Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Philipp Staib

Monitoring von Informationsportalen zur Erkennung von kompromittierten Organisationen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Philipp Staib

Monitoring von Informationsportalen zur Erkennung von kompromittierten Organisationen

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

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

Betreuender Prüfer : Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter : Prof. Dr.-Ing Olaf Zukunft

Abgegeben am 06.09.2016

Philipp Staib

Thema der Arbeit

Monitoring von Informationsportalen zur Erkennung von kompromittierten Organisationen

Stichworte

IT Sicherheit, Monitoring, Informationsportale, kompromittierte Organisationen

Kurzzusammenfassung

Ziel der Bachelorarbeit ist die Entwicklung einer Anwendung, welche zum Monitoring von Informationsportalen zur Erkennung von kompromittierten Organisationen eingesetzt werden kann. Im Konkreten bezieht sich die Anwendung auf das Informationsportal *Pastebin*, welches nach Mustern, die vom Anwender bereitgestellt werden, durchsucht wird. Der Monitoring-Prozess soll in Form eines Prototyps umgesetzt werden. Bei Erkennung einer kompromittierten Organisation soll ein Alarm ausgelöst werden. Die Verarbeitung solcher Alarme beziehungsweise die Information betroffener Organisationen ist anschließend Aufgabe nachgelagerter Prozesse.

Philipp Staib

Title of the paper

Monitoring of information portals for the detection of compromised organisations

Keywords

IT security, monitoring, information portals, compromised organisations

Abstract

The objective of this bachelor thesis is the development of an application for monitoring information portals in order to detect compromised organisations. In particular, the application relates to the information portal *Pastebin*, which is searched for user-specified patterns. The monitoring process is implemented in the form of a prototype. Detecting a compromised organization triggers an alarm. The processing of such an alarm and the information of the corresponding organisation is subsequently handled by downstream mechanisms.

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel und Motivation der Arbeit

Heutzutage sind Informationen beziehungsweise Daten von existenzieller Bedeutung, es handelt sich um schützenswerte Güter. Ob es nun Daten sind, welche im Besitz einer Organisation sind, beispielsweise Kundendaten einer Versicherung oder Daten über eine Organisation, welche etwa potenzielle Bedrohungen erhöhen und somit die Wahrscheinlichkeit eines erfolgreichen Angriffs steigern, wie die interne Netzwerktopologie oder das Sicherheitskonzept. Auch Daten, die an sich keine große Relevanz haben, jedoch auf den Datenabfluss innerhalb der Organisation schlussfolgern lassen, sind von Bedeutung. Eine Vertraulichkeitsverletzung dieser Daten oder eine ungewollte Veröffentlichung kann gravierende Folgen haben. Diese können finanziellen Schaden mit sich bringen, sie können juristischer Natur sein oder den Ruf und das Vertrauen gegenüber einer Organisation in Gefahr bringen. Es verlieren heute nach einer Schätzung des Bundesamtes für Sicherheit in der Informationstechnik deutsche Unternehmen pro Jahr noch immer 20 Milliarden Euro durch Wirtschaftsspionage (1). Schaden zu verhindern beziehungsweise zu mindern, ist Aufgabe der IT Sicherheit.
Überwiegend wird versucht, durch Firewalls, Intrusion Detection Systeme oder andere Sicherheitsvorkehrungen den Datenstrom zu kontrollieren und ungewollten Datenabfluss zu verhindern. Jedoch gelingt es Angreifern immer wieder -- an jeglichen Sicherheitssystemen vorbei -- an Daten zu gelangen beziehungsweise diese zu entwenden. Folglich ist es nicht der Ansatz dieser Arbeit, die Art der Informationsgewinnung auf zu klären oder potenziellen Angriffen vorzubeugen, sondern Portale nach sensiblen Daten zu monitoren. Damit kann nicht der Grund für einen Datenverlust gefunden werden, aber es kann belegt werden, dass eine Vertraulichkeitsverletzung vorliegt und die Organisation kompromittitert ist. Dieses Vorgehen ist besonders hilfreich, um Angriffe schnell zu entdecken, womit der

Schaden beziehungsweise die Folgen gemindert werden können (2). In dieser Arbeit werden ein Konzept und ein Prototyp erstellt, wobei der Prototyp zeigen soll, dass das Konzept realisierbar ist.

1.2 Zielgruppe der Arbeit

Diese Arbeit ist an Personen gerichtet, welche im IT Sicherheitsbereich tätig sind beziehungsweise Interesse an diesem Gebiet haben. Damit sind sowohl IT Sicherheitsbeauftragte von Unternehmen und Organisationen als auch Studenten gemeint. Zudem gehören zur Zielgruppe der Arbeit Datenschutzbeauftragte und Personen, welche für die Sicherheit, insbesondere den Schutz der Vertraulichkeit von Daten verantwortlich sind.

1.3 Abgrenzung

In der Arbeit geht es nicht darum, automatisch Muster zu erkennen, nach denen gesucht werden kann. Es wird nach vorgegebenen Mustern gesucht. Des Weiteren geht es auch nicht darum, dynamisch neue Informationsportale aufzuspüren. Die automatische Erkennung von potentiellen Suchmustern und die intelligente Suche nach neuen Informationsportalen liefern Ansatzpunkte für weiterführende Untersuchungen.

1.4 Gliederung

Nach einer Einführung in die Thematik erfolgt im zweiten Kapitel, dem Grundlagenkapitel, eine Erläuterung zur Relevanz der Anwendung im Kontext der IT Sicherheit. Es wird auf ein Beispiel eingegangen, in dem Informationsportale zur Veröffentlichung von entwendeten Daten eingesetzt werden. Außerdem werden in diesem Kapitel die Algorithmen erläutert, welche zum Aufspüren von Mustern verwendet werden. Anschließend wird auf Dateiformate eingegangen wobei der Unterschied zwischen Textdateien und Binärdateien aufgezeigt wird. Zudem werden verwandte Arbeiten beziehungsweise ähnliche Software-Lösungen betrachtet. Im dritten Kapitel werden die Anforderungsanalyse der Anwendung, sowie deren Spezifikation erläutert. Weiter werden die Architekturentscheidungen vorgestellt

und diskutiert. Abschließend wird in diesem Kapitel die Implementierung mit entsprechender Dokumentation beschrieben. Im vierten Kapitel wird das Testen der implementierten Anwendung dargestellt. Hier wird der Versuchsaufbau beschrieben und das Testergebniss dokumentiert und bewertet. Mit dem fünften Kapitel folgt eine abschließende Betrachtung in Form einer Zusammenfassung. Hier werden im Rahmen der Diskussion die Möglichkeiten weiterführender Arbeiten aufgezeigt.

1.5 Definitionen

Nachfolgend werden grundlegende Begriffe definiert, welche für das Verständnis der Arbeit dienlich sind.

Klassische IT Schutzziele

Integrität: „Die Daten sind vollständig und unverändert. Der Begriff „Information“ wird in der Informationstechnik für „Daten“ verwendet, denen je nach Zusammenhang bestimmte Attribute wie z. B. Autor oder Zeitpunkt der Erstellung zugeordnet werden können. Der Verlust der Integrität von Informationen kann daher bedeuten, dass diese unerlaubt verändert wurden oder Angaben zum Autor verfälscht wurden oder der Zeitpunkt der Erstellung manipuliert wurde“ (14, S. 14).

Verfügbarkeit: „Dem Benutzer stehen Dienstleistungen, Funktionen eines IT-Systems oder auch Informationen zum geforderten Zeitpunkt zur Verfügung“ (14, S. 14).

Vertraulichkeit: „Vertrauliche Informationen müssen vor unbefugter Preisgabe geschützt werden“ (14, S. 14).

Merkmale der Softwarequalität (nicht-funktionale Anforderung)

Performance: Performance beschreibt den Durchsatz und die Antwortzeit einer Anwendung. Durchsatz meint die zu bewältigende Arbeitslast im Bezug zur Zeit. Antwortzeit ist die Zeit zwischen einer Anfrage an das System und der Antwort des Systems (3).

Testbarkeit: Testbarkeit ist die Fähigkeit, die korrekte Funktionsweise eines Systems beweisen zu können. Hierzu zählt auch die Reproduzierbarkeit von Fehlern (3).

Verfügbarkeit: Verfügbarkeit definiert die absolute oder relative Zeit, in der ein System einsatzbereit sein muss. Verfügbarkeit sagt etwas aus über die Wahrscheinlichkeit einer Leistungserbringung in einer definierten Zeit. Hierzu zählt auch die Wiederherstellungszeit eines Systems nach einem Fehlerfall (3).

Bedienbarkeit: Bedienbarkeit beschreibt wie leicht ein System erlernt und effizient genutzt werden kann. Darüber hinaus soll der Nutzer vor fehlerhaften Eingaben geschützt werden (3).

Änderbarkeit: Änderbarkeit beschreibt die Möglichkeit, eine Software schnell und einfach an geänderte Bedürfnisse anzupassen. Ein hoher Grad an Modularisierung und lose Kopplung von Komponenten erleichtert dieses Reagieren auf Veränderungen. Ein Teilaspekt dabei ist Portabilität, sie beschreibt die Möglichkeit, ein System auf eine andere Umgebung zu portieren (3).

Skalierbarkeit: Skalierbarkeit ist ein Maßstab dafür, wie gut ein System seine Aufgaben bei steigender Arbeitslast vollrichten und sich an ein wachsendes Problem anpassen kann. Es wird zwischen horizontaler und vertikaler Skalierbarkeit unterschieden. Bei horizontaler Skalierbarkeit wird die Last auf mehrere Rechner verteilt. Hingegen wird bei vertikaler Skalierbarkeit die einzelne Maschine durch Erhöhung der Ressourcen leistungsfähiger gemacht (3).

NoSql Datenbanken

Key-Value-Store: Diese Datenbankstruktur besteht aus Schlüssel Wert Paaren. Die Werte können sowohl strukturiert als auch unstrukturiert sein. Der Zugriff erfolgt über einen eindeutigen Schlüssel. Key Value Stores zeichnen sich durch sehr gute Skalierbarkeit aus (4).

