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

Security Code Exporter für Github

Security Code Exporter for Github

Bachelorarbeit

im Studiengang Informatik

von

Christian Müller

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Javad Ghofrani
Betreuer: Fabien Viertel

Hannover, 16. März 2018

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig 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 16.03.2018

_____

Christian Müller

Zusammenfassung

Ein wesentlicher Aspekt der Softwareentwicklung ist das Bestreben Sicherheitslücken zu vermeiden und die Vertraulichkeit und Integrität der Daten zu gewährleisten. Dazu ist es notwendig eventuelle Schwachstellen frühestmöglich, also während der Entwicklung der Software, zu erkennen und zu beheben. Ein möglicher Ansatz ist die Verwendung von Code Clone Detection Software. Diese Software gleicht den entwickelten Code mit bekannten Schwachstellen aus einer Datenbank ab, um so mögliche Schwachstellen zu erkennen und diese dem Softwareentwickler mitzuteilen. Im Rahmen dieser Bachelorarbeit wurde eine Software mit dem Namen Security Code Exporter entwickelt, die eine solche Datenbank mit schadhaftem Code erstellt.

Der Security Code Exporter sucht automatisiert auf Github, einem der größten Hosts für Softwareprojekte, nach security relevantem Code und lädt diesen herunter. Als security relevanter Code wird Quellcode verstanden, der entweder eine bestimmte Schwachstelle enthält, schließt oder ausnutzt. Um security relevanten Code zu finden, kann nach CVE-IDs (Common Vulnerabilities and Exposures) und nach benutzerdefinierten Suchbegriffen gesucht werden. Die Ergebnisse sind nicht auf einzelne Programmiersprachen oder Dateitypen beschränkt. Der Benutzer kann die Ergebnisse aber durch verschiedene Filter in Bezug auf Precision, Recall und den jeweiligen Anwendungsfall optimieren. Die gefundenen Ergebnisse, bestehend aus Metadaten und Dateien, werden in einer Datenbank gespeichert und können angezeigt und durchsucht werden. Da ein kompletter Suchdurchlauf mehrere Stunden in Anspruch nimmt, besteht die Möglichkeit die Suche jederzeit abzubrechen oder zu pausieren, falls die Suche später fortgesetzt werden soll.

Mit Hilfe des Security Code Exporters können benutzerdefinierte Schwachstellendatenbanken erstellt werden, durch die Code Clone Detection Software für beliebige Programmiersprachen dem Entwickler helfen kann, Sicherheitslücken frühzeitig zu erkennen und diese zu beheben.

Abstract

A crucial goal in software development is to avoid as vulnerabilities to ensure a high level of security for the used data in terms of privacy and integrity. Therefor it is necessary to detect possible vulnerabilities as soon as possible while developing the software. One way of doing this is to use Code Clone Detection Software. This kind of software compares the developed code with the code of known vulnerabilities stored in a database to detect possible vulnerabilities and warn the developer. This bachelor thesis is about developing software named Security Code Exporter which constructs a database containing examples of vulnerable source code.

The software searches Github, one of the biggest hosts for software projects, for security relevant code and downloads it. Security relevant code is defined as source code which is vulnerable, fixes or is used to exploit a certain vulnerability. To find security relevant code it is possible to use CVE-IDs (Common Vulnerabilities and Exposures) user defined words as keywords for searching Github. The search results are not limited to certain programming languages or data types but it is possible to customize them by applying various filter to maximize the performance of the search in respect of recall and precision so the resulting database meets the requirements of its use case in the best possible way. All found results, consisting of meta data and files, are added to a vulnerability database and can be browsed by the user later on. Due to a complete search with CVE-IDs used as key words taking several hours, this process can be paused or canceled at any time.

Using the Security Code Exporter it is possible to create vulnerability databases with customizable content. These databases can be used as a base for code clone detection software working on different programming languages to help detect vulnerabilities in newly developed software.

Inhaltsverzeichnis

1 Einleitung

Viele Tätigkeiten der Menschen werden inzwischen durch Software unterstützt. Besonders wenn Software mit sensiblen Daten arbeitet, ist es wichtig, ein hohes Maß an Sicherheit für diese Daten zu gewährleisten. Die Vermeidung von Schwachstellen in der Software ist also ein wesentlicher Aspekt der Softwareentwicklung. Die nachträgliche Suche und Beseitigung von Schwachstellen erfordert viel Aufwand und ist somit mit hohen Kosten verbunden. Daher ist eine frühe Identifikation von möglichen Schwachstellen während des Entwicklungsprozesses wünschenswert. Wagner beschreibt in seiner Masterarbeit [1] eine Anwendung, die in Softwareprojekten, die Java verwenden, genutzt werden kann, um die verwendeten Bibliotheken auf Schwachstellen zu prüfen. Zusätzlich kann es sinnvoll sein, den vom Entwickler generierten Quellcode auf Schwachstellen zu überprüfen. Code Clone Detection Software erfüllt diese Aufgabe. Sie gleicht den vom Softwareentwickler generierten Code mit bekannten Schwachstellen aus einer Datenbank ab und warnt den Entwickler so frühzeitig vor möglichen Sicherheitslücken.

1.1 Problemstellung

Die Qualität der Schwachstellendatenbank hat einen wesentlichen Anteil am Erfolg dieses Ansatzes, da lediglich darin enthaltene Schwachstellen erkannt werden können. Außerdem sollte die Datenbank Informationen enthalten, wie eine bestimmte Schwachstelle beseitigt werden kann. Diese kann in Form eines fertigen Fixes, einer verbesserten Version des Codes, der die Schwachstelle behebt, oder des Exploits, der die Methodik beschreibt, mit der die Schwachstelle ausgenutzt werden kann, vorliegen. Ist kein Fix, aber ein Exploit vorhanden, kann der Entwickler anhand des Exploits gegebenenfalls einen eigenen Fix entwickeln oder den bereits vorhandenen Fix verbessern.

Es wird also eine Datenbank benötigt, die möglichst viele Schwachstellen und den zugehörigen Fix und Exploit enthält. Außerdem sollte die Datenbank möglichst gut auf den jeweiligen Anwendungsfall anpassbar sein. So kann es sinnvoll sein, nur bestimmte Programmiersprachen und Dateitypen in die Datenbank aufzunehmen. Die hinterlegten Dateien sollten normalisiert und die Datenbank von Duplikaten bereinigt werden, um dem Code Clone Detector eine gute Arbeitsgrundlage zu schaffen.

1.2 Lösungsansatz

Schwachstellen in Software werden im Allgemeinen geprüft, registriert und mit einem spezifischen Identifier, der CVE-ID versehen. Darunter werden Informationen, wie zum Beispiel Status, Aufnahme- und Bearbeitungsdatum und gegebenenfalls Verweise auf Webseiten mit näheren Informationen zu der Schwachstelle, abgelegt. Datenbanken, die CVE-IDs und die hinterlegten Metadaten beinhalten, sind zum Beispiel die „National Vulnerability Database“ (NVD[1]) und die „Common Vulnerabilities and Exposures“ (CVE[2]) Datenbank. Die CVE-IDs können als Suchbegriff für die Suche nach security relevantem Code verwendet werden. Als Datenquelle bietet sich Github[3] aufgrund seiner Größe und Popularität an. Es können Projekte, die in Zusammenhang mit einer, durch eine CVE-ID identifizierten, Schwachstelle stehen, gefunden, heruntergeladen und für eine Schwachstellendatenbank verwendet werden. Der Inhalt der Datenbank kann entsprechend den gestellten Anforderungen optimiert werden. So können auf Wunsch nur Projekte, die eine angegebene Programmiersprache verwenden, berücksichtigt und die heruntergeladenen Dateien auf angegebene Dateitypen beschränkt werden. Darüber hinaus soll die Möglichkeit bestehen die Datenbank händisch zu durchsuchen und einzelne Einträge anzuzeigen. Die Anwendung wird in Java entwickelt und als „Security Code Exporter“ bezeichnet.

1 https://nvd.nist.gov
2 https://cve.mitre.org/index.html
3 https://github.com

2 Grundlagen

In diesem Kapitel soll es um wichtige Grundlagen in Bezug auf die Umsetzung des Security Code Exporters gehen. Zunächst wird auf Schwachstellen und CVE-IDs als solche eingegangen. Anschließend werden die nötigen Grundlagen zu Git erläutert und auf Github eingegangen. Abschließend werden die benötigten Teile der von Github bereitgestellten Programmierschnittstelle (API[4]) aufgeführt und erklärt.

2.1 Schwachstellen

Das Bundesamt für Sicherheit in der Informationstechnik definiert eine Schwachstelle wie folgt:

„Eine Schwachstelle ist ein sicherheitsrelevanter Fehler eines IT-Systems ... . Ursachen können in der Konzeption, den verwendeten Algorithmen, der Implementierung, der Konfiguration, dem Betrieb ... liegen. Eine Schwachstelle kann dazu führen, dass eine Bedrohung wirksam wird und ... ein System geschädigt wird. Durch eine Schwachstelle wird ... ein System ... anfällig für Bedrohungen.“ [2]

Diese Definition schließt deutlich mehr Aspekte ein, als im Rahmen dieser Arbeit betrachtet werden. Ein Code Clone Detector kann nur solche Schwachstellen finden, die durch die verwendeten Algorithmen oder die Implementierung in die Software eingebracht wurden. Die für die Generierung der Schwachstellendatenbank genutzten Schwachstellen stammen aus den Datenbanken CVE und NVD und werden über die CVE-ID identifiziert. Dieser Identifikator besteht aus dem Präfix „CVE“, dem Veröffentlichungsjahr der Schwachstelle und einer für das Jahr spezifischen Nummer beliebiger Länge: *CVE-<YYYY>-<NNNN>*. Die CVE-Datenbank enthält zu jeder Schwachstelle eine kurze Beschreibung und, falls vorhanden weitere Referenzen [3]. Die NVD enthält darüber hinaus weitere Informationen wie den CVSS (Common Vulnerability Scoring System) Score, der ein Maß für die Schwere der Schwachstelle darstellt [4].

4 Application Programming Interface

2.2 Git

Git ist ein verteiltes Versionskontrollsystem (DVCS[5]), das ursprünglich zur Verwaltung des Linux Kernel Software Projekts entwickelt wurde. Zentrale Designziele waren Git vollständig verteilt und möglichst performant zu machen, um große Softwareprojekte leichter verwalten zu können. Diese Eigenschaften machen Git zu einem der beliebtesten Versionskontrollsysteme im Bereich der Softwareentwicklung.

Anders als von lokalen oder zentralisierten Versionskontrollsystemen, wird von DVCS die gesamte Historie eines Projekts lokal und zentralisiert gespeichert. Das hat den Vorteil, dass mehrere Entwickler gleichzeitig von verschiedenen Orten aus an einem Projekt arbeiten können, aber nicht auf eine Verbindung zum zentralen Speicherort des Projekts angewiesen sind. Außerdem sorgt die verteilte Struktur für eine höhere Ausfallsicherheit, da, wenn die Daten in der zentralisierten Version beschädigt wurden, es gleichwertige lokale Kopien gibt, durch die die zentralisierte Version wieder hergestellt werden kann.

