

**BACHELORTHESIS**  
Florian Nehmer

# Entwicklung eines Systems zur automatisierten statischen Sicherheitsüberprüfung von Software Repositories zur Integration in agile Softwareentwicklungsprozesse

---

**FAKULTÄT TECHNIK UND INFORMATIK**  
Department Informatik

Faculty of Computer Science and Engineering  
Department Computer Science

Florian Nehmer

Entwicklung eines Systems zur automatisierten  
statischen Sicherheitsüberprüfung von Software  
Repositories zur Integration in agile  
Softwareentwicklungsprozesse

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

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski  
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 13.12.2019

**Florian Nehmer**

**Thema der Arbeit**

Entwicklung eines Systems zur automatisierten statischen Sicherheitsüberprüfung von Software Repositories zur Integration in agile Softwareentwicklungsprozesse

**Stichworte**

IT Sicherheit, Softwareentwicklungsprozess, Agile Softwareentwicklung, Git, Shannon Entropie, Reguläre Ausdrücke, CVE, CWE, CVSS

**Kurzzusammenfassung**

Bei modernen agilen Softwareentwicklungsprozessen hat aufgrund der drei konkurrierenden Ressourcen Zeit, Geld und Qualität häufig einer dieser Aspekte das Nachsehen. Im Zuge von Zeitdruck kann es insbesondere passieren, dass die IT-Sicherheit während der Softwareentwicklung vernachlässigt wird. Als Folge daraus wurde in der Vergangenheit beobachtet, dass in Programmquelltexten, die in öffentlichen Softwarerepositories zu finden sind des Öfteren sensible Informationen bzw. Zugangsdaten im Klartext zu finden sind, oder auch Softwareabhängigkeiten zu Drittanbietersoftware bestehen, die Schwachstellen aufweisen. Diese Arbeit versucht durch die Konzeption und Entwicklung eines Systems, dass diese Probleme erkennt, eine Lösung anzubieten, welche gleichzeitig in der praktischen Anwendung minimalen Zusatzaufwand bedeutet.

**Florian Nehmer**

**Title of Thesis**

Design and implementation of a system for static security analysis of software repositories for the integration into agile software development processes

**Keywords**

IT security, software development process, agile software development, git, shannon entropy, regular expressions, CVE, CWE, CVSS

**Abstract**

---

Modern agile software development in many cases has to deal with three different planning resources of project management which form a trilemma: Time, money and quality. As a result software security sometimes gets neglected because of time pressure which is followed by lack of dilligence. Furthermore it was discovered that often times public available software repositories contain undisclosed confidential secrets in their containing source code, aswell as 3rd party libraries, which contain vulnerabilities or weaknesses. As a solution approach this piece of work provides a concept for a system, aswell as an implementation of this system, which tackles stated issues and at the same tries to generate as little extra work as possible in the daily software development routine.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>viii</b>
<b>Tabellenverzeichnis</b>	<b>x</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Ziel der Arbeit . . . . .	2
1.2 Zielgruppe der Arbeit . . . . .	2
1.3 Struktur der Arbeit . . . . .	3
<b>2 Grundlagen</b>	<b>4</b>
2.1 Git . . . . .	4
2.1.1 Funktionsweise von Git . . . . .	4
2.1.2 Nutzung von Git in der Softwareentwicklung . . . . .	5
2.1.3 Branching Modell nach Vincent Driessen . . . . .	5
2.1.4 Moderne Funktionen von Git Systemen . . . . .	9
2.2 Geheimniserkennung . . . . .	10
2.2.1 Geheimnis . . . . .	11
2.2.2 Erkennung potentieller Geheimnisse . . . . .	11
2.2.3 Shannon-Entropie . . . . .	11
2.2.4 Regulärer Ausdruck . . . . .	14
2.2.5 Verifizierung potentieller Geheimnisse . . . . .	16
2.3 Überprüfung von Softwareabhängigkeiten . . . . .	18
2.3.1 Softwareabhängigkeit . . . . .	18
2.3.2 Risiken durch Softwareabhängigkeiten . . . . .	18
2.3.3 Node Package Manager . . . . .	19
2.3.4 sonatype OSS Index . . . . .	20
2.4 Metriken . . . . .	21
2.4.1 CVE und CVSS . . . . .	21
2.4.2 CWE . . . . .	23

<b>3</b>	<b>Systemkonzept</b>	<b>25</b>
3.1	Ziel der Anwendung . . . . .	25
3.2	Funktionale Anforderungen . . . . .	25
3.3	Nichtfunktionale Anforderungen . . . . .	27
3.4	Architektur . . . . .	28
3.4.1	Kontextsicht . . . . .	28
3.4.2	Bausteinsicht . . . . .	29
3.4.3	Verteilungssicht . . . . .	30
<b>4</b>	<b>Entwicklung</b>	<b>32</b>
4.1	Verwendete Technologie . . . . .	32
4.1.1	Python Flask . . . . .	32
4.1.2	SQLAlchemy . . . . .	32
4.1.3	SQLite . . . . .	33
4.1.4	Angular . . . . .	33
4.1.5	Docker . . . . .	33
4.2	Verwendete Dienste . . . . .	34
4.2.1	Google SMTP Dienst . . . . .	34
4.2.2	Sonatype OSS Index . . . . .	34
4.2.3	truffleHog . . . . .	34
4.2.4	Common Words in Programming Languages . . . . .	35
4.2.5	Verwendete Entwicklungswerkzeuge . . . . .	35
4.3	Implementierungsdetails . . . . .	35
4.3.1	Autorisierung des Systems zum Scannen . . . . .	36
4.3.2	Verwendete reguläre Ausdrücke . . . . .	36
4.3.3	Wichtige Datentypen . . . . .	36
4.3.4	Gefundene Geheimnisse in bestimmten Dateien und Pfaden ausschließen . . . . .	37
<b>5</b>	<b>Integration in den agilen Softwareentwicklungsprozess</b>	<b>40</b>
5.1	Der agile Softwareentwicklungsprozess . . . . .	40
5.2	Einordnung des Systems in den agilen Softwareentwicklungsprozess . . . . .	41
5.3	Reaktion auf Befunde . . . . .	41
5.3.1	Reaktion auf korrekt gefundene Geheimnisse . . . . .	41
5.3.2	Reaktion auf fehlerhaft entdeckte Geheimnisse . . . . .	44
5.3.3	Reaktion auf gefundene Bibliotheken, die Sicherheitslücken enthalten	44

5.4	Benutzung des Systems . . . . .	44
5.4.1	Autorisierung des Systems zum Scannen . . . . .	44
5.4.2	Manuelles Scannen von Repositories . . . . .	45
5.4.3	Konfigurieren der E-Mail Empfängerliste . . . . .	46
5.4.4	Einrichten des Webhooks der Plattform Github für automatisiertes Scannen . . . . .	46
5.4.5	Abrufen von Reports . . . . .	48
<b>6</b>	<b>Fazit</b>	<b>49</b>
6.1	Zusammenfassung der Ergebnisse . . . . .	49
6.2	Bewertung des Ergebnisses . . . . .	49
6.3	Ausblick . . . . .	50
	<b>Literaturverzeichnis</b>	<b>51</b>
<b>A</b>	<b>Anhang</b>	<b>55</b>
A.1	Liste der verwendeten regulären Ausdrücke für Geheimniskandidatener- kennung . . . . .	55
	<b>Selbstständigkeitserklärung</b>	<b>57</b>

# Abbildungsverzeichnis

2.1	Äußere Struktur und Interaktionsmöglichkeiten des Branching Modells von Driessen <a href="https://nvie.com/posts/a-successful-git-branching-model/">https://nvie.com/posts/a-successful-git-branching-model/</a> .....	6
2.2	Beziehung zwischen develop-Branch und master-Branch <a href="https://nvie.com/posts/a-successful-git-branching-model/">https://nvie.com/posts/a-successful-git-branching-model/</a> .....	7
2.3	Beziehung zwischen develop-Branch und feature-Branches <a href="https://nvie.com/posts/a-successful-git-branching-model/">https://nvie.com/posts/a-successful-git-branching-model/</a> .....	8
2.4	Beziehung zwischen hotfix-Branch, master-Branch und develop-Branch <a href="https://nvie.com/posts/a-successful-git-branching-model/">https://nvie.com/posts/a-successful-git-branching-model/</a> .....	9
2.5	Weltkarte der Verteilung von CNAs (Stand: 21.11.2019) <a href="https://cve.mitre.org/cve/cna.html">https://cve.mitre.org/cve/cna.html</a> .....	22
3.1	Kontextsicht des reposer Systems (Eigene Darstellung) .....	28
3.2	Bausteinsicht des reposer Systems .....	29
3.3	Verteilungssicht des reposer Systems .....	30
4.1	Auszug der wichtigsten verwendeten Datenmodelle des Systems .....	38
4.2	Beispiel einer .reposecignore Datei .....	39
5.1	Integration des Systems in den agilen Entwicklungsprozess .....	42
5.2	Bereinigen der Commit Historie .....	43
5.3	Der “Public SSH Key” des Systems .....	45
5.4	Manuelles Scannen .....	46
5.5	Bereich API - Schnittstelle für den Github Webhook .....	47
5.6	Bereich Reports - Beispiel eines Reports .....	48



A.1 Benutzte Reguläre Ausdrücke im reposed System) <https://github.com/dxa4481/truffleHogRegexes/blob/master/truffleHogRegexes/regexes.json> . . . . . 56

# Tabellenverzeichnis

2.1	Syntax von regulären Ausdrücke in der Programmiersprache Python, vgl. [w3schools, 2019] . . . . .	15
-----	---	----

# 1 Einleitung

Der Softwareentwicklungsprozess wurde im Laufe der Zeit mehr und mehr optimiert. Es haben sich von Zeit zu Zeit immer neue Prozessmodelle etabliert. Aktuell ist agiles Projektmanagement laut dem Gartner Hype Cycle auf dem Höhepunkt seiner Beliebtheit [Schoen, 2018] und als Folge daraus erfreut sich agile Softwareentwicklung großer Popularität. Wichtig ist es, dass bei schnelllebigen dynamischen Prozessen die Softwaresicherheit nicht vernachlässigt wird. Trotzdem gibt es mehrere Gründe, weshalb die Sicherheit dennoch zum Teil vernachlässigt wird. Zum Beispiel sehen agile Projektvorgehensweisen wie “Scrum” in ihrer Grundform keine initiale detaillierte Risiko- und Bedrohungsanalyse vor [Pohl und Hof, 2015]. Des Weiteren zwingen Zeit- und Kostendruck Softwareprojekte dazu, an der Qualität zu sparen, denn sie stehen in direkter Konkurrenz zueinander und häufig müssen daher Abstriche gemacht werden [Schwalbe, 2015]. Um an der Qualität zu sparen, bietet sich dann der Sicherheitsaspekt an, da dieses Qualitätsmerkmal für den Kunden nicht immer direkt sichtbar ist. Dass Sicherheitslücken bezogen auf den Softwareentwicklungsprozess ein Problem darstellen, zeigt sich in der aktuellen OWASP Liste der 10 kritischsten Sicherheitsrisiken in der Webentwicklung. Auf Platz 9 befindet sich beispielsweise das Risiko durch die Nutzung von Komponenten mit bekannten Schwachstellen [OWASP Foundation, 2019]. Abhängigkeiten können Schwachstellen herbeiführen und “Moderne Webanwendungen verwenden in der Regel zahlreiche OpenSource-Bibliotheken (APIs) und -Frameworks in Form externer Abhängigkeiten (sogenannte ‘3rd-Party-Dependencies’). In den überwiegenden Fällen handelt es sich dabei um OpenSource-Komponenten.” [Rohr, 2018]. Gerade bei OpenSource-Abhängigkeiten liegt die Verantwortung dann beim Entwickler, der diese Abhängigkeit nutzen will, zu überprüfen, ob er diese sorgenfrei benutzen kann. Durch die zunehmende Nutzung solcher Abhängigkeiten, wird dies jedoch ohne adäquate Hilfsmittel schnell zur Mammutaufgabe, welche durch den angesprochenen Kosten- und Zeitdruck noch schwieriger zu bewältigen ist.