Document-Store: Document-Stores sind Datenbanksysteme, welche kein Schema besitzen, das die gesamte Datenbank umfasst. Auch hier werden Schlüssel Wert Paare gespeichert, jedoch können die abgespeicherten Dokumente unterschiedlicher Natur sein. Dadurch können komplexe Datenstrukturen gespeichert werden (4).

Wide-Column-Stores: Wide-Column-Stores sind spaltenorientierte Datenbanksysteme. Hierbei werden Einträge nicht nach ihren Zeilen gruppiert, sondern nach ihren Spalten (4).

Graph-Datenbanken: Graph-Datenbanken speichern die Daten und deren Relationen zueinander in Form von Knoten und Kanten ab. Dadurch wird es erleichtert, Beziehungen zwischen Daten mit hoher Komplexität abzubilden (4).

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der zu entwickelnden Software erläutert. Es geht darum, die Wichtigkeit der Anwendung im Kontext der IT Sicherheit aufzuzeigen. In diesem Zusammenhang wird erklärt was textbasierte Informationsportale sind. Es wird ein Beispiel für einen Hackerangriff aufgegriffen, bei dem Informationsportale zur Veröffentlichung der entwendeten Daten eingesetzt werden. Des Weiteren wird Patternmatching erläutert, was zur Mustererkennung in Texten zum Einsatz kommt. Anschließend wird auf Dateiformate eingegangen und dabei der Unterschied zwischen Textdateien und Binärdateien erläutert. Außerdem werden verwandte Arbeiten beziehungsweise ähnliche Software-Lösungen betrachtet.

2.1 Schwachstellen und Bedrohungen

Einem erfolgreichen Angriff geht meistens die Beschaffung von Informationen, über das Angriffsziel voraus. Was durch das Durchführen von Penetrationstests, was reales Angriffsvorgehen nachstellt, gezeigt wird. Wenn man zum Beispiel im Besitz von internen IP Adressen ist, kann man dadurch genaue Informationen über die vorhandenen Schwachstellen des Systems erlangen. Das bedeutet, je mehr über eine Organisation bekannt ist, umso wahrscheinlicher ist ein erfolgreicher Angriff auf die klassischen IT Schutzziele (5). Deshalb ist es für Organisationen extrem wichtig, schnellst möglich Kenntnis darüber zu erlangen, dass sich Informationen auf öffentlichen Informationsportalen befinden, welche zur Vorbereitung eines Angriffs dienen können. Dadurch ist es möglich, Gegenmaßnahmen einzuleiten und bekannte Schwachstellen zu schließen und damit potenzielle Bedrohungen zu vermindern.

2.2 Portale mit Informationen über Opfer

Es gibt zahlreiche Webseiten, welche dem Austausch von Textdateien dienen, sogenannte Textsharing Portale. Beispiele hierfür sind *Pastebin*, *Pastie*, *FrubarPaste*, *Codepad* und *Slexy*. Pastebin wurde erstellt, um Programmierern das Teilen von Quellcode zu erleichtern. Gegenwärtig ist Pastebin eines der beliebtesten Portale, um Textdateien zu veröffentlichen. Jedoch wird es nicht ausschließlich zum Teilen erlaubter Inhalte genutzt. Unter den öffentlich zugänglichen Beiträgen findet man nicht selten Datenbankauszüge oder private IP Adressen von kompromittierten Organisationen mit zugehörigen Nutzernamen und Passwörtern oder gestohlene Kreditkartendaten. Diese Daten können einen erfolgreichen Angriff beweisen und auf diesem Wege zum Kauf gesamter Datensätze animieren oder sie können zur Aufklärung im Rahmen eines Penetrationstests genutzt werden (6). Gründe dafür, dass Pastebin bevorzugt zur Veröffentlichung entwendeter Daten genutzt wird, sind:

- eine einfache Verwendung
- zum Veröffentlichen ist keine Anmeldung notwendig
- ein guter Umgang mit großen Textdateien
- Beiträge werden nicht im Voraus kontrolliert und
- es ist eine anonyme Veröffentlichung möglich.

Da Textsharingportale bei Angreifern immer beliebter werden, ist es enorm wichtig, ein Monitoring dieser durchzuführen, um eine Kompromittierung aufzudecken (7).

2.3 Hackerangriff auf Target

Target zählt zu den größten nordamerikanischen Einzelhandelsketten und wurde im November 2013 Opfer von einem Hackerangriff. Der Vorfall galt damals als einer der größten Hackerangriffe überhaupt. Der Angriff dauerte mehrere Wochen, wobei es den Angreifern gelungen ist, 40 Millionen Daten von Kredit- und Bankkarten sowie Postadressen, Telefonnummern und E-Mail-Adressen von 70 Millionen Kunden zu stehlen. Durch den Vorfall ist ein großer Vertrauensverlust für das Unternehmen entstanden, was nach sich gezogen hat, dass die Verkaufszahlen spürbar zurückgegangen sind. Darüber hinaus gab das Unternehmen 61 Millionen Dollar für die Aufklärung des Vorfalls aus (8). Im gesamten Schlussquartal gingen die Verkäufe um 2,5 Prozent zurück. Es mussten sogar acht Filialen geschlossen werden. Insgesamt kostete der Datendiebstahl Target in den Jahren 2013 und 2014 brutto 252 Millionen Dollar (2).

Noch bevor Target den Vorfall öffentlich gemacht hat, berichtete der IT-Sicherheitsexperte Brian Krebs darüber. Auf einschlägigen Portalen wurden bereits Millionen der gestohlenen Daten zum Kauf angeboten, obwohl der Angriff noch lief und die Angreifer sich noch weiterhin unbemerkt im Unternehmensnetz aufhielten (9). Die Hacker brauchten weniger als einen Tag, um sich im Unternehmensnetz einzunisten. Jedoch reagierte Target erst mehrere Wochen später. Das zeigt die große Verzögerung zwischen dem Eindringen und der Entdeckung von Angriffen, was für die betroffenen Unternehmen ein großes Problem darstellt. Hätte in diesem Fall ein Monitoring der eingesetzten Portale stattgefunden, wäre es möglich gewesen, den Angriff wesentlich früher aufzudecken. Dies wäre bereits möglich gewesen, als der Verkauf der Daten begann. Ab diesem Zeitpunkt hätten entsprechende Maßnahmen gegen den Angriff und deren Folgen eingeleitet werden können. Dadurch wären das Ausmaß des Angriffs und der dadurch verursachte Schaden für das Unternehmen erheblich geringer ausgefallen (2).

2.4 Patternmatching

Für das Verständnis der zu implementierenden Anwendung ist es unabdingbar, die unterschiedlichen Formen des Patternmatching darzustellen.

2.4.1 Definition Patternmatching

“Pattern Matching ist der Vorgang, bei dem die Existenz eines gegebenen Musters (Pattern) überprüft oder die Qualität der Gleichheit zum gegebenen Muster bewertet wird” (11, S. 1).

Anhand der Definition lässt sich erkennen, dass Patternmatching in zwei Kategorien unterteilbar ist, welche im Folgenden erklärt werden.

Weiche Vergleiche

Weiche Vergleiche geben Auskunft über die Qualität der Gleichheit. Es kann zum Beispiel ein Muster mit einer Menge an Daten verglichen werden. Dabei kann zu jedem Datum der Grad der Gleichheit festgestellt werden (10).

Harte Vergleiche

Harte Vergleiche machen eine exakte Aussage darüber, ob ein Muster gefunden wird oder nicht. Dadurch kann ein Teilstring in einem String nachgewiesen werden (10).

Im Folgenden werden einige Algorithmen zum exakten Patternmatching kurz vorgestellt und erläutert. Jedoch ist dies nur ein Ausschnitt aller Algorithmen und es wird kein Anspruch auf Vollständigkeit erhoben.

2.4.2 Algorithmen

Brute-Force Algorithmus

Hierbei handelt es sich um einen naiven und den einfachsten Algorithmus. Es wird für jede Position im Text überprüft, ob eine Übereinstimmung beginnend mit dieser Position vorliegt. Die Laufzeit beträgt im schlimmsten Fall O(m*n). Die durchschnittliche Laufzeit beträgt O(n+m), wobei m der Länge des Musters und n der Länge des Textes entspricht (11).

Beispiel:

Text	aaaaabaa
Pattern	aab
Vergleiche	aab
	aab
	aab
	aab
	aab

Knuth-Morris-Pratt Algorithmus

Der Ansatz ist ähnlich dem Brute-Force Algorithmus, jedoch werden die beim Missmatch bereits verglichenen Zeichen berücksichtigt. Dadurch entsteht der Vorteil, dass der nachfolgende Vergleich nicht wie beim Brute-Force Algorithmus in dem um 1 erhöhten Index des vorherigen Startindex beginnt, sondern ein Shift um mehrere Zeichen möglich ist. In diesem Verfahren beträgt durch den Wegfall einiger unnötiger Vergleiche die Laufzeit im schlimmsten Fall O(n+m) und im Durchschnitt O(n). Dabei entspricht m der Länge des Musters und n der Länge des Textes (11).

Boyer-Moore-Algorithmus

Bei diesem Verfahren beginnt der Vergleich des Patterns von hinten. Ähnlich wie beim Brute-Force Algorithmus wird Zeichen für Zeichen verglichen, jedoch wird der Index dekremtiert und nicht inkrementiert. Falls es zu einem Missmatch kommt, kann das Pattern so lange nach rechts verschoben werden, bis das im Text vorkommende Zeichen im Muster vorkommt. Falls das Zeichen im Text gar nicht im Muster vorkommt, kann das Muster um seine eigene Länge verschoben werden. Der Boyer-Moore Algorithmus besitzt eine deutlich kürzere Laufzeit als das Brute-Force oder Knuth-Morris-Pratt Vorgehen. Da dieser, auch nicht erfolgreiche Vergleiche zur Verbesserung nutzt. Die Laufzeit im schlimmsten Fall beträgt O(n+m), jedoch beträgt die durchschnittliche Laufzeit O(n/m), wobei m der Länge des Musters und n der Länge des Textes entspricht (11).