Für den Security Code Exporter sind im Wesentlichen zwei Strukturen von Git wichtig. Das durch Git verwaltete Dateisystem, zumeist Softwareprojekte, werden in Repositories gespeichert. Ein Repository enthält eine Datenbank mit einer Liste aller Versionen (Commits) des Projekts sowie den komprimierten Dateien der jeweiligen Version. Ein Commit besteht aus den einzelnen Dateien, die sich in Bezug auf die Vorgängerversion geändert haben, und Verweisen auf Dateien in der Datenbank des Repositories, die sich im Zuge des Commits nicht geändert haben. Commits und Dateien werden intern durch Hashwerte referenziert, die aus den Dateien erzeugt werden. Dadurch kann Git die Integrität der Versionen sicherstellen und zum Beispiel Übertragungsfehler erkennen [5].

5 Distributed Version Control System

2.3 Github

Github ist ein Online Dienst, auf dem Softwareentwickler ein zentrales Repository für ihr mit Git verwaltetes Projekt anlegen und anderen Benutzern zugänglich machen können. Github ist mit 24 Millionen Nutzern und 67 Millionen Repositories einer der größten Hosts für DVCS [6]. Diese Eigenschaft macht Github zu einer vielversprechenden Quelle für die Suche nach Schwachstellen.

2.4 Github API

Mit der REST API v3 bietet Github eine einfache Schnittstelle für die Nutzung von Github in Anwendungen und Projekten. Die API bietet unter anderem Zugriff auf viele Git Funktionalitäten, Repositories, Commits und Benutzer von Github. Sie bietet darüber hinaus eine Search API an, durch welche gezielt nach Repositoies, Commits, Code und anderen Informationen gesucht werden kann. Der Zugriff auf die Github API erfolgt über HTTPS (Hypertext Transfer Protocol Secure). Dabei wurde eine Zugriffsbeschränkung (Rate limiting) implementiert, die je nach Nutzungstyp eine begrenzte Anzahl von Anfragen pro Stunde an die Github API zulässt. Diese Anzahl unterscheidet sich darüber hinaus für Zugriffe, die durch ein Githubbenutzerkonto authentifiziert wurden und solche, die nicht authentifiziert wurden. Die API lässt im Allgemeinen 5000 authentifizierte und 60 nicht authentifiziere Zugriffe pro Stunde zu. Authentifizierte Zugriffe sind auf das jeweilige Githubbenutzerkonto bezogen. Das bedeutet, dass sich zwei Anwendungen, die, durch das gleiche Benutzerkonto authentifiziert, auf die Github API zugreifen, das Rate Limit von 5000 Zugriffen pro Stunde teilen müssen. Das Rate Limit von nicht authentifizierten Zugriffen bezieht sich auf die IP-Adresse, von der die Anfrage ausgeht. Informationen bezüglich des aktuellen Rate Limits, der verbrauchten Zugriffe und dem Zeitpunkt, an dem die verbrauchten Zugriffe zurückgesetzt werden, werden dem HTTPS Header der Antwort der API beigefügt [7].

Die Search API hat eigene Rate Limits und einige weitere Beschränkungen. Es werden 60 authentifizierte und 10 nicht authentifizierte Zugriffe pro Minute erlaubt. Diese Zugriffe zählen mit in das übergeordnete Rate Limit von 5000 oder 60 Zugriffen pro Stunde. Die Search API erzeugt eine Ergebnismenge bestehend aus bis zu 1000 Ergebnissen pro Suchanfrage, liefert aber nur maximal 100 Ergebnisse pro Zugriff. Die Ergebnisse sind in Abschnitte entsprechend der eingestellten Anzahl der Ergebnisse pro Anfrage

aufgeteilt. Auf diese Abschnitte kann separat zugegriffen werden. Bei sequenziellen Zugriffen auf eine bestimmte Ergebnismenge zählt nur der erste Zugriff, durch den die Ergebnismenge erzeugt wird, für das Rate Limit der Search API. Alle weiteren Zugriffe zählen ausschließlich für das übergeordnete Rate Limit. Außerdem ist jede Suchanfrage in ihrer Komplexität beschränkt. Es können nicht beliebig viele Suchbegriffe durch „und“ beziehungsweise „oder“ verknüpft werden [8].

3 Konzept des Security Code Exporters

Dieses Kapitel beschreibt das grundlegende Konzept, an dem sich der Aufbau des Security Code Exporters orientiert. Um die Aufgabe des Security Code Exporters zu lösen, wurde sie in folgende Teilaufgaben aufgeteilt: Suche, Klassifizierung, Download, Filterung und Speicherung der Ergebnisse. Abbildung 1 zeigt den Ablauf des Prozesses von der Suche nach Ergebnissen bis zur Speicherung in einer Datenbank. Die gezeigten Teilaufgaben sollen im Folgenden näher erläutert werden.

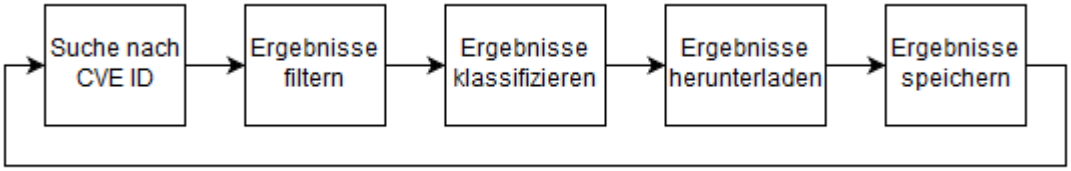

Abbildung 1: Ablaufkonzept

3.1 Suche

In dieser Teilaufgabe soll in Projekten auf Github nach Dateien, die möglicherweise security relevanten Code enthalten, gesucht und zugehörige Metadaten heruntergeladen werden. Dazu wird Github nach CVE-IDs und vom Benutzer definierten Schlüsselwörtern durchsucht. Die Suche erfolgt mittels der von Github bereitgestellten Search API. Diese bietet unter anderem die Möglichkeit, Repositories und Commits zu durchsuchen. Die Suche nach einer spezifischen CVE-ID oder einem anderen Schlüsselwort erfolgt zweistufig. Da jeglicher Code auf Github in Repositories organisiert ist, werden zunächst die Repositories durchsucht. Dabei wird mit der Search API im Titel, in der Beschreibung und in der README-Datei jedes Repositories nach dem Schlüsselwort gesucht. Wenn das Schlüsselwort gefunden wurde, werden die Metadaten des Repositories der Ergebnismenge beigefügt. Die Ergebnismenge wird nach Beendigung der Suche von der Search API zurückgegeben. Es gibt auf Github einige Repositories, die Informationen enthalten, wie explizite Schwachstelle ausgenutzt (PoC[6]) oder behoben werden können. Die erste Stufe der Suche bietet sich an, um solche Repositories zu finden, da oft mit Angabe der CVE-IDs auf die behandelten Schwachstellen verwiesen wird.

In der zweiten Stufe der Suche werden Commits betrachtet. Dabei wird der Kommentar

6 Proof of Concept

eines Commits nach dem jeweiligen Schlüsselwort durchsucht. Auch hier werden die entsprechenden Metadaten einer Ergebnismenge beigefügt, falls das Schlüsselwort gefunden wurde. Diese Art der Suche bietet sich an, um Schwachstellen und deren Fix zu finden. Dies basiert auf der Vermutung, dass, wenn im Kommentar angegeben wurde, dass eine Schwachstelle mit Verweis auf eine spezifische CVE-ID gefixt wurde, die Version des Quellcodes vor dem Commit die Schwachstelle und die Version nach dem Commit ihren Fix enthalten muss.

Die beiden erhaltenen Ergebnismengen werden zusammengefasst und als Ergebnismenge einer Suche betrachtet. Sind beide Stufen der Suche für ein Schlüsselwort erfolgt, werden die einzelnen Ergebnisse der Ergebnismenge klassifiziert.

3.2 Klassifizierung

In dieser Teilaufgabe soll anhand der Metadaten der gefundenen Repositories und Commits festgestellt werden, ob es sich bei einem bestimmten Repository oder Commit um Exploit Code, Vulnerable Code oder Unclassified Code handelt. Schadhafter Code und dessen verbesserte Version (Fix) wird unter Vulnerable Code zusammengefasst. Das liegt daran, dass die Schwachstelle und ihr Fix oft zusammen in einem Commit vorkommen. Es wurde jeweils eine Liste mit Zeichenketten, die auf Exploit Code beziehungsweise Vulnerable Code hinweisen, erstellt. Diese Zeichenketten sind Wörter oder Teilsätze, die auf eine bestimmte Klasse security relevanten Code hinweisen. Die Zeichenketten wurden aus Beispielen händisch ermittelt. Die Metadaten werden nach den Einträgen der Listen durchsucht. Dabei wird gezählt, wie oft Einträge aus den jeweiligen Listen gefunden wurden. Das ergibt zwei Zähler, aus deren Stand abgeleitet wird, um welche Klasse es sich handelt. Ist der Zähler für die Zeichenketten, die auf Exploit Code hinweisen höher als der für die Zeichenketten, die auf Vulnerable Code hinweisen, wird angenommen, dass es sich um Exploit Code handelt und andersherum. Sind beide Zähler gleich, lässt sich keine Aussage treffen und das Repository oder der Commit wird als Unclassified Code klassifiziert. Diese Vorgehensweise entspricht der einer Wortfrequenz Analyse. Die beiden Listen enthalten Standardeinträge, die vom Benutzer angepasst und erweitert werden können. Durch diese Anpassungsmöglichkeit kann der Benutzer die Klassifizierung der Ergebnisse aus Suchen nach CVE-IDs und besonders aus Suchen nach benutzerdefinierten Begriffen verbessern und so die Menge des scurity relevanten Codes steigern. Wurden alle Ergebnisse einer Suche klassifiziert, werden sie

heruntergeladen.

3.3 Download

In diesem Schritt werden die gefundenen Ergebnisse einer Ergebnismenge heruntergeladen. Dazu wird eine Liste mit Dateien erstellt, die zu dem jeweiligen Ergebnis gehören. Dieser Vorgang unterscheidet sich für Repositories und Commits. Bei Repositories wird via Git ein Klon des Repositories erstellt. Die geklonte Version ist „bare“, besitzt also nur die Datenbank mit der Historie des Repositories, aber kein Working Directorie und somit keine Dateien in ihrer ursprünglichen Form. Aus der Datenbank wird anhand der aktuellsten Version des Repositories eine Liste aller in dieser Version enthaltenen Dateien erstellt. Eine solche Liste wird auch für jedes Commit erstellt. Dafür wird mithilfe des für den Commit spezifischen Hashwertes eine .patch Datei von Github heruntergeladen. Diese enthält alle im Zuge des Commits vorgenommenen Änderungen. Aus dieser Datei wird dann die Liste der geänderten Dateien erzeugt. Dabei wird jeweils die Version der Datei nach dem Commit und, falls die Datei bereits vor dem Commit existierte, auch die Vorgängerversion in die Liste aufgenommen. Dies stellt sicher, dass, falls das betrachtete Ergebnis als Vulnerable Code klassifiziert wurde, sowohl die Schwachstelle als auch ihr Fix in den Dateien enthalten ist.