Ein weiteres Problem, welches durch zu wenig Sorgfalt entstehen kann, ist dass sensible Zugangsdaten in Klartext in Quelltexten oder Konfigurationsdateien geschrieben

und veröffentlicht werden. In einer Studie haben MELI ET AL. über einen Zeitraum von 6 Monaten über 200.000 solcher Informationen aus Repositories, die öffentlich auf der Plattform “github.com” gehostet sind, ausgelesen [Meli et al., 2019]. In einem Blogbeitrag der Firma github heißt es außerdem, wird ein Geheimnis versehentlich über ein Software Repository veröffentlicht, sollte man es als kompromittiert betrachten [Doll, 2013]. Diese Beobachtung ist natürlich nicht direkt auf den agilen Softwareprozess zurückzuführen, da nicht bekannt ist, wie diese betrachteten Projekte vorgehen. Trotzdem lohnt es sich anzuschauen, wie man von Anfang an dieses Risiko abwenden kann.

Die beiden dargestellten Probleme haben gemeinsam, dass sie schnell entstehen, wenn entweder der Raum für Sorgfalt nicht gegeben ist, bzw. ihnen keine Priorität gegeben werden, oder sie schlichtweg ignoriert werden. Es bedarf also strukturiertes Vorgehen und Hilfsmittel, damit nicht viele Ressourcen für das Verringern dieser Risiken benötigt werden.

### 1.1 Ziel der Arbeit

Die vorliegende Arbeit beschäftigt sich mit der Konzeption und Entwicklung eines Werkzeuges für den agilen Software Entwicklungsprozess, welches eine automatische Sicherheitsprüfung von Softwarerepositories durchführt. Bei dieser Überprüfung werden benutzte Bibliotheken von Drittanbietern gegen Schwachstellendatenbanken geprüft. Das Werkzeug sucht außerdem nach Klartext Geheimnissen, die im Quelltext sichtbar sind. Das Ziel ist es, dass das Werkzeug so gut in den agilen Softwareentwicklungsprozess integriert ist, dass es im Betrieb kaum Ressourcen in Form von Zeit in Anspruch nimmt und trotzdem die in der Einleitung besprochenen Risiken mindert. Dabei ist wichtig abzugrenzen, dass das Werkzeug keine Sicherheitsgarantie liefert bezogen auf die genannten Risiken, da es nur bereits bekannte Schwachstellen in Abhängigkeiten finden kann und auch nicht garantieren kann, dass es jedes Klartext Geheimnis erkennen wird.

### 1.2 Zielgruppe der Arbeit

Zielgruppe der Arbeit sind EntwicklerInnen, ProjektmanagerInnen und andere Beteiligte an Softwareprojekten sowie Interessierte der Bereiche Softwareengineering und IT Sicherheit. Grundlegendes Wissen über IT Sicherheit, Webentwicklung und agile Softwareent-

wicklung ist von Vorteil, aber keine Voraussetzung. Spezifische Begriffe und Konzepte werden an geeigneter Stelle erklärt.

### 1.3 Struktur der Arbeit

Im ersten Kapitel wurde das zugrunde liegende Problem innerhalb der modernen Softwareentwicklung dargestellt, das Ziel der Arbeit festgelegt und die Zielgruppe definiert. In Kapitel 2 werden grundlegende Begriffe und Werkzeuge aus den Bereichen Softwaretechnik und Informationstheorie erklärt, welche für das Verständnis der Arbeit benötigt werden. Auf diesen Grundlagen aufbauend wird in Kapitel 3 das Systemkonzept erstellt. Anforderungen werden abgesteckt und Architektursichten erstellt. Darauf folgt in Kapitel 4 die Beschreibung des Entwicklungsprozesses. In diesem Abschnitt wird auf verwendete Technologien und Anpassungen von diesen eingegangen sowie die strukturelle Erarbeitung der umgesetzten Konzepte dargestellt. In Kapitel 5 wird der agile Softwareentwicklungsprozess aufgegriffen und festgelegt, wie sich das in der Arbeit entstandene Werkzeug in diesen integriert, um das Sicherheitslevel zu erhöhen. Abschließend wird in Kapitel 6 ein Fazit über das Gesamtergebnis gezogen und betrachtet, inwieweit mit dem Ergebnis weiter gearbeitet werden kann.

## 2 Grundlagen

In diesem Kapitel wird zuerst grundlegend anhand von Konzepten und Begriffen der Aufbau von Git Repositories erläutert, welche im Rahmen dieser Arbeit nach Geheimnissen und Schwachstellen durchsucht werden. Danach werden verschiedene Mechanismen um Geheimnisse und Schwachstellen in Git Repositories zu finden dargestellt sowie Grundbegriffe hierfür definiert. Im letzten Teil der Grundlagen werden Metriken erklärt, die genutzt werden, um die erkannten Schwachstellen bewerten zu können.

### 2.1 Git

Git ist ein Werkzeug für das verteilte Arbeiten an Softwareprojekten. Es wurde ursprünglich von der Open Source Linux Gemeinschaft entwickelt, allen voran Linus Torvalds. Nachdem die Gemeinschaft 2005 aufgrund von Vertragsunstimmigkeiten mit dem Anbieter einer vorher eingesetzten Lösung diese nicht mehr einsetzen wollte, wurde eine neue Lösung für das verteilte Arbeiten an dem Linux Projekt benötigt. Es entstand die Software Git, welche heutzutage aus dem Arbeitsleben vieler SoftwareentwicklerInnen nicht mehr wegzudenken ist [Chacon und Straub, 2019a].

#### 2.1.1 Funktionsweise von Git

Git besteht im Grunde aus vier verschiedenen Objekttypen:

- blob - Enthält Inhalte einer Datei
- tree - Enthält Referenzen zu anderen tree Objekten und blob Objekten, ist vergleichbar mit einem Dateiverzeichnis
- commit - Verbindet den Zustand eines tree Objekts mit der Information, wie es zu diesem Zustand gekommen ist.

- tag - Enthält Metadaten und eine Referenz zu einem anderen Objekt, meistens zu einem commit Objekt

Mit dieser Umgebung ist es nun möglich, verteilt, nicht-linear an einem Projekt zu entwickeln und es zu versionieren. Der/die EntwicklerIn kann auf einer lokalen Version des Git Repositorys Commit-Objekte erstellen, die er/sie zu jedem Zeitpunkt mit einem anderen Git Repository synchronisieren kann. Desweiteren kann er/sie Kopien der Tree Objekte erstellen und diese auch lokal, bzw. zentral synchronisieren. Dieser Vorgang wird “Branching” genannt. Das Zurückgehen auf frühere Versionen ist damit auch möglich. Ein weiteres nützliches Feature ist u.a., dass das Identifikationsmerkmal von einem Commit-Objekt ein Hashwert ist, der auf der gesamten Revisionsgeschichte dieser Revision basiert, was das nachträgliche Fälschen der Projekthistorie erschwert (vgl. [Chacon und Straub, 2019b]). Daraus folgt, dass das nachträgliche Verändern der Revisionsgeschichte nur möglich ist, wenn man sie komplett neu aufbaut. Dieser Punkt ist besonders wichtig für den Verlauf der Arbeit, da die Geheimnisse, die gesucht werden Teil dieser Revisionsgeschichte sind.

Ein System, welches die Inhalte von Git Repositories durchsucht, sollte daher auch nach Möglichkeit alle Branches und Commits durchsuchen.

### 2.1.2 Nutzung von Git in der Softwareentwicklung

In der modernen Softwareentwicklung ist Git, aufgrund seiner dezentralen Struktur und der Möglichkeit sehr feingranular zu versionieren, ein gern genutztes Werkzeug. Der Einsatz von Git löst jedoch nicht alleine die Probleme und Anforderungen, die in einem Softwareprojekt auftreten. Vielmehr ist es wichtig, dieses Werkzeug strukturiert einzusetzen. Einen Ansatz für die strukturierte Verwendung von Git liefert das verbreitete “Driessens Branching Modell”.

### 2.1.3 Branching Modell nach Vincent Driessen

Das Branching Modell von Driessen, welches sich sehr gut für die Zusammenarbeit in Projekten sowie die Skalierung dieser eignet, wurde 2010 in einem Blog Beitrag [Driessen, 2010] veröffentlicht und wird im Folgenden erläutert. Anhand dieses Modells kann gesehen werden, wie ein Arbeitsablauf eines/einer EntwicklerIn aus der Perspektive des Quelltextes aussieht. Mit dieser Information kann später eingeordnet werden an, welchen

Stellen ein System, welches statisch den Quelltext innerhalb eines Softwarepositories überprüft, zum Einsatz kommen kann.

### Äußere Struktur

Das Modell beschreibt im Kern trotz der dezentralen Natur von Git ein zentrales Repository, welches die “Wahrheit” enthält und wird von DRIESSEN als “origin” bezeichnet. Jede/r EntwicklerIn “pushed” und “pulled” von diesem Repository. Das bedeutet in diesem Zusammenhang, dass lokale Änderungen der EntwicklerInnen nach origin synchronisiert werden (“Push”) und Änderungen von origin beim lokalen Stand der EntwicklerInnen integriert werden (“Pull”), jedoch können innerhalb von Unterteams auch EntwicklerInnen Änderungen von sich gegenseitig pullen.

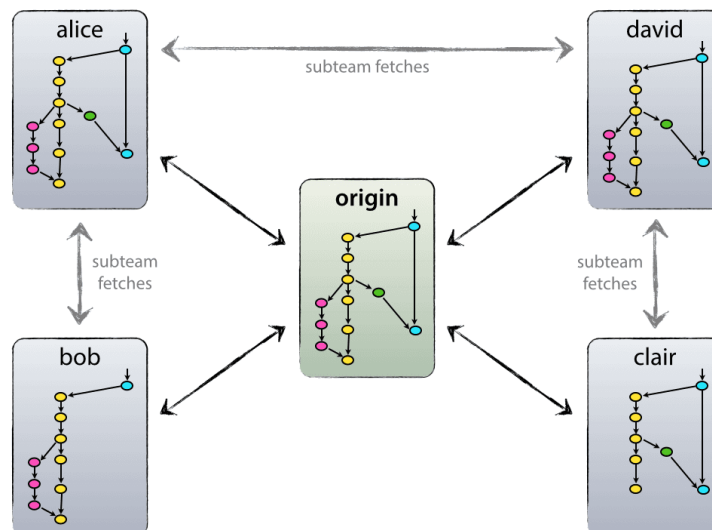


Abbildung 2.1: Äußere Struktur und Interaktionsmöglichkeiten des Branching Modells von Driessen

<https://nvie.com/posts/a-successful-git-branching-model/>

### Typen von Branches

Die beiden Haupt-Branches, die das Modell beschreibt, sind einmal der “master”-Branch und der “develop”-Branch. Der “master”-Branch enthält immer einen Stand, der als Pro-



duktivsystem geeignet ist. Der “develop”-Branch enthält die aktuellen Änderungen, die in das nächste Release kommen sollen.

