* Implementierung und bedarf noch weiterer Entwicklung und Forschung, um ein vollständiges Softwareprodukt realisieren zu können. Aus diesen Gründen gilt es, im abschließenden Kapitel offengebliebene Aspekte aufzuzeigen, die Anlass für zukünftige Arbeiten zu diesem Thema geben und mit denen an die vorliegende Arbeit angeknüpft werden kann.

6.2.1 Weitere Quellen

In Kapitel 4.6 wurden Herausforderungen beschrieben, die bei der Entwicklung des Tools *slughorn* dazu geführt haben, bestimmte Aspekte des Themas unbeleuchtet zu lassen. In erster Linie gilt es dabei auf die eingeschränkte Auswahl an Quellen hinzuweisen: Mit Facebook und Twitter wurden zwei äußerst beliebte soziale Netzwerke gewählt, die sich ausgezeichnet dazu eignen, die Funktionsweise zu testen und das Tool als *Proof of Concept* zu implementieren. Die schlechte Verwendbarkeit der API von Twitter und die Einschränkungen jener von Facebook führten dazu, dass die Entwicklung der jeweiligen Scraper mit großem Aufwand verbunden war und keine weiteren Quellen praktisch bearbeitet werden konnten.
Im Zuge der Entwicklung des Tools *slughorn* wurde ein großes Augenmerk auf die Erweiterbarkeit der Software gelegt. Im Prototyp wurden die Quellen Facebook und Twitter umgesetzt, wobei dies nur einen kleinen Ausschnitt einer breiten Masse an sozialen Netzwerken bildet, für die eigene Scraper Klassen zu implementieren sind. Alexa [88] zählt in einer Auflistung der *top 500 sites on the web* in der Kategorie *Social Networking* unter anderem die folgenden Webseiten auf, die für eine Erweiterung des Tools in Frage kommen:

- LinkedIn
- Pinterest
- Google+
- LiveJournal
- Flickr
- fiverr
- Badoo

- Meetup
- OkCupid
- Hootsuite
- XING
- last.fm
- Foursquare

6.2.2 Verknüpfung von Profilen

Die Evaluierung in Kapitel 5 hat gezeigt, dass in einigen Fällen eine erfolgreiche Wiederherstellung des Passwortes möglich gewesen wäre, wenn Profile von nahestehenden Personen wie Verwandten oder engen Freunden miteinbezogen worden wären. In einer Weiterentwicklung des Tools ist zu erwägen, bis zu welchem Grad diese Verknüpfung positive Effekte auf das Ergebnis hätte und wie dies technisch umgesetzt werden könnte.

6.2.3 Empirische Studie für Gewichtung (Ausgefallenheit - Häufigkeit)

In Kapitel 4.4.6 wurde die Approximation der Gewichtung der Parameter *Ausgefallenheit* und *Häufigkeit* für die Berechnung des Scores angesprochen. Aufgrund des begrenzten Rahmens dieser Arbeit ist eine genauere Untersuchung dieses Zusammenhangs nicht möglich, weshalb dies Anlass für eine anschließende Studie gibt. Mit empirischen Untersuchungsmethoden kann mit Hilfe realer Passwörter und dazugehöriger Profile sozialer Netzwerke festgestellt werden, welche Gewichtung gewählt werden müsste, um an relevante Wörter der Extraktion einen höheren Score zu vergeben und sie somit in der Wortliste weiter oben einzustufen.
Die Beschaffung realer Daten stellt hier mit Sicherheit eine der größten Herausforderungen dar. Es konnte bereits in Arbeiten zum Thema Passwörter festgestellt werden, dass bei empirischen Studien, die auf der ad-hoc Generierung von Passwörtern durch Testpersonen beruhen, eine große Gefahr der Beeinflussung besteht [11, 12]. So ist darauf zu achten, dass Teilnehmerinnen und Teilnehmer realistische Passwörter erstellen und nicht durch die ungewohnte Situation bedingt stärkere und schwächere Passwörter generieren als sie es in einem normalen Umfeld zu tun pflegen.
Beim Arbeiten mit durch Hacks veröffentlichte Passwortlisten besteht diese Gefahr zwar nicht, jedoch ist dabei die Herausforderung gegeben, die Passwörter Accounts in sozialen Netzwerken zuzuordnen.