Anschließend werden die einzelnen Dateien über die von Github bereitgestellte Subdomain raw.githubusercontent.com heruntergeladen. Dazu wird der vollständige Pfad, also *<Name des Repositorie Besitzers>/<Name des Repositories>/<Revision>/<Dateipfad innerhalb des Repositories>/<Datei>*, aus den Metadaten erzeugt. Unter diesem Pfad lässt sich die Datei herunterladen. Der lokale Pfad zu der Datei entspricht ab dem Downloadverzeichnis des Security Code Exporters dem generierten Pfad. Die Liste der heruntergeladenen Dateien wird den Metadaten des Ergebnisses hinzugefügt, bevor die Ergebnisse in einer Datenbank gespeichert werden. In der Datenbank werden die Metadaten der Ergebnisse zusammen mit Verweisen auf die einzelnen Dateien abgelegt. Dadurch lässt sich zu jeder Datei feststellen, zu welchem Ergebnis sie gehört. Die Datenbank befindet sich in einem eigenen Verzeichnis auf derselben Ebene wie das Downloadverzeichnis.

3.4 Filter

Der Benutzer hat die Möglichkeit verschiedene Filter festzulegen, um die Ergebnisse entsprechend der benötigten Informationen einzuschränken. Die einzelnen Filter werden teilweise während und teilweise zwischen den bereits erwähnten Arbeitsschritten angewendet. Der Reihenfolge ihrer Anwendung entsprechend wird zunächst ein Programmiersprachenfilter angewendet. Durch diesen kann der Benutzer die Ergebnismenge auf Repositories einschränken, deren Sprache in einer benutzerdefinierten Liste von Programmiersprachen enthalten ist. Alle Ergebnisse, die nicht den Vorgaben entsprechen, werden aus der Ergebnismenge entfernt. Dieser Filter wird zwischen Suche und Klassifizierung angewendet, sodass nur die gewünschten Ergebnisse die nächsten Schritte durchlaufen. Als Nächstes wird zwischen Klassifizierung und Download ein Klassenfilter angewendet. Durch diesen kann der Benutzer die Ergebnismenge auf bestimmte Klassen einschränken, um zum Beispiel nur Exploit Code zu erhalten. Dies schließt auch die Möglichkeit mit ein, alle unklassifizierbaren Ergebnisse aus der Ergebnismenge zu entfernen. Die letzte Filterinstanz bildet ein Dateitypenfilter. Durch diesen kann der Benutzer festlegen, welche Dateitypen heruntergeladen werden sollen. So lässt sich der heruntergeladene, security relevante Code zum Beispiel auf .java oder .c Dateien beschränken. Dieser Filter wird direkt vor dem Herunterladen der jeweiligen Datei angewendet und bestimmt, ob eine Datei heruntergeladen wird oder nicht. Dies ist der einzige Filter, der fest in einen der Arbeitsschritte integriert ist. Wird zu einem Ergebnis keine Datei heruntergeladen, wird das Ergebnis aus der Ergebnismenge entfernt und bereits heruntergeladene Metadaten gelöscht.

3.5 Ergebnisdatenbank

Nach vollständiger Bearbeitung einer Ergebnismenge, werden die enthaltenen Ergebnisse in einer Datenbank abgelegt. Die Datenbank befindet sich ausgehend vom Arbeitsverzeichnis des Security Code Exporters in einem Unterordner mit dem Namen „db“. Sie enthält sämtliche Ergebnisse, deren Klasse, die wichtigsten zugehörigen Metadaten und den Namen und Pfad jeder zugehörigen Datei. Darüber hinaus stellt die Schnittstelle zur Datenbank Funktionen bereit, durch die bestimmte Eckdaten extrahiert werden können. Dazu gehören Anzahl der gespeicherten Ergebnisse, Anzahl der Ergebnisse einer bestimmten Klasse sowie Anzahl und Gesamtgröße der heruntergeladenen Dateien.

Diese werden dem Benutzer während des Suchvorgangs präsentiert.

Abbildung 2 zeigt das Datenbankschema der Schwachstellendatenbank. Die Schlüsselwerte sind mit dem Kürzel PK (Primary Key) gekennzeichnet. Für die Tabellen „repositories“ und „github_user“ entspricht der Schlüssel der von Github verwendeten Identifikationsnummer für das jeweilige Repository beziehungsweise das jeweilige Benutzerkonto. Der Schlüsselwert der Tabelle „commits“ entspricht der URL[7], durch die der Commit auf Github gefunden werden kann. Die URL enthält den Namen des Besitzers des Repositories, den Namen des Repositories und den für den Commit errechneten Hashwert. Die Schlüssel der Tabellen „vulnerability_code“ und „vulnerability_file“ entsprechen jeweils einem für die Tabelle einzigartigen fortlaufenden Zahlenwert ohne weitere Bedeutung. Fremdschlüssel sind mit dem Kürzel FK (Foreign Key) gekennzeichnet. Von den beiden Fremdschlüsseln der Tabellen „vulnerability_code“ und „vulnerability_file“ enthält immer nur einer einen Wert. Dadurch wird angegeben, ob ein in der Tabelle „vulnerability_code“ gespeichertes Ergebnis ein Repository oder ein Commit ist. Wenn der Eintrag der Zeile „commit_reference“ NULL und der Eintrag der Zeile „repository_reference“ ungleich NULL ist, handelt es sich um ein Repository. Im umgekehrten Fall handelt es sich um ein Commit. Auf gleiche Art und Weise kann festgestellt werden, ob eine in der Tabelle „vulnerability_file“ gespeicherte Datei zu einem Repository oder Commit gehört.

7 Uniform Resource Locator

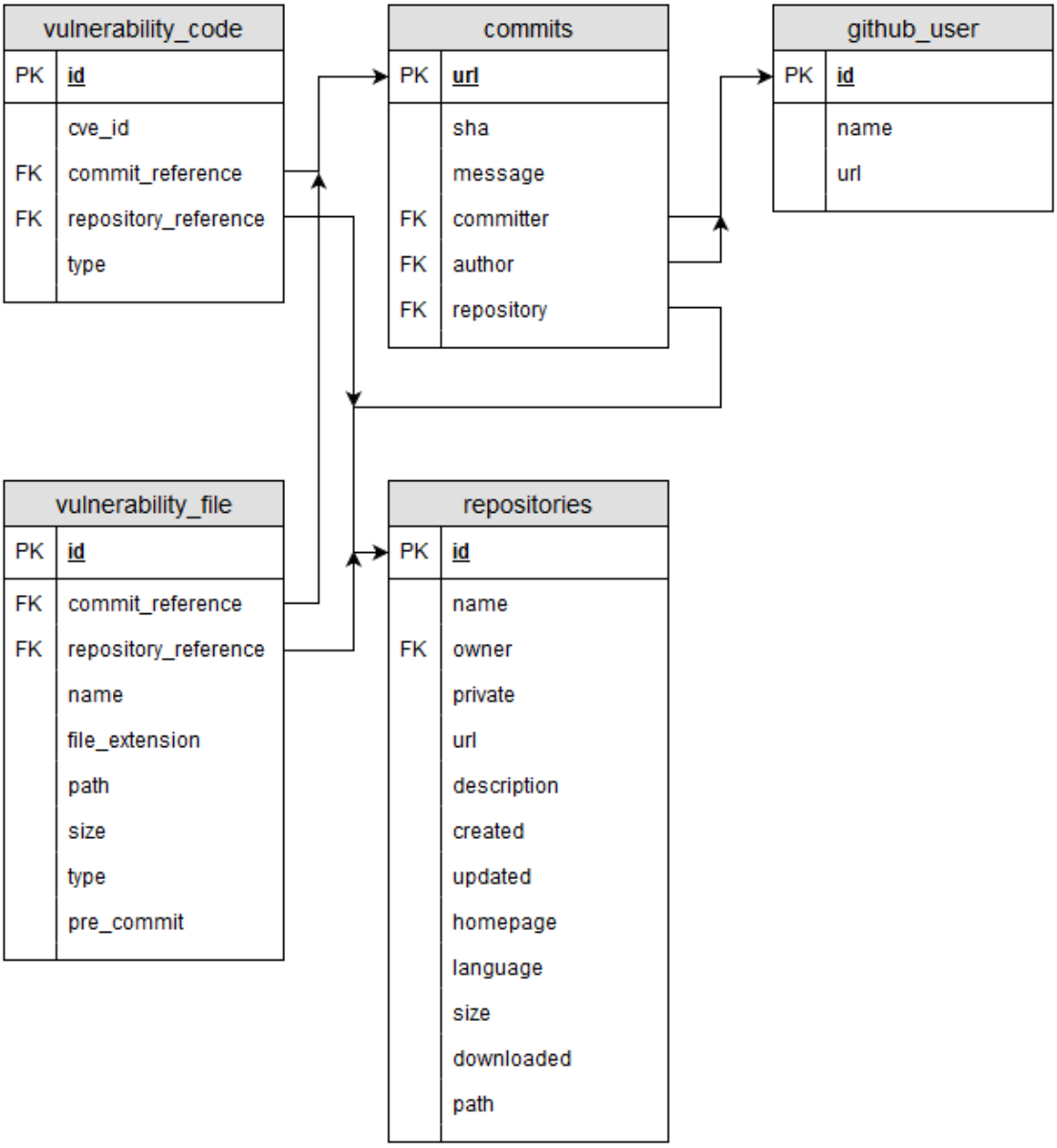

Abbildung 2: Datenbankschema der Schwachstellendatenbank

4 Implementierung

4.1 Architektur

Der Security Code Exporter wurde in Java implementiert. Seine Architektur folgt im Wesentlichen der im vorherigen Kapitel beschriebenen Aufteilung und orientiert sich grundsätzlich am Model-View-Controller (MVC) Paradigma. Bei dem MVC Programmierparadigma wird die Anwendung in drei Modulen organisiert. Das Modell (Model) enthält die Daten und Zustände der Anwendung. Die Präsentation (View) stellt die im Modell enthaltenen Informationen für den Benutzer dar und bietet Interaktionsmöglichkeiten mit der Anwendung. Die Steuerung (Controller) enthält die zur Steuerung der Anwendung notwendige Logik und verarbeitet Benutzerinteraktionen. In diesem Kapitel soll zunächst der grundlegende Aufbau des Security Code Exporters erläutert werden und anschließend auf einige wesentliche Datenstrukturen eingegangen werden.

4.1.1 Grundlegender Aufbau

Ein Faktor, der den Aufbau maßgeblich beeinflusst hat, war das Rate Limit der Github API. Derzeit werden rund 100000 CVE-IDs für die Suche auf Github verwendet. Ein kompletter Suchdurchlauf mit einer Anfrage an die Search API pro CVE-ID würde ohne Authentifikation *100000 Anfragen / 60 Anfragen pro Stunde = 1666,667 Stunden*, also fast 70 Tage dauern. Hier ist zu beachten, dass das übergeordnete Rate Limit mit 60 Anfragen pro Stunde greift, da die Zugriffe auf die Search API mit in das übergeordnete Rate Limit gezählt werden. Bei 10 erlaubten unauthentifizierten Zugriffen pro Minute auf die Search API, ist das übergeordnete Rate Limit bereits nach sechs Minuten aufgebraucht. Mit Authentifikation werden *100000 Anfragen / 60 Anfragen pro Minute = 1666,667 Minuten*, also fast 28 Stunden benötigt. Hinzu kommt, dass je nach Größe der Ergebnismenge weitere Zugriffe auf die API erfolgen. Diese Zugriffe zählen zwar nur in das übergeordnete Rate Limit, können aber, wenn das Kontingent aufgebraucht ist, zu deutlich längeren Wartezeiten führen, da das übergeordnete Rate Limit lediglich im Stundentakt zurückgesetzt wird.