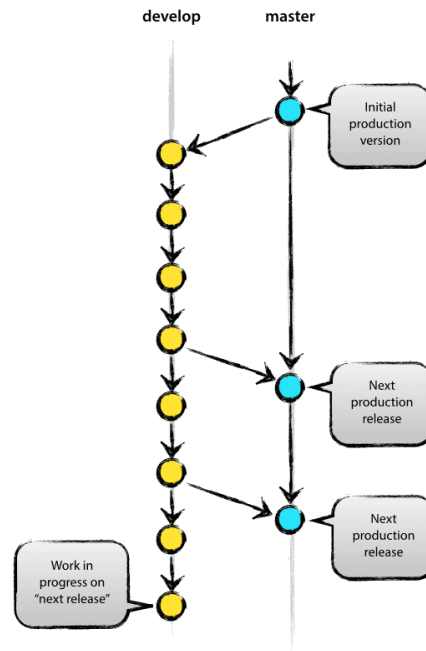


Abbildung 2.2: Beziehung zwischen develop-Branch und master-Branch

<https://nvie.com/posts/a-successful-git-branching-model/>

Darüber hinaus gibt es nun drei weitere Typen von Branches, die den verteilten Entwicklungsprozess, das Verfolgen von Features, das Vorbereiten von Produktivsystemen sowie das schnelle Lösen von Fehlern einer Produktivumgebung organisieren. Zum einen gibt es die “feature”-Branches, die benutzt werden um abgegrenzte Funktionen für ein zukünftiges Release zu entwickeln. Diese basieren auf dem “develop”-Branch.

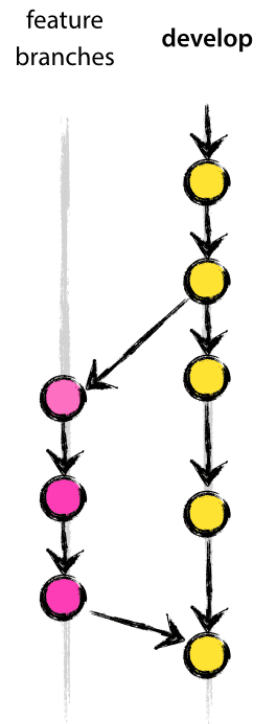


Abbildung 2.3: Beziehung zwischen develop-Branch und feature-Branches

<https://nvie.com/posts/a-successful-git-branching-model/>

Zum anderen gibt es den “release”-Branch, welcher vor dem Übergang in die Produktivumgebung erstellt wird und auf dem “develop”-Branch basiert. Zu dem Zeitpunkt an dem der “release”-Branch erstellt wird, sollten alle geplanten Features für das Release in den “develop”-Branch integriert worden sein. Auf dem nun erstellten “release”-Branch können nun letzte kleine Änderungen und Fehlerbehebungen vorgenommen werden, bevor es in die Produktivumgebung also dem “master”-Branch geht.

Zudem gibt es den “hotfix”-Branch. Dieser wird benutzt, wenn es dringende Fehler in der Produktivumgebung gibt, die schnell behoben werden müssen. Der “hotfix”-Branch basiert auf dem “master”-Branch. Innerhalb des “hotfix”-Branches wird nun der Fehler behoben. Danach wird der “hotfix”-Branch in den “master”-Branch integriert sowie in aktuelle “release”-Branches und dem “develop”-Branch.

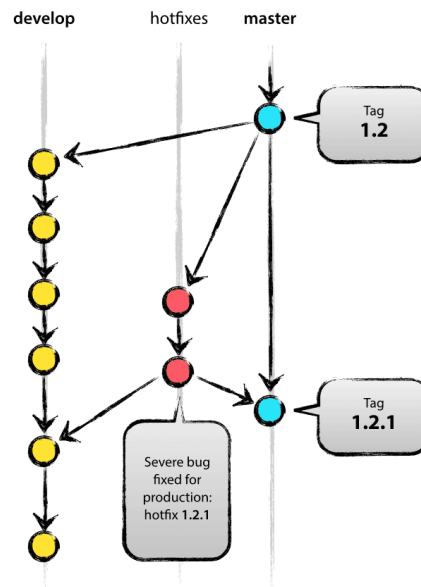


Abbildung 2.4: Beziehung zwischen hotfix-Branch, master-Branch und develop-Branch  
<https://nvie.com/posts/a-successful-git-branching-model/>

### 2.1.4 Moderne Funktionen von Git Systemen

Viele Anbieter wie "Github", "Gitlab" oder "Atlassian Bitbucket", die Git Repositories zur Verfügung stellen, liefern heutzutage noch zusätzliche Funktionen, die über das Kernsystem Git hinausgehen. Im Folgendem werden zwei Funktionen erläutert, die von dem in dieser Arbeit entwickeltem System genutzt werden.

#### Webhook

Ein Webhook ist generell eine Methode in der Web Entwicklung, um das Verhalten einer Webapplikation über Rückruffunktionen (engl: callbacks) zu verändern bzw. zu beeinflussen. Konkret im Zusammenhang mit Git Repositories werden Webhooks genutzt, um eine Rückruffunktion zu aktivieren, sobald Ereignisse im Zusammenhang mit dem Git Repository eintreten. Welche Ereignisse das sind, kann über die Systeme der Anbieter gesteuert werden. Beispiele für Ereignisse sind das Synchronisieren eines Commits, oder das Erstellen eines "Pull Requests". Die Software, die das Git Repository verwaltet, würde bei entsprechender Konfiguration nun die Rückruffunktion teilweise mit entsprechenden

Parametern aufrufen. Die Rückruffunktion ist im Kontext der Web Entwicklung häufig eine HTTP POST Schnittstelle. Da in der Arbeit das Ziel verfolgt wird, möglichst den ‘normalen’ Arbeitsfluss des Entwickelnden nicht zu verändern und trotzdem das Sicherheitslevel zu erhöhen, wird das Konzept des Webhooks als fester Bestandteil des Systems gesehen. Obwohl das System am Ende auch manuell benutzt werden kann, ist es trotzdem wichtig für eine gute Nutzungserfahrung eine Möglichkeit zu haben Webhooks einzusetzen.

### **Pull Request**

Ein Ereignis im Softwareentwicklungsprozess, um oben beschriebene Webhooks auszulösen ist der Pull Request, welcher je nach Anbieter auch Merge Request heißen kann. Ein Pull Request findet in dem Driessens Branching Modell immer dann statt, wenn eine fertiggestellte Änderung einen Schritt weiter Richtung master-Branch also “production“-Branch gemerged werden muss. Zum Beispiel ist dies der Fall, wenn ein feature-Branch fertiggestellt wurde und dieser in den develop-branch gemerged werden soll. Der/die EntwicklerInnen des Features kann nun bevor gemerged wird, einen “Pull Request” erstellen, also eine Anfrage, dass der develop-Branch die Änderungen des feature-Branch pulled. Diese Anfrage, muss nun je nach vereinbartem Prozess von einem/einer anderen EntwicklerIn überprüft und akzeptiert werden. Die meisten Anbieter bieten eine automatische Merge Funktion an, um das Feature zu integrieren. Ist dies nicht der Fall oder ist es nicht möglich, das Feature automatisch zu mergen, so müssen beteiligte EntwicklerInnen manuell den feature-Branch mergen [Github, 2019]. Genau an dem Punkt soll das für die Arbeit entwickelte System den überprüfenden EntwicklerInnen automatisch mit Zusatzinformationen über eventuelle Sicherheitsprobleme informieren, so dass sie in die Evaluierung des Pull Requests mit einfließen und Schwachstellen und Enthüllungen von Geheimnissen so schnell wie möglich abgewendet bzw. ungeschehen gemacht werden können.

## **2.2 Geheimniserkennung**

In diesem Abschnitt werden Begriffe, Konzepte und Technologien erklärt, die für das Aufspüren von Geheimnissen in Quelltexten benötigt werden.

### 2.2.1 Geheimnis

Ein Geheimnis im Kontext der Informationstechnologie ist ein kryptografischer Schlüssel oder ein Zugangspasswort zu einer Systemschnittstelle, welches aus Sicherheitsgründen geheim gehalten werden muss vgl. [Meli et al., 2019]. Ein Geheimnis sollte nur von einer Person lesbar sein, die entsprechend dazu berechtigt ist, dieses Geheimnis zu sehen bzw. zu verwalten.

#### Problem in der Praxis

Trotz der Vertraulichkeit von Geheimnissen kommt es immer wieder vor, dass sie sich in Klartext geschrieben in Umgebungen wieder finden, auf die auch Personen Zugriff haben, die eigentlich das Geheimnis nicht sehen dürften. Neben dem offensichtlichen Problem, dass ein Geheimnis den Zugang zu vertraulichen Daten ermöglichen könnte, gibt es auch API Token, die zum Beispiel dazu genutzt werden könnten kostenpflichtige Dienste bzw. Schnittstellen für Unbefugte nutzbar zu machen.

Wie diese Geheimnisse in großen Datenmengen auffindig gemacht werden, wird im folgenden erläutert.

### 2.2.2 Erkennung potentieller Geheimnisse

Die Erkennung von potentiellen Geheimnissen oder auch das Identifizieren von Geheimniskandidaten beschreibt den Schritt während des Geheimniserkennungsprozesses, in welchem Zeichenketten, die potentiell ein Geheimnis sein könnten aggregiert werden. Hierfür werden in dieser Arbeit zwei Prinzipien verwendet. Das erste dieser Prinzipien ist die sogenannte "Shannon Entropie".

### 2.2.3 Shannon-Entropie

Die Shannon Entropie, benannt nach ihrem Entdecker dem amerikanischen Mathematiker Claude Elwood Shannon, beschreibt den Informationsgehalt einer Nachricht innerhalb eines Zufallsexperimentes und wird mathematisch wie von FROCHTE in folgender Weise beschrieben:

“Die Entropie  $H$  eines Zufallsexperimentes  $V$  mit den möglichen Ausgängen  $A_1 \dots A_k$  ist definiert als

$$- \sum_{i=1}^k P(A_i) \cdot \log_2(P(A_i))$$

” [Frochte, 2019]

Um dieses Konzept nachzuvollziehen, stelle man sich vor, es gäbe eine zufällige diskrete Variable  $x$  und man frage sich, wie viel Information man erhält, wenn diese Variable einen bestimmten Wert annimmt. Je “überraschender”, also je unwahrscheinlicher der Wert ist, den die Variable annimmt, desto mehr Informationen hat man erhalten, desto mehr Informationsgehalt liefert der Wert, da man ihn eher nicht erwartet hätte. Im Grunde genommen wird die Wahrscheinlichkeit betrachtet, dass ein bestimmtes Ereignis eintritt. Je geringer die Wahrscheinlichkeit, desto mehr Informationen hat man erhalten. Diese Wahrscheinlichkeit wäre jedoch einfach das Produkt der Einzelwahrscheinlichkeiten:

$$\prod_{i=1}^k P(A_i)$$

Man kann sich vorstellen, dass wenn man dieses Produkt anhand von Texten mit zum Beispiel 101 Buchstaben betrachten würde, dieses Produkt aus 101 Einzelwahrscheinlichkeiten astronomisch gering werden würde, was nicht nur für Maschinen die Berechnung verkompliziert. SHANNON benutzte in seinem Werk aber den negativen Durchschnitt der logarithmischen Einzelwahrscheinlichkeiten, weil er beweisen konnte, dass das resultierende Ergebnis die durchschnittliche Anzahl an Ja-Nein Fragen ist, die benötigt werden, um den Wert der Variable zu erraten bzw. bei binärer Betrachtung, wie viele Bits durchschnittlich benötigt werden, um diese Information zu übermitteln. Dieses Konzept wird im Folgendem anhand eines Beispiels veranschaulicht.

