Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Security Code Crawler für Packetstormsecurity

Security Code Crawler for Packetstormsecurity

Bachelorarbeit

im Studiengang Informatik

von

Marvin Kuhnke

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Javad Ghofrani
Betreuer: M. Sc. Fabien Patrick Viertel

Hannover, 27.03.2018

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 27.03.2018

Marvin Kuhnke

Zusammenfassung

Security Code Crawler für Packetstormsecurity

Diese Bachelorarbeit beschäftigt sich mit der Website `packetstormsecurity.com`, welche eine Vielzahl an Quellcode- und anderen Dateien bereithält, die Exploits, Schwachstellen und andere sicherheitsrelevante Themen betreffen.

Primärer Fokus der Arbeit ist das Extrahieren von sicherheitsrelevantem Quellcode. Er ist interessant für die weitere Nutzung, zum Beispiel in Klonerkennungswerkzeugen.

Da die Website keine öffentliche *API* hat, wurde ein *Crawler* entwickelt, welcher zuerst eine lokale Datenbank aus dem *RSS-Feed* der Seiteneinträge erstellt. Einmal erstellt, können Einträge in beliebiger Reihenfolge heruntergeladen und ausgewertet werden. Die Auswertung erfolgt zuerst auf Basis von Dateiendungen, zusätzlich werden *Klassifizierer* vorgestellt, die Textdateien mit Quellcode erkennen. Auch ist es möglich, eigene Auswertungsmethoden zu entwickeln und diese als eine Art Plugin in den Crawler zu integrieren.

Die so gefundenen Dateien werden zusammen mit einer Datenbank, welche alle Meta-Informationen der Website-Einträge enthält, gespeichert. So lassen sich, falls im Eintrag vorhanden, zum Beispiel Dateien zu *Common-Vulnerabilities-and-Exposures-IDs* zuordnen.

Abstract

Security Code Crawler for Packetstormsecurity

This bachelor thesis concerns the website `packetstormsecurity.com`, which holds a multitude of source code and other data concerning exploits, vulnerabilities, and issues pertaining to security.

Primary matter of this thesis is the security pertinent source code, which is interesting for further use, for example in clone detection tools.

A *crawler* has been developed, since the website offers no public API. This crawler first creates a local database from the website's *RSS feed*, which can then be used to download entries in arbitrary order. Entries are then analyzed, which firstly makes use file extensions. Furthermore, classifiers are presented that are able to recognize text files containing source code. Additionally it is possible to develop different methods of analysis, which can then be intergrated into the crawler as a kind of plugin. The collected data is persisted together with a database that contains all meta data from the website entries. This way, if an entry has, for example, a *Common-Vulnerabilities-and- Exposures-ID*, the resulting snippet will be associated with the CVE-ID.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Am Fachgebiet Software Engineering wird gegenwärtig von Brunotte [10] das Eclipse-Plugin *Security Code Clone Detection* entwickelt. Ziel dieses Plugins soll es sein, Java-Entwickler schon während der Entwicklung vor unsicherem Quellcode zu warnen. Hierzu soll Klonerkennung angewandt werden. Damit diese Klonerkennung funktioniert, benötigt sie Quellcode von Schwachstellen. Quellcode findet sich an vielen Stellen im Internet, beispielsweise gibt es große Datensätze wie die *UCI Source Code Data Sets* [6]. Ähnliche Datensätze für Quellcode aus Schwachstellen existieren jedoch nicht.
Diese Arbeit beschäftigt sich mit der Website `packetstormsecurity.com` (PSS), die sicherheitsrelevante Dateien und Nachrichten veröffentlicht, und ob aus dieser Website sicherheitsrelevanter Quellcode gewonnen werden kann.

1.1 Problemstellung

PSS bietet über 100 000 Dateien aus dem Feld der Computersicherheit an, besitzt jedoch kein (öffentliches) *Application programming interface* (API). Unter diesen Dateien befinden sich möglicherweise viele Quelltextdateien und Dateien, die Quelltext enthalten. Diese Dateien sollen für die weitere Analyse zugänglich gemacht werden, insbesondere für Brunottes Security Code Clone Detection Eclipse Plugin.

1.2 Lösungsansatz

Um den Datensatz von PSS für eine weitere Analyse zugänglich zu machen, wird ein *Crawler* entwickelt, der selbstständig Dateien und ihrer Metadaten herunterlädt. Es kann nach verschiedenen Kriterien gefiltert werden, welche Dateien heruntergeladen werden, sodass nicht benötigte Einträge idealerweise nicht heruntergeladen werden. Die Dateien werden zusammen mit einer Datenbank von Metadaten gespeichert, so dass eine Weiternutzung erfolgen kann.

Weiterhin werden Statistiken über den kompletten Satz an Daten präsentiert. Um zusätzlichen Quellcode zu gewinnen, der nicht in Form von Quelltextdateien vorliegt, werden *Klassifizierer* entworfen, die Textdateien auf Quellcode untersuchen.

1.3 Struktur der Arbeit

In Kapitel 2 werden zunächst die an den Crawler gestellten Anforderungen beschrieben. Kapitel 3 geht auf alle für die folgenden Kapitel benötigten Grundlagen ein. In Kapitel 4 werden grundlegende Konzepte des Crawler erklärt. Kapitel 5 gibt Einblicke in die Architektur und die grafische Benutzeroberfläche des Crawlers. Kapitel 6 analysiert die auf PSS vorhandenen Dateien und stellt Klassifizierer für Textdateien mit Quellcode vor. Kapitel 7 geht auf verwandte Arbeiten ein. Abschließend enthält Kapitel 8 das Fazit und einen Ausblick.

Kapitel 2

Anforderungen

Dieser Abschnitt beschreibt die Anforderungen, die an den Crawler gestellt wurden. Abschnitt 2.1 beschreibt, welche Stakeholder das Projekt hat. In Abschnitt 2.2 werden die vom Fachgebiet Software Engineering (SE) gestellten Anforderungen genauer erläutert.

2.1 Stakeholder

Der Stakeholder des Crawlers ist das SE. Ziel ist es, den Crawler einzusetzen, um gefundenen Quellcode in einem Klonerkennungswerkzeug zu benutzen. Zusätzlich sind andere Anwendungsmöglichkeiten denkbar. So könnte der Crawler in Teilen oder komplett für weitere Forschungsprojekte verwendet werden, die auf größere Datenmengen aus dem Bereich der Sicherheitsforschung angewiesen sind.
Auch könnte er von anderen Forschern oder Entwicklern aus dem Bereich der Computersicherheit genutzt werden, die Dateien oder Quellcode weiterverwenden oder auswerten möchten.

2.2 Überblick

In diesem Abschnitt werden die Anforderungen, die vom SE gestellt wurden, genauer beschrieben.

[FR.1] Der Crawler ist plattformunabhängig

Nutzer sollen nicht durch ihre Betriebssystemwahl von der Nutzung des Crawlers ausgeschlossen werden.

[FR.2] Der Crawler ist eine Java-Anwendung

Die am SE mit Abstand am meisten eingesetzte Programmiersprache ist Java. Um die Erweiterung bzw. Weiternutzung möglichst einfach zu

gestalten, wird der Crawler in Java entwickelt.

[FR.3] Der Crawler kann Dateien und zugehörige Metadaten selbstständig von packetstormsecurity.com herunterladen

Der Crawler durchsucht die titelgebende Internetseite `packetstormsecurity.com` nach Dateien und lädt diese zusammen mit den zugehörigen Metadaten in ein vom Benutzer ausgewähltes Verzeichnis.

[FR.3.2] Der Dateidownload ist auf bestimmte Dateitags beschränkbar

Um die Menge an nicht benötigten heruntergeladenen Dateien zu reduzieren, kann der Download auf bestimmte Tags beschränkt werden. Es ist möglich, keine, eins oder mehrere Tags auszuwählen. Wenn kein Tag ausgewählt wurde, sollen Dateien unabhängig vom Tag heruntergeladen werden. Wenn mindestens ein Tag ausgewählt wurde, sollen Dateien nur heruntergeladen werden, wenn sie mindestens eins der ausgewählten Tags besitzen.

[FR.3.2] Der Dateidownload ist auf bestimmte Dateiendungen beschränkbar

Um die Menge an nicht benötigten heruntergeladenen Dateien zu reduzieren, kann der Download auf bestimmte, frei wählbare Dateiendungen beschränkt werden.

[FR.3.3] Der Dateidownload kann pausiert und zu einem späteren Zeitpunkt wiederaufgenommen werden

Da mitunter eine große Menge an Daten heruntergeladen wird, ist es möglich, den Download zu pausieren und zu einem späteren Zeitpunkt wieder aufzunehmen. Es ist gewährleistet, dass dabei keine Einträge übersprungen werden oder verloren gehen.

[FR.3.4] Heruntergeladene Archive können automatisch entpackt werden

packetstormsecurity.com hält neben vielen Textdateien auch Archive in verschiedenen Formaten. Daher gibt es die Möglichkeit, heruntergeladene Archive automatisch zu entpacken.

[FR.3.4] Zu heruntergeladenen Dateien zugehörige Metadaten werden in einer SQLite-Datenbank gespeichert

Metadaten, insbesondere eventuelle verknüpfte CVE-IDs, sind interessant zur weiteren Auswertung. Daher werden sie zusammen mit einem Verweis auf

zugehörige Dateien zusammen mit den Dateien in einer SQLite-Datenbank gespeichert.

[FR.4] Der Crawler besitzt eine grafische Benutzeroberfläche

Grafische Benutzeroberflächen vereinfachen die Bedienung und ermöglichen eine einfache Übersicht über große Mengen an Information. Daher besitzt der Crawler eine grafische Benutzeroberfläche.

[FR.4.1] Es gibt einen Dialog, der es ermöglicht neue Datenbanken anzulegen

Der Crawler ist nicht auf eine Datenbank beschränkt, sondern bietet die Möglichkeit, mehrere Datenbanken mit verschiedenen Einstellungen anzulegen.

[FR.4.2] Es gibt einen Dialog, der es ermöglicht, bestehende Datenbanken zu laden

Um bestehende Datenbanken zu öffnen und erneut zu betrachten oder zu aktualisieren gibt es die Möglichkeit, diese über einen Dateiauswahldialog zu öffnen.

[FR.4.3] Die grafische Benutzeroberfläche hat ein Master-Detail-Layout

Da ein einzelner Eintrag zu viele Metadaten hat, um sie alle übersichtlich in eine Tabellenzeile dazustellen, besonders da manche Metadaten mehrfach vorkommen können, bedient sich der Crawler eines Master-Detail-Layouts.

[FR.4.3.1] In der Master-Ansicht werden heruntergeladene Einträge in einer Tabelle gelistet

In einer Tabelle werden Einträge mit den wichtigsten Metadaten aufgelistet.

[FR.4.3.2] In der Detail-Ansicht werden zu einem in der Master-Ansicht ausgewählten Eintrag Metadaten angezeigt

Die Detailansicht zeigt alle in Abschnitt 3.3 aufgeführten Metadaten und zum Eintrag zugehörige Dateien in einem übersichtlichen Format.

[FR.4.3.3] Aus der Detail-Ansicht heraus ist es möglich, zu einem Eintrag zugehörige Dateien zu öffnen

Um gefundene Dateien einfach inspizieren zu können, lassen sich Dateien aus der Detail-Ansicht heraus öffnen.

[FR.4.3.3.1] Dateien werden mit dem im System für die entsprechende Dateiendung assoziierten Programm geöffnet

Um Präferenzen des Nutzers zu wahren und die sonst erforderliche Unterstützung für gängige Dateiformate zu vermeiden, werden geöffnete Dateien mit dem im System eingestellten Programm geöffnet.

[FR.4.3.3.2] Textdateien können optional in einer programminternen Ansicht angezeigt werden

Optional ist es möglich, pure Textdateien in einer programminternen Ansicht zu inspizieren, damit nicht zwingend eine weitere Anwendung geöffnet werden muss, um einfache Textdateien zu betrachten.

Kapitel 3

Grundlagen

Dieses Kapitel vermittelt essentielle Grundlagen zum Verständnis des Programms. In Abschnitt 3.1 wird zunächst definiert, was ein Crawler ist. In Abschnitt 3.2 wird die titelgebende Website PSS vorgestellt. Abschnitt 3.3 beschreibt alle zu Einträgen verfügbare Metadaten. Folgend wird in Abschnitt 3.4 das Format der Website erläutert. Abschließend werden in den Abschnitten 3.5 und 3.6 die beiden wichtigsten Schwachstellendatenbanken, die *Common Vulnerabilities and Exposures* (CVE) und die *National Vulnerability Database* (NVD) beschrieben.

3.1 Crawler

Ein Crawler bzw. Webcrawler wird eingesetzt, um Internetseiten zu indizieren oder archivieren. Seiten werden je nach gewünschter Funktion indiziert oder gespeichert und nach Hyperlinks durchsucht. Gefundene Hyperlinks werden in eine Warteschlange eingereiht und abgearbeitet. Die Suche wird von einer oder mehreren Startseiten, auch *seed* (Keim bzw. Samen) genannt, begonnen und dann rekursiv weitergeführt bis der Crawler stoppt, weil entweder keine Hyperlinks mehr in der Warteschlange sind oder aufgrund anderer Abbruchbedingungen wie das Erreichen einer vorher festgelegten maximalen Rekursionstiefe [12].

3.2 packetstormsecurity.com

In diesem Abschnitt wird die titelgebende Website `packetstormsecurity.com` (PSS) vorgestellt. PSS bietet aktuelle und vergangene Nachrichten und Dateien zu sicherheitsrelevanten Themen wie Exploits und Schwachstellen. Einträge werden von Einzelpersonen und Firmen eingereicht und folgend von Personal überprüft, bevor sie als Einträge auf der Website erscheinen [4].

PSS besteht seit 1998 und richtet sich laut eigener Aussage an Systemadministratoren, welche ihr System auf dem Laufenden halten, Sicherheitsforscher, welche von neue Entdeckungen berichten, Regierungen und Unternehmen, welche aktuelle Ereignisse beobachten müssen, Sicherheitsanbieter, welche neue Software entwickeln und viele mehr [1].
PSS erlaubt Nutzern das Erstellen von Konten, welche es möglich machen, Kommentare zu verfassen und Einträge als Favoriten zu markieren.
Dateilisten können sowohl als HTML-Seite, als auch als RSS-Feed abgerufen werden.
Einträge und somit Dateien besitzen eine laufende ID, beginnend bei 10 000. Aktuell (Stand März 2018) hat PSS über 100 000 Dateien.