Beispiel:
Text abcdefghallo
Pattern hallo
Vergleiche hallo
hallo
hallo

Nach dem ersten Misstmatch wird das Muster um seine gesamte Länge verschoben, da im Muster kein “d” vorkommt. Beim zweiten Misstmatch ist im Text ein “a”, welches auch im Muster vorkommt. Daher wird das Muster bis zum “a” im Muster verschoben.

In der folgenden Abbildung werden die Laufzeiten der Algorithmen übersichtlich dargestellt.

Algorithmus	worst case	average case
Brute-Force	O(m*n)	O(n+m)
Knuth-Morris-Pratt	O(n+m)	O(n)
Boyer-Moore	O(n+m)	O(n/m)

m entspricht der Länge des Musters
n entspricht der Länge des Textes

Abb. 1: Vergleich der Laufzeiten

2.5 Dateiformate

Da in dieser Arbeit die Unterscheidung zwischen textbasierten Dateien gegenüber Binärdateien eine wichtige Rolle spielt, wird im Folgenden der Unterschied erläutert.

Prinzipiell lassen sich Dateiformate nach verschiedenen Kriterien unterscheiden. Beispiele hierfür können sein:

- textuell gegenüber binär
- Daten gegenüber ausführbar
- nach dem Inhalt: Text, Bild, Video, Ton und
- offen gegenüber proprietär.

2.5.1 Binärdaten

Binärdaten sind Dateien mit numerischem Inhalt. Ohne die genaue Spezifikation des Dateiformats ist nicht erkennbar, ob es sich bei den gespeicherten Zahlen um Befehle oder um Nutzdaten handelt. Die meisten Formate trennen Befehle von den Nutzdaten. Verwaltungsinformationen werden an den Anfang der Datei -- in den Header -- geschrieben. Die Nutzdaten werden angehängt. Dadurch erfährt das Programm, welches die Datei interpretieren soll, wie es mit den Nutzdaten umzugehen hat. Für den Umgang mit Binärdaten, wird immer ein spezielles Programm benötigt, da nicht wie beim Texteditor, alles als ein konkretes Zeichen behandelt werden kann. Binärdaten können verschiedene Daten enthalten, zum Beipsiel:

- Text
- Ton
- Bild und
- Video
- … (12).

2.5.2 Textbasierte Dateiformate

Textdateien sind Dateien, in denen sich jedes Zeichen selbst repräsentiert. Es kommen keine binären Steuercodes vor. Jedoch existieren textbasierte Formate, bei denen Steuerungs-, Formatierungs- und Strukturbefehle im Klartext dargestellt werden. Beispiele hierfür sind:

- XML
- HTML
- LaTeX
- PostScript und
- Proprietäre Konfigurationsdateien
- … .

Genauer betrachtet sind Textdateien eine Abfolge gleich großer Speicherblöcke. Die Größe dieser Speicherblöcke hängt von dem gewählten Zeichensatz ab. Jeder Speicherblock bildet auf ein Zeichen ab. Somit ist zum Interpretieren von Textdateien keine formatabhängige Software notwendig, sondern lediglich ein Texteditor oder ein Programm, welches mit dem gewählten Zeichensatz umgehen kann (12).

2.6 Verwandte Arbeiten (Stand der Technik)

2.6.1 Terbium Labs

Das Start-up *Terbium Labs* wurde 2015 von zwei Forschern der John Hopkins University gegründet. Ziel ist es, frühzeitig festzustellen, ob private Daten im Web kursieren. Die so genannte Matchlight-Technologie durchsucht das Web sowie das Dark Web. Nach Angaben der Forscher erfasst Google nur etwa 5 bis 10 Prozent des gesamten Web. Deswegen verwendet das *Terbium* System hunderte Seed-Links und folgt Links, welche sich auf diesen Seiten befinden. Auf diese Art und Weise können große Teile des Web erfasst werden. Die gesammelten Daten werden in 14 Byte große Stücke geteilt und in einer Datenbank verschlüsselt gespeichert. Kunden können dann entgeldlich diesen Datenbestand nach eigenen Daten durchsuchen. Die Kundendaten werden auch verschlüsselt, somit kann die Suchmuster niemand im Klartext sehen (2).

2.6.2 Dump Monitor

Ein weiterer Ansatz um gestohlene Daten zu entdecken, ist *Dump Monitor*. Die Software durchsucht die auf Pastebin veröffentlichten Daten automatisch nach Merkmalen, welche auf möglicherweise gestohlene Daten hinweisen. Falls die Software fündig geworden ist, werden die Links auf Twitter gepostet. *Dump Monitor* wurde von Jordan Wright entwickelt. Eine ähnliche Software war *PastebinLeaks*. Jedoch ist *PastebinLeaks* heutzutage nicht mehr aktiv (7).

2.6.3 LeakedIn

LeakedIn ist eine ähnliche Software wie *Dump Monitor*. Sie durchsucht *Pastebin* nach bestimmten Schlüsselwörtern beziehungsweise regulären Ausdrücken. Falls ein Treffer erzielt wird, wird der dazugehörige Link auf der Webseite www.leakedin.com veröffentlicht.

3 System Design und Implementierung

Nachdem im vorherigen Kapitel die theoretischen Grundlagen erarbeitet werden, geht es in diesem Kapitel darum, die Software zu entwickeln. Zunächst werden dazu die Anforderungen an das System identifiziert. Hierbei werden die Anforderungen in funktionale und nicht-funktionale unterteilt und nummeriert, damit später deren Umsetzung sichergestellt und dokumentiert werden kann. Grundlegende Architekturentscheidungen werden erläutert, ein Entwurf des Systems erstellt und die Implementierung dokumentiert. Darüber hinaus werden die verwendeten Bibliotheken aufgezeigt.

3.1 Anforderungsanalyse

Um an die funktionalen Anforderungen des Systems zu gelangen, müssen die wichtigsten Anwendungsfälle identifiziert werden. Da das System wenig Interaktion mit dem Anwender aufweist, sind nur zwei Anwendungsfälle (Anwender startet Matching-Vorgang, Anwender fügt neue Muster hinzu) zu beachten. Deshalb wird auf eine detaillierte Darstellung der Anwendungsfälle verzichtet.

3.1.1 Funktionale Anforderungen

Ziel ist es, eine Anwendung zum Monitoring von Informationsportalen zu erstellen. Dies bedeutet, dass die Daten, welche auf dem Informationsportal *Pastebin* befinden, nach Mustern durchsucht werden. Hierfür sind folgende Anforderungen an das System beim Entwurf und der Implementierung zu beachten:

FR1: In dem zu bauenden System werden diese Muster, nach denen gesucht wird, vom Anwender vorgegeben.

FR2: Die Muster müssen persistent gehalten werden, wohingegen die zu durchsuchenden Daten nicht langfristig gespeichert werden.

FR3: Das Monitioring bezieht sich auf das Informationsportal *Pastebin*.

FR4: Das Matching der Daten geschieht textbasiert.

FR5: Die in der anwendungsinternen Datenbank gespeicherten Muster werden nur mit Daten verglichen, welche ab diesem Zeitpunkt und zukünftig veröffentlicht werden. Es ist nicht möglich Muster mit Daten aus der Vergangenheit zu vergleichen.

FR6: Wird eine Übereinstimmung gefunden, heißt das, dass sich das zu suchende Muster innerhalb einer sich auf *Pastebin* befindlichen Datei wiederfindet. Der Nutzer muss darüber in Kenntnis gesetzt werden.

FR7: Es muss dem Anwender möglich sein, Muster in der anwendungsinternen Datenbank zu speichern. Dies muss über eine direkte Eingabe der Daten möglich sein. Alternativ muss es auch möglich sein, textbasierte Dateiformate einzulesen.

FR8: Der Anwender muss den Matching-Vorgang starten können.

3.1.2 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen soll die Anwendung folgende nicht-funktionale Anforderungen erfüllen.

NFR1: Performance - Das System muss in der Lage sein, die Daten, welche auf *Pastebin* hoch geladen werden, in Echtzeit zu verarbeiten. Es darf zu keinem Rückstau der Daten kommen.

NFR2: Testbarkeit - Definierte Grenzfälle muss das System korrekt bearbeiten.

NFR3: Verfügbarkeit - Das System muss weiterlaufen, auch wenn versucht wird nicht unterstützte Dateiformate einzulesen oder der Pfad einer Datei falsch angegeben wird.

NFR4: Änderbarkeit - Es muss möglich sein, neue Informationsportale hinzuzufügen, ohne dafür die Anwendung grundlegend ändern zu müssen. Dies muss in der Architektur berücksichtigt werden. Auch muss es möglich sein, die Matchingalgorithmen beziehungsweise die Logik, welche eine Übereinstimmung auslöst, auszutauschen oder zu erweitern. Es muss gewährleistet sein, das System auf verschiedenen Umgebungen ohne Anpassungen des Systems zu starten.

NFR5: Skalierbarkeit - Die Anzahl der Muster, nach denen die Anwendung sucht, beeinflusst nicht die Performance des Systems. Dies gilt auch für die Anzahl der Informationsportale.

NFR6: Bedienbarkeit - Der Anwender muss davor geschützt werden fehlerhafte Eingaben machen zukönnen.

3.2 Architekturentscheidungen

Nachfolgend werden grundlegende Archtiekturentscheidungen ausgeführt und diskutiert.