Um den Zeitaufwand für die Suche so gering wie möglich zu halten, wird die Suche und der Download der Suchergebnisse in zwei verschiedenen Threads ausgeführt. Dadurch soll erreicht werden, dass immer die volle Anzahl der erlaubten Zugriffe auf die Github

API auch genutzt wird. Des Weiteren werden mehrere Suchbegriffe und damit ihre Anfragen zusammengefasst, um die Gesamtzahl der Zugriffe zu verringern. Zusätzlich kann der Benutzer mehrere Githubbenutzerkonten verwenden, um mit diesen parallel zu suchen. Auf das Zusammenfassen von Suchbegriffen und die Verwendung mehrerer Benutzerkonten wird im Kapitel „Aufbau der Suche“ näher eingegangen. Abbildung 3 zeigt den grundlegenden Aufbau des Security Code Exporters. Die einzelnen Komponenten werden in den folgenden Kapiteln näher erläutert.

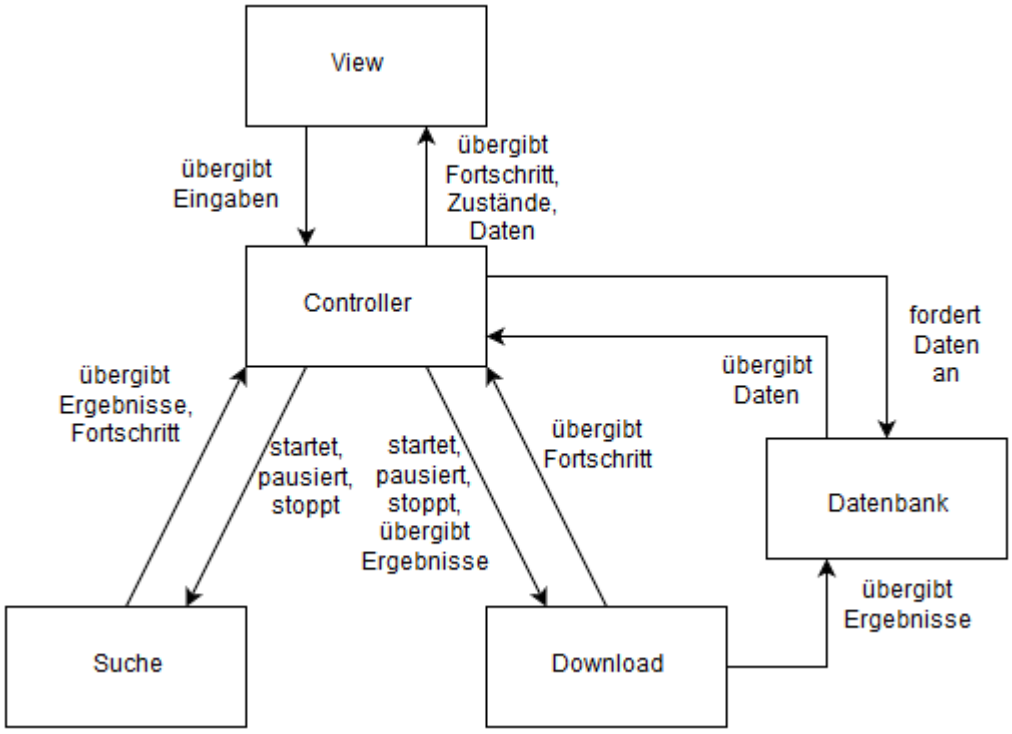

Abbildung 3: Grundlegender Aufbau des Security Code Exporters

4.1.1.1 Controller

Der Controller initialisiert den Such- und Downloadprozess in jeweils eigenen Threads. Er kann die Prozesse pausieren und stoppen. Werden die Prozesse pausiert, gehen die Threads in einen Wartezustand über und können auf Wunsch fortgesetzt werden. Die Daten und Zustände werden nicht gespeichert und gehen somit beim Beenden der Anwendung verloren. Werden die Prozesse gestoppt, beenden sich die Threads. Die Daten und Zustände können auf Wunsch gespeichert und sofort oder nach einem Neustart in neue Prozesse übernommen werden. Das ermöglicht ein besseres Zeitmanagement im Umgang mit den hohen Laufzeiten der Such- und Downloadprozesse. Der Controller

dient außerdem als Schnittstelle zwischen der Präsentation (View), den Such- und Downloadprozessen und der Datenbank. Die von den Suchprozessen erzeugten Ergebnismengen werden über den Controller an den Downloadprozess weitergeleitet. Dieser speichert die Ergebnismengen nach ihrer Verarbeitung in der Datenbank und signalisiert dem Controller eine Änderung der Datenbank. Der Controller stößt dann eine Aktualisierung der Präsentation an. Dafür werden einige Eckdaten, wie zum Beispiel die Gesamtzahl der gefundenen Ergebnisse, Anzahl der verschiedenen Repositories und Commits, Anzahl der heruntergeladenen Dateien und deren Gesamtgröße, aus der Datenbank geladen und an die Präsentation übergeben. Tätigt der Benutzer Eingaben, die die Suchprozesse, den Downloadprozess oder die Datenbank betreffen, werden diese von der Präsentation an den Controller weitergeleitet und von diesem verarbeitet.

4.1.1.2 Suche

Abbildung 4 zeigt den Aufbau der Suche. Wenn ein neuer Suchdurchlauf gestartet wird, prüft der Controller ob der Benutzer Githubbenutzerkonten hinterlegt hat. Danach entscheidet sich, ob authentifiziert oder nicht authentifiziert auf die Github API zugegriffen wird. Falls der Benutzer keine Konten hinterlegt hat oder diese nicht nutzen möchte, wird vom SearchThread Controller ein einziger SearchThread, also eine einzige suchende Instanz, erzeugt. Hat der Benutzer ein oder mehrere Konten hinterlegt und möchte diese auch nutzen, erzeugt der SearchThread Controller für jedes Konto einen SearchThread. Die Liste der Suchwörter wird von allen SearchThreads parallel bearbeitet. Dadurch kann die Dauer des Suchdurchlaufs signifikant verringert werden. Dies ist möglich, da das Rate Limit bei authentifizierten Zugriffen auf die Github API an das Benutzerkonto gebunden ist. Durch die Nutzung mehrerer Konten wird das Rate Limit für die gesamte Anwendung um 5000 Zugriffe pro Stunde beziehungsweise 60 Zugriffe pro Minute für jedes verwendete Konto erhöht. Diese Strategie funktioniert nur für authentifizierte Zugriffe, da das Rate Limit für nicht authentifizierte Zugriffe an die IP-Adresse gebunden ist. Dadurch müssten sich weitere SearchThreads ein Kontingent teilen, was keine Vorteil bieten würde.

Jeder SearchThread folgt dem gleichen internen Ablauf. Zunächst wird geprüft, ob die Suche pausiert oder gestoppt wurde. Dies wird den SearchThreads von ihrem Controller mitgeteilt. Wurde die Suche pausiert, geht der SearchThread in einen Wartezustand, verbleibt 250 Millisekunden in diesem und prüft den Zustand anschließend erneut. Wurde

die Suche gestoppt, wird der SearchThread beendet. Ist beides nicht der Fall, fordert der SearchThread sechs Suchbegriffe von der SearchItemList an. Diese Klasse ist eine spezielle Datenstruktur, die die Verteilung der Suchbegriffe an die einzelnen SearchThreads übernimmt. Sie wird im Kapitel „Wesentliche Datenstrukturen“ näher erläutert. Wenn keine Suchbegriffe zurückgegeben wurden, bedeutet das, dass alle Suchbegriffe bearbeitet wurden und der Suchdurchlauf damit beendet ist. In diesem Fall wird der SearchThread ebenfalls beendet. Wurden Suchbegriffe zurückgegeben, werden diese mit „oder“ verknüpft und für eine Anfrage an die Search API genutzt. Die Ergebnismenge enthält nun die Vereinigung alle Ergebnisse, die für die einzelnen Suchbegriffe gefunden wurden. Durch das Zusammenfassen der Suchbegriffe sinkt die Anzahl der benötigten Zugriffe auf die Github API und es können so mehr Suchbegriffe pro Rate Limit Kontingent bearbeitet werden. Es ist nicht möglich mehr als sechs Suchbegriffe zusammenzufassen, da die Search API sonst lediglich die Meldung, dass die Suchkomplexität zu groß sei, zurückgibt.

Die Suche auf Github erfolgt in zwei Stufen. Zunächst werden die Repositories durchsucht und die Ergebnisse einem Objekt der Klasse „SearchResults“ hinzugefügt. Ein Objekt dieser Klasse stellt eine Ergebnismenge dar. Auch diese Klasse wird im Kapitel „Wesentliche Datenstrukturen“ näher erläutert. Anschließend wird eine zweite Anfrage für die Suche nach Commits gestellt und auch deren Ergebnisse der Ergebnismenge hinzugefügt. Zu diesem Zeitpunkt besteht jedes Ergebnis lediglich aus den Metadaten zu einem Repository oder Commit. Als Nächstes erfolgt die Anwendung des Sprachfilters. Dabei wird die in den Metadaten der Ergebnisse angegebene Programmiersprache mit der Liste der gewünschten Sprachen verglichen und alle Ergebnisse, deren Sprache nicht gewünscht ist, aus der Ergebnismenge entfernt. Die Ergebnismenge enthält nun eine Reihe von Ergebnissen, die jeweils zu einem oder mehreren Suchbegriffen gehören. Daher wird nun versucht festzustellen, zu welchem Suchbegriff ein jedes Ergebnis gehört. Dazu werden die Metadaten der Ergebnisse nach den einzelnen Suchbegriffen durchsucht. Falls die Suche erfolgreich war, werden die Ergebnisse der zum jeweiligen Suchbegriff gehörenden Ergebnismenge hinzugefügt. Die dabei entstandenen Ergebnismengen werden nun klassifiziert. Es wird also festgelegt, ob die Ergebnisse als Exploit Code, Vulnerable Code oder Unclassified Code in der Datenbank abgelegt werden. Dieser Vorgang folgt dem im Kapitel „Klassifizierung“ beschriebenen Konzept. Wurden alle Ergebnisse einer Ergebnismenge klassifiziert, wird die Ergebnismenge dem Con-

troller übergeben und ein Suchfortschritt gemeldet.