Sei unser Zufallsexperiment ein Münzwurf mit zwei möglichen Ergebnissen  $A_1, A_2$  (Kopf/Zahl) mit einer Wahrscheinlichkeit von  $P(A_{1,2}) = 0,5$ , dann erhalten wir folgende Entropie  $H$ :

$$H = - \sum_{i=1}^2 P(A_i) \cdot \log_2(P(A_i)) = -2 \cdot (0,5 \log_2(0,5)) = 1$$

Für diese Art von Münzwurf benötigen wir nur eine Frage, bzw. ein Bit, um die Information zu übermitteln, nämlich “Ist das Ergebnis Kopf?”, bzw.  $0 = \text{Zahl}, 1 = \text{Kopf}$ .

Betrachten wir nun ein Spiel, in dem zwei Spieler jeweils eine Münze werfen und der Spieler der Kopf wirft gewinnt. Wenn beide Spieler Kopf/Zahl werfen ist das Spiel unentschieden. Die Wahrscheinlichkeit für ein Unentschieden liegt bei  $P(A_3) = 0.5$  und die Wahrscheinlichkeit für einen Sieg liegt jeweils bei  $P(A_{1,2} = 0, 25)$

$$H = -0,25 \log_2(0,25) - 0,25 \log_2(0,25) - 0,5 \log_2(0,5) = 1,5$$

Wenn man geschickt fragt, stellt man als erste Frage “Ist das Ergebnis unentschieden?” und als zweite Frage “Hat Spieler 1 gewonnen?” dann ergibt sich, dass in der Hälfte der Fälle nur eine Frage gestellt wird ( $P(A_3) = 0.5$ ) und in der anderen der Hälfte der Fälle beide Fragen gestellt werden, was einen Durchschnitt von 1.5 Fragen ergibt.

### Entropie der englischen Sprache

Das vorangegangene Konzept der Entropie hat SHANNON in seinem Artikel auf die englische Sprache angewendet. Er kam zu dem Ergebnis, dass die englische Sprache nur eine Entropie von etwas mehr als 1 hat bei Sequenzen von 100 Buchstaben bzw. bei Sequenzen mit bis zu 8 Zeichen eine Entropie von 2,62. SHANNON führt dies auf eine hohe Redundanz, also auf die Größe der Einschränkung auf einen Text durch die statistische Struktur der Sprache zurück. Er nennt hierfür das Beispiel, dass der Buchstabe ‘E’ sehr häufig vorkommt und das ‘H’ häufig auf ‘T’ folgt und das ‘U’ auf den Buchstaben ‘Q’ folgt [Shannon, 1951]. Das ist besonders hilfreich für diese Arbeit, da wir die Annahme treffen können, dass eine rein zufällige Folge von Buchstaben und Zahlen häufig eine höhere Entropie haben wird als eine gleich lange Sequenz englischer Sprache. Da die englische Sprache in Form von Schlüsselwörtern einerseits fest in Programmiersprachen integriert ist, und andererseits in der Softwareentwicklung verbreitet ist, nutzt diese Arbeit diese Annahme, um Geheimniskandidaten zu entdecken, da Geheimnisse in sehr vielen Fällen eine Folge von zufallsgenerierten Zahlen und Buchstaben enthalten.

Teilweise haben die Geheimnisse, die gesucht werden, trotz zufälliger Zahlen und Buchstaben eine bekannte Struktur, da ihre Generierung immer auf die gleiche Art und Weise nach bestimmten Regeln passiert. Diese bergen die Gefahr, eine niedrigere Entropie zu haben. Dieses Wissen über Struktur können wir uns nicht mit der Shannon Entropie zu

Nutze machen. Im Gegenteil, die Abfolge der Zeichen ist nun weniger zufällig. Daher wird im Folgenden noch ein zweites Prinzip erläutert, welches genutzt wird, um Geheimnis-kandidaten mit bekannter Struktur zu entdecken, nämlich das Abgleichen mit regulären Ausdrücken.

### 2.2.4 Regulärer Ausdruck

Unter einem regulären Ausdruck versteht man eine formale Sprache zur Beschreibung von Zeichenketten. Sie kann unter anderem dazu genutzt werden, Zeichenketten zu filtern, zu durchsuchen und zu verändern. Reguläre Ausdrücke haben die Mächtigkeit eines endlichen Automaten und haben eine zentrale Bedeutung für viele Anwendungen der Textverarbeitung. Desweiteren gehören sie zur Sprachdefinition bzw. zum Funktionsumfang von nahezu allen heutzutage verbreiteten Programmiersprachen. Hierbei ist zu beachten, dass die verschiedenen Sprachen teilweise eine leicht unterschiedliche Syntax für reguläre Ausdrücke haben vgl. [Fischer und Hofer, 2010]. Da die hier in der Arbeit erstellte Software die Sprache Python benutzt, folgt eine tabellarische Übersicht der Syntax für reguläre Ausdrücke in Python:

Tabelle 2.1: Syntax von regulären Ausdrücken in der Programmiersprache Python

Character	Description
<b>Metacharacters</b>	
<code>[]</code>	A set of characters
<code>\</code>	Signals a special sequence (can also be used to escape special characters)
<code>.</code>	Any character (except newline character)
<code>^</code>	Starts with
<code>\$</code>	Ends with
<code>*</code>	Zero or more occurrences
<code>+</code>	One or more occurrences
<code>{}</code>	Exactly the specified number of occurrences
<code> </code>	Either or
<code>()</code>	Capture and group
<b>Special Sequences</b>	
<code>\A</code>	Returns a match if the specified characters are at the beginning of the string



<code>\b</code>	Returns a match where the specified characters are at the beginning or at the end of a word
<code>\B</code>	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word
<code>\d</code>	Returns a match where the string contains digits (numbers from 0-9)
<code>\D</code>	Returns a match where the string DOES NOT contain digits
<code>\s</code>	Returns a match where the string contains a white space character
<code>\S</code>	Returns a match where the string DOES NOT contain a white space character
<code>\w</code>	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore <code>_</code> character)
<code>\W</code>	Returns a match where the string DOES NOT contain any word characters
<code>\Z</code>	Returns a match if the specified characters are at the end of the string
<b>Sets</b>	
<code>[arn]</code>	Returns a match where one of the specified characters (a, r, or n) are present
<code>[a-n]</code>	Returns a match for any lower case character, alphabetically between a and n
<code>[^arn]</code>	Returns a match for any character EXCEPT a, r, and n
<code>[0123]</code>	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
<code>[0-9]</code>	Returns a match for any digit between 0 and 9
<code>[0-5][0-9]</code>	Returns a match for any two-digit numbers from 00 and 59
<code>[a-zA-Z]</code>	Returns a match for any character alphabetically between a and z, lower case OR upper case
<code>[+]</code>	In sets, <code>+</code> , <code>*</code> , <code>.</code> , <code> </code> , <code>()</code> , <code>\$</code> , <code>{}</code> has no special meaning, so <code>[+]</code> means: return a match for any <code>+</code> character in the string

Tabelle 2.1: Syntax von regulären Ausdrücke in der Programmiersprache Python, vgl. [w3schools, 2019]

Diese formale Sprache kann nun genutzt werden, um Strukturen zu beschreiben, die bestimmte Schlüssel haben.

### Beispiel: Google API Key

Google API Token, die zur Authentisierung für viele Services von Google für EntwicklerInnen genutzt werden, starten laut der eigenen Aussage von GOOGLE meistens mit der Buchstabenkombination “AIza” [Google, 2019d]. Mit dieser Information kann man folgenden regulären Ausdruck bilden:

$$GoogleTokenRegex = AIza.+$$

oder wenn die Tokenlänge z.B. 39 wäre:

$$GoogleTokenRegex = AIza.{35}$$

Der erste reguläre Ausdruck beschreibt, dass Zeichenketten, die dieser Struktur entsprechen, mit der Buchstabenkombination “AIza” beginnen müssen, gefolgt von mindestens einem beliebigen Zeichen bzw. beim zweiten Ausdruck gefolgt von 35 beliebigen Zeichen.

Je nachdem wie spezifisch die genutzten regulären Ausdrücke sind, kann man mit ziemlicher Sicherheit sagen, dass eine Zeichenkette, die einem solchen regulären Ausdruck entspricht, ein Geheimnis ist. Bei den Zeichenketten, die durch Entropie gefunden wurden, ist es jedoch sinnvoll, diese noch weiter zu untersuchen, da sie nur aufgrund der Tatsache entdeckt wurden, dass sie möglicherweise eine zufällige Aneinanderreihung von Buchstaben sind. Techniken hierfür werden im Folgenden erläutert

### 2.2.5 Verifizierung potentieller Geheimnisse

Bei der Verifizierung werden verschiedene Arten von Filtern auf potentielle Geheimnisse angewendet, um Geheimniskandidaten, die keine Geheimnisse sind, herauszufiltern. Diese Filter müssen genauestens durchdacht sein, denn nur ein einziger zu starker Filter kann dafür sorgen, dass Geheimnisse nicht erkannt werden (“False Negatives”). Zu schwache Filter können dazu führen, dass Zeichenketten erkannt werden, die gar keine Geheimnisse sind (“False Positives”). Im Folgendem werden die für die Arbeit wichtigen Filterprinzipien erklärt

### **Wörter Filter**

Der Wörter Filter beseitigt Geheimniskandidaten, die ein englisches Wort ab einer gewissen Länge enthalten, denn die gesuchten Geheimnisse sollten englische Wörter nur rein zufällig enthalten. Wenn ein Geheimnis aus Zufall ein englisches Wort enthält, dann wird dieses als “False Negative” weggefiltert, jedoch ist dies ab einer gewissen Wortlänge für englische Begriffe, die durch den Filter angewendet werden sehr unwahrscheinlich. Wenn die Länge für die betrachteten Wörter zu lang ist, werden wiederum vermehrt “False Positive” Geheimnisse erkannt. Das zu Grunde liegende Wörterbuch wird wie bei MELI ET AL. aus den am meisten verwendeten Wörtern in Programmen auf Github [Kashcha, 2018] gebildet.

### **Struktureller Filter**

Ein weiterer Filter, der angewendet wird, ist ein Zeichenkettenstrukturfilter. Dieser beseitigt logische Strukturen in Zeichenketten, wie das Herauf- und Herabzählen von Buchstaben wie z.B. “ABCDEFGH” oder “DCBA”. Des Weiteren werden Buchstaben, die sie sich zu oft wiederholen, weggefiltert, wie zum Beispiel “XXXXXXXX”. Die Nachforschungen von MELI ET AL. haben ergeben, dass solche Zeichenketten häufig auf Testeingaben basieren. Trotzdem ist auch hier wieder das Gleichgewicht zwischen False-Positive und Wrong-Negative zu beachten. Laut MELI ET AL. eignet sich hier eine Zeichenkettenlänge von mindestens 5.

### **Software: truffleHog**

Eine Software, die die beiden zuvor erläuterten Prinzipien zur Geheimniserkennung in Git Repositories implementiert, ist die Open Source Software “truffleHog” entwickelt von Dylan Ayrey [Ayrey, 2018]. Diese nach GPL-2.0 veröffentlichte Software wurde jedoch zuletzt durch MELI ET AL. kritisiert, da sie die Suchergebnisse nicht validiert. Vielmehr bezeichnen MELI ET AL. die Software als “ineffektiv”, da sie in ihren Tests eine Vielzahl von False-Positives lieferte [Meli et al., 2019].