3.2.1 Datenbanktyp

In der Anwendung werden auschließlich Muster persistent gehalten, die vom Nutzer bereitgestellt werden. Die Daten, welche von einem Portal zum Matching herangezogen werden, werden nicht dauerhaft gespeichert. Es muss gewährleistet sein, wie in den nichtfunktionalen Anforderungen (unter 3.1.2 Performance, Skalierbarkeit) erwähnt, die Daten in Echtzeit zu verarbeiten und bei großen eintreffenden Datenmengen dieses Performacekriterium zu erhalten, sodass kein Rückstau der zu untersuchenden Daten entsteht. Da dies jedoch bei großen Mengen an Mustern auf einer Maschine schwierig zu garantieren ist, sollte es möglich sein, die persistent gehaltenen Muster auf mehrere Knoten zu verteilen. Somit kann das Matching verteilt und nebenläufig geschehen. Dadurch wird die Erfüllung der Anforderungen an Skalierbarkeit *NFR5* und Performance *NFR1* sichergestellt. Da die Muster vorraussichtlich nicht oft verändert werden, spielt Konsistenz eine untergeordnete Rolle. Auch wiederspricht das Konzept von klassischen relationalen Datenbanken dem Konzept der Verteilung. Dadurch bietet sich hier die Verwendung von NoSql Datenbanken an, da diese eine Verteilung erleichtern und hier eine Aufweichung des Konsistenzkriteriums zugunsten von Verfügbarkeit und Skalierung vorgenommen wird. Demnach fällt die Entscheidung auf eine NoSql Datenbank (4).
Weiter ist die Frage zu beantworten, welche Form der NoSql Datenbank verwendet werden sollte. Wie in Kapitel 1.5 erläutert, gibt es vier verschiedene Arten von NoSql Datenbanken. Es wird davon ausgegangen, dass die Muster keine Beziehungen zu anderen Mustern haben, welche durch die Datenbank abgebildet werden müssen. Dies wäre prinzipiell möglich, hier wird jedoch davon abgesehen. Auch ist es nicht notwendig, spaltenweise Berechnungen auf Tabellen durchzuführen. Es bietet sich daher die Verwendung von Key Value Stores an, weil ein Schlüssel mit Informationen über den Anwender auf das Muster zeigt. Somit fällt die Entscheidung auf eine Key Value Datenbank.

3.2.2 Nebenläufige Bearbeitung

Wie in Abbildung 2 ersichtlich ist, unterteilt sich die Anwendung in die verschiedenen Kernbereiche Datenhaltung, Datenbeschaffung und Datenverarbeitung, welche unterschiedlich rechenintensiv sind. Deshalb muss eine optimale Ressourcenausnutzung garantiert werden. Die in der Abbildung gezeigte Bausteinsicht wird unter 3.3.3 detailliert erläutert.
Zum Beispiel soll das Beschaffen der Daten unabhängig vom Rest der Anwendung geschehen. Es wäre möglich, dass eine sequenzielle Bearbeitung die vorhandenen Ressourcen nicht optimal nutzt. Das Matching beziehungsweise die Suche nach den Mustern ist in dieser Anwendung eine zeitkritische Komponente, welche unabhängig vom Rest der Anwendung durchgeführt werden muss. Eine nebenläufige Bearbeitung der genannten Kernbereiche sorgt für eine optimale Ressourcenausnutzung. Des Weiteren sorgt diese Architekturentscheidung dafür, das eine spätere Verteilung dieser Aufgaben auf unterschiedliche Knoten beziehungsweise das Hinzufügen weiterer Matchingknoten leicht durchzuführen ist. Dadurch werden die nicht-funktionalen Anforderungen NFR 1 und NFR 5 sichergestellt (13).

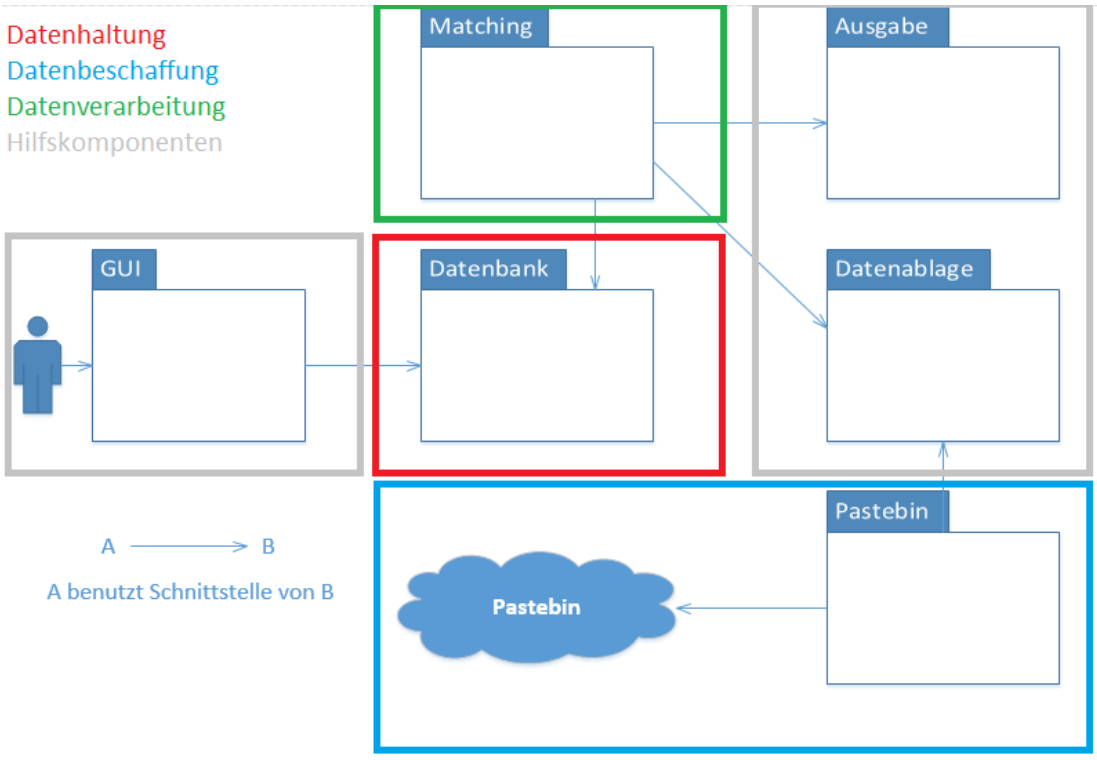

Abb. 2: Aufteilung der Anwendung

3.2.3 Kommunikation zwischen Threads

Durch die in Punkt 3.2.2 beschriebene nebenläufige Bearbeitung der bereits genannten Kernbereiche muss ein Datenaustausch zwischen den Threads erfolgen. Dies kann auf unterschiedlichem Wege geschehen. Zum einen wäre es möglich, diesen Austausch über direkte Nachrichten zu realisieren. Eine zweite Möglichkeit besteht darin, die Daten ähnlich dem Shared Memory Prinzip in einer zentralen Datenablage zu hinterlegen. Ein direkter Austausch über Nachrichten bietet sich in der vorliegenden Anwendung nicht an. Weil es bedeutet, dass beispielsweise die Datenbeschaffungskomponente jeden Thread, welcher ein Matching durchführt, kennen muss, um diesem die neu beschafften Daten zu schicken. Dadurch würde eine direkte Kopplung dieser beiden Komponenten vorliegen, was nicht erwünscht ist. Des Weiteren ist die Datenbeschaffungskomponente ausschließlich für das Beschaffen der Daten und nicht für das Verwalten anderer Komponenten zuständig. Deshalb ist es optimaler eine zentrale Datenablage zu benutzen. Hierauf legt die Datenbeschaffungskomponente die neuen Daten ab. Die Matchingkomponenten können sich von dort die neuen Daten holen, unabhängig davon, wieviele Matchingkomponenten im Einsatz sind. Diese Architekturentscheidung erleichtert die spätere Verteilung und Skalierung und sorgt damit für eine klare Aufgabenteilung der Threads. Dadurch ist es einfacher die Datenbeschaffungskomponente oder Matchingkomponente durch neue Implementierungen zu erneuern oder komplett auszutauschen, da sie jeweils nur die zentrale Datenablage kennen müssen.

3.3 Systementwurf

In diesem Abschnitt wird das System entworfen und es werden die verschiedenen Sichten auf das System beschrieben. Es wird auf die Kontextsicht, Verteilungssicht und die Bausteinsicht eingegangen. Hierbei wird gekennzeichnet durch welche Entscheidungen Anforderungen umgesetzt werden.

3.3.1 Kontextsicht

In der Kontextsicht wird das System mit seinen Schnittstellen zur Außenwelt gezeigt. Diese Schnittstellen können Fremdsysteme, technische Systemumgebung, Anwender oder Betreiber sein. Das hier zu implementierende System besitzt zwei Schnittstellen. Die erste Schnittstelle ist die zum Anwender, welcher dadurch die Anwendung starten, neue Muster zum Matching hinzufügen und Muster löschen kann. Die zweite Schnittstelle ist die Verbindung zu *Pastebin*, hierüber bekommt das System neue Daten.

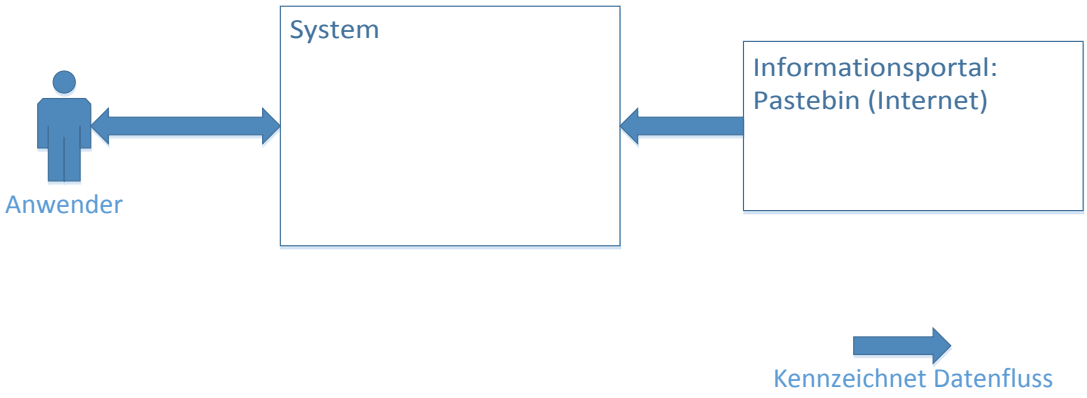

Abb. 3: Kontextsicht

3.3.2 Verteilungssicht

Die Verteilungssicht beschreibt die technische Umgebung, in der das System abläuft. Da es in dem Rahmen dieser Bachelorarbeit darum geht, einen Prototyp zu entwickeln, wird auf den Einsatz einer zentralen Datenbankinstanz verzichtet. Es wird eine embbeded Datenbank verwendet. Dadurch ist als technische Vorraussetzung für das System ein Host mit einer Java virtuellen Machine notwendig. Des Weiteren muss dieser Host über eine funktionierende Internetverbindung verfügen.