3.3 Verfügbare Metadaten

Zu Dateien, die auf PSS publiziert werden, sind die folgenden Metadaten verfügbar:

Titel Kurzer Titel der Datei oder auch oft nur der Dateiname.

Veröffentlichungsdatum Veröffentlichungsdatum der Datei.

Beschreibung Eine Zusammenfassung der Datei, oft mit Ursprung, Inhalt und Funktionsweise der Datei.

Autoren Ein oder mehrere Autoren, bestehend aus Name und einem von vier Typen: Firma, Gruppe, Magazin oder Person.
Es sollte jedoch beachtet werden, dass Autoren nicht immer korrekt getaggt sind: So sind keine der zehn am häufigsten veröffentlichten Autoren Personen, jedoch werden vier davon (*Red Hat*, *Debian*, *HP* sowie *Google Security Research*) als solche markiert (Siehe Anhang A.2.4).

Tags Eine Beliebige Anzahl von Dateitags.

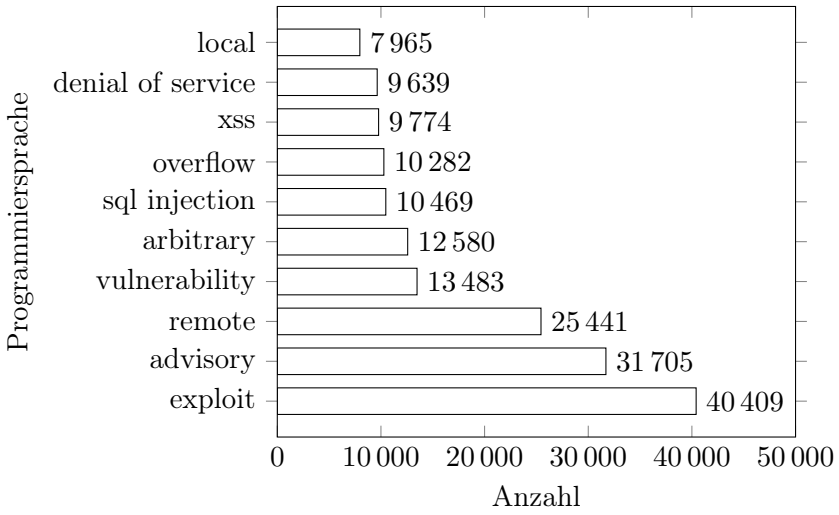


Abbildung 3.1: Die 10 häufigsten Dateitags

Systeme Falls zutreffend werden betroffene Systeme aufgelistet.

Advisories Falls die Datei eine oder mehrere CVE-IDs betrifft, werden diese hier aufgelistet.

Via Ein Link zu der Quelle, falls die entsprechende Datei nicht aus erster Hand kommt.

3.4 Format der Einträge

Alle zuvor erwähnten Metadaten finden sich auf der Kommentarseite einer Datei, jedoch weder in der HTML-Liste noch dem RSS-Feed. Um zu einer Datei alle Metadaten zu erhalten, muss folglich ihre Kommentarseite besucht werden. Um alle Einträge zu crawlen bietet es sich also an, den RSS-Feed zu benutzen. Er bietet dieselben Informationen in kleinerer Dateigröße, da die HTML-Liste zusätzlich zu den Einträgen die komplette Seite enthält.

3.5 Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) ist ein freies Verzeichnis für Informationen zu öffentlich verfügbaren Sicherheitslücken und anderen Schwachstellen. Betrieben wird *Common Vulnerabilities and Exposures* (CVE) vom *United States Computer Emergency Readiness Team* (US-CERT) aus dem *Office of Cybersecurity and Communications* beim *U.S. Department of Homeland Security.*
Vor 1999, dem Schöpfungsjahr von CVE, gab es keinen Industriestandard für die Benennung von Sicherheitslücken. Dadurch konnte es vorkommen, dass gleiche Sicherheitslücken verschiedene Bezeichnungen hatten.
CVE vereinfacht die Kommunikation zwischen Personen und Werkzeugen, indem CVE-Einträge andere Datenbanken referenzieren und sie so verschiedene Bezeichnungen für dieselbe Sicherheitslücke verknüpfen.
CVE ist laut eigener Aussage ein Verzeichnis und keine Datenbank, da CVE-Einträge immer mindestens eine öffentliche Referenz benötigen. Sie enthalten keine weiteren Informationen wie Risiko, Auswirkungen, Anleitungen zur Behebung oder andere detaillierte technische Informationen. Ein CVE-Eintrag besteht so lediglich aus einer Identifikationsnummer, einer Beschreibung und mindestens einer öffentlichen Referenz [2].

3.6 National Vulnerability Database

Die *National Vulnerability Database* (NVD) ist ein Spiegel des CVE-Verzeichnisses, der es um eine zusätzliche Analyse erweitert.
Sie wird betrieben durch das *National Institute of Standards and Technology* (NIST) des *United States Department of Commerce.*
Die NVD ist mit dem CVE-Verzeichnis synchronisiert, sodass Aktualisierungen im CVE sofort in der NVD auftauchen. Spätestens zwei Werktage nach Veröffentlichung eines CVE-Eintrags wird er von Analysten des NVD durch Schwachstellenattribute ergänzt. Diese Attribute sind Auswirkungsmetriken (*Common Vulnerability Scoring System*, CVSS), Schwachstellentyp (*Common Weakness Enumeration, CWE*) und Anwendungsbezug (*Common Platform Enumeration*, CPE) [3].

Kapitel 4

Konzepte des Crawlers

Dieses Kapitel beschreibt zuerst die grundlegende Funktionsweise des Crawlers. Anschließend wird in Abschnitt 4.1 das Ausgabeformat des Crawlers gezeigt. Folgend wird in Abschnitt 4.2 beschrieben, welche lokalen Datenbanken der Crawler pflegt. Abschließend erläutert Abschnitt 4.3, welche Daten der Crawler herunterlädt.
Der entwickelte Crawler hat Ähnlichkeiten mit einem konventionellen Webcrawler, jedoch besitzt er eine Rekursionstiefe von eins und extrahiert nur ausgewählte Hyperlinks. Die Seiten des Rich Site Summary Feeds (RSS-Feeds) werden als Seed benutzt und der Crawler extrahiert alle zu Kommentarseiten führende Hyperlinks sowie die im RSS-Feed verfügbaren Metadaten. Gesammelte Kommentarseiten werden im Anschluss separat besucht und relevante Daten extrahiert.

4.1 Ausgabeformat

Die Ausgabe des Crawlers besteht aus einer SQLite-Datenbank *out.sqlite* und mehreren Ordnern.

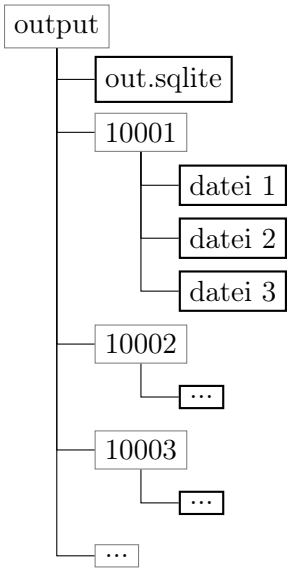


Abbildung 4.1: Beispielhafte Ausgabe

Die Ordner enthalten mindestens die komplette Datei eines einzelnen PSS-Eintrags und gegebenenfalls mehrere weitere Dateien mit Codeschnipseln aus der ursprünglichen Datei. Der Name eines Ordners korrespondiert mit der ID des zugehörigen PSS-Eintrags.
Die Datenbank enthält alle Metadaten der ursprünglichen Einträge und einige Daten zum Fortschritt und der Einstellung des Crawlers.
Die folgenden Abbildungen zeigen die Schemas der Datenbanken, wobei Primärschlüssel **fett**, und Fremdschlüssel *kursiv* stilisiert sind. Der Großteil der Datenbank besteht aus sieben Tabellen und speichert die Metadaten der Einträge.

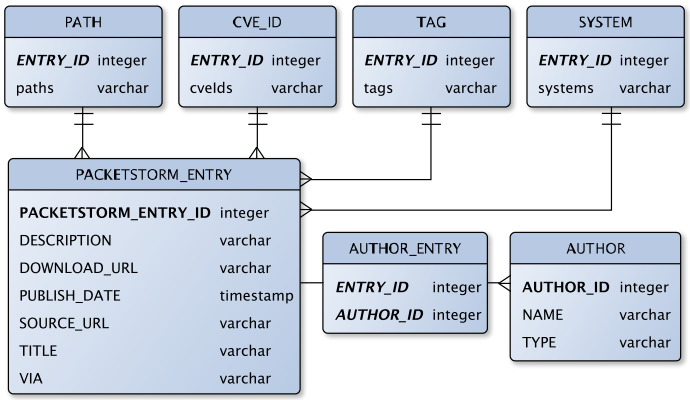


Abbildung 4.2: Entity-Relationship-Modell (ER-Modell) des Datenteils

Die wichtigste Tabelle dieses Teils ist „PACKETSTORM_ENTRY“, welche alle Metadaten zu einem Eintrag enthält, die maximal einmal zu einem Eintrag bestehen. Sie hält nur Einträge derer Dateien, die für die endgültige Ausgabe relevant sind. Ihr Primärschlüssel, „PACKETSTORM_ENTRY_ID“, ist gleichzeitig Fremdschlüssel in den Tabellen „CVE_ID“, „TAG“ sowie „SYSTEM“. Die Daten in diesen Tabellen haben eine *Many-To-Many-Beziehung* zu Einträgen. Da sie jedoch alle nur eine Nicht-Schlüssel-Zeile haben, wurde bewusst auf die Komplexität einer zusätzlichen Join-Tabelle verzichtet. Zusätzlich ist „PACKETSTORM_ENTRY_ID“ ein Fremdschlüssel in „PATH“. Diese Tabelle enthält relative Pfade von zu Einträgen zugeordneten Dateien. Weiterhin besitzt „AUTHOR“ die *Join-Tabelle* „PACKETSTORM_AUTHOR“.

Abbildung 4.3: Entity-Relationship-Modell der Tabelle „CONSIDERED“

Die Tabelle „CONSIDERED“ enthält alle IDs von Einträgen, die bereits für die endgültige Ausgabe betrachtet wurden, unabhängig davon, ob diese Einträge in der Tabelle „PACKETSTORM_ENTRY“ enthalten sind oder nicht. Ohne sie wäre es nicht möglich das Programm zu pausieren, da nicht zwischen verworfenen und noch nicht heruntergeladenen Einträgen unterschieden werden könnte.
Die letzte Tabelle, „META“, hält die Konfiguration des Crawlers. Sie ist nicht Teil des Outputs, sondern wird lediglich genutzt, um nach Schließen der Anwendung die Konfiguration wiederherzustellen.

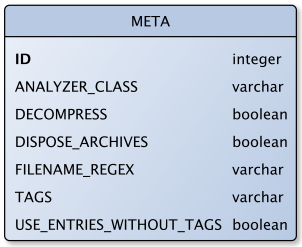


Abbildung 4.4: Entity-Relationship-Modell der Tabelle „META“

4.2 Lokale Datenbanken

Zusätzlich zu den Ausgabedatenbanken verwaltet der Crawler auch eine Globale Datenbank. Sie dient als Cache und enthält alle Tabellen von Abbildung 4.2. Zusätzlich gibt es die Tabelle „RSS_ENTRY“.

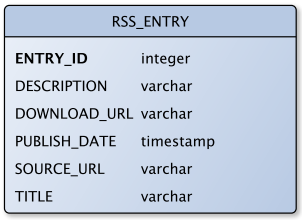


Abbildung 4.5: Entity-Relationship-Modell der Tabelle „RSS_ENTRY“

Sie enthält alle Metadaten, die sich bereits im RSS-Feed von PSS befinden und dient als Cache für den Feed. Einmal in den Cache aufgenommene Einträge können so in beliebiger Reihenfolge heruntergeladen werden.

4.3 Datendownload

Der RSS-Feed von PSS ist kein einzelnes Dokument, sondern ist in mehrere Seiten geteilt. Die Seite mit dem kleinsten Index (Index Eins) ist dabei die neueste Seite, was bedeutet, dass die Seiten des RSS-Feeds nicht statisch sind. Neue Einträge auf Seite eins verdrängen ältere Einträge, die schließlich auf Folgeseiten rutschen. Diese Dynamik macht es schwer, den RSS-Feed als Index für alle Einträge zu benutzen, wenn die Möglichkeit bestehen soll, den Crawler zu pausieren. Wird der Crawler pausiert und werden während der Pause neue Einträge in den RSS-Feed eingefügt, verschieben sich alle Einträge im Feed nach hinten.
Stattdessen baut der Crawler vor der ersten Benutzung zunächst eine lokale Datenbank auf, welche alle Einträge des RSS-Index beinhaltet. Ein Pausieren

ist hier nicht möglich, allerdings auch nicht nötig, da der Download des kompletten Indexes nur wenige Minuten beansprucht. Nachdem einmal ein lokaler Index erstellt wurde, kann er innerhalb weniger Sekunden aktualisiert werden, indem die neuesten Seiten heruntergeladen wurden.
Es besteht das Problem, dass die Einträge nicht in Reihenfolge der ID im RSS-Feed auftauchen. Hierfür gibt es keine erkenntlichen Gründe.
Um wirkliche Einträge herunterzuladen benutzt der Crawler nun die zuvor aufgebaute lokale Datenbank. Kommentarseiten, zuvor aus dem RSS-Feed extrahiert, werden, beginnend mit dem kleinsten Index, besucht und Metadaten extrahiert. Folgend wird die zugehörige Datei heruntergeladen.

Kapitel 5

Implementierung

In diesem Kapitel werden in Abschnitt 5.1 zunächst die Architektur und folgend in 5.2 die grafische Oberfläche des Crawlers beschrieben. Abschließend zeigt Kapitel 5.3, wie der Crawler um weitere Analyseklassen erweitert werden kann.