Trotzdem bildet truffleHog die Grundlage der in dieser Arbeit entwickelten Geheimniserkennung, da die verwendeten Grundprinzipien tendenziell gute Ergebnisse liefern können. Es werden zusätzlich jedoch die genannten Prinzipien zur Geheimnisverifizierung angewendet.

## 2.3 Überprüfung von Softwareabhängigkeiten

Der zweite Teil der statischen Sicherheitsanalyse welche das in dieser Arbeit entwickelte System durchführt, bezieht sich auf Softwareabhängigkeiten von Anwendungen. Grundbegriffe und Mechanismen werden im folgenden Abschnitt erläutert.

### 2.3.1 Softwareabhängigkeit

Eine Softwareabhängigkeit ist eine Art der Kopplung in der Softwareentwicklung. Es gibt sowohl externe als auch interne Softwareabhängigkeiten. Die externen Abhängigkeiten, die in dieser Arbeit im Fokus stehen, sind Abhängigkeiten zu externen Bibliotheken, die benötigt werden, um die eigene Software zu kompilieren und in einer produktionsähnlichen Umgebung auszuführen [Toth, 2019]. Softwareabhängigkeiten entstehen zum Beispiel in Form von Bibliotheken, die in ein Projekt importiert werden, damit bestimmte Funktionalitäten nicht von dem/der EntwicklerIn selbst implementiert werden müssen.

### 2.3.2 Risiken durch Softwareabhängigkeiten

In dem Beispiel der externen Bibliotheken gibt es immer das Risiko, dass der/die UrheberIn dieser Softwareabhängigkeit Fehler gemacht hat, welche dann wiederum zu Schwachstellen oder Verwundbarkeiten in der Software führen können oder sogar der/die UrheberIn bewusst Hintertüren in seine/ihre Software eingebaut hat, um NutzerInnen seiner/ihrer Softwarebibliothek zu kompromittieren. Außerdem gibt es das Risiko, dass der Entwicklungsprozess des/der Entwickelnden der Software kompromittiert wurde und darüber Schadcode in die Software eingeführt wird, welche Hintertüren für AngreiferInnen öffnet. Ein Beispiel hierfür lieferte ein weit verbreitetes Plugin mit dem Namen “jquery-file-upload”.

#### Beispiel jquery-file-upload

Ein Beispiel für eine Sicherheitslücke, die durch verwundbare Abhängigkeiten entstehen kann, gab es Ende 2018 mit einer Bibliothek mit dem Namen “jquery-file-upload”, wenn diese auf Apache-HTTPD-Servern zum Einsatz kam. AngreiferInnen war es möglich, Dateien in beliebige Verzeichnisse auf dem Zielsystem hochzuladen. Somit konnte beliebiger

Schadcode auf den Servern der Opfer ausgeführt werden. Das Problem war, dass die Bibliothek ihre Zugriffsrechte auf Verzeichnisse über die sogenannte `.htaccess` Datei von Apache Servern ermittelte. Eine Änderung in dem Umgang mit diesen Dateien von Apache aktivierte aber standardmäßig eine Option, dass die übernommenen Einstellungen durch diese Dateien ignoriert wurden. Es war über diese Bibliothek nun möglich Dateien in beliebige Verzeichnisse auf dem Apache Server hochzuladen. Der `jquery-file-upload` wurde zu diesem Zeitpunkt auf der Plattform Github über 7800 Mal geforked also kopiert, und möglicherweise in andere Projekte integriert, was für eine weite Verbreitung dieser Software und auch der Verwundbarkeit spricht [Schirmacher, 2018].

Der `jquery-file-upload` ist neben vielen anderen Bibliotheken in einer großen Datenbank erhältlich, welche in dieser Arbeit auch genutzt wird, indem bekannte Verwundbarkeiten von Bibliotheken aus dieser Datenbank an anderer Stelle abgefragt werden. Die besagte Datenbank heißt Node Package Manager Register. Ihr Aufbau wird im Folgendem erläutert.

### 2.3.3 Node Package Manager

Laut eigener Aussage ist der “Node Package Manager” kurz “npm” das weltgrößte Softwareregister. Es besteht grundlegend aus drei Komponenten:

- Die Webseite, die Dokumentationen und Versionsinformationen zu den einzelnen Paketen enthält sowie die Möglichkeit eigene Inhalte zu dem Register beizutragen und zu verwalten.
- Ein Kommandozeilenprogramm (CLI), welches von den meisten Entwicklern zur Interaktion mit dem Node Package Manager benutzt wird.
- Das Register, eine große öffentliche Datenbank, die JavaScript Software und Meta-informationen enthält, also die Bibliotheken.

Der Node Package Manager wird von EntwicklernInnen benutzt um externe Abhängigkeiten, die durch das Register bereitgestellt werden, in die eigene Software zu integrieren. Der Node Package Manager ist die größte Sammlung an Javascript und Typescript Bibliotheken, die es gibt [npm Inc., 2019].

Sind einmal in einem Projekt alle Bibliotheken mitsamt der Versionsnummer entdeckt, gilt es zu überprüfen, ob diese anfällig für bekannte Schwachstellen und Verwundbarkeiten

sind. Hier gibt es auch Datenbanken, die diese Informationen standardisiert pflegen und der Öffentlichkeit bereitstellen. Eine davon ist der OSS Index der Firma sonatype.

### 2.3.4 sonatype OSS Index

Der Sonatype OSS Index ist ein freier Service der Firma sonatype, welcher Informationen über bekannte Schwachstellen oder Verwundbarkeiten von Softwarebibliotheken über eine REST Schnittstelle anbietet. Hierfür werden Informationen über Schwachstellen und Verwundbarkeiten aus öffentlichen Datenbanken benutzt, wie der “NVD” der National Vulnerability Database, welche von der US-amerikanischen Regierung unterhalten wird [Sonatype Inc., 2019].

Da die im Rahmen der Arbeit entwickelte Software eine Webanwendung ist, eignet sich dieser Service sehr gut für die Integration in die Software. Die Schnittstelle nimmt eine Liste von sogenannten Koordinaten entgegen und liefert Verwundbarkeits und Schwachstelleninformationen über diese Koordinaten zurück falls vorhanden. Eine Koordinate ist ein Bezeichner für eine Softwarebibliothek. Sie basiert auf der “Package-URL Specification” [Schuberth et al., 2019]. Eine Koordinate hat folgendes Format:

$$pkg : type/namespace/name@version?qualifiers\#subpath$$

Die einzelnen Bestandteile einer Koordinate haben folgende Bedeutungen:

- type: Der Typ der Komponente zum Beispiel “npm” oder “pypi”) (erforderlich)
- namespace: Ein Namensprefix zum Beispiel ein NPM package scope wie “@angular” (optional)
- name: Der Name der Komponente zum Beispiel “jquery-file-upload” (erforderlich)
- version: Die Version der Komponente zum Beispiel “0.22” (optional)
- qualifiers: Zusatzdaten zu der Komponente wie Betriebssystem, Architektur (optional, wird von OSS Index ignoriert)
- subpath: Extra Subpfad innerhalb der Komponente, relativ zu der Paketquelle (optional, wird von OSS Index ignoriert)

Die Verwundbarkeits- und Schwachstelleninformationen die zum Schluss von OSS Index erhalten werden, enthalten wiederum Informationen und Metriken die inzwischen Teil weit verbreiteter Standards sind. Diese werden im folgendem und letzten Abschnitt der Grundlagen erläutert.

### 2.4 Metriken

Für die Risikobewertung von gefundenen Sicherheitslücken in Softwareabhängigkeiten ist es unabdingbar die Auswirkungen, die Angriffsvektoren und die gefährdeten Güter bzw. Werte einschätzen zu können. Hierfür benötigt man Informationen über die gefundenen Schwachstellen oder Verwundbarkeiten. Bis 1999 gab es keine einheitliche Lösung für die Bewertung und Hersteller haben eigene Metriken erstellt, um über Schwachstellen und Verwundbarkeiten zu informieren bzw. diese einzustufen. Diese Metriken sind nicht untereinander vergleichbar gewesen, was dazu führte, dass sie außerhalb der einzelnen Organisationen schwierig anwendbar waren. Um dieses Problem zu lösen, entstanden ab 1999 u.a. 3 Metriken, die auch im Laufe dieser Arbeit verwendet werden und daher jeweils erläutert werden. Zum einen entstanden die “Common Vulnerabilities and Exposures (CVE)” und das “Common Vulnerability Scoring System” und zum anderen die “Common Weakness Enumeration”. Diese Standards werden durch die US-amerikanische Non-Profit-Organisation “MITRE” und die Organisation “FIRST.Org Inc.” in Zusammenarbeit mit großen Technologiekonzernen und Einrichtungen gepflegt und angeboten. Die Forschungen und Dienste von MITRE und FIRST.Org werden finanziert von der US-amerikanischen Regierung, u.a. dem “U.S. Department of Homeland Security (DHS)”, sowie der “Cybersecurity and Infrastructure Security Agency (CISA)”. Laut eigener Aussage haben sich CVE, CVSS und CWE heutzutage zu Industriestandards entwickelt [MITRE Corporation, 2019a] [MITRE Corporation, 2019b].

#### 2.4.1 CVE und CVSS

Die “Common Vulnerabilities and Exposures” ist eine Liste mit Einträgen über bekannte Verwundbarkeiten in IT-Systemen. Des Weiteren liefert sie einen eindeutigen Bezeichner pro Verwundbarkeit und eine Beschreibung dieser Verwundbarkeit. Diese Liste wird nicht nur durch die MITRE Corporation gepflegt, sondern durch eine Menge von IT Konzernen

und Organisationen, welche als sogenannte “CVE Numbering Authority (CNA)” eingetragene sind. Aktuell (Stand: 21.11.2019) gibt es weltweit 108 CNAs, davon alleine 67 in den USA, 9 in China und 6 in Deutschland. Die relative Verteilung dieser CNAs ist ungefähr in Abbildung 2.5 dargestellt [MITRE Corporation, 2019c].

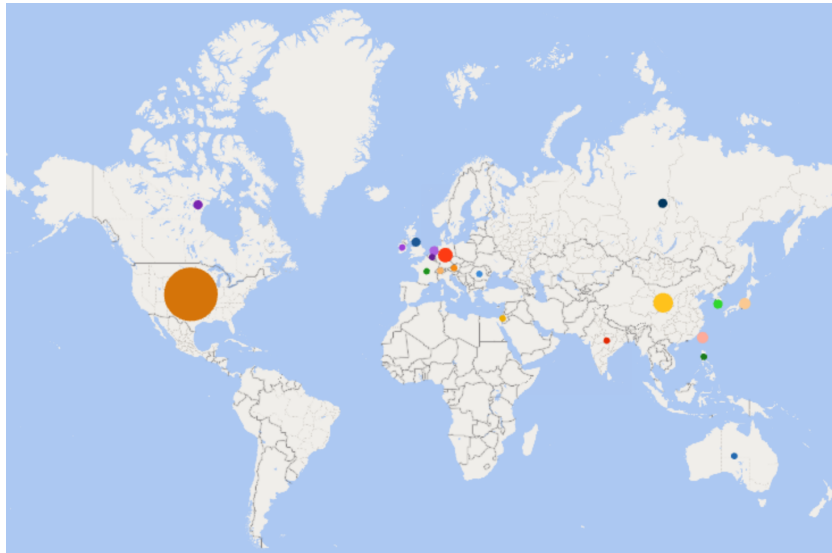


Abbildung 2.5: Weltkarte der Verteilung von CNAs (Stand: 21.11.2019)

<https://cve.mitre.org/cve/cna.html>

### Aufbau eines CVE Eintrags

Ein CVE Eintrag besteht aus drei Teilen.