6.2.4 Spezialisierung auf weitere Sprachen

Im entstandenen Tool *slughorn* wurde der Fokus der Akquise und Extraktion auf die deutsche Sprache gelegt. So ist beim Account, der für den Scraper für das soziale Netzwerk Facebook verwendet wird darauf zu achten, dass Deutsch in den Einstellungen gewählt ist, da sonst gewisse Teile der HTML-Seite, die für die Navigation notwendig sind, nicht gefunden werden können. In einer Weiterentwicklung des Tools ist auf eine dynamische Erkennung der Spracheinstellungen zu achten, um die Software international verwendbar zu machen.
Weiters wurde in Kapitel 4.4.3 ein umgesetzter Bearbeitungsschritt von deutschen Wörtern beschrieben, bei dem grammatikalisch deklinierte Wörter in ihre Stammform zurückgeführt werden. Wenngleich die Notwendigkeit eines solchen Schrittes in der deutschen Sprache besonders gegeben ist, kann dies auch auf andere Sprachen zutreffen. Zudem ist zu recherchieren, welche weiteren sprachlichen Eigenheiten existieren, die einer zusätzlichen Bearbeitung bedürfen.

6.2.5 Weitere Verarbeitungsschritte von Begriffen

In Kapitel 4.4 wurde beschrieben, mit welchen Methoden die gesammelten Informationen der Zielperson bearbeitet werden, um relevante Wörter in einer sortierten Liste zu erhalten. Im Zuge der *Proof of Concept* Implementierung wurden diese beschriebenen Schritte als Funktionen umgesetzt. Die Notwendigkeit einer Erweiterung dieser Methoden kann in zukünftigen Arbeiten abgewogen werden. Naheliegende Schritte umfassen dabei eine genauere Filterung von unwichtigen Wörtern bevor diese in die Wortliste aufgenommen werden und das Hinzufügen von neuen Begriffskategorien wie Datum, URL, Name oder Hashtag zu den beiden bestehenden, Wort und Zahl.
Ein weiterer möglicher Verarbeitungsschritt, dessen Nutzen es zu evaluieren gilt, ist das Bilden von Assoziationen auf Basis von Wortähnlichkeiten. So ist es möglich, von einer häufigen Verwendung von Wörtern aus einem Themengebiet, wie etwa *kicken*, *Verein* und *Mittelstürmer* zum Oberbegriff *Fußball* zu gelangen. Die Berechnung des linguistischen Kontextes zwischen zwei oder mehreren Wörtern ist mit Hilfe der von Mikolov et al. vorgestellten Modelle *word2vec* möglich [89] und beruht auf *Machine Learning* Algorithmen.
Die Evaluierung in Kapitel 5 hat zudem gezeigt, dass die Trennung von zusammengehörigen Begriffen wie *Monty Python* zu separaten Wörtern in etwa 10% der Fällen negativen Einfluss auf die Wiederherstellung von Passwörtern gehabt hat, weshalb dieser Aspekt ebenfalls als verbesserungswürdig zu erachten ist.

6.2.6 Machine Learning

Eine abschließende Möglichkeit zur Verbesserung des in dieser Arbeit vorgestellten Verfahrens ist der Einsatz von *Machine Learning* Algorithmen in den Bereichen der Passwortgenerierung aus Wortlisten und Bewertung von Wörtern. Während diese Schritte derzeit nach zuvor fest definierten Regeln durchgeführt werden, ist es denkbar, eine künstliche Intelligenz zu trainieren, die Anhand realer Passwörter und dazugehörigen Profilen aus sozialen Netzwerken selbstständig jene Muster erkennt, die relevant sind und eine Extraktion und Reihung der wichtigen Wörter und daraus folgend eine Passwortgenerierung vornimmt.