5.1 Architektur

Die Architektur des Crawler basiert auf dem Entwurfsmuster *Model-View-ViewModel* (MVVM), welches dem Muster *Model-View-Controller* (MVC) ähnelt. Der *View* definiert, wie die grafische Benutzeroberfläche aussieht. Im View wird keine Logik definiert. Beispielsweise weiß der View, ob ein Knopf existiert, jedoch nicht, wann er aktiviert oder deaktiviert ist oder was geschieht, wenn er ausgelöst wird. Das *ViewModel* repräsentiert den Zustand des Views. Es weiß, ob ein Kontrollelement aktiviert oder deaktiviert ist, jedoch nicht, wie es aussieht oder ob es überhaupt existiert. Verbunden werden View und ViewModel, indem öffentlich zugängliche Properties und Methoden aus dem ViewModel im View mit Elementen der grafischen Oberfläche verknüpft werden. Das *Model* enthält domänenspezifische Logik, Persistenzlogik sowie Zustand und existiert im ViewModel [5].
Wird also beispielsweise der Knopf „Index updaten“ in der grafischen Oberfläche betätigt, ruft der View die zuvor für diesen Knopf festgelegte Methode im ViewModel auf. Das ViewModel wiederum fragt die Model-Klasse nach fehlenden Einträgen. Währenddessen wird im ViewModel ein Property „updating“ auf `wahr` gesetzt, welches mit der „deactivated“-Property des „Index updaten“-Knopfs verbunden ist, wordurch der Knopf deaktiviert wird. Hat das Modell die Anfrage abgeschlossen, wird das Resultat an das ViewModel zurückgegeben, welches es für den View aufbereitet. Anschließend wird die „updating“-Property wieder auf falsch gesetzt, was den Knopf „Index updaten“ wieder aktiviert.

5.1.1 Packetstormsecurity-API

Die Datenabfrage geschieht in drei Schritten. Zunächst wird ein `RssIndex` instanziiert. Über ihn kann der lokale Index des RSS-Feeds aktualisiert und abgefragt werden. Wird eine Aktualisierung des lokalen Indexes gestartet, holt sich der `RssIndex` zunächst die Anzahl der Einträge im RSS-Feed von der Website und vergleicht sie mit der Anzahl der Einträge in der Datenbank. Gibt es neue Einträge, werden die neuesten Seiten des RSS-Feeds heruntergeladen, geparst und neue Einträge in die Datenbank eingefügt. Nun können vom `RssIndex` alle `RssEntry`s abgerufen und weiter verarbeitet werden.
Der zweite Schritt ist das Herunterladen der eigentlichen Einträge. Hierzu gibt es die `PacketstormApi` und den `PacketstormFetcher`. `PacketstormApi` dient dazu, einen `RssEntry` zu einem kompletten `PacketstormEntry` zu ergänzen. `PacketstormFetcher` verbindet `PacketstormApi` und `RssIndex`. Um im `PacketstormFetcher` fehlende Einträge in die Warteschlange zu stellen, werden die IDs aller Einträge, die schon heruntergeladen wurden, übergeben und mit den Einträgen im `RssIndex` verglichen. In der übergebenen Liste fehlende Indizes werden in die Warteschlange gestellt. Nun kann der Fetcher so lange abgefragt werden, bis die Warteschlange leer ist und fehlende Einträge heruntergeladen wurden. Der `PacketstormFetcher` ist so konzipiert, dass Abfragen seriell oder parallel geschehen können. Ein paralleles Abfragen steigert die Performanz erheblich (Abb. 5.1.1).

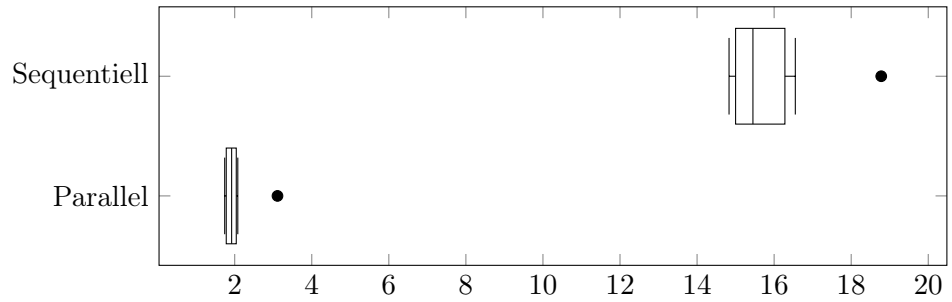


Abbildung 5.1: Vergleich von parallelen und sequentiellen Downloadzeiten von je 100 Einträgen, siehe auch A.1

5.1.2 Datenbanken

Auf Datenbanken wird mittels Data Access Objects (DAOs) zugegriffen. Um beispielsweise neue Einträge in die Datenbank einzupflegen, ruft der `RssCache` eine vorher im Konstruktor injizierte Implementierung eines `RssEntryDao`s auf. Durch diesen Umweg wird eine zusätzliche Abstraktionsschicht zwischen Implementation und Datenbank geschaffen. Diese

ermöglicht es, verschiedene Persistenzverfahren einzusetzen, ohne dass an der Implementation Änderungen vorgenommen werden müssen.
Die Implementierung der DAOs erfolgt durch *Hibernate*, eine von Red Hat unter der *GNU Lesser General Public License* (LGPL) vertriebenen Implementation der *Java Persistence API* (JPA). Solche Implementationen werden auch *JPA Provider* genannt. Die DAO-Implementationen des Crawlers sind so konzipiert, dass sie mit beliebigen JPA-Providern funktionieren. Die abstrakte Klasse `JpaDao<E extends Serializable>` implementiert alle Methoden des Interfaces `Dao<E extends Serializable>`. Dies bietet eine grundlegende Implementation, die für alle zu persistierenden Objekte funktioniert und gleichzeitig unabhängig vom *JPA-Provider* sowie der benutzten Datenbank ist. Durch JPA ist es nicht nötig, wie mit der sonst üblichen Bibliothek *Java Database Connectivity* (JDBC) händisch `ResultSet`s auszuwerten, auch SQL-Anfragen werden nur für spezialisiertere Anfragen benötigt. Es wurde sich aufgrund der Fehleranfälligkeit und Aufwändigkeit von händischen JDBC-Implementationen sowie der bereits genannten Gründe, Einfachheit und Erweiterbarkeit, für JPA mit Hibernate entschieden.

5.2 Grafische Benutzeroberfläche

In diesem Abschnitt wird der Aufbau und die Funktionsweise der grafischen Benutzeroberfläche erläutert. Zunächst werden in den Abschnitten 5.2.1 und 5.2.2 die Menü- und die Funktionsleiste gezeigt. Da ein Eintrag zu viele Metadaten besitzt, um sie alle komfortabel in einer Tabellenspalte anzuzeigen, hat die grafische Oberfläche ein Master-Detail-Layout, welche im Detail in den Abschnitten 5.2.3 und 5.2.4 beschrieben sind. In Abschnitt 5.2.5 findet sich die dritte Hauptansicht, die es dem Nutzer erlaubt, Textdateien zu sichten. Die Abschnitte 5.2.6 und 5.2.8 behandeln die in Extrafenster ausgelagerten Sichten für Statistiken und zum Verwalten von Analyse-Klassen. Abschließend wird in Abschnitt 5.2.7 der Dialog zum Erstellen neuer Datenbanken vorgestellt.

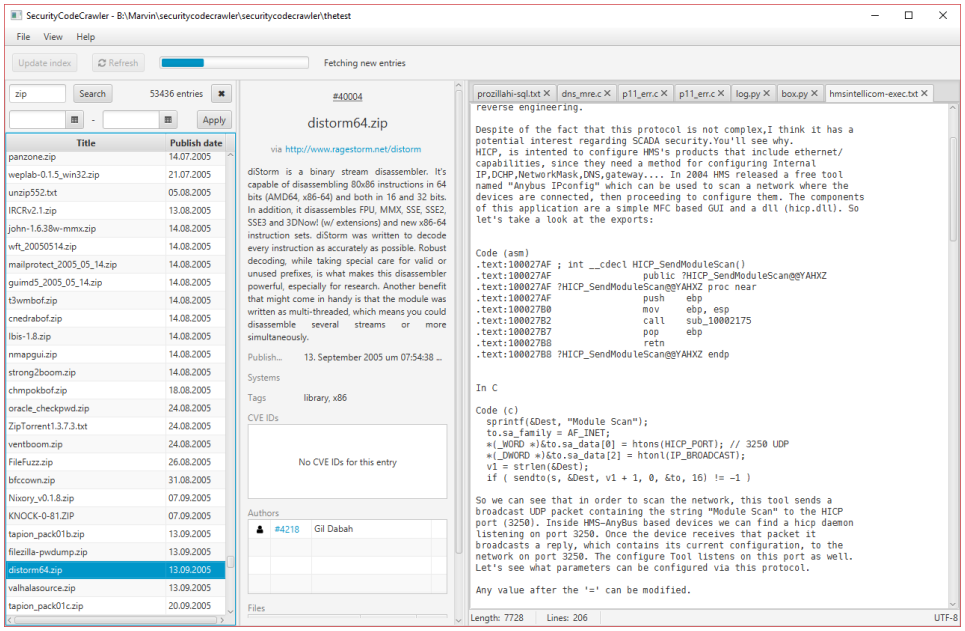


Abbildung 5.2: Übersicht über die Grafische Benutzeroberfläche

5.2.1 Menüleiste

Der Crawler besitzt eine Menüleiste mit den drei Menüs *File*, *View* sowie *Help*.

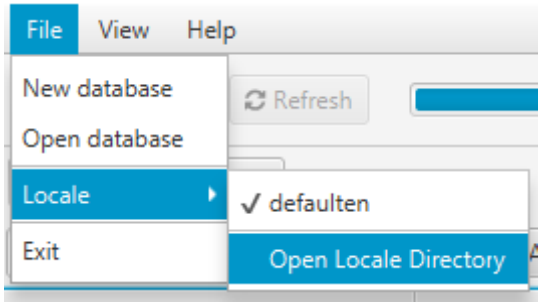


Abbildung 5.3: Das *File*-Menü

Unter *File* (Abb. 5.3) lässt sich die Sicht zum Konfigurieren und Erstellen einer neuen Datenbank öffnen (Abschnitt 5.2.7), eine bestehende Datenbank öffnen sowie der Crawler schließen. Hier kann der Nutzer auch die Lokalisierung ändern und den Lokalisierungsordner im Dateimanager des Systems öffnen, um eine neue Lokalisierung hinzuzufügen.

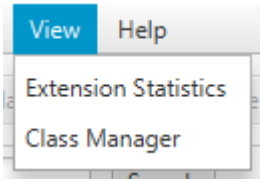


Abbildung 5.4: Das *View*-Menü

Unter *View* (Abb. 5.3) lassen sich die Statistik-Sicht (Abschnitt 5.2.6) und der Analyseklassenmanager (Abschnitt 5.2.8) öffnen.

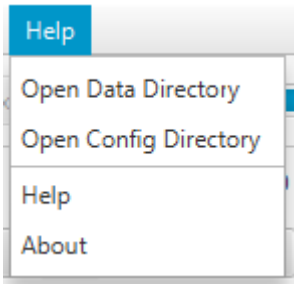


Abbildung 5.5: Das *Help*-Menü

Abschließend lässt sich unter *Help* (Abb. 5.5) der Konfigurationsordner im Dateimanager öffnen, beispielsweise um die Konfiguration anzupassen

oder die globale Datenbank zu manipulieren, sowie der Datenordner im Dateimanager öffnen, um Daten oder die Datenbank zu kopieren oder extern zu öffnen.

5.2.2 Funktionsleiste

Auf der Funktionsleiste (Abb. 5.6) gibt es zwei Knöpfe, jeweils zum Aktualisieren der globalen und gerade geöffneten Datenbank. Ist gerade eine Aktualisierung aktiv, lässt sich keine weitere durchführen, die Buttons werden deaktiviert. Neben den Buttons gibt es einen Fortschrittsbalken und ein Label, welches den aktuellen Status anzeigt. Beim Aktualisieren der globalen Datenbank zeigt die Fortschrittsleiste nur unbestimmten Fortschritt. Ist die Aktualisierung abgeschlossen, wird im Label eine Erfolgsnachricht zusammen mit der Anzahl heruntergeladener Einträge angezeigt.

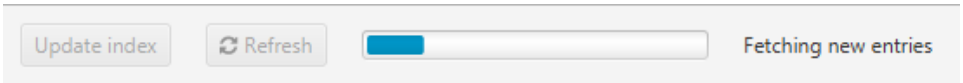


Abbildung 5.6: Funktionsleiste während der Suche nach neuen Einträgen

Wird die aktuell ausgewählte Datenbank aktualisiert, zeigt der Fortschrittsbalken den Fortschritt der zu aktualisierenden Dateien an. Auch hier wird bei Abschluss der Aktualisierung eine Erfolgsnachricht inklusive Anzahl heruntergeladener Dateien angezeigt.

5.2.3 Master-Sicht

In der Master-Sicht (Abb. 5.8) werden nur Titel und Veröffentlichungsdatum als wichtigste Metadaten in einer Tabelle angezeigt. Sie ist nach Titel oder Veröffentlichungsdatum sortierbar, zusätzlich kann nach Titel und Veröffentlichungsdatum gefiltert werden. Die Anzahl der in der Tabelle angezeigten Dateien werden in einem separaten Label gezeigt. Dies macht es ersichtlich, wie viele Dateien momentan in der Datenbank sind, wie viele Dateien bei einer Suche gefunden wurden sowie ob aktuell neue Dateien hinzugefügt werden.

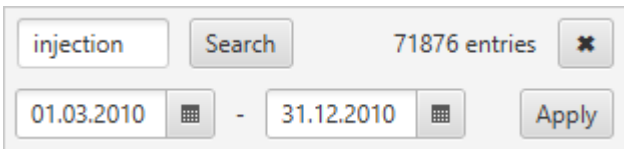


Abbildung 5.7: Suchleiste der Master-Sicht mit Beispielsuche

Markiert der Nutzer eine Datei, öffnet sich diese in der Detail-Sicht. Ist der Crawler gerade dabei, ein Update der Datenbank durchzuführen, wird

die Tabelle laufend aktualisiert, das heißt, neue Einträge werden sofort an das Ende der Tabelle angefügt und sind sofort ersichtlich. So kann der Nutzer direkt Dateien sichten und muss nicht auf Vollendung des Crawlens warten, was mitunter sehr lange dauern kann (Siehe Abschnitt 6.1).