- CVE ID, welche der eindeutigen Identifizierung des Eintrages dient
  - Struktur: CVE-<Jahr>-<Identifikationsnummer (mind. 4 Stellen)>
  - Beispiel: CVE-2008-4250
- Kurze Beschreibung der Sicherheitslücke
- Sachdienliche Referenzen z.B. zu Schwachstellenreports und Maßnahmenbeschreibungen



### CVSS

Ein Bestandteil von CVE Beschreibungen ist eine Einstufung nach dem “Common Vulnerability Scoring System (CVSS)”. Diese Pseudometrik wird durch eine Reihe von subjektiven und objektiven Parametern errechnet. Beispiele für diese Parameter sind Angriffsvektoren, welche Bereiche betroffen sind (Vertraulichkeit, Integrität oder Verfügbarkeit) oder auch welche Umgebungen betroffen sind (Netzwerke, benachbarte Netzwerke oder nur lokal). Der errechnete Wert befindet sich nun in einem von fünf Intervallen, die jeweils natürlichsprachlich die Schwere der Verwundbarkeit beschreiben: None, Low, Medium, High und Critical. Bis Version 2 des CVSS wurde noch die mathematische Formel für die Berechnung des CVSS veröffentlicht, inzwischen seit Version 3 wird die Berechnung des Scores als Pseudocode veröffentlicht [FIRST.Org, 2019] und Werkzeuge zur Verfügung gestellt, um einen CVSS zu berechnen [NIST].

### 2.4.2 CWE

Die “Common Weaknesses Enumeration (CWE)” verfolgt den Ansatz Softwareschwachstellentypen zu benennen und zu kategorisieren. Die CWE kann also dazu genutzt werden, verbreitete Schwachstellen in einer Software abzu prüfen, abzuschwächen und im besten Fall abzuwenden. Diese Liste wird weltweit von Wissenschaftlern und Forschern von aktuell 55 verschiedenen Universitäten, Unternehmen und Organisationen gepflegt, darunter “Red Hat”, “Oracle” und “OWASP” [MITRE Corporation, 2019d]. Ein CWE Eintrag besteht aus einer ID mit folgender Struktur <CWE-*<einzigartige Nummer>*> und folgenden Einzelheiten:

- Kurzbeschreibung
- Lange Beschreibung
- Beziehungen zu anderen CWEs
- Zeitpunkt des Auftretens der Schwachstelle im Lebenszyklus einer Software
- Betroffene Plattformen und Programmiersprachen
- Wahrscheinlichkeit für eine Ausnutzung dieser Schwachstelle
- Beispiele für die Ausnutzung der Schwachstelle

- Abschwächungs oder Beseitigungsmaßnahmen
- Beziehungen zu diesem CWE innerhalb der Datenbank
- Notizen

Innerhalb des Systems, welches im Rahmen der Arbeit entwickelt wird, wird versucht bei den gescannten Softwareabhängigkeiten, die eine Schwachstelle enthalten, auch immer die Referenz zu dem CWE Eintrag herzustellen.

## 3 Systemkonzept

### 3.1 Ziel der Anwendung

Das Zielsystem ist eine Webanwendung, welche einen Service anbietet, welcher eine statische Sicherheitsanalyse eines gegebenen Git-Repositories vornimmt. Hierbei sollen externe Abhängigkeiten auf Schwachstellen und Verwundbarkeiten überprüft werden und ein entsprechender Report erstellt werden. Des Weiteren soll nach sensiblen Geheimnissen innerhalb des Quelltextes und der Repository-Historie gesucht werden und mögliche Befunde eben dieser gemeldet werden. Das übergeordnete Ziel der Anwendung ist es, das Sicherheitsniveau für den/die AnwenderIn zu erhöhen. Dies soll so einfach wie möglich in den Softwareentwicklungsprozess integriert werden können und in der Anwendung wenig Mehraufwand bedeuten. Der Arbeitstitel des Systems ist “reposec” was für “Repository Security” steht und den Nutzen des Systems zum Ausdruck bringen soll.

### 3.2 Funktionale Anforderungen

**FA1: Als Benutzer bin ich in der Lage mit dem System Geheimnisse innerhalb eines Git Repositorys zu detektieren**

Beim Scannen durchsucht die Software unter anderem mit den in den Grundlagen besprochenen Techniken wie Entropie, Regulären Ausdrücken, Wörter Filter und Wortstruktur Filter ein gegebenes Git Repository, um Geheimnisse zu entdecken. Dieser Dienst wird über eine REST-Schnittstelle angeboten und speichert durchgeführte Scans in einer Datenbank. Hierbei ist wichtig zu erwähnen, dass die gefundenen Geheimnisse selbst nicht gespeichert werden, sondern nur der Ort, an welchem sie sich befinden, da für den Benutzer der Ort ausreichen sollte, um zu identifizieren, um was für ein Geheimnis es sich handelt.

#### **FA2: Als BenutzerIn bin ich in der Lage, mit dem System verwundbare externe Abhängigkeiten innerhalb eines Softwareprojektes zu detektieren**

Die Software durchsucht bei einem gegebenen Softwareprojekt die Verzeichnisstruktur, um typische Dateien zu finden, in welchen Projektabhängigkeiten stehen. In dieser Arbeit wird sich auf die “package.json”, welche Abhängigkeiten aus dem npm Repository enthält, beschränkt. Da das npm Repository die größte Softwarebibliothek ist, reicht es im Rahmen dieser Arbeit aus, um das Konzept der Software zu verdeutlichen. Dieser Dienst wird über eine REST Schnittstelle angeboten und Ergebnisse werden in einer Datenbank abgespeichert.

#### **FA3: Als BenutzerIn bin ich in der Lage mir Reports über vergangene Scans anzuschauen**

Eine Komponente des Systems ist eine Webapplikation, in welcher der Benutzer Reports über vergangene Scans einsehen kann. In der Darstellung der Reports ist es möglich aus einer Liste von Reports einzelne Reports detaillierter anzuschauen.

#### **FA4: Als System bin ich in der Lage auf Github-Webhooks zu reagieren**

Es ist möglich die Reporterstellung über Webhooks der Plattform Github zu initiieren. Die Webhook-Funktionalität beschränkt sich hierbei auf Webhooks, die durch Github-Pull-Request-Ereignisse ausgelöst werden.

#### **FA5: Als BenutzerIn werde ich über einen Scan per E-Mail informiert**

Sofort nachdem ein Scan passiert ist, sendet das System eine E-Mail an eine gegebene Liste von E-Mail Adressen. Diese E-Mail enthält einen Link zu dem erstellten Report. Die E-Mails werden bei automatischen Scans über Webhooks immer versendet und bei manuellen Scans optional versendet.

**FA6: Als BenutzerIn bin ich in der Lage, über Public-Key-Authentifizierung das System meine Repositories scannen zu lassen**

Das System erzeugt ein SSH Schlüsselpaar. Der Public Key wird dem Benutzer bereitgestellt, so dass er das System mit Hilfe dieses Keys berechtigen kann, seine Repositories zu scannen. Git über HTTPS wird nicht unterstützt. Das bedeutet, dass jedes Repository, das gescanned werden soll, vorher das System mit Hilfe des Public Keys authentisieren muss.

**FA7: Als BenutzerIn bin ich in der Lage, über eine .ignore Datei zu steuern, welche Dateien und Verzeichnisse nicht gescanned werden**

Beim Scannen durchsucht das System den "Default Branch" des Repositories nach einer Datei in der Informationen über Pfade und Dateien stehen, die von den Scanreports ausgeschlossen werden sollen, um zum Beispiel das Scannen von kompilierten Dateien zu vermeiden.

### 3.3 Nichtfunktionale Anforderungen

**NFA1: Bestehendes integrieren**

Es wird an erster Stelle schon existierende Software genutzt. Diese wird an Stellen mit Verbesserungspotential modifiziert und für die Systemumgebung angepasst. Trotzdem soll darauf geachtet werden, dass nicht unnötig viele Abhängigkeiten geschaffen werden.

**NFA2: Einfachheit in der Bedienung**

Die REST Schnittstelle sowie die Weboberfläche sollen intuitiv bedienbar sein. Bei der REST Schnittstelle wird dieses durch eine übersichtliche Dokumentation und einem REST konformen Schnittstellendesign erreicht. Bei der Weboberfläche wird dies durch die Konzentration auf und das Hervorheben der Kernfunktionalitäten erreicht.

### NFA3: Wenig Aufwand in der Bedienung

Aufgrund der Wichtigkeit für diese Arbeit ist diese Anforderung extra aufgeführt und nicht in NFA2 integriert. Das System soll im Betrieb im besten Fall eigenständig arbeiten. Das bedeutet konkret, dass es auf Aktionen des Entwicklers in seinem normalen Softwareentwicklungsprozess reagiert und Signale sendet.

## 3.4 Architektur

Die Architektur des Systems wird im Folgenden durch drei Architekturdiagramme dargestellt.

### 3.4.1 Kontextsicht

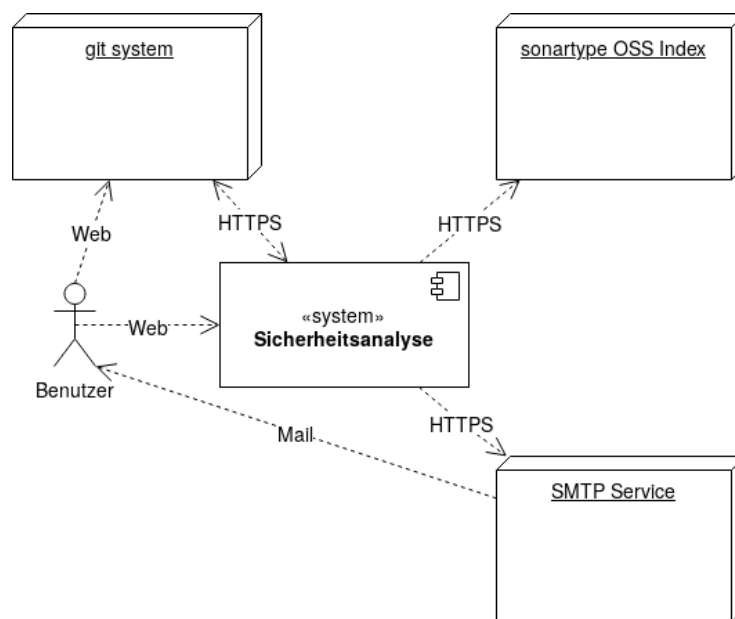


Abbildung 3.1: Kontextsicht des repossec Systems (Eigene Darstellung)

Die Abbildung 3.1 verdeutlicht, dass der/die BenutzerIn einerseits mit dem repossec System agieren kann, andererseits aber auch mit dem Git-System, welches an Stelle des Benutzenden dann mit dem repossec Systems agiert, welches wiederum über den SMTP Service den/die BenutzerIn benachrichtigt.