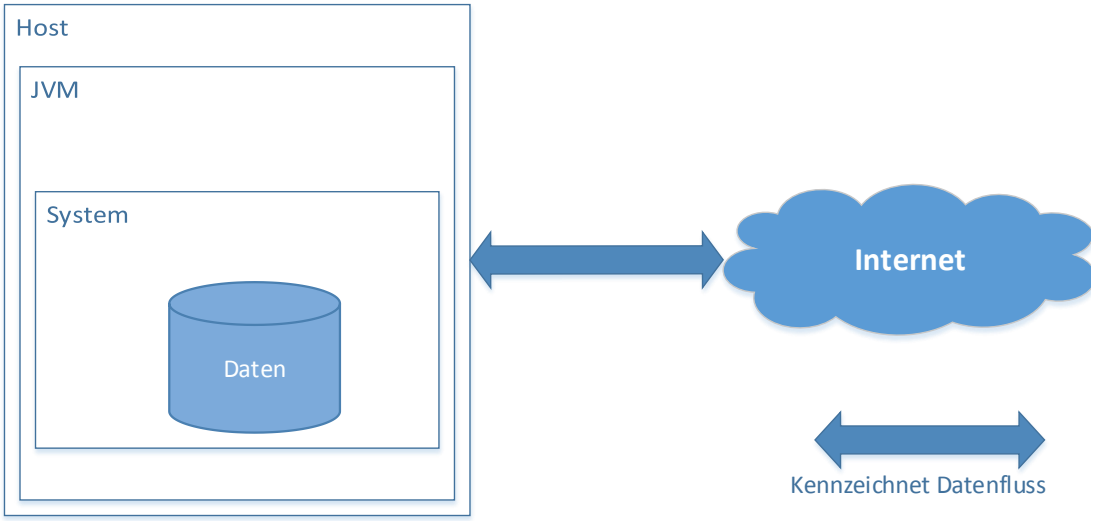

Abb. 4: Verteilungssicht

3.3.3 Bausteinsicht

Die Bausteinsicht beschreibt die Softwarekomponenten des Systems und deren Schnittstellen untereinander. Jedoch wird vom Quellcode abstrahiert. Dadurch wird die Struktur zwischen den Bausteinen der Software erkennbar. Es handelt sich dabei um eine statische Sicht auf das System. Auf eine detaillierte Abbildung der Klassen innerhalb der Komponenten wird verzichtet, da die Bausteine eine geringe Komplexität aufweisen und dies zum allgemeinen Verständnis in diesem Fall nicht notwendig ist.

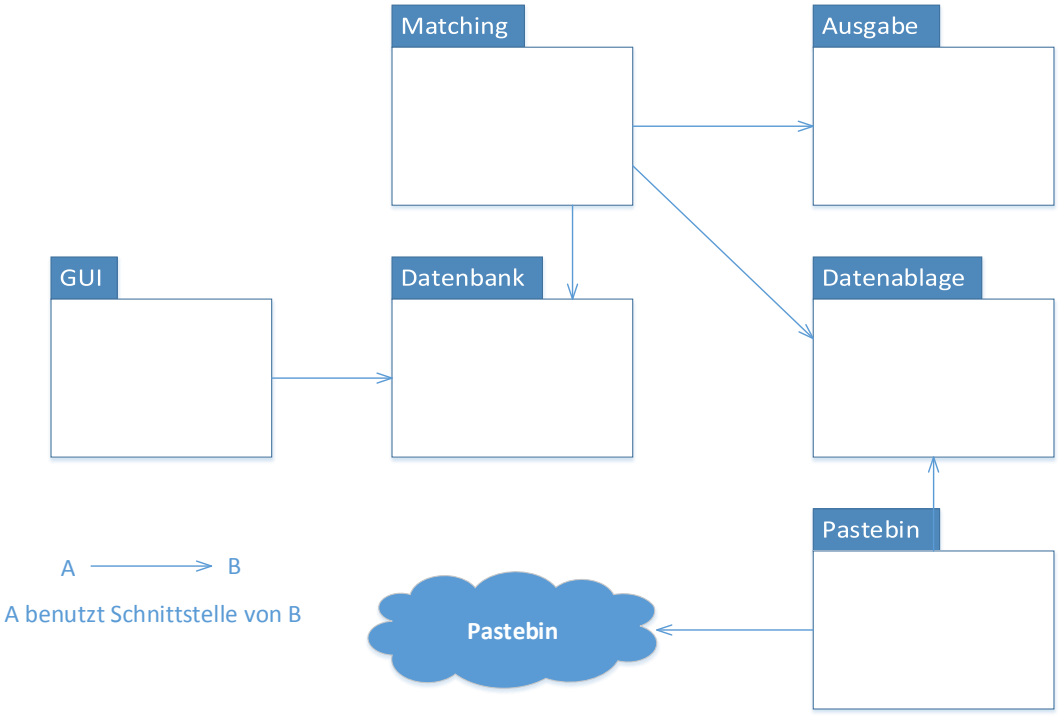

Abb. 5: Bausteinsicht

Zunächst werden die Komponenten mit ihren Aufgaben und Verantwortlichkeiten erklärt und anschließend wird auf die Schnittstellen der Komponenten eingegangen.

Die ***GUI-Komponente*** ist die Schnittstelle zum Benutzer. Darüber wird die Anwendung gestartet. Über die GUI können neue Muster in der anwendungsinternen Datenbank gespeichert, aber auch die Muster gelöscht werden. Die GUI-Komponente greift nur auf die Datenbank-Komponente zu.

Die ***Datenbank-Komponente*** dient ausschließlich dem Verwalten der Muster. Intern ist eine embedded key-value Datenbank eingebunden. Der Nutzer kann hier Muster speichern und löschen. Die Matching-Komponente kann sich die Muster holen, die sie benötigt, um nach Matchingtreffern in der Pastebindaten zu suchen. Die Datenbank-Komponente greift auf keine anderen Schnittstellen zu. Sie wird lediglich von der GUI- und der Matching-Komponente verwendet.

Die ***Matching-Komponente*** ist dafür verantwortlich, das Patternmatching welches unter 2.4 erklärt wird, durchzuführen. Dazu holt sie sich die Muster aus der Datenbank. Die zu durchsuchenden Daten werden von der zentralen Datenablage genommen. Wie in den Grundlagen erläutert, gibt es verschiedene Ansätze, um das Matching umzusetzen. Zum einen, muss entschieden werden, ob man harte oder weiche Vergleiche bevorzugt. Hier ist die Entscheidung auf harte Vergleiche gefallen. Folglich muss dann entschieden werden, welcher Algorithmus zum Einsatz kommt. Da im Bezug auf den Vergleich der durchschnittlichen Laufzeiten der Algorithmus von Boyer Moore am effinzientesten ist, kommt dieser zum Einsatz. Falls es ein Matchingtreffer gibt, wird ein Dienst der Ausgabe-Komponente verwendet. Wenn die Matching Logik erweitert und komplett ersetzt werden soll, muss dies in dieser Komponente geschehen oder sie muss ausgetauscht werden. Die Matching-Komponente greift auf Schnittstellen der Datenbank-, Datenablage und Ausgabe-Komponente zu.

Die ***Ausgabe-Komponente*** ist eine rein technische Komponente. Im Fall eines erfolgreichen Treffers dient sie dazu, eine Ausgabe zu erzeugen bzw. eine gewünschte Aktion durchzuführen. Da eventuell andere Aktionen als Folge eines Matchingtreffers denkbar wären, ist dies in eine eigene Komponente ausgelagert, somit ist es einfacher weitere Aktionen hinzuzufügen.

Die ***Datenablage-Komponente*** wird dazu genutzt, Daten zu verwalten. Diese Daten werden von der Matching-Komponenente und von der Pasetbin-Komponente benötigt. Die neuen Pastebindaten werden abgelegt und die Matching-Komponenete entnimmt diese Daten. Dies dient dazu, dass sich die Pastebin- und die Matching-Komponente nicht direkt kennen müssen.

Die ***Pastebin-Komponente*** ist dafür verantwortlich, die aktuellen Daten von Pastebin zu beschaffen. Sie muss sicherstellen, dass alle Daten gedownloaded und zum Matching bereitgestellt werden. Die neuen Daten werden dann auf der zentralen Datenablage hinterlegt und dadurch dem Matching zur Verfügung gestellt. Falls neue Informationsportale hinzugefügt werden sollen, muss dies in dieser Komponente geschehen oder die komplette Komponente gegebenenfalls ausgetauscht werden. Die Pastebin-Komponente greift dabei auf die Schnittstelle der Datenablage und auf die Scrapping API von Pastebin zu.