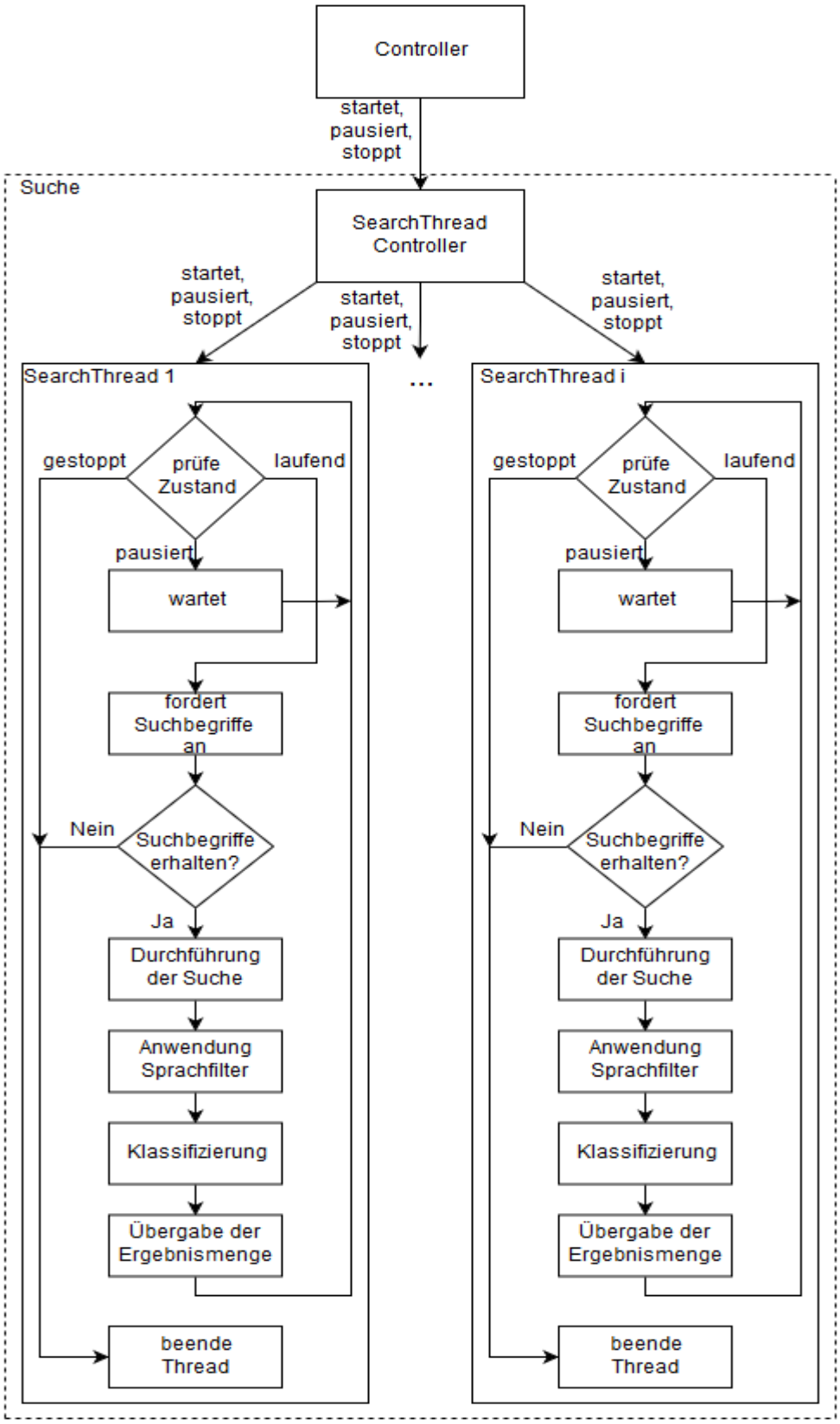

Abbildung 4: Aufbau der Suche

4.1.1.3 Download

Abbildung 5 zeigt den Aufbau des Downloadprozesses (ResultDownloader). Wie bereits bei der Suche wird auch hier zunächst geprüft, ob der Downloadprozess pausiert oder gestoppt wurde. Wurde er pausiert, geht der Downloadprozess für 250 Millisekunden in einen Wartezustand über und prüft den Zustand anschließend erneut. Wurde der Downloadprozess gestoppt, wird der Thread, in dem der Prozess läuft, beendet und auf Wunsch die Warteschlange (Queue) gespeichert. Die Warteschlange enthält die Ergebnismengen, die heruntergeladen werden sollen. Wenn der Downloadprozess weder pausiert noch gestoppt wurde, wird geprüft, ob die Warteschlange Einträge enthält. Ist dies nicht der Fall, geht der Downloadprozess in den Wartezustand über und prüft nach der Wartezeit erneut seinen Zustand. Enthält die Warteschlange Ergebnismengen, wird die nächste Ergebnismenge geladen. Für jedes enthaltene Ergebnis, also gefundene Repository oder Commit, wird aus den Metadaten eine Liste mit Dateien erstellt. Auf diese Liste wird der Dateifilter angewendet. Dieser gleicht die Dateitypen der einzelnen Dateien mit einer benutzerdefinierten Liste mit gewünschten Dateitypen ab und entfernt alle nicht gewünschten Einträge aus der Dateiliste. Anschließend wird versucht, alle Dateien über raw.githubusercontent.com herunterzuladen. Jede Datei, die nicht erfolgreich heruntergeladen werden konnte, wird aus der Dateiliste entfernt. Anschließend wird die Dateiliste den Metadaten des Ergebnisses hinzugefügt. Wurden alle Ergebnisse einer Ergebnismenge bearbeitet, werden sie der Datenbank hinzugefügt.

Der Downloadprozess kann sich im Gegensatz zu den SearchThreads nicht selbst beenden, da anders als bei der Suche nicht absehbar ist, zu welchem Zeitpunkt der Prozess vollständig bearbeitet wurde. Die einzige Möglichkeit den Thread, in dem der Downloadprozess läuft, zu beenden, ist durch das Stoppen der Such- und Downloadprozesse durch den Benutzer.

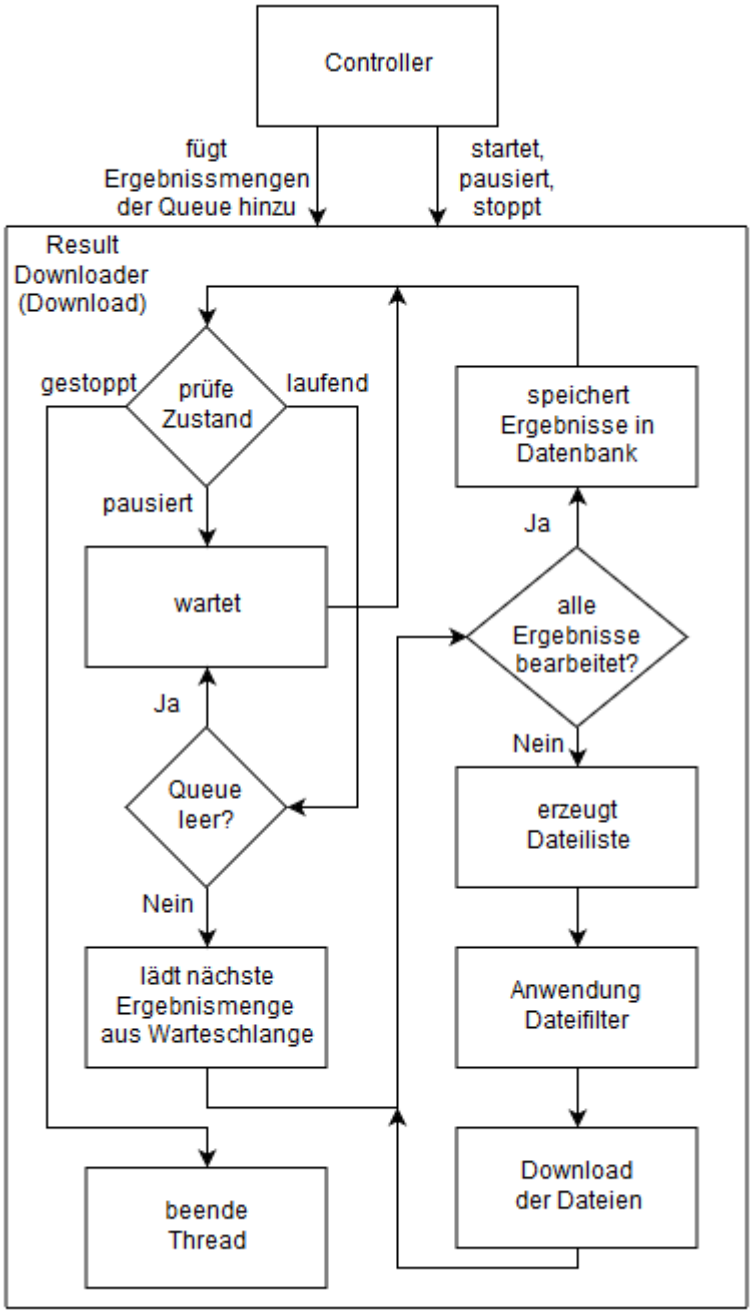

Abbildung 5: Aufbau des Result Downloaders

4.1.2 Wesentliche Datenstrukturen

Es wurden einige wesentliche Datenstrukturen erstellt, mit denen der Security Code Exporter arbeitet. Die Klasse „SearchItemList“ ist eine davon. In einem Objekt dieser Klasse werden zu Beginn der Suche alle zu verwendenden Suchbegriffe in einer Liste gespeichert. Suchbegriffe können vom Benutzer definierte Zeichenketten oder CVE-IDs sein. Die CVE-IDs werden aus einer, durch das Updatemodul des Schwachstellenprüfers für Javabibliotheken von Leif Erik Wagner [1] erstellten, lokalen CVE-Datenbank geladen. Die Liste wird, wenn mehrere SuchThread-Instanzen verwendet werden, von vorne und von hinten beginnend abgearbeitet. Die Klasse enthält daher zwei Zeiger, die auf den jeweils nächsten Suchbegriff zeigen. Fordert ein SearchThread Suchbegriffe an, wird anhand seiner Nummer bestimmt, ob er die Liste von vorne nach hinten oder von hinten nach vorne abarbeitet. Dem SearchThread werden, falls noch Suchbegriffe bearbeitet werden müssen, die jeweils nächsten Suchbegriffe übergeben und der entsprechende Zeiger wird aktualisiert. Die SearchThreads werden nacheinander bedient, wenn mehrere SearchThreads parallel Suchbegriffe anfordern. Dadurch wird verhindert, dass ein Suchbegriff von mehreren SearchThreads bearbeitet wird. Diese Klasse ist also Thread safe.

Die Klasse „SearchResults“ ist eine weitere wesentliche Datenstruktur. Ein Objekt dieser Klasse nimmt jeweils eine von der Suche produzierte Ergebnismenge auf. Es enthält jeweils ein Liste mit Objekten der Klasse „Repository“ und eine mit Objekten der Klasse „Commit“. Diese beiden Listen enthalten die Ergebnisse der Suche. Des Weiteren enthält das Objekt eine Liste mit Suchbegriffen, die zu dieser Ergebnismenge gehören. Dabei handelt es sich um die sechs zu einer Anfrage zusammengefassten Begriffe. Nachdem die einzelnen Ergebnisse den Suchbegriffen zugeordnet wurden, enthält die Liste nur noch einen Suchbegriff und die dazu gefundenen Repositories und Commits.