6.3 Resümee

Im Zuge der vorliegenden Arbeit wurde der theoretische Hintergrund zu Passwörtern und der Akquise von personenbezogenen Informationen dargelegt. Die niedrigen Erfolgsaussichten und hohe Zeitintensität von klassischen Brute-Force Attacken erlauben die Annahme, dass eine automatisierte Generierung von personenbezogenen Passwortlisten eine aussichtsreiche Alternative bilden kann. Die erfolgreiche Umsetzung der Erkenntnisse und positive Evaluierung des Tools *slughorn* konnten diese These bestätigen. Damit wurde gezeigt, welche zusätzliche, vielleicht unerwartete Bedeutung personenbezogene Informationen im Internet haben können und Raum für zahlreiche zukünftige Arbeiten in diesem spannenden und relevanten Gebiet gegeben.

Akronymverzeichnis

API Application Programming Interface. 25, 26, 44, 47, 51, 64, 66, 82

ARP Address Resolution Protocol. 34

CeWL Custom Word List generator. 31

CLI Command Line Interface. 39–42, 44, 61, 67, 89

CUPP Common User Passwords Profiler. 21

DES Data Encryption Standard. 6

HTML Hypertext Markup Language. 45, 49, 84

HTTP Hypertext Transfer Protocol. 45

HTTPS Hypertext Transfer Protocol Secure. 48

ISO International Organization for Standardization. 46, 52

JSON JavaScript Object Notation. 45, 49

MD5 Message Digest 5. 18, 19, 35, 69, 70

MITM Man-in-the-Middle. 34

NLTK Natural Language Toolkit. 31, 46, 54, 56

NTLM NT LAN Manager. 35

PDF Portable Document Format. 18

PIN Personal identification number. 13

POS Part-of-speech. 46, 47, 54

PRTK Passwort Recovery Tool Kit. 29, 34

SAM Security Account Manager. 34

SHA256 Secure Hash Algorithm. 18, 35

UML Unified Modeling Language. 38–40, 51

UNIX Uniplexed Information and Computing Service. 6

URL Uniform Resource Locator. 49, 84

WLAN Wireless Local Area Network. 34

XML Extensible Markup Language. 45

Abbildungsverzeichnis

2.1 Dauer, alle Kombinationen zu probieren, abhängig von der Länge des Passwortes 20

4.1 Logo des Scrapingtools *slughorn* 37
4.2 UML-Klassendiagramm 40
4.3 UML-Sequenzdiagramm 41
4.4 Abfolge der Subprozesse mit Zwischenspeicherungen 43
4.5 UML-Aktivitätsdiagramm über den Prozess der Wortextraktion 51
4.6 Histogramm der Häufigkeit von extrahierten Wörtern im Sprachgebrauch . . . 57
4.7 Histogramm der Zipfhäufigkeit von extrahierten Wörtern im Sprachgebrauch . 57
4.8 Kombinierter Score bei gleicher Gewichtung der Parameter 61

Tabellenverzeichnis

2.1 Notation für Zeichengruppen nach Chou et al. 8
2.2 Beispiele für Standardwörter 9
2.3 Beispiele für Buchstabensubstitutionen 10
2.4 Beispiele für Standardwörter mit Substitution und/oder Anhang 10
2.5 Beispiele für mnemonische Passwörter 12
2.6 Beispiele für Zufallspasswörter 13
2.7 Beispiele für einen Übersetzungsmechanismus des Kryptonizers 14
2.8 Anzahl verschiedener Kombinationen abhängig von Passwortlänge 18
2.9 Dauer um alle Kombinationen auszuprobieren abhängig von Passwortlänge . . 19

4.1 Optionen des CLI 42
4.2 Unicode Whitelist 52
4.3 Beziehung der relativen Häufigkeit zur Zipfhäufigkeit 58
4.4 Regeln nach denen Zahlen mit Wörtern kombiniert werden 62
4.5 Performanceanalyse der Akquise- und Verarbeitungsschritte 64

5.1 Beschreibung der Mängelkategorien 69
5.2 Ergebnisse für Testperson 1 70
5.3 Ergebnisse für Testperson 2 71
5.4 Ergebnisse für Testperson 3 72
5.5 Ergebnisse für Testperson 4 73
5.6 Ergebnisse für Testperson 5 74