Im Folgenden werden die Schnittstellen erläutert. Damit wird ein möglicher Austausch von Komponenten erleichtert. Aus dem Namen der Schnittstelle ist ersichtlich, welche Komponente sie anbietet und welche Komponente auf diese zugreift. Beispiel: I*anbietendeKomponente*Fuer*zugreifendeKomponente*

IdatenbankFuerGui

```
public int speicherDaten(String user, String data);
```

Diese Methode erwartet als Parameter einen Schlüssel (user) und die dazugehörigen Daten (data), welche als Muster gespeichert werden sollen. Wenn das Speichern erfolgreich ist, wird 0 zurückgegeben. Falls der Schlüssel bereits vergeben ist, wird eine 1 zurückgegeben. Sollte die Eingabe keine lesbaren Zeichen enthalten, wird eine 2 zurückgegeben.

```
public boolean clear();
```

Diese Schnittstelle löscht die komplette Datenbank und alle in ihr gespeicherten Muster. Sie erwartet keine Parameter. Ist das Löschen erfolgreich, wird true zurückgegeben sonst false.

```
public int speicherDateiAusPfad(String user, String pfad);
```

Als Paramteter muss ein Pfad (pfad) angegeben werden, unter dem eine .txt-Datei abgelegt ist. Der Inhalt dieser Datei wird dann unter dem Schlüssel (user) in der Datenbank als Muster gespeichert. Wenn das Speichern erfolgreich ist, wird eine 0 zurückgegeben. Soll ein nicht zulässiges Dateiformat eingelesen werden, wird eine 1 zurückgegeben. Falls ein falscher Dateipfad angegeben wird oder die Datei nicht existiert, wird eine 2

zurückgegeben. Eine 5 wird zurückgegeben, wenn der Schlüssel (user) bereits vergeben ist. Sollte die Datei keine lesbaren Zeichen enthalten, gibt diese Methode eine 5 zurück.

IdatenbankFuerMatching

```
public ArrayList<String[]> gibAlleDaten();
```

Über den Aufruf dieser Methode werden alle Muster der Datenbank ausgegeben. Ein Paramter ist hierfür nicht notwendig. Es wird eine Liste von zweielementigen Stringarrays zurückgegeben. Das erste Element ist der Schlüssel, unter dem die Daten abgespeichert sind. Das zweite Element enthält das Suchmuster.

```
public ArrayList<String[]> gibNeuesteDaten();
```

Diese Schnittstelle gibt alle Muster aus, welche seit der letzten Abfrage hinzugekommen sind. Das sorgt dafür, dass Daten nicht unnötig in der Matching-Komponente mehrfach gehalten werden. Eine Parametereingabe ist dafür nicht notwendig. Es wird eine Liste von zweielementigen Stringarrays zurückgegeben. Das erste Element ist der Schlüssel, unter dem die Daten abgespeichert sind. Das zweite Element enthält das Suchmuster.

IausgabeFuerMatching

```
public void erstellenAusgabe(String dbeintrag, String[] pastebin,
String pfad);
```

Diese Schnittstelle bekommt als Eingabeparameter den Datenbankeintrag (Muster), ein zweielementiges Stringarray, welches an erster Position den eindeutigen Schlüssel für die URL des Pastebineintrags und an zweiter Position die zugehörigen Daten. Zudem wird der Schlüssel (pfad) übergeben, unter dem der Datenbankeintrag abgespeichert ist. Es wird eine Ausgabe HTML Datei mit dem Namen des übergebenen Schlüssels (Pfad) erstellt und in dem Ordner ‚ausgabe' abgelegt. Eine Rückgabe erfolgt nicht.

IdatebablageFuerMatching

```
public ArrayList<String[]> gibNeueDaten();
```

Mit Hilfe dieser Methode kann sich die Matchingkomponente die hinterlegten Pastebindaten holen. Es wird eine Liste von zweielementigen Stringarrays zurückgegeben. An erster Position steht der eindeutige Schlüssel, mit dem die eindeutige URL des Pastebineintrags gebildet werden kann. An zweiter Position stehen die dazugehörigen Daten.

```
public boolean isNeuePastebinDaten();
```

Diese Schnittstelle dient der Abfrage, ob neue Daten hinterlegt sind. Wenn neue Daten hinterlegt sind, gibt die Methode true zurück sonst false.

IdatebablageFuerPastebin

```
public void hinterlegeNeueDaten(ArrayList<String[]> neu);
```

Diese Methode macht es der Pastebinkomponente möglich, neue Daten in der Datenablage zu hinterlegen. Es wird eine Liste von zweielementigen Stringarrays übergeben. An erster Position steht der Schlüssel, mit dem die URL des Pastebineintrags eindeutig identifizierbar ist. An zweiter Position stehen die dazugehörigen Daten. Es gibt keine Rückgabe.

Durch den modularen Aufbau des Systems, die klar abgegrenzten Aufgaben der einzelnen Komponenten und die eindeutig definierten und beschriebenen Schnittstellen ist es möglich, die Anwendung zu ändern oder zu erweitern. Dadurch wird die Berücksichtigung der nicht-funktionalen Anforderung *NFR4* sichergestellt.

3.4 Implementierung

Die Implementierung wird gemäß den Architekturentscheidungen und des erarbeiteten Entwurfs durchgeführt. Der gesamte Implementierungsprozess wird in Java umgesetzt. Nachfolgend werden die verwendeten Bibliotheken aufgelistet.

3.4.1 Verwendete Bibliotheken

Json

Json ist ein Format zum Austausch von Daten, welches von Programmiersprachen unabhängig ist. In dem System wird eine in Java implemetierte Json Bilbiothek verwendet, um mit den Daten arbeiten zu können, welche von Pastebin gedownloaded werden. Die Bibliothek steht unter folgender URL zur freien Verfügung https://github.com/stleary/JSON-java.

MapDB

MapDB ist eine embedded open source Datenbank. Durch MapDB ist es möglich, die typische Java Collection API zu verwenden. In dem System wird MapDB zur Speicherung der Muster genutzt, da diese persistent gehalten werden müssen. Wie im Rahmen der Architekturentscheidungen erläutert, ist hier der Einsatz eines Key-Value-Stores optimal. Die Bibliothek steht unter http://www.mapdb.org/doc/quick-start/ zur freien Verfügung.

StringSearch

StringSearch ist eine Java Bibliothek. Sie stellt Implementierungen vom Boyer-Moore Algorithmus und weiteren Algorithmen zum Patternmatching zur Verfügung. Diese Implementierungen werden zum effizienten Auffinden der Muster benötigt.
Unter http://johannburkard.de/software/stringsearch/ steht die Bibliothek zur freien Verfügung.

3.4.2 Umsetzung von Vorgaben und Implementierungsentscheidungen

Im Folgenden werden die getroffenen Implementierungsentscheidungen ausgeführt und dabei auf die einzelnen Komponenten eingegangen.

GUI-Komponente

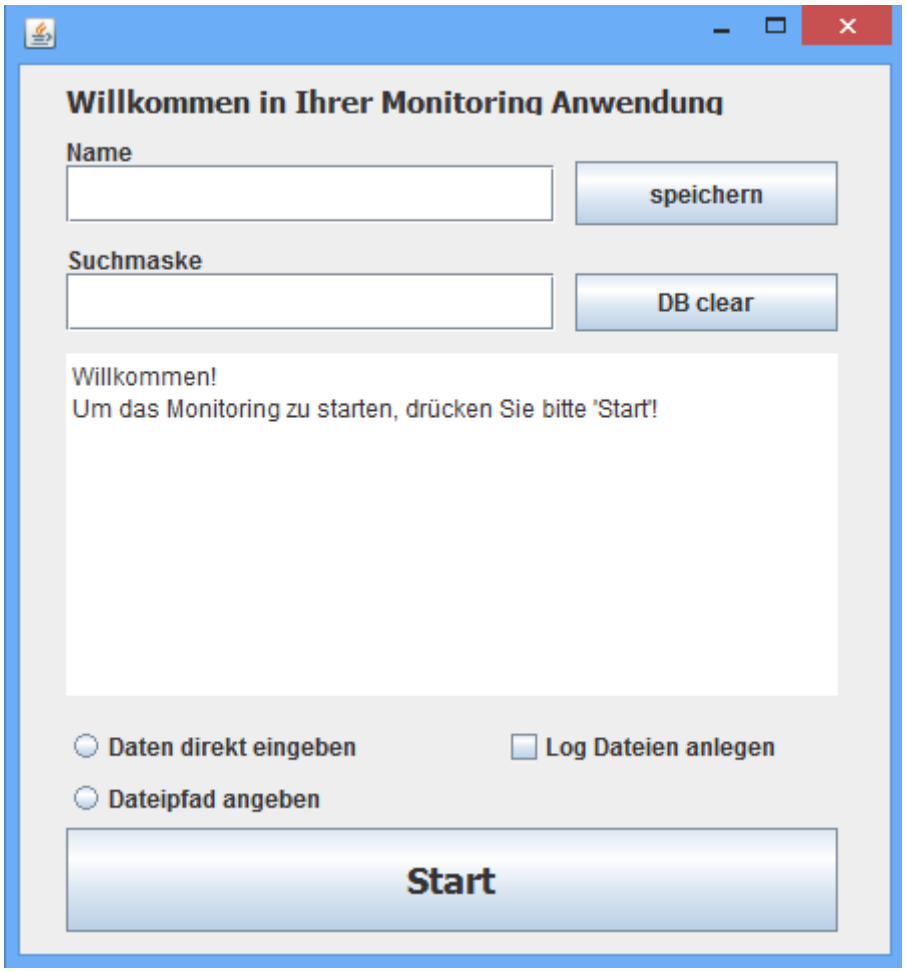

Abb. 6: Graphische Oberfläche

In Abbildung 6 wird die graphische Oberfläche (GUI) des Systems dargestellt. Nachfolgend werden die Funktionen der einzelnen Buttons erklärt.

Durch den Button ***Start*** wird die Anwendung gestartet. Dadurch beginnt implizit der Matchingalgorithmus mit den Mustern, welche in der Datenbank gespeichert sind. Damit ist sichergestellt, dass die funktionale Anforderung *FR8* umgesetzt ist. Der Nutzer kann auswählen, ob während der Laufzeit der Anwendung Logdateien angelegt werden oder nicht. Dies geschieht über die Checkbox ***Log Dateien anlegen***. Sobald die Anwendung gestartet ist, ist dieses Feld nicht mehr verfügbar. Wenn die Option gewählt ist, wird eine Datenbank-Logdatei angelegt, welche die Interaktion mit der Datenbank wiedergibt. Zusätzlich wird für jeden Matchingdurchlauf eine Logdatei angelegt. Die Dateien werden dabei in einem gesonderten Ordner abgelegt.