Title	Publish date
CVE Checker 3.5	30.09.2013
Gentoo Linux Security Advisory 201...	28.07.2014
Debian Security Advisory 2990-1	28.07.2014
Debian Security Advisory 2991-1	28.07.2014
Red Hat Security Advisory 2014-094...	28.07.2014
ZeroCMS 1.0 Cross Site Scripting	28.07.2014
Web Encryption Extension Authentic...	28.07.2014
CMSimple 4.4.4 RFI / Code Executio...	28.07.2014
Ground Zero Summit (G0S) 2014 Ca...	28.07.2014
MasterCard Open Redirect	28.07.2014
WordPress FBGorilla SQL Injection	28.07.2014
WordPress Lead Octopus Power SQ...	28.07.2014
Sagem F@st 3304-V1 Denial Of Ser...	28.07.2014
DirPHP 1.0 Local File Inclusion	28.07.2014
Parallels Tools 9.0 Privilege Escalation	28.07.2014
SILC (Secure Internet Live Conferenc...	23.07.2014
WordPress Slider Revolution Respo...	28.07.2014
Barracuda Networks Spam / Virus Fi...	28.07.2014
dtSearch Desktop Untrusted Library ...	28.07.2014
iTunes Manifest.mbdb Parser	28.07.2014
Redis Portscan Utility	28.07.2014
HP Security Bulletin HPSBGN02936	28.07.2014
SQLmap Cheatsheet 1.0	28.07.2014
Android SDK SQL Injection	28.07.2014
Oxwall 1.7.0 Cross Site Request Forg...	28.07.2014
Oxwall 1.7.0 Remote Code Execution	28.07.2014

Abbildung 5.8: Tabelle der Master-Sicht

5.2.4 Detail-Sicht

In der Detail-Sicht (Abb. 5.9) werden alle verbleibenden Metadaten in einem übersichtlichen Format angezeigt.

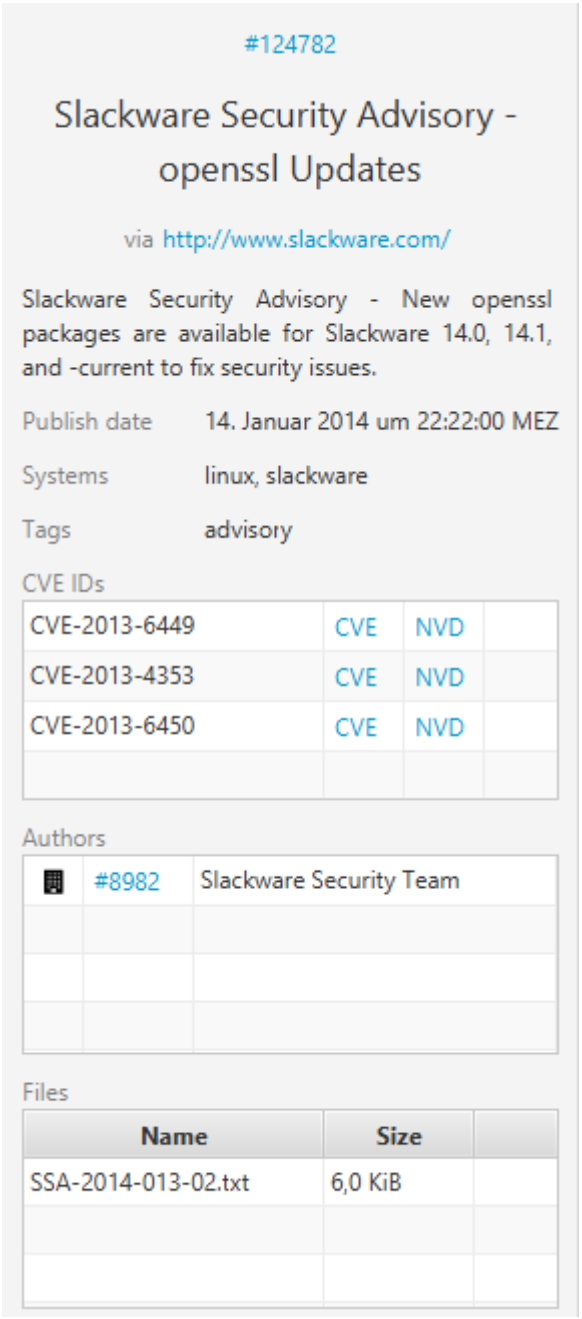


Abbildung 5.9: Detailsicht, die zu einem Eintrag zugehörige Metadaten anzeigt

Zuerst steht die ID des Eintrags als Hyperlink, verweisend auf die Kommentarseite des Eintrags, in dieser Sicht. Darunter folgen Titel, Beschreibung, Veröffentlichungsdatum, eine kommaseparierte Liste von betroffenen Systemen sowie eine kommaseparierte Liste von Tags. Abschließend zeigen drei Tabellen jeweils zu dem gewählten Eintrag zugehörige CVE-IDs, Autoren mit Hyperlink zu der Autorenseite auf PSS sowie Dateien und ihre Größe auf dem Datenträger. Doppelklickt der Nutzer eine Textdatei, öffnet sich in der in Reiter, der diese Textdatei anzeigt. Mit einem Rechtsklick auf eine Datei lässt sich ein Kontextmenü (Abb. 5.10) öffnen.

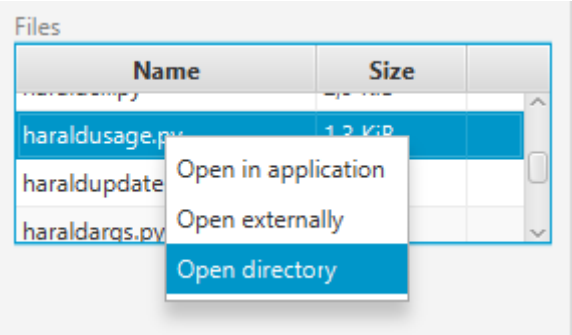


Abbildung 5.10: Kontextmenu der Detail-Sicht

Um es dem Nutzer zu erleichtern, Dateien extern zu manipulieren, lassen sich aus diesem Menü Dateien mit dem im System für die entsprechende Dateiendung assoziierten Programm, beispielsweise *Notepad++* für Textdateien, zu öffnen. Auch ist es möglich, den Dateimanager am Speicherort des Programms zu öffnen.

5.2.5 Datei-Sicht

In der Datei-Sicht werden geöffnete Textdateien in Reitern angezeigt, Text kann hier direkt herauskopiert werden (Abb. 5.11).

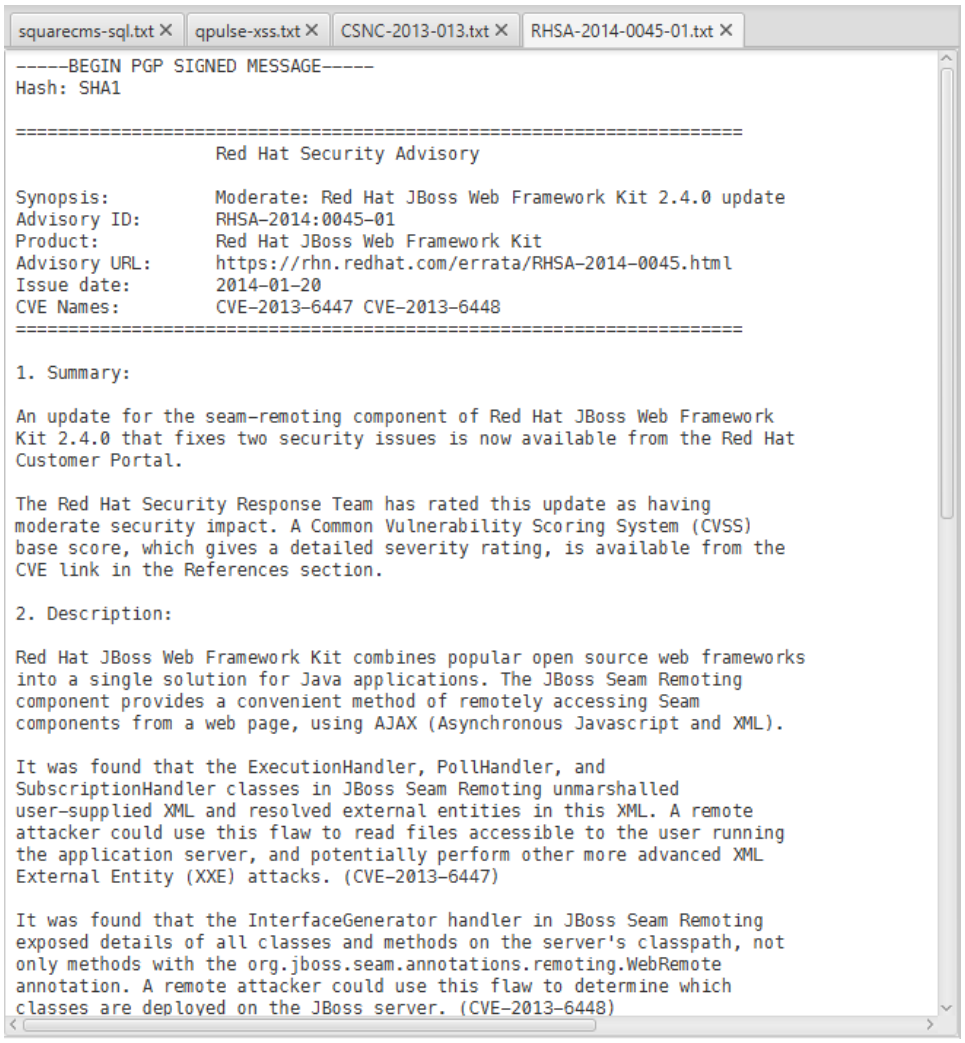


Abbildung 5.11: Datei-Sicht mit mehreren geöffneten Textdateien

Am unteren Sichtrand werden Zeichen- und Zeilenanzahl der Datei präsentiert (Abb. 5.12).

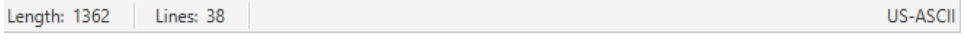


Abbildung 5.12: Informationsleiste der Dateisicht

5.2.6 Statistik-Sicht

Damit der Nutzer schnell einen Überblick über die gefundenen Dateien bekommt, kann die Statistik-Sicht benutzt werden.

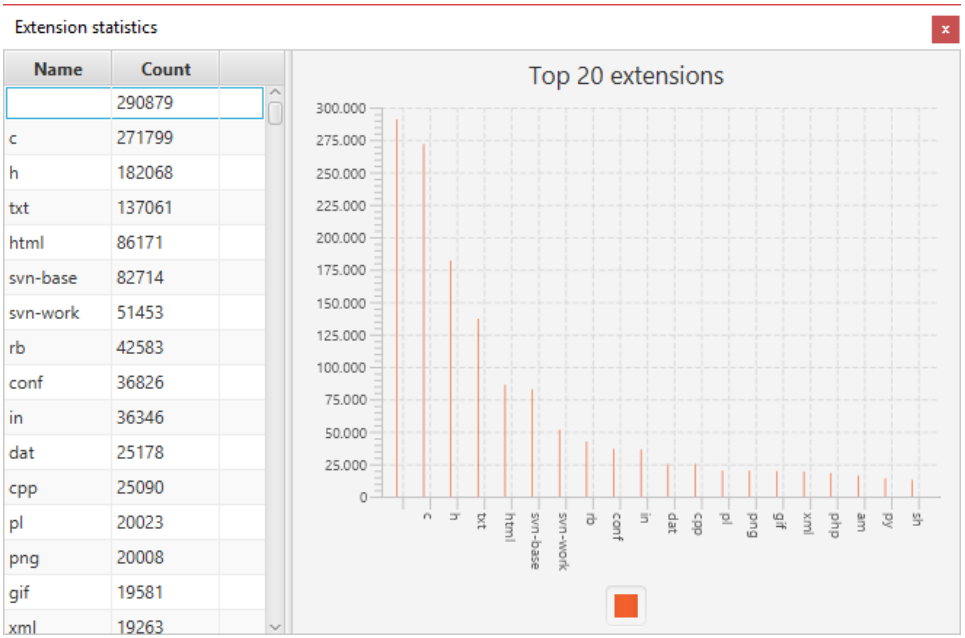


Abbildung 5.13: Sicht der Statistik von Dateiendungen

Sie zeigt links in einer Tabelle alle Dateiendungen zusammen mit der Anzahl und rechts einen Balkengraph der 20 häufigsten Dateiendungen.

5.2.7 Sicht für neue Datenbanken

Aus dieser Sicht können neue Datenbanken konfiguriert und erstellt werden. Zunächst wählt der Nutzer in einem Datei-Auswahldialog den Ort, an dem die neue Datenbank angelegt werden soll. Folgend wählt er Dateiendungen, die heruntergeladen werden sollen. Sind keine Dateiendungen gewählt, werden alle Dateien heruntergeladen. In einer Tabelle können nun Tags ausgewählt werden. Werden ein oder mehrere Tags gewählt, werden nur Dateien heruntergeladen, die mindestens eines dieser Tags besitzen. Ist kein Tag gewählt, werden Dateien unabhängig von den gegebenen Tags heruntergeladen. Wird der Auswahlkasten *Use entries without tags* gewählt, werden Einträge, die keine Tags besitzen, ebenfalls heruntergeladen, auch wenn nach anderen Tags gefiltert wird.
Ist der Auswahlkasten *Extract archives* gewählt, werden Archive heruntergeladen und extrahiert, ansonsten werden sie unabhängig von den gewählten Dateiendungen nicht mit heruntergeladen. Wenn Archive extrahiert werden

sollen, kann über den Auswahlkasten *Dispose archives* gesteuert werden, ob Archive nach dem Entpacken gelöscht werden sollen oder nicht.