Auflistungsverzeichnis

1 Fragebogen von CUPP (Ausschnitt) 33

2 Beispielbefehl mit Optionen 42

3 Algorithmus für das Auslesen von Tweets 49

4 Deutsche Stoppwörter 55

5 Zusätzliche deutsche Stoppwörter 55

6 Befehle für *hashcat* 68

Literaturverzeichnis

[1] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[2] Marcus K Rogers and Kate Seigfried. The future of computer forensics: a needs analysis survey. *Computers & Security*, 23(1):12–16, 2004.

[3] M Al Fahdi, Nathan L Clarke, and Steven M Furnell. Challenges to digital forensics: A survey of researchers & practitioners attitudes and opinions. In *Information Security for South Africa, 2013*, pages 1–8. IEEE, 2013.

[4] Darren Quick and Kim-Kwang Raymond Choo. Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294, 2014.

[5] Polybius. *The Histories of Polybius: Translated from the Text of F. Hultsch*, volume 1 of *Cambridge Library Collection - Classics*. Cambridge University Press, 2012.

[6] M. Bando. *101st Airborne: The Screaming Eagles in World War II*. MBI Publishing Company, 2011.

[7] David C Walden and Tom Van Vleck. *The compatible time sharing system (1961-1973): Fiftieth anniversary commemorative overview*. IEEE Computer Society, 2011.

[8] Fernando J Corbató, RJ Creasy, RC Daley, and MM Daggett. *The Compatible Time-Sharing System: A Programmer's Guide*, volume 45. Mit Press Cambridge, Mass., 1963.

[9] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[10] Gilbert Notoatmodjo and Clark Thomborson. Passwords and perceptions. In *Proceedings of the Seventh Australasian Conference on Information Security-Volume 98*, pages 71–78. Australian Computer Society, Inc., 2009.

[11] SM Haque, Matthew Wright, and Shannon Scielzo. A study of user password strategy for multiple accounts. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 173–176. ACM, 2013.

[12] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.

[13] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[14] Susan Wiedenbeck, Jim Waters, Jean-Camille Birget, Alex Brodskiy, and Nasir Memon. Authentication using graphical passwords: Effects of tolerance and image choice. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 1–12. ACM, 2005.

[15] Hsien-Cheng Chou, Hung-Chang Lee, Hwan-Jeu Yu, Fei-Pei Lai, Kuo-Hsuan Huang, and Chih-Wen Hsueh. Password cracking based on learned patterns from disclosed passwords. *IJICIC*, 2013.

[16] Katherine Blashki and Sophie Nichol. Game geek's goss: linguistic creativity in young males within an online university forum (94/\/\3 933k'5 9055oneone). *Australian journal of emerging technologies and society*, 3(2):71–80, 2005.

[17] Xavier de Carné de Carnavalet and Mohammad Mannan. From very weak to very strong: Analyzing password-strength meters. In *NDSS*, volume 14, pages 23–26, 2014.

[18] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. Password memorability and security: Empirical results. *IEEE Security & privacy*, 2(5):25–31, 2004.

[19] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human selection of mnemonic phrase-based passwords. In *Proceedings of the second symposium on Usable privacy and security*, pages 67–78. ACM, 2006.

[20] Johannes Kiesel, Benno Stein, and Stefan Lucks. A large-scale analysis of the mnemonic password advice. In *Proceeding of NDSS*, 2017.

[21] Morrie Gasser. A random word generator for pronounceable passwords. Technical report, MITRE CORP BEDFORD MA, 1975.

[22] Unicode 6.0.0. http://unicode.org/versions/Unicode6.0.0/. Aufgerufen am 13. November 2017.

[23] Maximilian Golla, Dennis Detering, and Markus Dürmuth. Emojiauth: quantifying the security of emoji-based authentication. In *Proceedings of the Usable Security Mini Conference (USEC)*, 2017.