## 3.4.2 Bausteinsicht

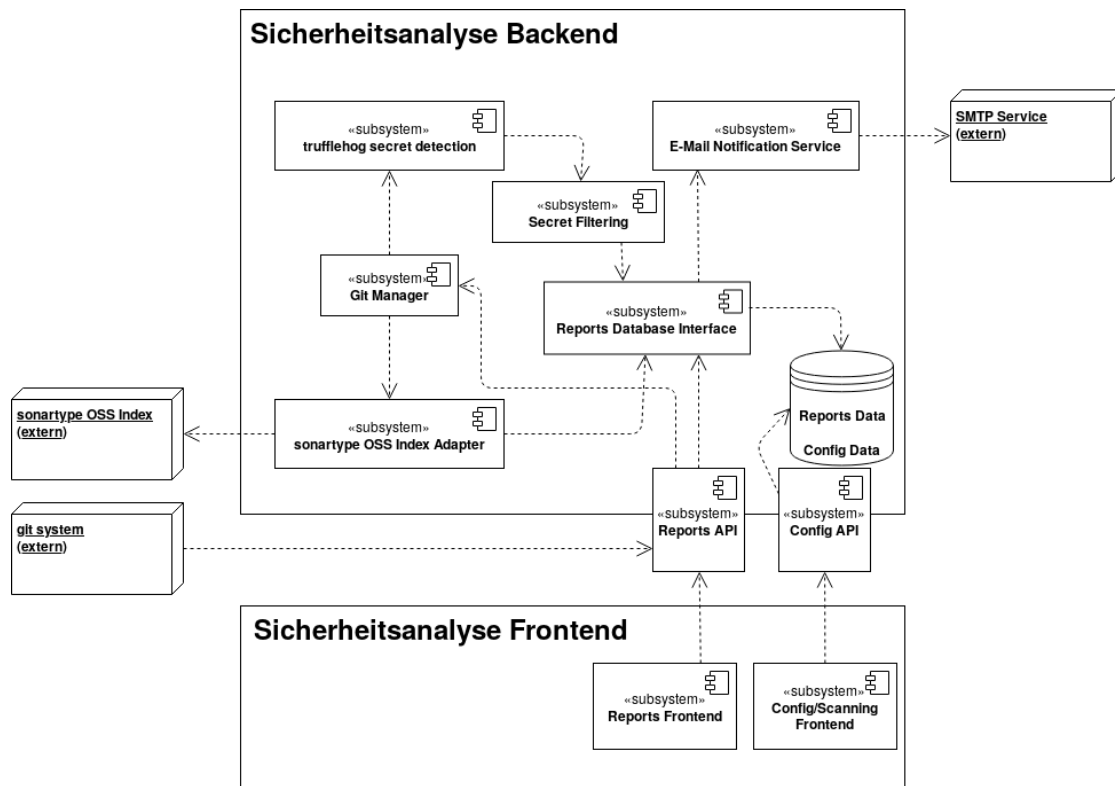


Abbildung 3.2: Bausteinsicht des reposer Systems

In der Abbildung 3.2 der Bausteinsicht sieht man die verschiedenen Bestandteile des Systems. Die grobe Struktur zeigt ein Frontend, ein Backend, eine Datenbank und drei externe Systeme. Die Reports API wird als Einstiegspunkt für das Frontend sowie für das externe Git-System genutzt. Sobald ein Client anfordert, dass ein neuer Report erstellt wird, wird im Backend der Git-Manager aktiviert, welcher das zu scannende Repository kloniert und die Referenz auf dieses an die truffleHog secret detection und den sonartype OSS Index Adapter weitergibt. Die truffleHog secret detection sucht nun Geheimniskandidaten, welche sie dann zum Secret Filtering gibt. Beim Secret Filtering werden dann der Wörter Filter und der Wortstrukturfilter angewendet. Die bereinigten Geheimnisse werden dann über das Reports Database Interface an die Reports Datenbank erreicht. Danach überprüft der sonartype OSS Index Adapter wiederum das zuvor erhaltene Repository, indem es die verwendeten Bibliotheken extrahiert und von der externen sonartype OSS Index Schnittstelle scannen lässt. Die gewonnenen Informationen gelangen ebenfalls über das Reports Database Interface in die Datenbank. Wenn das Reports Database In-

terface seine Arbeit beendet hat, sendet der E-Mail Notification Service mit Hilfe des externen SMTP Services E-Mails. Des Weiteren werden die in der Vergangenheit erstellten Reports auch über die Reports API angeboten.

Die Config API wird dazu genutzt, die Authentisierung und die E-Mail Notifications des Systems zu konfigurieren.

#### 3.4.3 Verteilungssicht

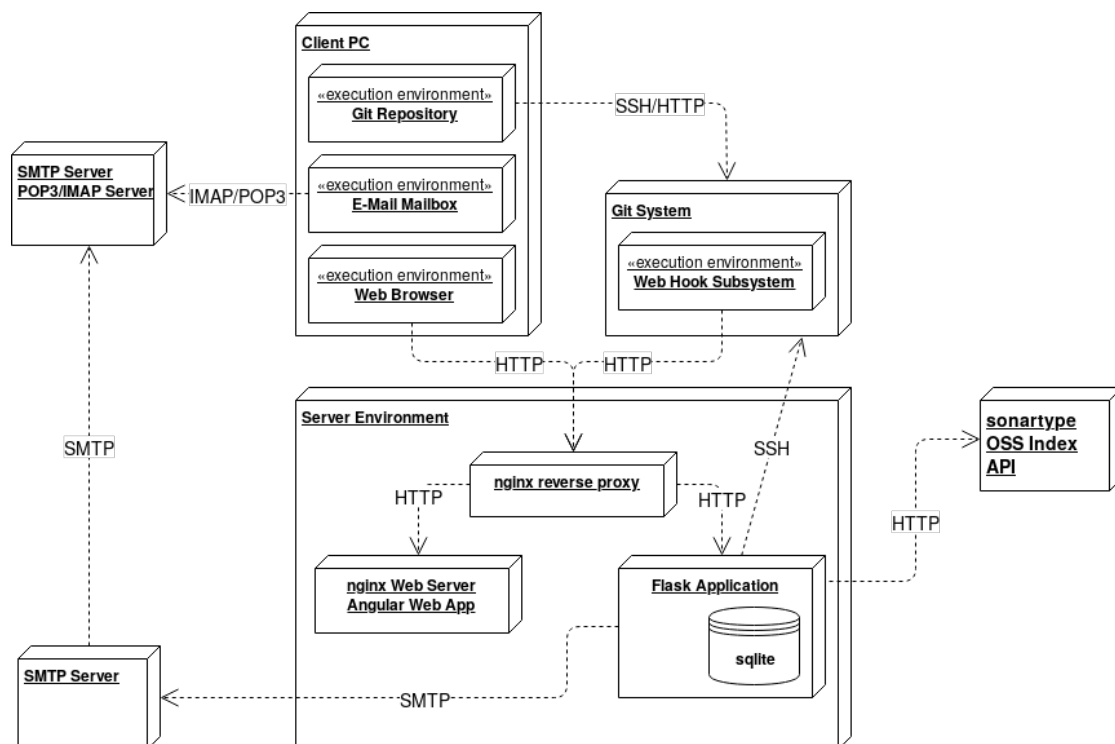


Abbildung 3.3: Verteilungssicht des reposec Systems

In der Abbildung 3.3 sieht man die Verteilungssicht des reposec Systems. Der Client benötigt für die Benutzung des Systems einen Web Browser, eine E-Mail Mailbox und ein Git-Repository, welches er von dem Git-System geklont hat. Der verwendete Git Service (in diesem Fall Github) benötigt einen Webhook-Service. Die Umgebung für die reposec Webapplikation besteht aus mehreren Services, weshalb für die Umsetzung der reposec Software die Containervirtualisierungssoftware "Docker" benutzt wird. Die Webapplikation wird nun über drei Docker-Container realisiert. Es gibt einmal das Backend welches auf Flask basiert und eine sqlite Datenbank benutzt, welche auf dem Docker-Host liegt



und vom Flask Server Container gemounted wird. Der zweite Container ist das Frontend in Form einer Angular Web App. Der dritte Container ist ein nginx Webserver, der als reverse Proxy agiert, welcher HTTP Anfragen entsprechend an das Frontend oder das Backend weiterleitet. Die drei Container werden über das Tool “Docker Compose” für ein einfaches Bauen und Deployen gemanaged. Eine detaillierte Beschreibung der verwendeten Dienste folgt in Kapitel 4.

# 4 Entwicklung

## 4.1 Verwendete Technologie

### 4.1.1 Python Flask

Für die Umsetzung der Logik im Backend sowie das Implementieren der Web Schnittstelle wurde die Bibliothek Python “Flask” verwendet. Flask ist ein Webentwicklungs Framework, welches einen minimalen Satz an Kernfunktionen liefert. Diese lassen sich durch eine Vielzahl von Plugins erweitern [Ronacher et al., Flask 1.1.1]. Es wurde gewählt, da es wenig Boilerplate Code und Verwaltungsaufwand mit sich bringt und somit die Kernfunktionalität des Systems im Vordergrund steht.

#### **Flask-RESTPlus**

Eine Erweiterung, die für Flask verwendet wurde, heißt “Flask-RESTPlus”. Diese wurde im Laufe der Entwicklung hinzugefügt, da sie eine Vielzahl von Annotationen und Mechanismen liefert, um programmatisch Schnittstellendokumentationen zu generieren [Haustant, Flask-RESTPlus 0.13.0].

### 4.1.2 SQLAlchemy

Python “SQLAlchemy” liefert eine Sammlung an Werkzeugen zur Benutzung von SQL Datenbanken sowie einen “Object Relation Mapper (ORM)”. ORMs werden dazu genutzt Daten, die in relationalen Datenbanken liegen, auf Objekte abzubilden, die in Applikationsquelltexten benutzt werden. SQLAlchemy ist kompatibel mit einer Vielzahl von Datenbanksystemen und liefert Migrationsfunktionalitäten, um diese Datenbanksysteme auch ggf. migrieren zu können [Bayer, SQLAlchemy 1.3.10].

### 4.1.3 SQLite

Als Datenbanksystem wurde eine “SQLite” Datenbank verwendet. SQLite ist eine Bibliothek, die eine serverlose, relationale SQL Datenbankengine implementiert. SQLite arbeitet direkt auf Dateien, die auf dem Dateisystem liegen, wahlweise auch auf Daten, die im Arbeitsspeicher liegen (In-Memory) [Hipp, SQLite 3.30.1]. Sie wurde hier aufgrund der Einfachheit in der Architektur und den somit eingesparten Verwaltungsaufwand verwendet. SQLite ist nicht für die Skalierung des Systems geeignet. Vielmehr wurde mit der Benutzung von SQLAlchemy darauf geachtet, dass die Datenbank austauschbar ist, falls es das Ziel sein sollte, die Anwendung zu skalieren.

### 4.1.4 Angular

Für das Web Frontend wurde das Javascript/Typescript Framework “Angular (2+)” verwendet. Angular ist ein Single Page Application Framework, welches viele Funktionalitäten von Haus aus mitliefert [Google, Angular 8.1.3]. Im Rahmen dieser Arbeit fällt die Auswahl der Frontend Technologie nicht groß ins Gewicht. Deshalb wurde das Framework gewählt, bei dem die meiste Praxiserfahrung vorhanden war.

### Angular Material

Als Erweiterung von Angular wurde die Komponenten Bibliothek Angular Material verwendet. Diese liefert einen Satz an Standard UI-Komponenten zur Benutzung für das Frontend. Durch die strikte Umsetzung des “Material Design Konzepts” innerhalb von Angular Material wird dieses automatisch durch die Benutzung der Bibliothek erzwungen [Google, Angular Material 8.1.3].

### 4.1.5 Docker

Für das Deployment der einzelnen Anwendungen wurde die Containervirtualisierungssoftware “Docker” verwendet. Container enthalten alle notwendigen Systemfunktionen und Pakete, um die einzelnen Anwendungen auszuführen. Die Konfiguration der Container beschränkt sich jeweils auf eine Datei, was sie sehr transportabel macht [Docker, Docker 19.03].