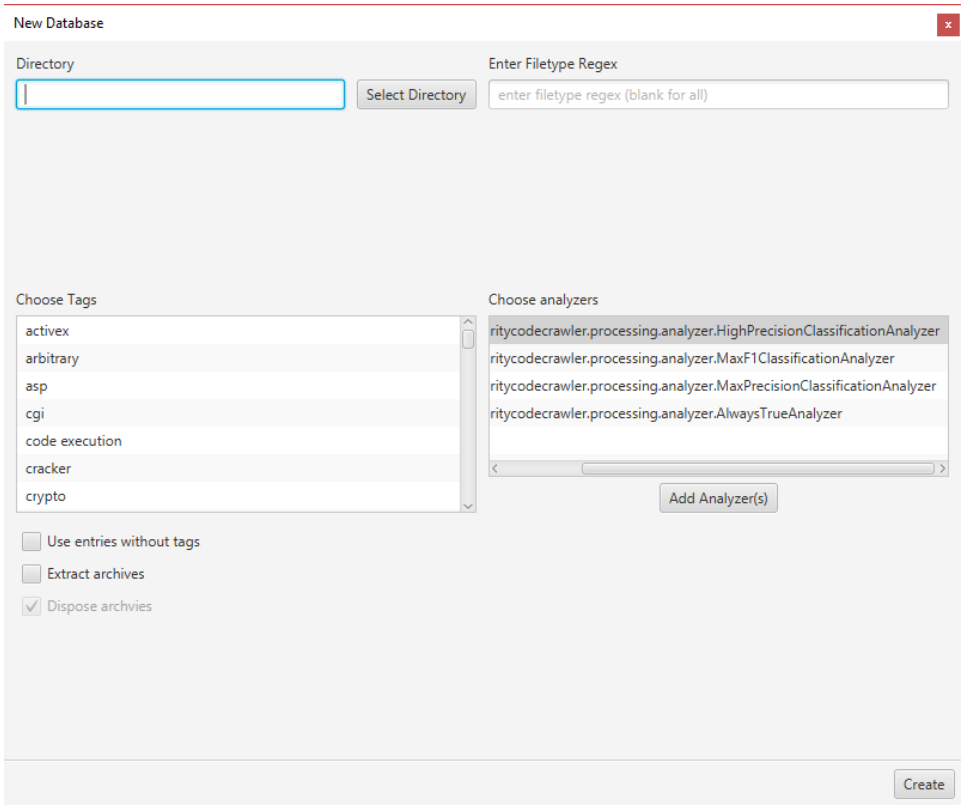


Abbildung 5.14: Sicht für neue Datenbanken

5.2.8 Analyseklassen-Manager-Sicht

Diese Sicht erlaubt es dem Nutzer, selbstgeschriebene Klassen oder Klassen Dritter zur Analyse hinzuzuziehen (Siehe Abschnitt 5.2.8). In einer Tabelle werden geladene Klassen und ihr Speicherpfad angezeigt. Sind sie verschoben oder gelöscht worden, werden sie rot hinterlegt. Am rechten Rand der Tabelle lassen sich Klassen löschen, wenn sie nicht mehr benötigt werden. Bereits mitgelieferte Analyse-Klassen werden in dieser Sicht nicht angezeigt, sie soll lediglich zum Verwalten externer Klassen dienen.

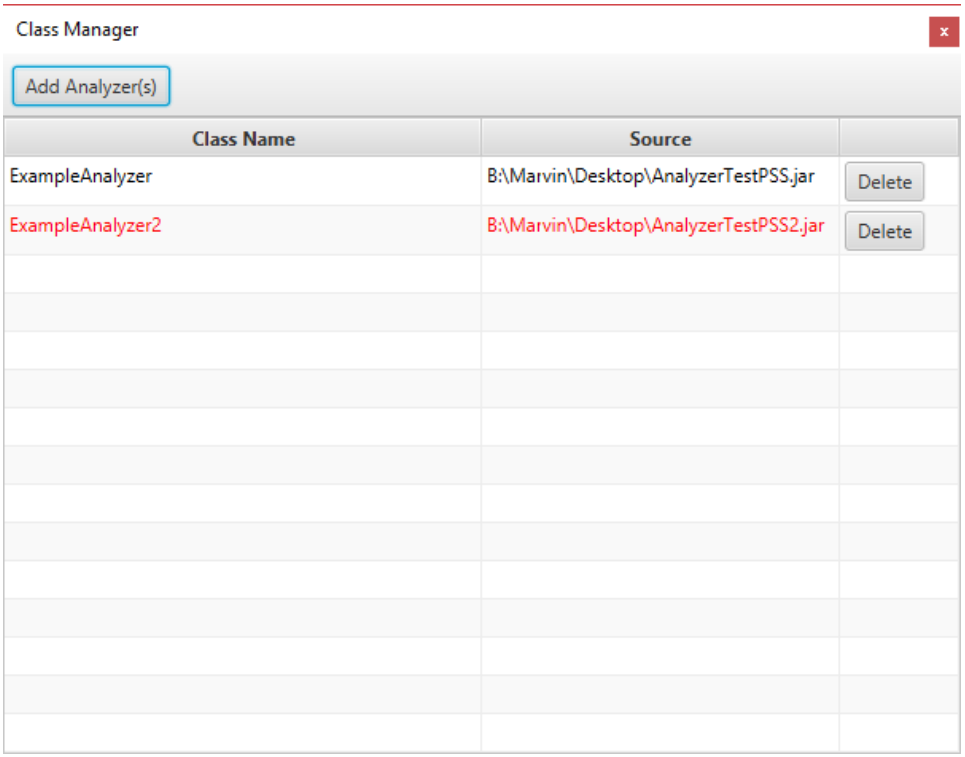


Abbildung 5.15: Analyseklassen-Manager-Sicht

5.3 Erweiterbarkeit durch Analyseklassen

Über den assen sich beliebig viele externe Analyseklassen in den Crawler laden, jedoch nur eine pro Datenbank. Analyseklassen müssen lediglich das in Abbildung 5.16 gezeigte Interface `Analyzer <E extends Entry>` implementieren, so lässt sich die Dateianalyse erweitern, ohne dass jedes mal das komplette Projekt neu kompiliert werden muss.

```java
package net.marvk.securitycodecrawler.processing;

import java.nio.file.Path;

@FunctionalInterface
public interface Analyzer<E extends Entry> {
    boolean analyze(final E entry, final Path workingDirectory);
}
```

Abbildung 5.16: Die Klasse Analyzer <E extends Entry>

Es handelt sich um ein Interface mit einer Methode. Sie bekommt einen Eintrag und das Arbeitsverzeichnis (um Pfade des Eintrags auflösen zu können) übergeben und gibt zurück, ob der Eintrag behalten werden soll. Eine Beispielimplementation, die repräsentativ nur den Titel eines Eintrags ausdruckt und nur Einträge, deren Titel mit „A“ bzw. „a“ beginnen, behält, ist in Abbildung 5.17 zu sehen.

```java
import net.marvk.securitycodecrawler.packetstorm.model.
    PacketstormEntry;
import net.marvk.securitycodecrawler.processing.Analyzer;

import java.nio.file.Path;

public class ExampleAnalyzer implements Analyzer<
    PacketstormEntry> {
    @Override
    public boolean analyze(final PacketstormEntry entry, final
        Path path) {
        final String title = entry.getTitle();
        System.out.println(title);

        return title.charAt(0) == 'A' || title.charAt(0) == 'a';
    }
}
```

Abbildung 5.17: Beispielhafte Implementation einer Analyseklasse

Kapitel 6

Evaluation

6.1 Vorhandene Dateien

Um die auf PSS vorhandenen Daten auszuwerten zu können, wurden zunächst alle Daten heruntergeladen. Es wurden zwei Datensätze erstellt, einer inklusive dekomprimierter Archive sowie einer, bei dem die Archive nicht dekomprimiert wurden. Dateien wurden gezählt und die Gesamtgröße des Datensatzes ermittelt. Anschließend wurden die Dateiendungen in beiden Datensätzen gezählt (Siehe A.2.1 u. A.2.2). Hier fällt auf, dass beide neben Quellcode auch eine hohe Anzahl an Textdateien beinhalten, beim Datensatz ohne Dekompression sind es über 70% der Dateien. Diese Dateien könnten weiteren Quellcode enthalten. Im Datensatz ohne Dekompression sind nur sehr wenige Java-Dateien enthalten, die für Brunottes Eclipse Plugin [10] von Interesse sind. Der Datensatz mit Dekompression enthält zwar circa 40 000 Java-Dateien, diese zahl wird aber durch in Java geschriebene komplette Projekte, die als Archiv auf PSS veröffentlicht wurden, aufgeblasen. Java-Code explizit mit Schwachstellen ist kaum vorhanden.

Datensatz	ohne Dekompression	mit Dekompression
Anzahl Dateien	100 212	3 312 291
Größe	31,08GB	66,75GB
Datum des Downloads	13.02.2018	22. und 23.02.2018
Laufzeit	1:49h	16:00h

Abbildung 6.1: Statistik der Datensätze

Die erhöhte Größe und längere Laufzeit für den Datensatz mit Dekompression wird ausschließlich durch das Entpacken der Archive verursacht, heruntergeladen wurden dieselben Daten. Aus den knapp 17 500 Archiven konnten über 3,2M Dateien gewonnen werden. Der Testcomputer ist ein Desktop-Computer mit einem Intel Core i5-4690K@3,5GHz mit 16GB RAM

sowie einer Internetleitung mit 200 000kbit/s Downloadrate. Der Crawler hatte beim Erstellen des Dekompressionsdatensatzes eine konstante CPU-Auslastung von über 60%.
Des Weiteren wurden untersucht, in welchen Jahren wie viele Einträge (Siehe Anhang A.2.3) sowie welche Autoren am meisten veröffentlicht wurden (Siehe Anhang A.2.4).

6.2 Quellcode-Erkennung

In diesem Abschnitt werden *Klassifizierer* gezeigt, die Quellcode in Textdateien erkennen sollen. Unter einem Klassifizierer versteht man eine Methode zur Klassifizierung von Elementen einer Menge in verschiedene Klassen. Die Klassen der Elemente sind dem Klassifizierer jedoch nicht bekannt, er muss anhand von Klassifizierungsregeln entscheiden, welcher Klasse ein Element angehört. Geht es um die Unterscheidung zwischen genau zwei Klassen, wie im vorliegenden Fall (Quellcode vs. kein Quellcode), wird auch von einem *binärer Klassifizierer* gesprochen.
In Abschnitt 6.2.1 wird ein Benchmark erstellt, mit dem die Klassifizierer getestet werden könnten. Anschließend werden in Abschnitt 6.2.2 Metriken vorgestellt, die einen Vergleich von verschiedenen Klassifizierern erleichtern sollen. Abschnitt 6.2.3 widmet sich dem eigentlichen klassifizieren. Schließlich wird in Abschnitt 6.2.4 die Performanz der Klassifizierer verglichen.

6.2.1 Benchmark

Dateien	Anzahl	Stichprobenumfang(N)	daraus mit Quellcode
.txt	70 674	383	84

Abbildung 6.2: Umfang der Textdateien und Stichprobe

Um zu bestimmen, wie effektiv eine Methode der Quellcode-Erkennung ist, wird zunächst ein Benchmark benötigt. Der Benchmark beschränkt sich hier auf Textdateien aus dem Datensatz ohne Dekompression. Da es nicht praktikabel ist, alle 70 674 Textdateien händisch auszuwerten, wird eine zufällige Stichprobe entnommen. Um eine ausreichend große Stichprobe zu entnehmen, wird für die Größe folgende Formel benutzt [9]:

$$\hat{p} = 1 - \hat{q} \tag{6.1}$$

$$\sigma = \hat{p}\hat{q} \tag{6.2}$$

$$n_0 = \frac{(z_{\alpha/2})^2 \sigma}{E^2} \tag{6.3}$$

$$n = \frac{n_0 N}{n_0 + (N-1)} \tag{6.4}$$

$\hat{p}$ ist ein wert in $[0;1]$, er repräsentiert den Anteil an Textdateien, der Quellcode enthält. $\hat{q}$ ist der Anteil an Textdateien, der keinen Quellcode enthält, daraus folgt (6.1). Das Produkt aus $\hat{p}$ und $\hat{q}$ ist die Standardabweichung σ (6.2). Durch (6.3) erhält man die Stichprobengröße für unendlich große Populationen n_0. Wird nun für die Populationsgröße N korrigiert (6.4), erhält man die Stichprobengröße n.

Da $\hat{p}$ und somit $\hat{q}$ unbekannt sind, weil es keine bekannte Analyse zu Textdateien auf PSS gibt, wird der schlimmste mögliche Fall angenommen, also der, bei dem die Stichprobe am größten ist. Dies ist $\hat{p} = \hat{q} = 0,5$ und somit $\sigma = 0.25$.
Das Konfidenzintervall wird auf 95% und der Fehler E auf 5% festgelegt. Daraus folgt die in Abbildung 6.2 ersichtliche Stichprobengröße von 383. Wenn in f% aller Textdateien Quellcode vorkommt, ist durch Nutzen obiger Formel zu 95% sicher, dass in $(f \pm 5)$% der Dateien in der Stichprobe auch Quellcode vorkommt.
Um das Sichten von Dateien einfach zu machen wurde ein spartanisches Tool entwickelt.