Über den Button ***speichern*** kann der Anwender neue Muster in der Datenbank hinterlegen. Es muss zwingend ein Name eingegeben werden, unter dem das Muster gespeichert wird. Namen dürfen nur einmal verwendet werden. Falls ein Name bereits existiert, wird der Anwender darauf hingewiesen. Es muss entschieden werden, ob eine Datei eingelesen oder das Muster direkt eingegeben wird. Deshalb muss der Anwender sich zwischen ***Daten direkt eingeben*** oder ***Dateipfad eingeben*** entscheiden. Wenn die Option Dateipfad gewählt wird, muss ein gültiger Pfad zu einer Textdatei angegeben werden. Wenn die Datei nicht existiert, diese ein unzulässiges Format besitzt oder ein falscher Pfad angegeben wird, wird der Anwender darüber informiert. Dadurch wird die nicht-funktionale Anforderung *NFR3* realisiert. Es müssen sowohl in Dateien als auch in direkt eingegebenen Mustern zwingend lesbare Zeichen enthalten sein. Dies wird von der Anwendung geprüft und dem Anwender gegebenenfalls mitgeteilt. Durch diese Funktionen sind die funktionalen Anforderungen *FR2*, *FR7* sowie die nicht-funktionale *NFR6* umgesetzt. Da die Muster vom Anwender bereitgestellt werden, ist die funktionale Anforderung *FR1* umgesetzt.

Durch den Button **DB clear** kann der Anwender die gesamte Datenbank löschen.

Alle Mitteilungen an den Anwender oder das Auffinden eines Matchingtreffers werden über das Textfenster angezeigt. Im Falle eines Treffers werden dem Nutzer der konkrete Ort, der von der Anwendung hinterlegten Datei und der Name dieser genannt. In der abgelegten Datei befinden sich detailierte Informationen zum Auffinden des Musters, die für den Nutzer relevant sind.

Um den Matchingvorgang zu stoppen, muss die Anwendung geschlossen werden.

Ausgabe-Komponente

Wie im Entwurf festgelegt wird, muss es möglich sein, dem Nutzer erfolgreiche Treffer mitzuteilen. Dabei ist die fraglich, wie ein Treffer mitgeteilt wird. Die Entscheidung ist hierbei darauf gefallen, ein HTML Dokument zu erstellen. Dieses Dokument wird auf der lokalen Maschine abgelegt. Zudem wird dem Nutzer der entsprechende Pfad zu dem Dokument mitgeteilt. Inhalt dieser Datei sind zum Beispiel der Zeitpunkt des Auffindens, die URL zu dem Originalpaste und weitere Informationen. Es wird bewusst ein HTML Dokument gewählt, da davon aus gegegangen werden kann, dass unabhängig von der Plattform und der Arbeitsumgebung jeder im Besitz eines Browsers ist. Somit ist sichergestellt, dass jeder dieses Dokument öffnen kann. Durch die Ausgabekomponente wird die Umsetzung der funktionalen Anforderung *FR6* umgesetzt.

Datenablage-Komponente

Die Datenablage ist der zentrale Punkt, über den die Matchingkomponenente und die Pastebinkomponente Daten austauschen. Um unnötiges Austauschen von Daten zu vermeiden, wird eine Flag implementiert, wodurch anzeigt wird, ob neue Daten hinterlegt sind.

Datenbank-Komponente

In der Datenbankkomponente werden die Muster gehalten, die zum Matching benötigt werden. Auch hier ist es wichtig, dass nicht Daten unnötig oft oder wiederholt an die Matchingkomponente geschickt werden. Daher ist es möglich, die Daten abzurufen, die seit der letzten Anfrage hinzugekommen sind. Ein Flag zeigt an, ob es Veränderungen an der Datenbasis gibt. Es ist möglich, Daten über einen Dateipfad einzulesen, jedoch wird nur das .txt-Format unterstützt. Darüber hinaus ist darauf zu achten, dass die Dateien UTF-8 codiert sind, da die Anwendung komplett mit UTF-8 arbeitet. Vor der Speicherung der Daten werden diese normalisiert, was bedeutet, dass nicht lesbare Zeichen (Leerzeichen, Zeilenumbruch und Tabulator) entfernt werden. Zudem werden alle Buchstaben vereinheitlicht als Kleinbuchstaben dargestellt. Es ist nicht möglich, Dateien oder Daten zu speichern, welche keine lesbaren Zeichen enthalten. Dadurch, dass in der Datenbank ausschließlich die Muster und nicht die Pastebindaten gespeichert werden, ist die funktionale

Anforderung *FR5* erfüllt. Die Muster werden nur mit den jeweils neu gedownloadeten Daten verglichen. Daten, welche keine Treffer ergeben, werden verworfen.

Matching- und Pastebin-Komponente

Wie in den Architekturentscheidungen erläutert, ist eine nebenläufige Bearbeitung der einzelnen Kernbereiche Datenhaltung, Datenbeschaffung und Matching für einen effizienten Ablauf wichtig. Wie in Abbildung 7 deutlich wird, werden die Matching- und die Pastebin-Komponente explizit in eigenen Threads ausgeführt. Die anderen Komponenten laufen in dem von Java implizit ausgeführten Thread. Die Pastebin-Komponente bedient sich bei der Beschaffung der neuesten Daten an der von Pastebin angebotenen Scraping API. Dabei werden alle 60 Sekunden die neuesten Pastebindaten runtergeladen. Dadurch wird sichergestellt, dass die funktionale Anforderung *FR3* umgesetzt wird. Die genaue Verwendung der Scraping API kann unter http://pastebin.com/api_scraping_faq eingesehen werden.
Wie in der Bausteinsicht unter 3.3 beschrieben, wird der Algorithmus von Boyer Moore verwendet. Dafür wird die SearchString Bibliothek eingesetzt, welche Implementierungen des Boyer Moores Algorithmus anbietet. Dadurch wird die funktionale Anforderung *FR4* umgesetzt. Die Daten, die von Pastebin gedownloaded werden, werden normalisert bevor sie nach Mustern untersucht werden. Das bedeutet, dass nicht lesbare Zeichen (Leerzeichen, Zeilenumbruch und Tabulator) entfernt werden. Weiter werden alle Buchstaben vereinheitlicht als Kleinbuchstaben dargestellt. Damit wird verhindert, dass zum Beispiel Groß- und Kleinschreibung oder einfache Zeilenumbrüche Daten unauffindbar machen. Falls leere Dateien von Pastebin gedownloaded werden, werden diese verworfen.

```
public static void starten() {
    db.initDB();
    Thread paste = new PastebinImpl();
    Thread match = new Matching();
    ordnererstellen();
    paste.start();
    match.start();

}
```

Abb. 7: Startroutine

4 Ergebnisse

In diesem Kapitel geht es darum, die implementierte Anwendung zu testen und somit nachzuweisen, dass das System, die in der Einleitung der Arbeit formulierten Ziele erfüllt und sich der gewünschte Effekt mit Blick auf die Erkennung von kompromittierten Organisationen erzielen lässt. Dafür wird ein Szenario entwickelt, in dem die Anwendung ein vordefiniertes Verhalten zeigen soll. Das tatsächliche Verhalten wird protokolliert und anschließend ausgewertet. Hierbei wird die nicht-funktionale Anforderung *NFR2* umgesetzt.

4.1 Testdaten

Um den Versuch realitätsnah aufzubauen, werden verschiedene Dateien von Pastebin gedownloaded. Es wird darauf geachtet, dass die Daten verschiedene Zeichen beinhalten wie zum Beispiel chinesiche oder kyrillische Zeichen. Somit kann gezeigt werden, dass die Anwendung Universalität bietet. Zudem wird darauf geachtet, dass die Daten verschieden groß sind, damit wird nachgewiesen, dass die Anwendung auch mit großen Daten umgehen kann. Aus diesen Dateien werden dann beliebig Teile rausgeschnitten, welche in die Anwendung eingefügt werden oder die Dateien werden als Ganzes in der Anwendung gespeichert.

4.2 Versuchsaufbau

Die Originaldaten werden in verschiedenen Zeitabschnitten auf Pastebin hochgeladen. Anschließend wird kontrolliert, ob die Anwendung die Daten findet und wie viel Zeit sie dafür benötigt. Dabei hilft die Auswertung der angelegten HTML-Dateien. Es wird berechnet, wie lange die Anwendung durschnittlich mit dem Matchingvorgehen beschäftigt ist. Die Ergebnisse werden im nächsten Absatz aufgeführt und erläutert.