Die Klassen „Repository“, „Commit“ und „GithubUser“ enthalten die Metadaten der Ergebnisse. Abbildung 6 zeigt, welche Informationen die Klassen enthalten. Das Feld „id“ der Klassen „Repository“ und „GithubUser“ enthalten die Identifikationsnummern, unter denen Github die Repositories und Nutzer speichert. Mithilfe dieser Nummer kann einfach und ohne die Search API nutzen zu müssen auf die auf Github hinterlegten Daten zugegriffen werden. Die Felder „url“ enthalten einen direkten Link zu dem hinterlegten Repository, Commit oder Benutzerkonto auf Github. Der Link wird genutzt,

um den Benutzer des Security Code Exporters auf Github weiterleiten zu können. Das Feld „size“ der Klasse Repository enthält die Größe des gesamten Repositories, wie es auf Github hinterlegt ist, und nicht etwa die Summe der Größen der heruntergeladenen Dateien. Das Feld „downloaded“ gibt an, ob ein lokaler Klon des Repositories angelegt wurde. Das passiert ausschließlich, wenn das Repository als eigenständiges Ergebnis gefunden wurde. Ein Repository, dass zu einem Commit gehört, wird nicht geklont. Die Felder „type“ der Klassen „Repository“ und „Commit“ enthalten nach der Klassifizierung die Klasse des Ergebnisses. Die Felder „files“ der Klasse „Repository“ und „pre_Commit_Files“ und „post_Commit_Files“ der Klasse „Commit“ werden nach dem Download der gewünschten Dateien gesetzt und enthalten eine Liste mit allen zu diesem Ergebnis heruntergeladenen Dateien.

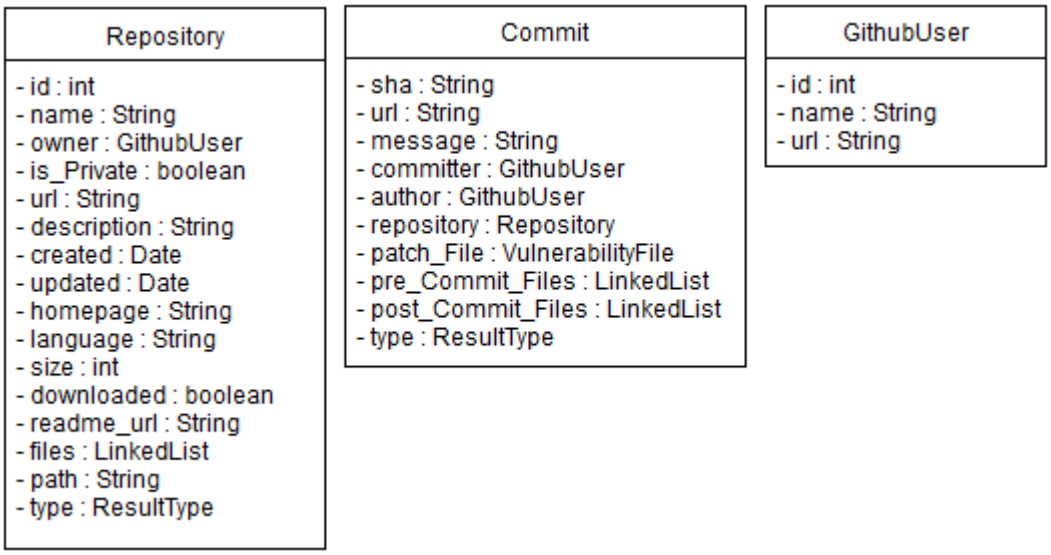

Abbildung 6: Wesentliche Datenstrukturen

4.2 GUI

Die Benutzeroberfläche (GUI[8]) des Security Code Exporters besteht aus zwei Hauptansichten. Die Ansicht „Create Database“ wird während der Erzeugung der Schwachstellendatenbank verwendet. Die Ansicht „Browse Database“ ist die zweite Ansicht und wird verwendet, um die Schwachstellendatenbank manuell zu durchsuchen.

4.2.1 Ansicht „Create Database“

Die Ansicht „Create Database“ bietet während der Erzeugung der Schwachstellendatenbank einen Überblick über den aktuellen Suchfortschritt, den Fortschritt des Downloadprozesses bei der Bearbeitung einer bestimmten Ergebnismenge, Eckdaten des derzeitigen Standes der Schwachstellendatenbank und eine Liste mit Statusmeldungen des Security Code Exporters. Abbildung 7 zeigt die Ansicht zu einem Zeitpunkt, an dem die Suche abgeschlossen ist und nur der Downloadprozess aktiv ist.

Der Benutzer kann den Such- und Downloadprozess in dieser Ansicht starten, pausieren und stoppen. Stoppt der Benutzer die Prozesse, wird er gefragt, ob der aktuelle Fortschritt gespeichert werden soll. Des Weiteren gelangt der Benutzer über diese Ansicht zu den Einstellungen des Security Code Exporters.

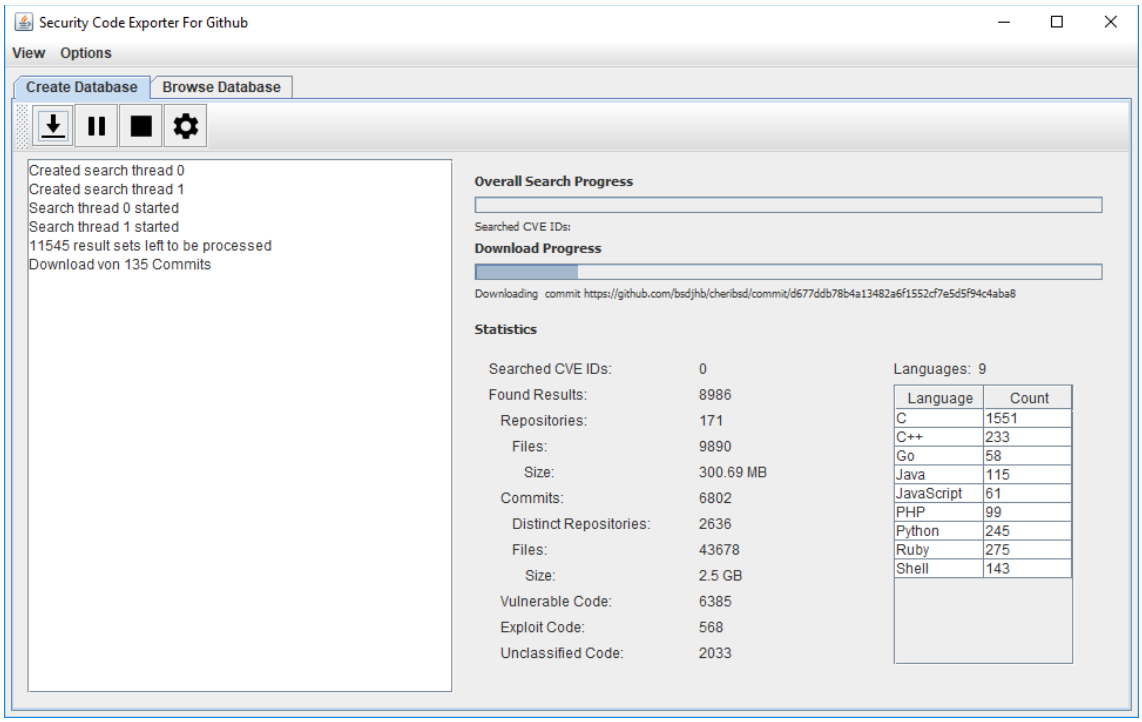

Abbildung 7: Ansicht "Create Database"

8 Graphical User Interface

4.2.1.1 Einstellungen

Die Einstellungen des Security Code Exporters sind in fünf Kategorien aufgeteilt. Abbildung 8 zeigt die Kategorien und ihre Einstellungsmöglichkeiten. In der Kategorie „Search“ kann der Benutzer eine Liste mit eigenen Suchwörtern anlegen und verwalten. Er kann Einträge zu der Liste hinzufügen und die Einträge bearbeiten oder löschen. Außerdem kann festgelegt werden, ob die CVE-IDs aus der lokalen CVE-Datenbank, die Einträge der benutzerdefinierten Liste oder beides als Suchbegriffe für die Suche verwendet werden soll. In der Kategorie „Download“ kann das Arbeitsverzeichnis des Security Code Exporters festgelegt werden. Unter diesem Verzeichnis werden die Schwachstellendatenbank und die heruntergeladenen Ergebnisse gespeichert. Darüber hinaus kann durch den Menüpunkt „File Structure“ festgelegt werden, ob die heruntergeladenen Ergebnisse, ihrer Klassifizierung entsprechend, in Unterordnern abgelegt oder alle Ergebnisse in einem einzigen Ordner zusammengefasst werden sollen. Welche Klassen heruntergeladen werden sollen, lässt sich durch den Menüpunkt „Download Result Type“ festlegen. Diese Einstellung stellt den Klassenfilter dar. Die Sprach- und Dateifilter lassen sich unter „Filter“ konfigurieren. Der Benutzer kann zum einen angeben, ob der jeweilige Filter angewendet werden soll, und zum anderen festlegen, wonach gefiltert werden soll. Die Filter beachten keine Groß- und Kleinschreibung. Die Einträge „Java“ und „java“ im Sprachfilter haben also denselben Effekt. In der Kategorie „Accounts“ kann festgelegt werden, ob der Security Code Exporter die hinterlegten Githubbenutzerkonten zur Authentifizierung verwenden soll. Des Weiteren kann der Benutzer Benutzerkonten hinterlegen und verwalten. Die letzte Kategorie enthält die Listen mit den zur Klassifizierung verwendeten Schlüsselwörtern. Die Listen enthalten Standardeinträge, können aber vom Benutzer angepasst und optimiert werden.