[24] Kryptonizer Ihr individueller Übersetzer für Krypto-Passwörter. http://www.kryptonizer.de/hintergrund/. Aufgerufen am 13. November 2017.

[25] Martin MA Devillers. Analyzing password strength. *Radboud University Nijmegen, Tech. Rep*, 2, 2010.

[26] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 523–537. IEEE, 2012.

[27] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. Adaptive password-strength meters from markov models. In *NDSS*, 2012.

[28] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[29] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[30] Diceware German - Benjamin Tenne. http://world.std.com/~reinhold/diceware_german.txt. Aufgerufen am 20. Februar 2018.

[31] Brutalis. https://sagitta.pw/hardware/gpu-compute-nodes/brutalis/. Aufgerufen am 22. März 2018.

[32] 8x Nvidia GTX 1080 Hashcat Benchmarks. https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40. Aufgerufen am 22. März 2018.

[33] Twitter Blog Tweeting Made Easier. https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html. Aufgerufen am 5. Januar 2018.

[34] Twitter Help How to Tweet. https://help.twitter.com/en/using-twitter/how-to-tweet. Aufgerufen am 5. Januar 2018.

[35] Twitter Help About public and protected Tweets. https://help.twitter.com/en/safety-and-security/public-and-protected-tweets. Aufgerufen am 5. Januar 2018.

[36] TechCrunch Twitter Data Analysis: An Investor's Perspective. https://techcrunch.com/2009/10/05/twitter-data-analysis-an-investors-perspective-2/. Aufgerufen am 5. Januar 2018.

[37] Twoblog The last 100 million Twitter accounts. http://www.twopblog.com/2012/05/last-100-million-twitter-accounts.html. Aufgerufen am 5. Januar 2018.

[38] Twitter Developers Search Tweets. https://developer.twitter.com/en/docs/tweets/search/overview. Aufgerufen am 5. Januar 2018.

[39] Facebook - Info. https://www.facebook.com/pg/facebook/about/. Aufgerufen am 16. Februar 2018.

[40] Facebook - Datenrichtlinie. https://www.facebook.com/privacy/explanation. Aufgerufen am 16. Februar 2018.

[41] Facebook - Die Graph API. https://developers.facebook.com/docs/graph-api/. Aufgerufen am 16. Februar 2018.

[42] Khawla Al-Wehaibi, Tim Storer, and William Bradley Glisson. Augmenting password recovery with online profiling. *digital investigation*, 8:S25–S33, 2011.

[43] Grigorios Fragkos and Theodore Tryfonas. A cognitive model for the forensic recovery of end-user passwords. In *Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on*, pages 48–54. IEEE, 2007.

[44] DigiNinja. https://digi.ninja. Aufgerufen am 15. Februar 2018.

[45] CeWL. https://github.com/digininja/CeWL/. Aufgerufen am 15. Februar 2018.

[46] SmeegeScrape. http://www.smeegesec.com/2014/01/smeegescrape-text-scraper-and-custom.html. Aufgerufen am 15. Februar 2018.

[47] SmeegeSec. http://www.smeegesec.com/. Aufgerufen am 15. Februar 2018.

[48] Crunch. https://sourceforge.net/projects/crunch-wordlist/. Aufgerufen am 15. Februar 2018.

[49] Kali. https://www.kali.org/. Aufgerufen am 15. Februar 2018.

[50] CUPP. https://github.com/Mebus/cupp. Aufgerufen am 15. Februar 2018.

[51] Cain & Abel. http://www.oxid.it/cain.html. Aufgerufen am 15. Februar 2018.

[52] PRTK. https://accessdata.com/product-download/password-recovery-toolkit-prtk-version-7.6.0. Aufgerufen am 15. Februar 2018.

[53] John the Ripper password cracker. http://www.openwall.com/john/. Aufgerufen am 15. Februar 2018.

[54] hashcat - advanced password recovery. https://hashcat.net/hashcat/. Aufgerufen am 15. Februar 2018.

[55] THC Hydra. https://github.com/vanhauser-thc/thc-hydra. Aufgerufen am 15. Februar 2018.