4.3 Ergebnisse

Muster	Größe in KB	Original mit Muster	Zeitpunkt des Veröffentlichens	Zeitpunkt des Auffindens	Benötigte Zeit in Minuten	Ergebnisdatei
test1.txt	2	test1original.txt	04.08.2016 - 16.36	04.08.2016 - 16.36	< 1	test1.html
test2.txt	1	test2original.txt	04.08.2016 - 16.36	04.08.2016 - 16.37	1	test2.html
test3.txt	13	test3original.txt	04.08.2016 - 16.37	04.08.2016 - 16.38	1	test3.html
test4.txt	3	test4original.txt	04.08.2016 - 16.38	04.08.2016 - 16.38	< 1	test4.html
test5.txt	15	test5original.txt	04.08.2016 - 16.38	04.08.2016 - 16.38	< 1	test5.html
test6.txt	28	test6original.txt	04.08.2016 - 16.39	04.08.2016 - 16.40	1	test6.html
test7.txt	39	test7original.txt	04.08.2016 - 16.40	04.08.2016 - 16.40	< 1	test7.html
test8.txt	1	test8original.txt	04.08.2016 - 16.40	04.08.2016 - 16.40	< 1	test8.html
test9.txt	3	test9original.txt	04.08.2016 - 16.41	04.08.2016 - 16.41	< 1	test9.html
test10.txt	4	test10original.txt	04.08.2016 - 16.41	04.08.2016 - 16.41	< 1	test10.html
test11.txt	1	test11original.txt	04.08.2016 - 16.47	04.08.2016 - 16.47	< 1	test11.html
test12.txt	1	test12original.txt	04.08.2016 - 16.47	04.08.2016 - 16.48	1	test12.html
test13.txt	1	test13original.txt	04.08.2016 - 16.48	04.08.2016 - 16.48	< 1	test13.html
test14.txt	5	test14original.txt	04.08.2016 - 16.48	04.08.2016 - 16.48	< 1	test14.html
test15.txt	1	test15original.txt	04.08.2016 - 16.48	04.08.2016 - 16.49	1	test15.html
test16.txt	2	test16original.txt	04.08.2016 - 16.49	04.08.2016 - 16.50	1	test16.html
test17.txt	3	test17original.txt	04.08.2016 - 16.50	04.08.2016 - 16.50	< 1	test17.html
test18.txt	4	test18original.txt	04.08.2016 - 16.50	04.08.2016 - 16.50	< 1	test18.html
test19.txt	1	test19original.txt	04.08.2016 - 16.51	04.08.2016 - 16.51	< 1	test19.html
test20.txt	6	test20original.txt	04.08.2016 - 16.51	04.08.2016 - 16.52	1	test20.html

Abb. 8: Tabelle Testlauf

Wie die Abbildung 8 belegt, werden alle Muster entdeckt. Beim durchgeführten Testlauf wird die Zeit gemessen, welche die Anwendung benötigt, um den Matchingdurchlauf für alle Muster und die aktuellen Pastebindaten durchzuführen. Diese Berechnungszeit beträgt mit dem vorliegenden Datenbestand im Durchschnitt 19 ms pro Durchlauf. Da das System alle 60 Sekunden die neuesten Daten von Pastebin downloaded und nach Mustern durchsucht, ist die benötigte Zeit zum Auffinden eines Treffers von maximal 1 Minute logisch erklärbar. Dadurch, dass die Testdaten dokumentiert sind, ist der Test reproduzierbar. Damit kann aussgeschlossen werden, dass es sich um ein zufälliges Ergebnis handelt. Die verwendeten Testdaten sowie die Ergebnisdateien sind auf der CD Rom einsehbar, die der vorliegenden Arbeit beiliegt.

Der durchgeführte Test zeigt, dass das System alle Daten findet, in denen Muster enthalten sind. Da ein `Nichtfinden` von Daten nicht nachgewiesen werden kann, kann nur ein erfolgreiches Auffinden gezeigt werden. Der Testlauf hat die gewünschten Ergebnisse geliefert, sodass gezeigt wird, dass mit Hilfe der Anwendung ein funktionierendes Monitoring von Pastebin möglich ist. Durch die korrekte Funktionalität der Anwendung bietet sich für Organisationen die Möglichkeit, Pastebin nach selbstdefinierten Mustern zu durchsuchen. Es kann selbstständig entschieden werden, welche Daten sensibel beziehungsweise relevant sind. Es können entweder ganze Dateien sein oder nur konkrete Emailadressen, IP Adressen, Kreditkartendaten oder ähnliches, die in einer größeren Auflistung auftauchen. Durch das Monitoring von Informationsportalen können Organisationen zeitnah darüber informiert werden, wenn auf diesen Portalen Daten einsehbar sind, die auf eine Kompromittierung hindeuten.

5 Zusammenfassung

5.1 Was wird erarbeitet?

Daten sind heutzutage von existenzieller Bedeutung. Sie müssen in besonderem Maße vor fremder Einsicht oder Verlust geschützt werden. Daher ist es für Organisationen wichtig, zu erfahren, falls eine Kompromittierung vorliegt, weil ihre Daten in Gefahr sind beziehungsweise die Integrität der Daten. Es kann sich dabei um Daten über die Organisationen oder Daten, die sich in dessen Besitz befinden, handeln. Es ist wichtig, eine Kompromittierung möglichst schnell zu entdecken, damit potenzieller Schaden gemindert werden kann. Dieser Schaden kann finanzieller Natur sein, juristische Folgen mit sich bringen oder zu einem Vertrauenverlust gegenüber der Organistation führen. Häufig spielen Informationsportale eine wichtige Rolle bei der Veröffentlichung entwendeter Daten. Auch Daten, die Teil einer möglichen Angriffsvorbereitung sein können, sollten schnell entdeckt werden, wenn diese auf Informationsportalen einsehbar sind, weil dadurch Gegenmaßnahmen zeitnah eingeleitet werden können. Der Angriff auf Target zeigt, dass ein gezieltes Monitoring von Informationsportalen den Schaden gemindert hätte.
Mit Hilfe der entwickelten Anwendung ist es möglich, durch das Monitoring von Informationsportalen eine Kompromittierung schnell aufzuspüren. Damit entsteht ein erheblicher Vorteil gegenüber herkömmlichen Vorgehen. Eine Kompromittierung kann Minuten nach der Veröffentlichung von Daten entdeckt werden. Dazu werden funktionale und nicht-funktionale Anforderungen erhoben, aus welchen sich die Architekturentscheidungen und der Systementwurf ableiten. Nach der Implementierung des Systems wird dessen Funktionalität und damit der Mehrwert für die IT Sicherheit beziehungsweise für die Entdeckung von Kompromittierung gezeigt. Die Funktionalität des Ansatzes wird durch einen Testlauf belegt.

5.2 Wird das Ziel der Arbeit erreicht?

Ziel der Arbeit ist es, einen Prototypen zu entwickeln, welcher in der Lage ist, kompromittierte Organisationen zu erkennen. Mit Hilfe des entwickelten Prototyps lässt sich das Informationsportal Pastebin gezielt nach Daten monitoren wodurch eine Kompromitterung entdeckt werden kann. Es wird gezeigt, dass dieser Ansatz funktioniert. Außerdem wird gezeigt, dass man im Zusammenhang von Angriffen auf Organisationen nicht ausschließlich darauf achten sollte, was innerhalb einer Organisation geschieht. Es ist ebenso wichtig, auf öffentlich zugängliche Informationen zu achten, da sich aus diesen wichtige Rückschlüsse ziehen lassen. Auch dadurch ist es möglich, Kompromittierungen zu entdecken. Werden Daten gefunden, die auf eine Kompromittierung hindeuten, kann nicht auf die Quelle der Daten geschlussfolgert werden, aber es kann belegt werden, dass eine Vertraulichkeitsverletzung vorliegt. Da Informationsportale in diesem Kontext eine wichtige Rolle spielen, sollte ihnen ein hohes Maß an Aufmerksamkeit beigemessen werden.

5.3 Ausblick

Insgesamt kann die Entwicklung der Anwendung als Erfolg gewertet werden, weshalb eine weitere Verfolgung dieses Ansatzes wünschenswert ist. Da im Rahmen dieser Arbeit nur ein erster Prototyp erarbeitet wird, gibt es verschiedene Anknüpfungspunkte für Weiterentwicklungen. Zum Beispiel ist es denkbar, aus eingegebenen Mustern automatisch weitere Muster abzuleiten. Auch können zum Monitoring weitere Informationsportale herangezogen werden. Es ist möglich, diese neuen Portale automatisch aufzuspüren, wodurch sich der Suchradius des Monitorings erweitern würde. Darüber hinaus ist es möglich, aus weiteren Dateiformaten Informationen in eine texttuelle Darstellungen abzuleiten, nach denen gesucht werden kann. Außerdem könnte man das Monitoring von Informationsportalen als Webdienst anbieten. Es ist denkbar, Informationen über das Auffinden eines Treffers zentral zu sammeln beziehungsweise mit weiteren Interessenten oder Institutionen zu teilen.

Literaturverzeichnis

1. **Eckert, Claudia.** *IT-Sicherheit.* München : Oldenbourg Wissenschaftsverlag GmbH, 2014. 978-3-486-77848-9.

2. **[Online] 6. April 2016. [Zitat vom: 6. April 2016.] http://www.heise.de/tr/artikel/Start-up-will-im-Dark-Web-nach-gestohlenen-Daten-suchen-2686264.html.**

3. Schäfer, Werner. *Software-Entwicklung.* München : Pearson Studium, 2010.

4. Ha, Tran Ngoc. *Charakteristika und Vergleich von SQL.* http://dbs.uni-leipzig.de/file/seminar_1112_tran_ausarbeitung.pdf : s.n., 12. 12 2011.

5. Informationstechnik, Bundesamt für Sicherheit in der. *Durchführungskonzept für Penetrationstests.* Bonn : s.n., 2016.

6. Zeltser, Lenny1. [Online] 4. 5 2016. [Zitat vom: 4. 5 2016.] https://zeltser.com/paste-sites-for-pen-testing-reconnaissance/.

7. Zeltser, Lenny. [Online] 29. 4 2016. [Zitat vom: 29. 4 2016.] https://zeltser.com/pastebin-used-for-sharing-stolen-data/.

8. Handelsblatt. [Online] 09. 05 2016. [Zitat vom: 09. 05 2016.] http://www.handelsblatt.com/unternehmen/handel-konsumgueter/target-hackerangriff-kostet-us-handelskette-millionen/9543006.html.

9. Heise. [Online] 09. 05 2016. [Zitat vom: 09. 05 2016.] http://www.heise.de/newsticker/meldung/Target-Hack-Auch-70-Millionen-Kundendaten-erbeutet-2083521.html.

10. Windhorst, Maik. *Pattern Matching.* Bremen : s.n.

11. Reichelt, Stephan. *Pattern Matching.* 2002.

12. Kersken, Sascha. *Kompedium der Informationstechnik.* Bonn : Galileo Press GmbH, 2003.

13. Peschel-Findeisen, Thomas. *Nebenläufige und Verteilte Systeme.* Bonn : mitp-Verlag, 2006.

14. BSI, Bundesamt für Sicherheit in der Informationstechnik. *Leitfaden Informationssicherheit.* Bonn : s.n., Februar 2012.

Abbildungsverzeichnis

Versicherung über Selbstständigkeit

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

Hamburg, den _____ _____