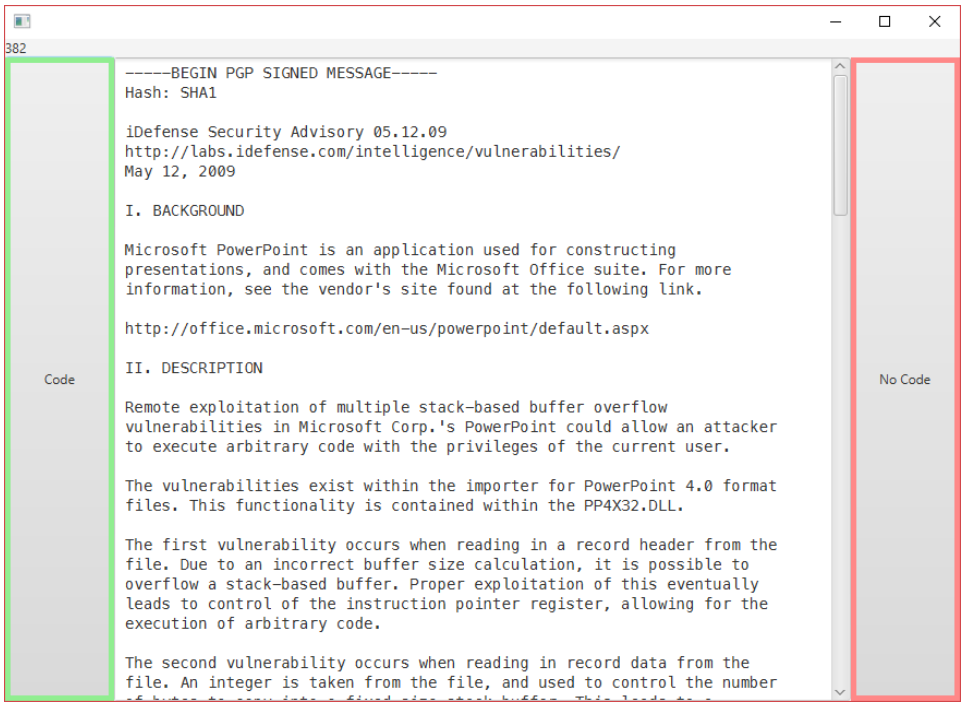


Abbildung 6.3: Tool zum Sichten und Klassifizieren von Textdateien

Hier werden nacheinander alle 383 Textdateien angezeigt, der Nutzer kann sie komplett sichten und dann mit einem Knopfdruck auf entweder *Code* oder *No Code* klassifizieren und in separate Ordner kopieren. So müssen sie nicht händisch in einem Texteditor geöffnet und manuell kopiert werden. Zusätzlich wurde händisch gezählt, welche Programmiersprache(n) Dateien mit Quellcode enthalten (Abb. 6.2.1). Vier Dateien enthalten hier je zwei verschiedene Programmiersprachen: Zwei Mal C++ und Python, ein Mal C und Python sowie ein Mal PHP und SQL.

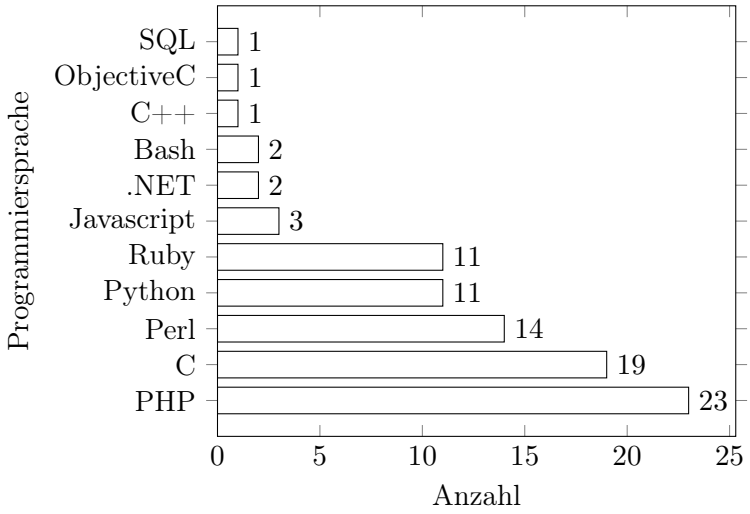


Abbildung 6.4: Programmiersprachenhäufigkeit in der Stichprobe

6.2.2 Bewertung

Um Klassifizierer vergleichen zu können, werden die für binärer Klassifizierer bekannten Metriken *Precision* (*Genauigkeit*) und *Recall* (*Empfindlichkeit*) verwendet. Precision ($\frac{TP}{TP+FP}$) ist das Verhältnis zwischen *True Positive* (*richtig Positiv*, TP) und der Summe aus TP und *False Positive* (*falsch Positiv*, FP). Sie gibt den Anteil der korrekt als `wahr` klassifizierten Elemente im Verhältnis zu der Gesamtheit der als `wahr` klassifizierten Elemente an. Recall ($\frac{TP}{TP+FN}$) ist das Verhältnis zwischen TP und der Summe aus TP und *False Negative* (*falsch Negativ*, FN). Er beschreibt den Anteil von korrekt als `wahr` klassifizierten Elementen im Verhältnis zu den tatsächlich wahren Elementen.
Eine dritte Metrik ist der *F₁-Score*. Der *F₁-Score* ($\frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}$) bildet das harmonische Mittel zwischen Recall und Precision, durch ihn lässt sich die Güte von unterschiedlichen Klassifizierer mit einer einzelnen Metrik vergleichen.
Um nun eine Messbasis zu haben, soll noch berechnet werden, was ein naiver, zufällig wählender Klassifizierer leisten könnte. Sei N die Anzahl an Dateien in der Stichprobe und n_{code} die Anzahl an Dateien in der Stichprobe, die Code enthalten. Dann ist $p = \frac{n_{code}}{N}$ die Wahrscheinlichkeit, dass eine zufällig gewählte Datei Quellcode enthält. Sei nun q die Wahrscheinlichkeit, dass der naive Klassifizierer ein Element als positiv („hat Code“) klassifiziert.

Es ergeben sich

$$TP = p \quad * q \tag{6.5}$$
$$FP = (1-p) * q \tag{6.6}$$
$$FN = p \quad * (1-q) \tag{6.7}$$

und

$$Precision = \frac{TP}{TP+FP} = \frac{p*q}{p*q+(1-p)*q} = \frac{p*q}{q} = p \tag{6.8}$$
$$Recall = \frac{TP}{TP+FN} = \frac{p*q}{p*q+p*(1-q)} = \frac{p*q}{p} = q \tag{6.9}$$
$$F_1 = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}} = \frac{2}{\frac{1}{q}+\frac{1}{p}} \tag{6.10}$$

also ist die Precision unabhängig von q:

$$Precision = p = \frac{n_{code}}{N} = \frac{84}{383} \approx 0,219 \tag{6.11}$$

Ein möglichst hoher Recall wird erreicht, indem q auf den Maximalwert von 1,0 gesetzt wird. Für den Klassifizierer bedeutet dies, dass jede Datei als positiv klassifiziert wird. Nun ergibt sich der maximale *F_1-Score*:

$$F_1 = \frac{2}{\frac{1}{q}+\frac{1}{p}} = \frac{2}{\frac{1}{1}+\frac{1}{0,219}} \approx 0,36 \tag{6.12}$$

Zusammenfassend hat ein naiver Klassifizierer also eine eine Precision von 0,219, einen maximalen Recall von 1,0 sowie einen maximalen *F_1-Score* von 0,36. Dies sei also die Messbasis für die folgenden Klassifizierer.

6.2.3 Klassifizierer

In diesem Abschnitt werden Versuche gezeigt, Dateien, die Quellcode enthalten, zu klassifizieren.

Schlüsselwörter

Schlüsselwörter bilden wesentliche Tokens von Programmiersprachen. Dieser Klassifizierer untersucht die Frequenz von Schlüsselwörtern unter allen Tokens einer Datei. Die Tokens werden gewonnen, indem der Inhalt einer Datei jeweils an einem Zwischenraumzeichen getrennt wird, also an Leerzeichen, Tabulatoren, Zeile- bzw. Seitenvorschüben, Wagenrückläufe u. Ä. Als Schlüsselwörter werden hier die zehn häufigsten Schlüsselwörter aller in der Stichprobe mehr als zwei Mal vorkommenden Programmiersprachen genutzt.

Die Schlüsselwörter wurden dem Projekt *common-words* von Kashcha [13] entnommen, welches einen Schlüsselwort-Index von circa drei Millionen quelloffenen GitHub-Repositorys enthält.
Diese Schlüsselwörter (Siehe Anhänge A.3.1 und A.3.2) wurden nun einzeln auf ihrer Precision bei verschiedenen Schwellenwerten getestet. Schlüsselwörter, die nie eine Precision über 0,25 erreichen, wurden entfernt.

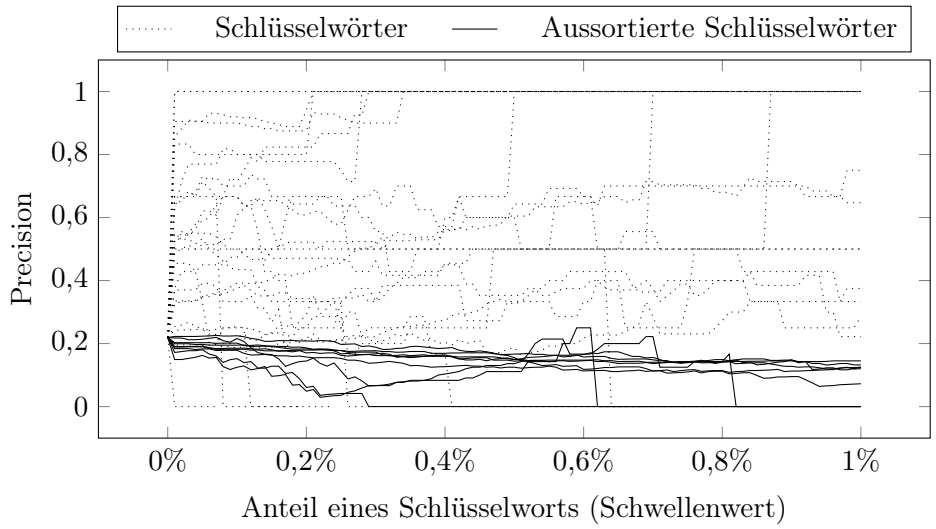


Abbildung 6.5: Precision für einzelne Schlüsselwörter

Die entfernten Schlüsselwörter „a“, „for“, „in“, „public“, „should“, „the“, „this“, „to“ sowie „use“ sind häufige Wörter in der englischen Sprache und kommen daher nicht vorwiegend in Quellcode, sondern in natürlichsprachlichem Text vor. Mit den verbleibenden Wörtern (Siehe Anhang A.3.3) wurde nun ein Klassifizierer erstellt, der die Häufigkeit aller Schlüsselwörter berechnet.
Der Klassifizierer (Abb. 6.2.3) hat einen *F_1-Score* von bis zu 0,56 bei einem Schwellenwert von 7,5 Wörtern pro Datei, hier erreicht er eine Precision von 0,52 und Recall von 0,60, er ist also dem naiven Klassifizierer überlegen. Ist eher hohe Precision gewünscht, erreicht er bei einem Schwellenwert von 50,5 einen Precision-Wert von 0,88 bei einem Recall von 0,095. Die maximale Precision von 1,0 erreicht er bei einem Schwellenwert von 86,5, bei einem Recall von nur noch 0,024.
Dieses Ergebnis kann noch verbessert werden, indem statt einer absoluten Anzahl Schlüsselwörter der Anteil an Schlüsselwörtern in einer Datei berechnet wird (Abb. 6.2.3). Dieser verbesserte Schlüsselwort-Klassifizierer hat bei einem Schwellenwert von 1,85% einen maximalen *F_1-Score* von 0,59 bei einer Precision von 0,70 und einem Recall von 0,52. Er erreicht bei einem Schwellenwert von 3,42% seine maximale Precision von 0,93 bei einem Recall von 0,17. Dies stellt zwar eine Verbesserung im Vergleich zum

vorangegangenen Klassifizierer da, allerdings wird die maximal mögliche Precision von 1,0 mit diesem Klassifizierer nicht erreicht.

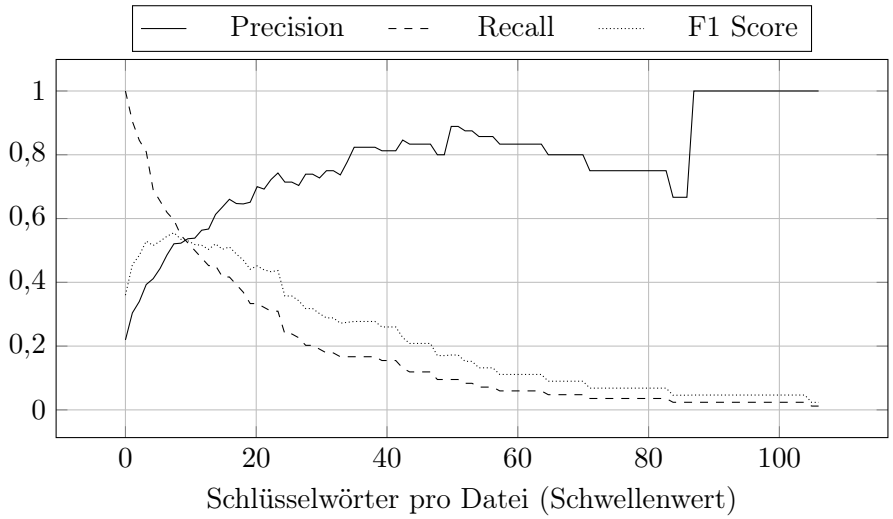


Abbildung 6.6: Dateiklassifikation mit dem Schlüsselwort-Klassifizierer

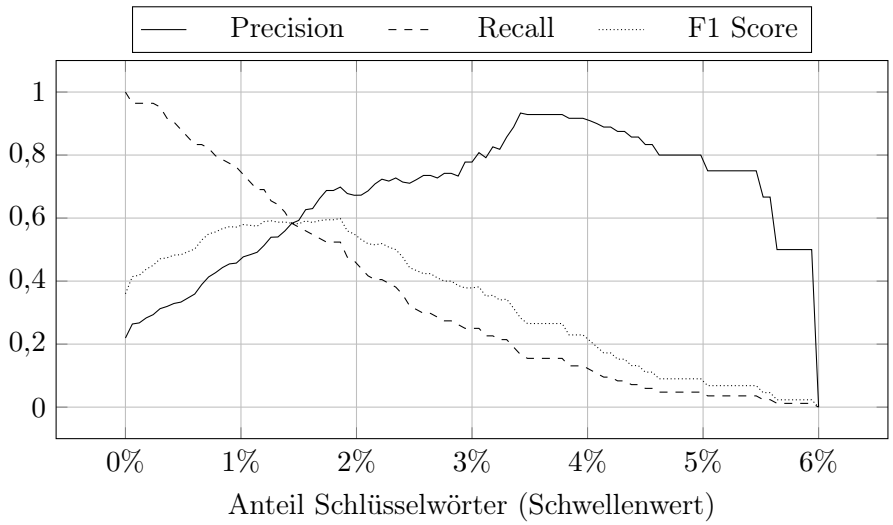


Abbildung 6.7: Dateiklassifikation mit dem relativen Schlüsselwort-Klassifizierer

Besonderes Zeilenende

Ein Großteil der in der Stichprobe vertretenen Programmiersprachen nutzt Semikolons, um ein Zeilenende zu markieren, was bei Text in natürlicher Sprache eher selten vorkommt. Dies lässt sich als einen Ansatz zur Klassifizierung nutzen.
Für diesen Klassifizierer wurde darauf verzichtet, absolute Vorkommen zu zählen, stattdessen wurde direkt ein relativer Ansatz genutzt. Semikolons an Zeilenenden werden gezählt und durch die Anzahl Zeilen geteilt.