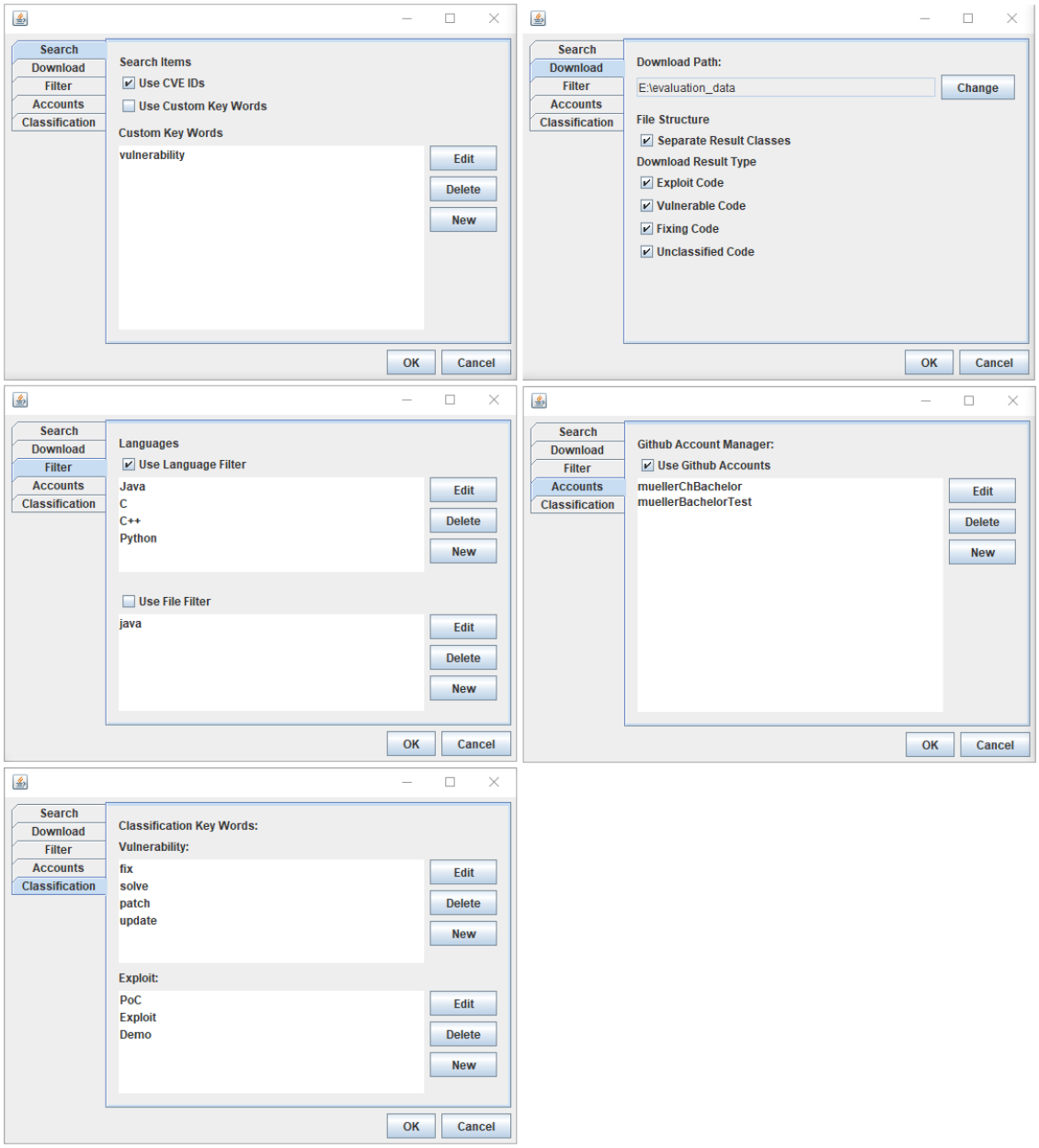

Abbildung 8: Einstellungen

4.2.2 *Ansicht „Browse Database“*

Diese Ansicht gibt dem Benutzer des Security Code Exporters die Möglichkeit manuell nach Einträgen in der Datenbank zu suchen und diese einzusehen. Es wird nach Repositories, Commits und GithubUsern gesucht und die Suchergebnisse in einer Liste angezeigt. Ein Icon links neben dem Eintrag gibt an, ob es sich um ein Repository, Commit oder GithubUser handelt. Wählt der Benutzer einen Eintrag aus, werden die Metadaten aus der Datenbank abgerufen und angezeigt. Zusätzlich zu den regulären Metadaten werden weitere Informationen, wie zum Beispiel eine Liste mit Commits, die zu einem Repository in der Schwachstellendatenbank existieren, oder Listen mit Repositories und Commits, die zu einem GithubUser gehören, angezeigt. Wählt der Benutzer einen Eintrag dieser Listen durch einen Doppelklick aus, lädt der Security Code Exporter die zum Eintrag gehörenden Metadaten aus der Datenbank und zeigt sie an. Zu den regulären Metadaten gehört auch eine Liste mit Dateien, die zum jeweiligen Repository oder Commit gehören. Mittels eines Doppelklicks kann der Ordner, in dem die Datei abgelegt ist, geöffnet werden. Darüber hinaus wird der Benutzer bei einem Doppelklick auf eine hinterlegte URL auf die entsprechende Internetseite unter Verwendung des Standardbrowsers weitergeleitet. Abbildung 9 zeigt die Ansicht „Browse Database“.

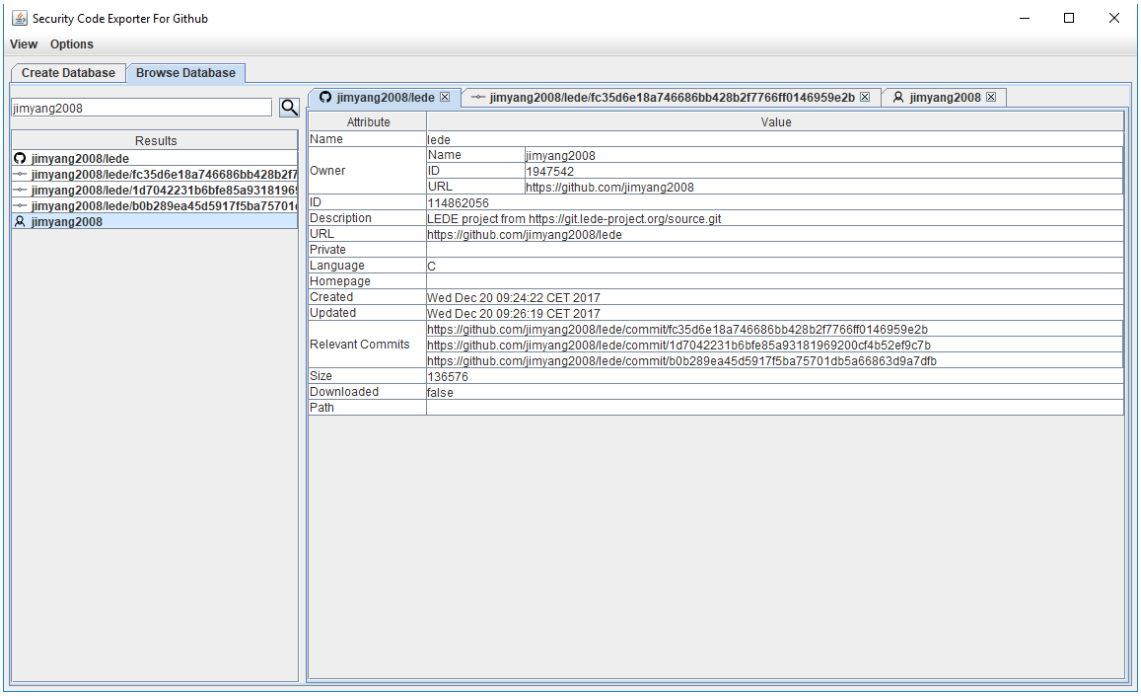

Abbildung 9: Ansicht "Browse Database"

5 Evaluation

In diesem Kapitel sollen die vom Security Code Exporter gefundenen Ergebnisse evaluiert werden. Dazu wurde eine Schwachstellendatenbank erstellt, die auf lediglich neun Programmiersprachen beschränkt wurde. Es wurden keine Klassen- und Dateifilter angewendet. Die genutzte Schwachstellendatenbank ist auf der beigelegten DVD zu finden. Zunächst wird die Suche, dann die Klassifizierung und abschließend die Laufzeit evaluiert.

5.1 Suche

5.1.1 Precision

Die von einer Suchanfrage zurückgegebenen Ergebnisse müssen nicht zwangsläufig alle relevant sein. Die irrelevanten Ergebnisse werden als False Positivs (FP) bezeichnet. Die relevanten Ergebnisse hingegen als True Positives (TP). Die Precision einer Suchanfrage ergibt sich aus dem Anteil der relevanten Ergebnisse an allen zurückgegebenen Ergebnissen. Der kleinste erreichbare Wert ist eine Precision von 0. Das würde bedeuten, dass in den zurückgegebenen Ergebnissen kein relevantes Ergebnis enthalten ist. Der größte erreichbare Wert ist eine Precision von 1. Das würde wiederum bedeuten, dass alle zurückgegebenen Ergebnisse relevant sind. Ein möglichst hoher Wert für die Precision ist also erstrebenswert.

$$Precision = \frac{TP}{TP + FP}$$

Da die zur Evaluation erzeugte Schwachstellendatenbank mit 9058 enthaltenen Ergebnissen zu groß ist, um alle Ergebnisse händisch auf ihre Relevanz zu prüfen, wurden stattdessen 30 durch einen Zufallsgenerator ausgewählte Ergebnisse betrachtet. Unter den Ergebnissen befinden sich 29 Commits und 1 Repository. Der Anteil der Repositories an der betrachteten Ergebnismenge entspricht somit 3,3%. Insgesamt sind 282 von 9058 Ergebnissen Repositories. Dies entspricht einem Anteil von 3,1 %. Die generierte Ergebnismenge erhält also die Verteilung der Ergebnisse. Von den 30 Ergebnissen wurden 19 nach händischer Begutachtung der Metadaten und zugehörigen Dateien als relevant eingestuft. Demnach weist die Suche eine Precision von 0,633 auf. Einige der Ergebnisse wurden als irrelevant eingestuft, da sie keine Dateien mit Quellcode enthalten.

Diese Ergebnisse können durch den Dateifilter entfernt werden. Dadurch lässt sich die Precision der Suche erhöhen. Eine Liste mit den Schlüsselwerten der betrachteten Ergebnisse befindet sich auf der beigelegten DVD.

5.1.2 Recall

So wie nicht alle durch eine Suche erhaltenen Ergebnisse relevant sein müssen, kann es auch passieren, dass nicht alle relevanten Ergebnisse von der Suche gefunden wurden. Die relevanten Ergebnisse, die nicht von der Suche gefunden wurden, werden als False Negatives (FN) bezeichnet. Der Recall einer Suche ist der Anteil der gefundenen relevanten Ergebnisse an allen relevanten Ergebnissen. Der kleinste mögliche Wert für den Recall einer Suche beträgt 0. Das würde bedeuten, dass kein relevantes Ergebnis von der Suche gefunden wurde. Der größte mögliche Wert beträgt 1. Das wiederum würde bedeuten, dass alle relevanten Ergebnisse von der Suche gefunden wurden. Auch für den Recall einer Suche ist ein möglichst hoher Wert erstrebenswert.

$$Recall = \frac{TP}{TP + FN}$$

Aufgrund der Größe von Github ist es nicht möglich, die Menge aller relevanten Ergebnisse zu ermitteln. Auch die Einschränkung der Suche auf einen einzigen Suchbegriff löst die Problematik nicht, da auch dann für jedes auf Github hinterlegte Repository und für jeden Commit geprüft werden müsste, ob das Repository oder der Commit für die Suche relevant ist. Aus diesem Grund kann keine Aussage zum Recall der Suche des Security Code Exporters gemacht werden.

5.2 Klassifizierung

5.2.1 Precision

Die Precision einer Klassifizierung bezieht sich auf die unterschiedlichen Klassen. Die Precision, mit der eine Klasse erkannt wird, ist der Anteil der korrekt zu dieser Klasse zugeordneten Ergebnisse (TP) an allen zu dieser Klasse zugeordneten Ergebnissen. Die fälschlicherweise zu dieser Klasse zugeordneten Ergebnisse werden als False Positives (FP) bezeichnet.

$$Precision_{Klasse\,i} = \frac{TP}{TP + FP}$$

Da auch in diesem Fall die Menge aller Ergebnisse zu groß ist, um jede Klassifizierung händisch zu prüfen, wurde erneut eine Testmenge mit zufällig ausgewählten Ergebnissen erzeugt. Um die Precision jeder Klasse berechnen zu können, wurden für jede Klasse jeweils 10 Ergebnisse zufällig ausgewählt. Aus der Klasse „Unclassified Code“ wurden von 10 Ergebnissen 4 richtig zugeordnet. Demnach ist die Presision der Klassifizierung für diese Klasse 0,4. Aus der Klasse „Exploit Code“ wurden 9 und aus der Klasse „Vulnerable Code“ wurden ebenfalls 9 Ergebnisse richtig zugeordnet. Demnach ist die Precision für die Klassifizierung dieser Klassen 0,9.