[56] Brutus. http://www.hoobie.net/brutus/. Aufgerufen am 15. Februar 2018.

[57] Medusa Parallel Network Login Auditor. http://foofus.net/goons/jmk/medusa/medusa.html. Aufgerufen am 15. Februar 2018.

[58] pickle Python object serialization. https://docs.python.org/3/library/pickle.html. Aufgerufen am 12. Februar 2018.

[59] The Python Standard Library. https://docs.python.org/3/library/index.html. Aufgerufen am 12. Februar 2018.

[60] Click. http://click.pocoo.org/5/. Aufgerufen am 12. Februar 2018.

[61] click-spinner. https://github.com/click-contrib/click-spinner. Aufgerufen am 12. Februar 2018.

[62] Facebook SDK for Python. https://facebook-sdk.readthedocs.io/en/latest/. Aufgerufen am 12. Februar 2018.

[63] Einführung der Suche im Social Graph. https://www.facebook.com/notes/facebook/einf%C3%BChrung-der-suche-im-social-graph/569087396437436/. Aufgerufen am 12. Februar 2018.

[64] Beautiful Soup. https://www.crummy.com/software/BeautifulSoup/. Aufgerufen am 12. Februar 2018.

[65] Requests. http://docs.python-requests.org/. Aufgerufen am 15. März 2018.

[66] fake-useragent. https://github.com/hellysmile/fake-useragent. Aufgerufen am 15. März 2018.

[67] User Agent String.Com. http://useragentstring.com/. Aufgerufen am 15. März 2018.

[68] w3schools - Browser Statistics. https://www.w3schools.com/browsers/default.asp. Aufgerufen am 15. März 2018.

[69] lxml - XML and HTML with Python. http://lxml.de/. Aufgerufen am 15. März 2018.

[70] Selenium. http://selenium-python.readthedocs.io/. Aufgerufen am 12. Februar 2018.

[71] fastText. https://fasttext.cc/. Aufgerufen am 12. Februar 2018.

[72] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[73] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[74] pycountry. https://pypi.python.org/pypi/pycountry. Aufgerufen am 12. Februar 2018.

[75] NLTK Natural Language Toolkit. http://www.nltk.org/. Aufgerufen am 12. Februar 2018.

[76] ClassifierBasedGermanTagger. https://github.com/ptnplanet/NLTK-Contributions/tree/master/ClassifierBasedGermanTagger. Aufgerufen am 12. Februar 2018.

[77] GermaLemma. https://github.com/WZBSocialScienceCenter/germalemma. Aufgerufen am 12. Februar 2018.

[78] pyphen. http://pyphen.org/. Aufgerufen am 12. Februar 2018.

[79] Robert Speer, Joshua Chin, Andrew Lin, Sara Jewett, and Lance Nathan. Luminosoinsight/wordfreq: v1.7, September 2017.

[80] Winfried Ulrich. Wörterbuch. 2002.

[81] Patrick Brandt, Rolf-Albert Dietrich, and Georg Schön. Sprachwissenschaft. ein roter faden für das studium der deutschen sprache. 2006.

[82] TIGER Corpus. http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html. Aufgerufen am 12. Februar 2018.

[83] Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. Tiger: Linguistic interpretation of a german corpus. *Research on language and computation*, 2(4):597–620, 2004.

[84] Walter JB Van Heuven, Pawel Mandera, Emmanuel Keuleers, and Marc Brysbaert. Subtlex-uk: A new and improved word frequency database for british english. *Quarterly Journal of Experimental Psychology*, 67(6):1176–1190, 2014.

[85] George Kingsley Zipf. *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books, 2016.

[86] Selim Aksoy and Robert M Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern recognition letters*, 22(5):563–582, 2001.

[87] FacebookBot auf GitHub. https://github.com/hikaruAi/FacebookBot. Aufgerufen am 15. Februar 2018.

[88] Alexa - Top 500 sites on the web, by category Social Networking. https://www.alexa.com/topsites/category/Computers/Internet/On_the_Web/Online_Communities/Social_Networking. Aufgerufen am 22. März 2018.

[89] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.