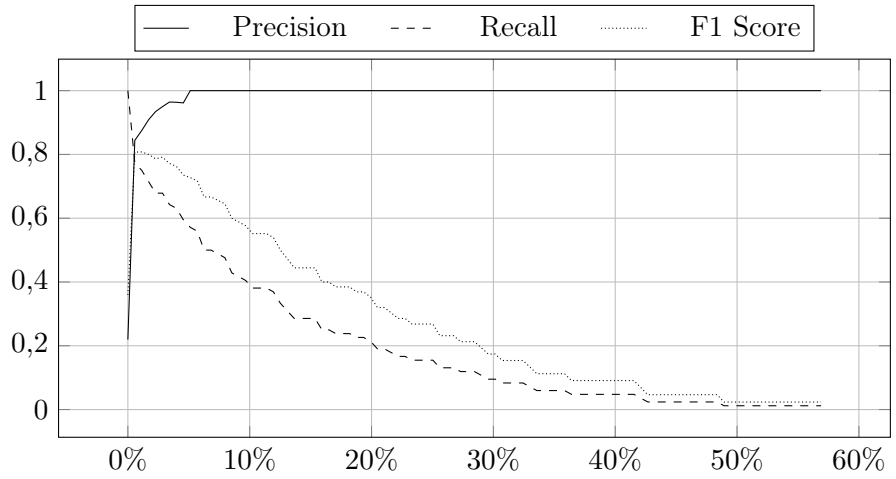


Abbildung 6.8: Dateiklassifikation mit dem Semikolon-Klassifizierer

Bereits bei einem Schwellenwert von ungefähr 0,5% erreicht dieser Klassifizierer seinen maximalen *F_1-Score* von 0,81 bei einer Precision von 0,84 und einem Recall von 0,77. Ist maximale Precision von 1,0 gewünscht, erreicht er diese bei einem Schwellenwert von ungefähr 5% mit einem Recall von 0,57. Bei Precision von 0,95 hat er einen Recall von 0,68.
Dieser Klassifizierer übertrifft den Schlüsselwort-Klassifizierer, er hat einen höheren maximalen *F_1-Score* (0,81 vs. 0,59) und bei einer höheren maximalen Precision (1,0 vs. 0,93) einen deutlich höheren Recall (0,77 vs. 0,17).

Kombiniert

Abschließend wird untersucht, ob eine gewichtete Kombination der vorhergegangenen Klassifizierer eine Verbesserung nach sich zieht. Um ein Gewicht zu entscheiden, werden alle Gewichte getestet und auf den maximalen *F₁-Score* untersucht (Abb. 6.9).

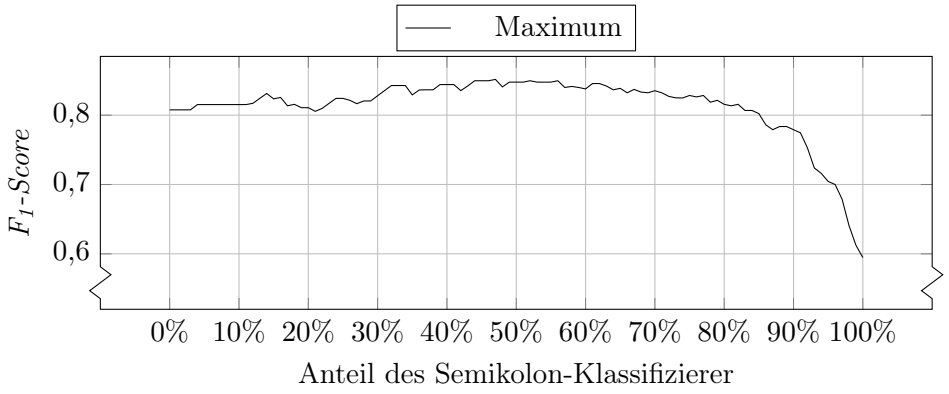


Abbildung 6.9: Maximale *F₁-Score*s für den Kombinierten Klassifizierer bei verschiedenen Gewichten

Zu erkennen ist, dass der kombinierte Klassifizierer einen höheren maximalen *F₁-Score* hat als der Semikolon-Klassifizierer. Das Maximum von 0,85 befindet sich bei einem Gewicht von 47% für den Semikolon-Klassifizierer und somit 53% für den Schlüsselwort-Klassifizierer.

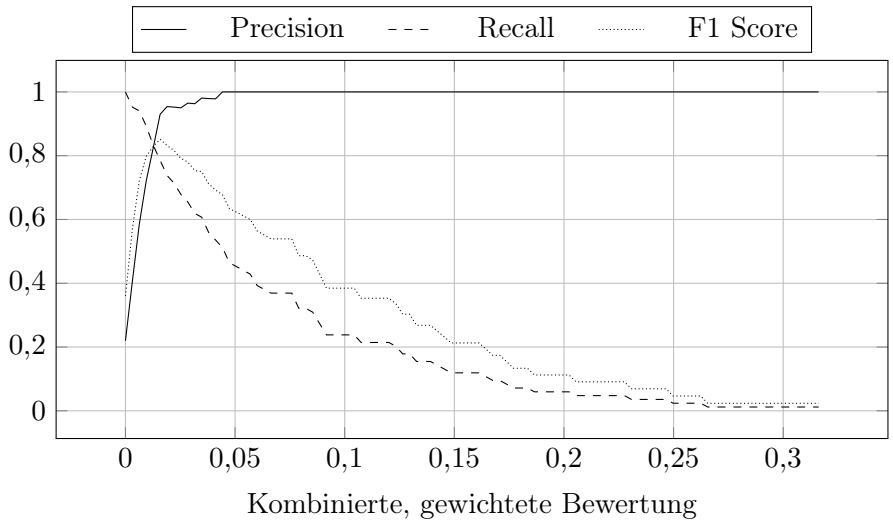


Abbildung 6.10: Dateiklassifikation mit dem kombinierten Klassifizierer und einer 47-53-Gewichtung

Der kombinierte Klassifizierer mit der bestimmten Gewichtung (Abb. 6.2.3) leistet teilweise mehr als der Semikolon-Klassifizierer. Neben dem höheren maximalen F_1-*Score* hat erreicht er eine Precision von 0,95 bei einem höheren Recall von 0,73 verglichen mit 0,68. Den maximalen Precision-Wert von 1,0 erreicht dieser Klassifizierer allerdings erst mit einem Recall von nur noch 0,51.

6.2.4 Fazit

Abhängig von der benötigten Metrik sollte entweder den Semikolon- oder den kombinierten Klassifizierer gewählt werden. Ist ① perfekte Precision gewünscht, sollte man den Semikolon-Klassifizierer nutzen, für „nur“ ② hohe Precision bzw. einen ③ hohen *F_1-Score* den kombinierten Klassifizierer. Ist ein noch höherer Recall gewünscht, geht dies auf Kosten der Precision. Um beispielsweise von ③ einen um 0,1 erhöhten Recall zu erreichen, werden 0,2 Precision geopfert.

Wunsch	Klassifizierer	Schwellenwert	Precision	Recall	F_1-Score
① Precision = 1,0	Semikolon	5,1%	1,0	0,57	0,73
② Precision > 0,95	Kombiniert	0,019	0,95	0,73	0,83
③ Hoher *F_1-Score*	Kombiniert	0,016	0,93	0,79	0,85

Abbildung 6.11: Empfohlene Klassifizierer

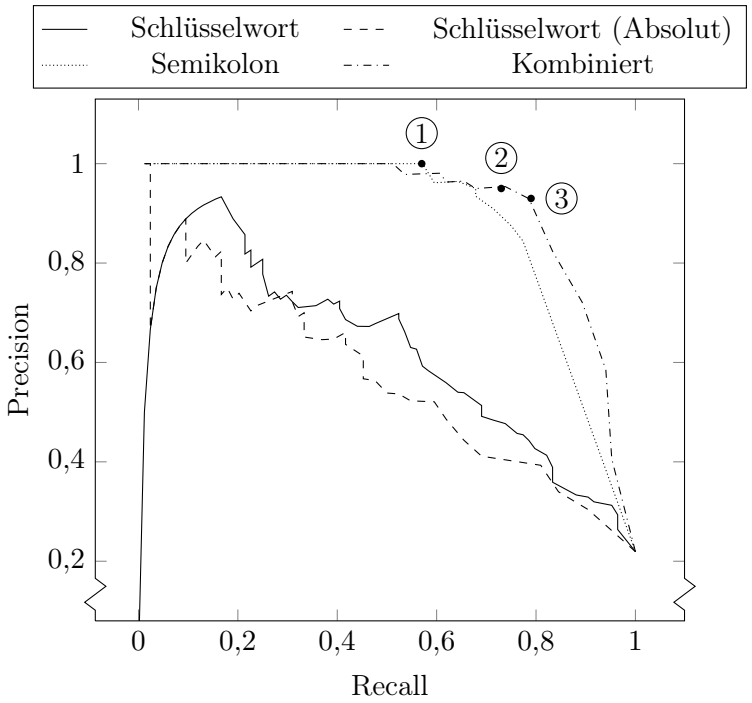


Abbildung 6.12: Performanz aller Klassifizierer im Vergleich

Kapitel 7

Verwandte Arbeiten

Müller [15] entwickelt gegenwärtig einen ähnliches Tool wie das in dieser Arbeit vorgestellte. Er nutzt jedoch `github.com` als Quellcode-Quelle.
Bajracharya et al. [8] präsentieren *Sourcerer*, eine Infrastruktur für umfassende Quellcodesammlung und -analyse. Sie beschreiben, wie Sourcerer quelloffenene Repositorys parallel crawlt sowie dass für verschiedene Repositorys verschiedene Crawling-Plugins bereitgestellt werden. Anders als diese Arbeit widmen sie sich vollständig expliziten Quellcode-Repositorys und nicht heterogenen Quellen wie PSS.
Chatterjee et al. [11] zeigen einen Ansatz zur Extrahierung von Quellcode und assoziiertem beschreibenden Text aus wissenschaftlichen Arbeiten. Sie beschreiben das Extrahieren von Quellcode allerdings als unkompliziert, vermutlich aufgrund der in solchen Arbeiten vorhandenen Quellcodeformatierung, und fokussieren ihre Bemühungen auf den beschreibenden Text.
Bacchelli et al. [7] zeigen, wie sie Quellcode in Emails erkennen. Als Benchmark nutzen sie eine Untermenge von Mails aus verschiedenen Mailing-Listen von quelloffenen Java-Projekten. Sie stellen verschiedene Klassifizierer vor: Zuerst ein Verfahren, dass dem in 6.2.3 vorgestellten Verfahren zur Klassifizerung nach Schlüsselwörtern ähnelt, es jedoch auf Java-Schlüsselwörter beschränkt und um Sonderzeichen erweitert. Im zweiten Verfahren matchen sie Zeilen gegen einen regulären Ausdruck, der Funktionsaufrufe oder Teile eines solchen erkennt und überprüfen Zeilenenden auf Semikolons, was dem in 6.2.3 vorgestellten Semikolon-Klassifizierer ähnelt. Abschließend suchen sie nach Zeilen, die den Anfang eines Blocks darstellen.
Zhang et al. [16] sowie Liu et al. [14] nutzen *auf Buchstabenbasis arbeitende rekurrente neuronales Netzwerke*, um Texte zu klassifizieren. Dies ist ein von der in dieser Arbeit gezeigten Klassifizierer abweichender Ansatz, in dem nicht händisch Metriken zur Klassifizierung erdacht werden. Stattdessen wird ein neuronales Netzwerk mit einem Trainingsdatensatz gefüttert und gegen ein Testdatensatz erprobt.

Kapitel 8

Zusammenfassung und Ausblick

In diesem Kapitel wird zuerst in Abschnitt 8.1 die Arbeit zusammengefasst. Abschließend wird in Abschnitt 8.2 ein Ausblick gegeben, welche Themen dieser Arbeit weiter untersucht werden könnten.

8.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Crawler für die Website PSS entwickelt, welcher alle oder eine begrenzte Auswahl an Dateien herunterladen und auswerten kann. Er kann verschiedene Datenbanken verwalten, Dateien und Metadaten betrachten und ist um eigene Analyseklassen erweiterbar.
Mit dem Crawler wurden Datensätze erstellt, die alle Dateien von PSS enthalten. Es wurde analysiert, wie viele Dateien welcher Dateitypen diese Datensätze enthalten.
Es wurden verschiedene Klassifizierer entworfen, die Textdateien mit Quellcode erkennen. Für sie wurde ein Benchmark aus einer zufälligen Auswahl von Textdateien aus einem der vorher erstellten Datensätze konstruiert. Anschließend wurden die Klassifizierer mit verschiedenen Parametern auf Precision, Recall und F_1-Score untersucht und die Ergebnisse verglichen.
Zusammenfassend lässt sich sagen, dass die Zielsetzung bezüglich des Crawlers erreicht wurde. Leider enthält PSS allerdings nicht viele Java-Quellcodedateien mit Schwachstellen, die für das Plugin von Brunotte [10] benötigt werden.