5.2.2 Recall

Der Recall einer Klassifizierung bezieht sich ebenfalls auf die jeweilige Klasse. Er ergibt sich aus dem Anteil der richtig zu der Klasse zugeordneten Ergebnisse (TP) an allen Ergebnissen, die zu dieser Klasse gehören. Ergebnisse, die zu einer bestimmten Klasse gehören, aber nicht als solche klassifiziert wurden, werden als Flase Negatives (FN) bezeichnet.

$$Recall_{Klassei} = \frac{TP}{TP + FN}$$

Die Werte für den Recall der Klassifizierung einer bestimmten Klasse werden aus der im vorherigen Kapitel beschriebenen Testmenge berechnet. So beträgt der Recall für die Klasse „Unclassified Code“ 0,8, für die Klasse „Exploit Code“ 0,9 und für die Klasse „Vulnerable Code“ 0,6.

5.3 Laufzeit

Je nach Einstellung des Security Code Exporters braucht der Downloadprozess deutlich mehr Zeit als die Suche. Im besten Fall braucht der Downloadprozess etwa so viel Zeit wie die Suche. Das ist der Fall, wenn die Ergebnisse durch die verschiedenen Filter stark eingeschränkt werden. Abgesehen davon lässt sich der Downloadprozess nicht weiter beschleunigen. Die Suche hingegen kann durch das Hinzufügen weiterer Githubbenutzerkonten weiter beschleunigt werden. So dauerte ein Suchdurchlauf mit einem Githubbenutzerkonto etwa 49 Stunden, mit zwei Benutzerkonten etwa 23 Stunden und mit drei etwa 14 Stunden.

Der Download der Testdatenbank hingegen wurde nach etwa vier Tagen mit einem Fortschritt von etwa 35% gestoppt. Bei der Testdatenbank wurden die Ergebnisse auf neun beliebte Programmiersprachen beschränkt und sonst keine weiteren Filter angewendet. Beschränkt man die Ergebnisse jedoch auf die Programmiersprache Java und lässt den Security Code Exporter lediglich .java Dateien herunterladen, braucht der Downloadprozess lediglich einige Minuten länger als die Suche mit zwei Benutzerkonten. Das zeigt, wie wirksam die Filter bei der Begrenzung der Laufzeit des Downloadprozesses sind.

6 Related Work

Die Aufgabe des Security Code Exporters besteht in der Erzeugung einer benutzerdefinierten Schwachstellendatenbank. Die Datenbank enthält security relevanten Code und soll von einer Code Clone Detection Software genutzt werden, um mögliche Schwachstellen möglichst früh in der Entwicklungsphase eines Softwareprojekts zu identifizieren. Liu et al. beschreiben in ihrem Artikel „VFDETECT: A vulnerable code clone detection system based on vulnerability fingerprint“ [9] einen ähnlichen Ansatz. Dazu werden zu Schwachstellen „Fingerprints“ generiert. Anhand der Fingerprints sollen Klone der Schwachstelle identifiziert werden. Zur Erzeugung der Fingerprints werden die bei der Beseitigung der Schwachstelle entstandenen .diff Dateien verwendet. Diese enthalten alle geänderten Zeilen in der ursprünglichen und geänderten Form in einem oder mehreren Ausschnitten des Quellcodes. Diese Ausschnitte werden normalisiert und, nach entfernten, also ursprünglichen, und hinzugefügten, also geänderten, Zeilen getrennt. Die getrennten Zeilen werden anschließend in Hashwerte umgerechnet. Die zu den .diff Dateien erzeugten Hashwerte werden als Fingerprint der Schwachstelle in einer Schwachstellendatenbank abgelegt. Es wird ausgeführt, dass es wichtig sei, alle bei der Beseitigung einer Schwachstelle gemachten Änderungen zu betrachten, damit der Kontext und die Bedingungen, die erfüllt seien müssen, um eine Schwachstelle auszulösen, erhalten blieben. Als Datenbasis wurden Schwachstellen aus großen Open Source Softwareprojekten, wie zum Beispiel „Linux kernel“, „Ffmpeg“ und „Firefox“, genutzt.

Der Security Code Exporter speichert zu gefundenen Commits .patch Dateien ab. Diese enthalten alle im Zuge des Commits gemachten Änderungen am Projekt. Die enthaltenen Informationen entsprechen denen aller .diff Dateien, die zu den geänderten Dateien erstellt werden können. Dies entspricht der in VFDETECT verwendeten Datenbasis zur Erzeugung der Fingerprints. Zusätzlich werden vom Security Code Exporter nicht nur die Bereiche des Quellcodes, die geändert wurden, sondern die ganzen Dateien gespeichert. Anders als bei VFDETECT, werden die vom Security Code Exporter gefundenen Suchergebnisse nicht normalisiert und durch spezifische Hashwerte dargestellt. Der Security Code Exporter sucht und speichert lediglich vermeintlich security relevanten Code. Es wird keine Nachbearbeitung der Ergebnisse durchgeführt.

7 Resümee

7.1 Zusammenfassung

Der Security Code Exporter wurde entwickelt, um anwendungsspezifische Schwachstellendatenbanken zu erstellen. Dadurch soll einer Code Clone Detection Software eine angepasste Datengrundlage gegeben werden, durch die die Software in der Lage ist, mögliche Schwachstellen in Softwareprojekten so früh wie möglich während der Entwicklungsphase zu finden. Der Decurity Code Exporter durchsucht dafür die auf Github hinterlegten Repositories und Commits nach CVE-IDs und weiteren festlegbaren Begriffen. Die Ergebnisse werden klassifiziert und heruntergeladen. Sie können durch verschiedene Filter auf bestimmte Klassen, Programmiersprachen und Dateitypen reduziert werden. Der Security Code Exporter besitzt eine Graphische Benutzeroberfläche, durch die der Benutzer der Anwendung Einstellungen tätigen, den Fortschritt der Such- und Downloadprozesse einsehen, deren Fortgang steuern und die in der Schwachstellendatenbank gespeicherten Informationen durchsuchen und einsehen kann. Die heruntergeladenen Dateien werden nicht weiter verarbeitet. Es wurde keine Normalisierung und Duplikatsprüfung umgesetzt.

7.2 Ausblick

Im nächsten Schritt müsste eine Normalisierung des heruntergeladenen Quellcodes umgesetzt werden. Die Normalisierungsmethode muss an die jeweils betrachtete Programmiersprache angepasst werden. So kann es zum Beispiel Sinn machen, bei Sprachen wie Java oder C, Leerzeilen und Leerzeichen zu entfernen. Die Programm- und Kontrollstuktur des Quellcodes bleibt aufgrund der Klammerung der Blöcke erhalten. Werden Leerzeilen und Leerzeichen aber aus Python-Code entfernt, gehen Programm- und Kontrollstrukturen verloren. Wurde ein Normalisierungsverfahren für eine Sprache entwickelt, können anschließend Duplikate einer Schwachstelle aus der Schwachstellendatenbank entfernt werden. Dadurch kann die Menge des für die Code Clone Detection Software relevanten Codes verringert werden, ohne Informationen zu Schwachstellen zu verlieren. Außerdem kann die Laufzeit des Such- und Downloadprozesses weiter optimiert werden. Um einige Metadaten zu Ergebnissen zu erhalten, sind weitere Zugriffe auf die Github API notwendig. Es könnte geprüft werden, ob die benötigten Metadaten

bereits in der Datenbank enthalten sind, bevor eine Anfrage an die Github API gestellt wird. Dies ist zum Beispiel der Fall, wenn mehrere Commits zum selben Repository heruntergeladen werden. Anstelle die zum Repository gehörenden Metadaten für jeden Commit erneut von der API zu beziehen, können die bereits vorhandenen Metadaten aus der Datenbank geladen werden. Eine ähnliche Vorgehensweise bietet sich auch für den Downloadprozess an. Derzeit wird für jede Datei geprüft, ob diese bereits existiert. Nur, wenn die Datei nicht existiert, wird sie heruntergeladen. Stattdessen könnte geprüft werden, ob das Repository oder der Commit bereits heruntergeladen wurde. Ist dies der Fall und es wurde keine Änderung des Dateifilters vorgenommen, kann die erneute Bearbeitung des Ergebnisses übersprungen werden. Des Weiteren kann geprüft werden, ob die Verteilung der Ergebnismengen auf mehrere parallel arbeitende Downloadprozesse eine Verringerung der Laufzeit bewirkt. Damit könnte den hohen Laufzeiten des Downloadprozesses bei geringem Einsatz der Filter entgegengewirkt werden. Eine weitere Verbesserung der Benutzbarkeit der Schwachstellendatenbank könnte erzielt werden, indem die in der Datenbank hinterlegten Dateipfade von absoluten zu relativen Pfaden geändert werden. Zurzeit wird in der Datenbank ein absoluter Pfad zu einer bestimmten Datei gespeichert. Das hat den Nachteil, dass sobald die Datenbank und die zugehörigen Ergebnisse an eine andere Stelle im Dateisystem kopiert werden, die absoluten Pfade nicht mehr gültig sind. Das ist der Grund dafür, dass der Security Code Exporter die Ordner der Dateien auf der DVD nicht öffnen kann. Eine Änderung der Dateipfade zu einer relativen Repräsentation beginnend mit dem gewählten Arbeitsverzeichnis des Securtity Code Exporters, würde das Problem lösen.

8 Anhang

8.1 Literaturverzeichnis

[1] Wagner, Leif Erik; Konzept und Entwicklung eines Schwachstellenprüfers für Java-Bibliotheken. Gottfried Wilhelm Leibniz Universität Hannover Fakultät für Elektrotechnik und Informatik Institut für Praktische Informatik Fachgebiet Software Engineering, 2017

[2] BSI: Glossar Schwachstelle. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html, . - [abgerufen 10.03.2018]

[3] CVE: FAQS What is a CVE ID Entry? https://cve.mitre.org/about/faqs.html, . - [abgerufen 10.03.2018]

[4] NVD: FAQ What is the difference between the NVD and the Common Vulnerabilities and Exposures (CVE) standard vulnerability dictionary? https://nvd.nist.gov/general/faq, . - [abgerufen 10.03.2018]

[5] Chacon, Scott; Straub, Ben: Pro Git EVERYTHING YOU NEED TO KNOW ABOUT GIT. 2. Auflage. Apress, 2014, S. 8 - 16

[6] Github: Octoverse. https://octoverse.github.com/, 2017. - [abgerufen 10.03.2018]

[7] Github: Github Developer REST API v3. https://developer.github.com/v3/, . - [abgerufen 09.03.2018]

[8] Github: Github Developer REST API v3 Search. https://developer.github.com/v3/search/ . - [abgerufen 09.03.2018]

[9] Liu, Zhen; Wei, Qiang; Cao, Yan: VFDETECT: A vulnerable code clone detection system based on vulnerability fingerprint. 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, 2017, S. 548-553

8.2 Abbildungsverzeichnis

8.3 DVD

8.3.1 Inhalt

- Security Code Exporter Executable
- Security Code Exporter Projektarchiv
- Internetquellen
- Schriftliche Ausarbeitung
- Daten der Evaluation
- Testdatenbank mit Dateien