8.2 Ausblick

Die in Kapitel 7 angesprochenen Klassifizierer zur Erkennung von Quellcode in Emails von Bacchelli et al. [7] ließen sich möglicherweise ganz oder in abgewandelter Form für die Klassifizierung von Textdateien bzw. Gewinnung

von Quellcodeausschnitten nutzen.
Auch sind andere Verfahren zur Ausschnittgewinnung denkbar, beispielsweise die auf Buchstabenbasis arbeitenden rekurrenten neuronalen Netzwerke von Zhang et al. [16] sowie Liu et al. [14] , die in Kapitel 7 angesprochen wurden. Sie könnten Textdateien beispielsweise Zeile für Zeile klassifizieren und so Quellcodeausschnitte gewinnen.

Anhang A

Anhang

A.1 Zeitmessung von parallelem vs. sequentiellem Download

Parallele und sequentielle Downloadzeiten in Sekunden von jeweils 100 Einträgen. Für die sequentielle Messung wurden 10 Threads genutzt.

Parallel	Sequentiell
1.743411	14.832703
1.7477	14.985585
1.784515	15.001869
1.806026	15.089445
1.912039	15.126877
1.921442	15.771359
1.956813	15.980033
2.044751	16.282726
2.082177	16.553442
3.108771	18.782334

A.2 Datensätze

A.2.1 Häufigste Dateiendungen für Datensatz ohne Dekompression

Die 35 häufigste Dateiendungen aus dem Datensatz ohne Dekompression.

Dateiendung	Anzahl
txt	70674
gz	7649
zip	5708
tgz	3129
c	2820
pdf	2013
html	835
asc	551
bz2	507
pl	490
exe	308
htm	292
gif	189
sh	161
ps	148
rar	134
jpg	132
box	126
arj	85
tar	80
iso	65
Z	64
patch	62
cpp	61
ZIP	57
xz	57
vulnerability	56
ppt	52
vul	52
doc	47
7z	44
diff	40
png	34
TXT	33
README	31

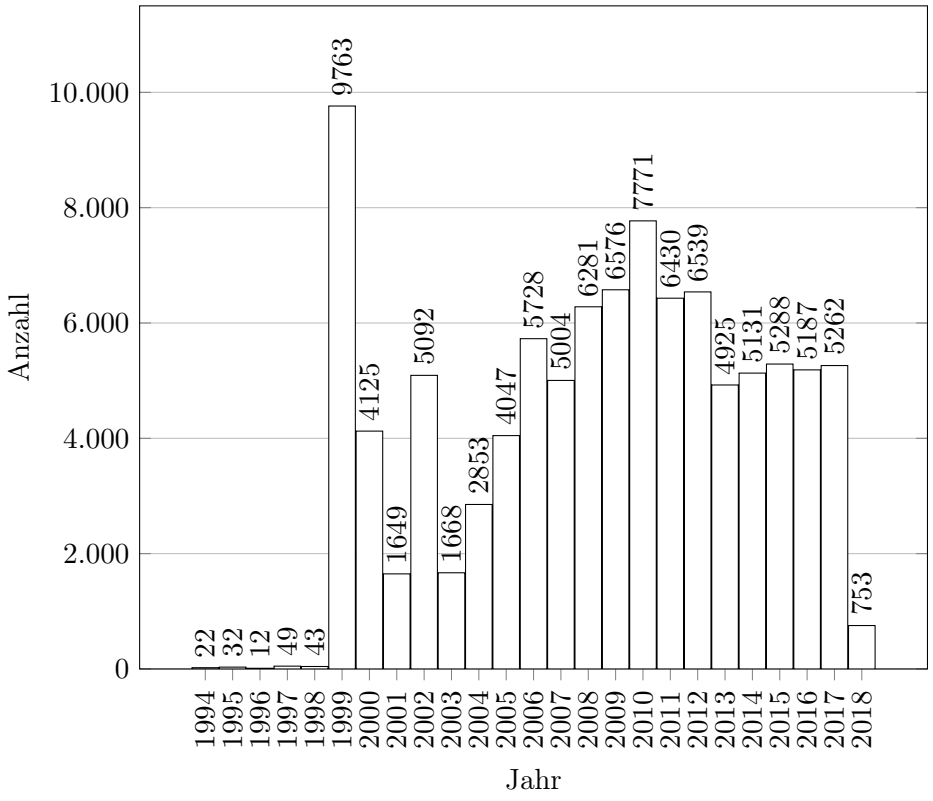
A.2.2 Häufigste Dateiendungen für Datensatz mit Dekompression

Die 35 häufigste Dateiendungen aus dem Datensatz mit Dekompression.

Dateiendung	Anzahl
c	494512
h	342485
txt	236537
svn-base	143969
html	124033
svn-work	84112
png	64808
in	62385
rb	62280
conf	61753
3	55115
cpp	53905
pm	53036
short	41149
py	40639
java	39468
xml	38917
dat	35738
m4	33626
am	32801
pl	31953
sh	29378
php	25874
gif	23523
pem	23248
Makefile	22909
README	20278
pod	19727
js	18729
t	14743
entries	12916
crt	11920
nse	11753
po	11349
ll	11115

A.2.3 Veröffentlichte Dateien nach Jahr

Stand: 13.03.2018



A.2.4 Häufigste veröffentlichte Autoren

Die 40 am häufigsten veröffentlichte Autoren auf PSS.

Name	Typ	Anzahl
Red Hat	person	4002
Ubuntu	company	3980
Debian	person	3352
Mandriva	company	2707
Gentoo	company	2340
Tipping Point	company	1126
HP	person	1120
Google Security Research	person	994
iDefense Labs	company	741
Cisco Systems	company	643
Luigi Auriemma	person	613
High-Tech Bridge SA	person	578
indoushka	person	553
LiquidWorm	person	540
Ihsan Sencan	person	526
Benjamin Kunz Mejri	person	479
Slackware Security Team	company	438
Hewlett Packard	company	431
Apple	company	343
rgod	person	304
juan vazquez	person	293
MustLive	person	284
AutoSec Tools	person	272
US-CERT	company	268
KedAns-Dz	person	254
Todd J.	person	234
H D Moore	person	222
hyp3rlinx	person	221
MC	person	218
mjurczyk	person	203
The FreeBSD Project	group	190
Core Security Technologies	person	189
sinn3r	person	171
thc	group	170
Moudi	person	160
Aliaksandr Hartsuyeu	person	159
S@BUN	person	154
Ashiyane Digital Security Team	person	153
VMware	person	148
mr_me	person	140

A.3 Schlüsselwörter

A.3.1 Häufigste Schlüsselwörter nach Programmiersprachen

Die folgende Tabelle zeigt die 10 häufigsten Schlüsselwörter der 6 häufigsten Programmiersprachen aus der Stichprobe [13].

	PHP	const	Perl	Python	Ruby	Javascript
1.	this	if	my	self	end	this
2.	return	define	the	if	do	function
3.	if	the	self	def	def	if
4.	array	return	if	return	the	var
5.	the	int	return	import	to	return
6.	function	const	use	the	if	the
7.	php	void	to	in	should	i
8.	class	include	a	None	it	a
9.	div	to	_	for	require	to
10.	public	h	sub	from	a	value

A.3.2 Schlüsselwörter-Menge

Die folgende Tabelle zeigt die Schlüsselwörter aus A.3.1 ohne Duplikate.

Schlüsselwort
None
_
a
array
class
const
def
define
div
do
end
for
from
function
h
i
if
import
in
include
int
it
my
php
public
require
return
self
should
sub
the
this
to
use
value
var
void

A.3.3 Filtrierte Schlüsselwörter-Menge

Die folgende Tabelle zeigt die verbleibenden Schlüsselwörter, nachdem sie mit dem Verfahren in Abschnitt 6.2.3 beschriebenen Verfahren aussortiert wurden.

Schlüsselwort
None
_
array
class
const
def
define
div
do
end
from
function
h
i
if
import
include
int
it
keyword
my
php
require
return
self
sub
value
var
void

Literaturverzeichnis

[1] About Packet Storm. https://packetstormsecurity.com/about/. accessed 2018-03-27.

[2] CVE - Frequently Asked Questions. http://cve.mitre.org/about/faqs.html. accessed 2018-03-27.

[3] NVD - FAQ. https://nvd.nist.gov/general/faq. accessed 2018-03-27.

[4] Submissions Packet Storm. https://packetstormsecurity.com/submit/. accessed 2018-03-27.

[5] The MVVM Pattern. https://msdn.microsoft.com/en-us/library/hh848246.aspx. accessed 2018-03-27.

[6] UCI Source Code Data Sets. https://www.ics.uci.edu/~lopes/datasets/. accessed 2018-03-27.

[7] A. Bacchelli, M. D'Ambros, and M. Lanza. Extracting Source Code from E-Mails. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 24–33, June 2010.

[8] S. Bajracharya, J. Ossher, and C. Lopes. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming*, 79:241 – 259, 2014. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[9] M. L. Berenson, D. M. Levine, and T. C. Krehbiel. *Basic Business Statistics: Concepts and Applications*, Section 8.7 auf CD-ROM. Prentice Hall, 10th edition, April 2005.

[10] W. Brunotte. Security Code Clone Detection entwickelt als Eclipse Plugin. Unpublished master thesis at the Software Engineering Group at Leibniz Universität Hannover, 2018.

[11] P. Chatterjee, B. Gause, H. Hedinger, and L. Pollock. Extracting Code Segments and Their Descriptions from Research Articles. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 91–101, May 2017.

[12] J. Cho. *Crawling the Web: Discovery and Maintenance of Large-scale Web Data*. PhD thesis, Stanford University, November 2001.

[13] A. Kashcha. common-words. `https://github.com/anvaka/common-words`, 2016. Commit 6cd73a5d414e659900ee59792219df187b66b749, accessed 2018-03-27,.

[14] J. Liu, F. Meng, Y. Zhou, and B. Liu. Character-Level neural networks for short text classification. In *2017 International Smart Cities Conference (ISC2)*, pages 1–7, September 2017.

[15] C. Müller. Security Code Exporter für Github. Unpublished bachelor thesis at the Software Engineering Group at Leibniz Universität Hannover, 2018.

[16] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level Convolutional Networks for Text Classification. *CoRR*, abs/1509.01626, 2015.

Glossar

$\mathbf{F_1}$ *Siehe: F_1-Score.*

$\mathbf{F_1}$-Score ($\mathbf{F_1}$-Maß, $\mathbf{F_1}$) Harmonisches Mittel aus Recall und Precision. $F_1 = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}$ 35–37, 39–42, 57

API Application programming interface Schnittstelle eines Software-Systems v, vii, 1, 57, 58

binärer Klassifizierer Ein Klassifizierer, der zwischen genau Zwei Klassen unterscheidet. 33, 35

Crawler *Siehe: Abschnitt 3.1.* v, vii, 1–5, 7, 11–15, 19, 21, 22, 29, 32, 45

CVE Common Vulnerabilities and Exposures *Siehe: Abschnitt 3.5.* v, vii, 7, 10

DAO Data Access Object Entwurfsmuster zum abkapseln von Datenquellen. 18, 19

False Negative (falsch Negativ, FN) Anzahl der fälschlicherweise als `falsch` klassifizierten Elemente 35, 57

False Positive (falsch Positiv, FP) Anzahl der fälschlicherweise als `wahr` klassifizierten Elemente 35, 57

FN *Siehe: False Negative.*

FP *Siehe: False Positive.*

Hibernate *Object-Relational Mapping* (ORM) Framework für Java, implementiert *Java Persistence API* (JPA) 19

JDBC Java Database Connectivity Java-Datenbank-Schnittstelle 19

Join-Tabelle Tabelle, die Indizes aus zwei oder mehr anderen Tabellen enthält, über die die Tabellen gejoint werden. 13

JPA Java Persistence API Java-Datenbank-Schnittstelle 19, 57

Klassifizierer *Siehe: Abschnitt 6.2.* v, vii, 2, 33, 35–43, 45, 57

LGPL GNU Lesser General Public License Lizenz für Quelloffene Software 19

Many-To-Many-Beziehung (n, m)-Beziehung zwischen zwei Datenbank-Entitäten. Beispielsweise kann ein Autor mehrere Einträge verfassen und ein Eintrag von mehreren Autoren verfasst sein. 13

Model *Siehe: Abschnitt 5.1.* 17

MVC Model-View-Controller Entwurfsmuster zum erstellen von Software, trennt Applikationen in die namensgebenden Teile Model, View und Controller 17

MVVM Model-View-ViewModel Entwurfsmuster zum erstellen von Software, trennt Applikationen in die namensgebenden Teile Model, View und ViewModel 17

Notepad++ Beliebter Texteditor für Microsoft Windows 25

NVD National Vulnerability Database *Siehe: Abschnitt 3.6.* 7, 10

ORM Object-Relational Mapping Abbildung von Objekten einer Programmiersprache zu Tabellen einer Datenbank 57

Precision (Genauigkeit) Anteil der korrekt als `wahr` klassifizierten Elemente im Verhältnis zu der Gesamtheit der als `wahr` klassifizierten Elemente. $Precision = \frac{TP}{TP+FP}$ 35–39, 41, 42, 57

PSS `packetstormsecurity.com` v, vii, 1, 2, 4, 7–9, 12, 14, 25, 31, 34, 43, 45, 51

Recall (Empfindlichkeit) Anteil der korrekt als `wahr` klassifizierten Elemente im Verhältnis zu den tatsächlich `wahr`en Elementen. $Recall = \frac{TP}{TP+FN}$ 35–37, 39, 41, 42, 57

Red Hat Amerikanisches Software-Unternehmen, was für seine quelloffenen Softwareprojekte bekannt ist. 9, 19

RSS-Feed Rich Site Summary Feed Eine Erweiterung von XML. Ein standardisierter Weg, um Nachrichten und anderen sich aktualisierende Onlineinhalte abzurufen. v, vii, 11, 14, 15

TN *Siehe: True Negative.*

TP *Siehe: True Positive.*

True Negative (richtig Negativ, TN) Anzahl der korrekt als `falsch` klassifizierten Elemente 59

True Positive (richtig Positiv, TP) Anzahl der korrekt als `wahr` klassifizierten Elemente 35, 59

View *Siehe: Abschnitt 5.1.* 17

ViewModel *Siehe: Abschnitt 5.1.* 17