### **Docker Compose**

In der Applikation werden drei Container verwendet, die untereinander kommunizieren können. Dies wird mithilfe des Multi-Container Werkzeugs “Docker Compose” konfiguriert und verwaltet. Ein weiterer Vorteil ist die Abstraktion des Deployments der drei Container, so dass sie sich am Ende sehr einfach bauen und deployen lassen [Docker, Docker Compose 1.24.1].

## **4.2 Verwendete Dienste**

### **4.2.1 Google SMTP Dienst**

Für das Senden von E-Mails wird der SMTP Dienst von Google über SSL verwendet [Google, Google Mail]. Die Anbindung in Flask wurde über die Python Standard Bibliothek “smtplib”, welche das “SMTP” Protokoll (RFC 821) und die “SMTP Service Extensions” (RFC 1869) implementiert, gelöst [Python Software Foundation, Python smtplib]. Die Absenderadresse, die die Benachrichtigungen sendet, lautet “reposecreporter@gmail.com”.

### **4.2.2 Sonatype OSS Index**

Für das Finden von Schwachstellen innerhalb von Softwarebibliotheken wird der Sonatype OSS Index verwendet. Diese Schnittstelle wurde in den Grundlagen bereits erläutert [Grundlagen: sonatype OSS Index].

### **4.2.3 truffleHog**

Für das Detektieren von Geheimniskandidaten wurde die Software truffleHog in das Flask Backend integriert, welche in den Grundlagen bereits erläutert wurde [Grundlagen: Software: truffleHog].

### 4.2.4 Common Words in Programming Languages

Für das Herausfiltern von beliebten Wörtern in Programmiersprachen aus Geheimnis-kandidaten wurden die Daten von der Internetseite <https://anvaka.github.io/common-words/> extrahiert und verwendet [Kashcha, 2018].

### 4.2.5 Verwendete Entwicklungswerkzeuge

Es wurde hauptsächlich die Software Visual Studio Code verwendet. Visual Studio Code ist ein Open-Source Quelltexteditor der Firma Microsoft, der Plugins für alle hier verwendeten Programmiersprachen und Frameworks anbietet, die den Arbeitsfluss erleichtern [Microsoft, VSCode 1.40.1]. Für das Management von Abhängigkeiten wurde im Frontend “npm” [Grundlagen: Node Package Manager] und für das Backend “virtualenv” verwendet [Bicking, virtualenv 16.7.2]. Des Weiteren wurde auf Textbearbeitungsebene der Texteditor “Vim” verwendet [Moolenaar, Vim 8.0], welcher u.a. als Plugin in Visual Studio Code integriert wurde.

## 4.3 Implementierungsdetails

### Entwicklungs- und Systemsprache

Als Sprache für Benennungen und Kommentare während der Entwicklung wurde Englisch gewählt. Diese Entscheidung wurde getroffen, weil es dann zu keiner Vermischung von Deutsch und Englisch mit den ohnehin existierenden reservierten Wörtern kommt und weil mit truffleHog eine Software integriert wurde, die ebenfalls Englisch als Entwicklersprache benutzt. Das Frontend, die Backendschnittstelle und die Schnittstellendokumentation sind ebenfalls aus Konsistenzgründen in englischer Sprache. Weitere Systemsprachen ließen sich mit der Hilfe von “i18n”-Bibliotheken hinzufügen, dies ist nicht Bestandteil dieser Arbeit.

### 4.3.1 Autorisierung des Systems zum Scannen

Das System unterstützt ausschließlich Authentifizierung und Klonen von Repositories über Public-Key Authentifizierung. Das heißt, dass bei der Installation des Systems einmalig ein SSH Schlüsselpaar erzeugt wird. Der Public Key muss von dem/der BenutzerIn aus dem System über das Frontend ausgelesen werden und dem Github Account hinzugefügt werden, der Zugriff auf die Repositories Repositories hat, die gescannt werden sollen. Beim Autentifizierungsprozess kann nun Github eine Nachricht mit dem Public-Key verschlüsseln, die dann von dem reposer System entschlüsselt wird und das Ergebnis Github mitgeteilt wird. So kann Github das reposer System authentifizieren und für den Zugriff auf die Repositories autorisieren.

### 4.3.2 Verwendete reguläre Ausdrücke

Für die Geheimniskandidatenerkennung über reguläre Ausdrücke wurde die Standardliste der Software truffleHog verwendet. Diese ist im Anhang dargestellt (siehe A.1).

### 4.3.3 Wichtige Datentypen

In der Abbildung 4.1 sieht man die drei wichtigsten Datenmodelle des Systems, aus denen die Reports gebildet werden. Wichtig zu erwähnen ist, dass das Modell “SecretIssue” für ein erkanntes Geheimnis verwendet wird und nur den Weg zu der Datei (“path”-Feld), in der das erkannte Geheimnis steht, enthält, nicht das Geheimnis selber. Das “reason”-Feld enthält den Grund für die Erkennung als Geheimnis also entweder durch Entropie, oder regulären Ausdruck.

Bei dem Scan von Softwareabhängigkeiten entstehen bei Befunden “DependencyIssues”, welche auf “Vulnerabilities” also Verwundbarkeiten referenzieren. Wichtige Felder hier sind “cwe” und “cve”, welche den CWE, bzw. wenn vorhanden den CVE Identifikator enthalten. Der “cvss\_vector” enthält eine Textrepräsentation der Werte, die verwendet wurden, um den “cvss\_score” zu errechnen. Das “reference”-Feld enthält einen Hyperlink zu der OSS Index Plattform, wo eine detaillierte Beschreibung der Verwundbarkeiten und Schwachstellen hinterlegt ist.

#### 4.3.4 Gefundene Geheimnisse in bestimmten Dateien und Pfaden ausschließen

Programmatisch generierte Binärdateien, die teilweise in Repositories liegen, haben die Eigenschaft, dass sie eine hohe Entropie besitzen. Sie würden daher schnell als Geheimniskandidat identifiziert werden und eventuell nicht weggefiltert. Daher wurde ein Mechanismus implementiert, der es dem/der BenutzerIn erlaubt Daten explizit vom Report auszuschließen. Hierfür muss der/die BenutzerIn in den "Default-Branch" des Repositories eine Datei anlegen, die ".reposecignore" heißt. Diese Datei enthält pro Zeile eine Beschreibung eines Pfades bzw. einer Datei, die vom Report ausgeschlossen werden soll. Die Beschreibung ist in Form eines regulären Ausdrucks zu geben. Dateien, die im Wurzelverzeichnis des Repositories liegen, benötigen *keine* Zusatzzeichen am Anfang bei der Angabe wie z.B. "./" oder "/". Zeilen in der .reposecignore Datei, die mit einem "#" beginnen, werden ignoriert. In Abbildung 4.2 sieht man ein Beispiel für eine solche Datei.

```
1     Vulnerability
2     {
3         "id": int,
4         "dependency_issue_id": int,
5         "oss_id": string,
6         "title": string,
7         "description": string,
8         "cvss_score": string,
9         "cvss_vector": string,
10        "cwe": string,
11        "cve": string,
12        "reference": string
13    }

1     SecretIssue
2     {
3         "id": int,
4         "report_id": int,
5         "branch": string,
6         "commit": string,
7         "commitHash": string,
8         "date": string,
9         "path": string,
10        "reason": string,
11    }

1     DependencyIssue
2     {
3         "id": int,
4         "report_id": int,
5         "branch": string,
6         "coordinate": string,
7         "description": string,
8         "reference": string,
9         "vulnerabilities": Vulnerability[],
10        "package_name": string,
11    }
```

Abbildung 4.1: Auszug der wichtigsten verwendeten Datenmodelle des Systems



```
1      # example .reposecignore
2
3      (*.*)?env/(.*)$
4      angular-app/package-lock\.json
5      angular-app/src/app/app\.component\.html
```

Abbildung 4.2: Beispiel einer .reposecignore Datei

# 5 Integration in den agilen Softwareentwicklungsprozess

## 5.1 Der agile Softwareentwicklungsprozess

Im agilen Softwareentwicklungsprozess wird auf eine Sammlung von Methoden und Vorgehensweisen gesetzt, die daraufhin optimiert sind, bei spezifischen Problemen zu helfen. Diese Methoden sind einfach gehalten und lassen sich recht unkompliziert umsetzen. Des Weiteren beziehen sich diese Methoden auf alle Bereiche der klassischen Softwareentwicklung, also auch Projektmanagement, Softwaredesign und -architektur sowie Prozessverbesserungen.

Außerdem ist es in der agilen Softwareentwicklung wichtig, dass Informationen im Team untereinander geteilt werden, um Projektentscheidungen gemeinsam zu treffen anstatt diese durch einen Projektmanager alleine treffen zu lassen. Es geht also darum, Planung, Design und Prozessverbesserungen für das gesamte Team zu öffnen. Hierfür sind eine offene Denkweise der ProjektteilnehmerInnen, einfache Kommunikationswege und flexible Arbeitsstrukturen notwendig, um auf Änderungen und Probleme reagieren zu können [Stellman und Greene, 2019].

Ein Beispiel für so eine flexible Arbeitstruktur ist im Bereich des Konfigurationsmanagements die Verwendung des Branching Modells nach Driessen, welches in den Grundlagen bereits erläutert wurde [Grundlagen: Branching Modell nach Vincent Driessen].

## 5.2 Einordnung des Systems in den agilen Softwareentwicklungsprozess

Im Folgenden wird an einem Beispiel gezeigt, wie sich das im Rahmen dieser Arbeit entwickelte System in den agilen Softwareentwicklungsprozess einordnen lässt. Es wurde bereits durch die Konzeption des Systems festgelegt, dass der Scan durch das Auslösen eines Pull oder Merge Request ausgelöst werden kann und eine Liste von Personen Bescheid bekommt. Das heißt, dass sich dann eine Person aus dieser Liste dazu angehalten fühlen muss, Befunde zum Beispiel über die Aufnahme in ein Ticketsystem dem Team zu zuteilen, wie es in agilen Teams üblich ist. Ein Beispiel, wie diese Pull- oder Mergerequests in agilen Methoden entstehen wurde anhand des Driessen Branching Modells im Abschnitt Grundlagen: Pull Request beschrieben und wird in Abbildung 5.1 visualisiert.

Neben der Bearbeitung dieser Befunde und noch dem initialen Aufwand des Einrichten des Systems ist der beschriebene Prozess der Minimalaufwand, der betrieben werden muss, um das Sicherheitslevel eines Softwareprojektes zu erhöhen. Jedoch ist es sehr unwahrscheinlich, dass jede Änderung in jedem Team in einem Pull oder Merge Request auftaucht bzw. diese Hilfsmittel überhaupt genutzt werden. Deshalb ist es sinnvoll, wenn Repositories regelmäßig durch einen Sicherheitsverantwortlichen mit Hilfe des Systems manuell gescanned werden und wiederum in einem Bearbeitungsprozess landen, wie zum Beispiel das initiale Anlegen eines Tickets in einem Ticketsystem.

## 5.3 Reaktion auf Befunde

Wenn nun Befunde in den Reports auftauchen gilt es diese zu behandeln. Möglichkeiten dies zu tun sind im Folgendem exemplarisch dargestellt. ,

### 5.3.1 Reaktion auf korrekt gefundene Geheimnisse

Sobald ein Geheimnis womöglich öffentlich in einem Software Repository auftaucht, sollte man es als kompromittiert betrachten [Doll, 2013]. Es muss also geändert werden. Es gilt aber trotzdem, in solchen Situationen vorerst Ruhe zu bewahren. Das sofortige Ändern verhindert zwar, dass das Geheimnis weiterhin ausgenutzt werden könnte, aber man sollte sich im Klaren sein, was für Systeme im Einsatz sind, die momentan abhängig von diesem

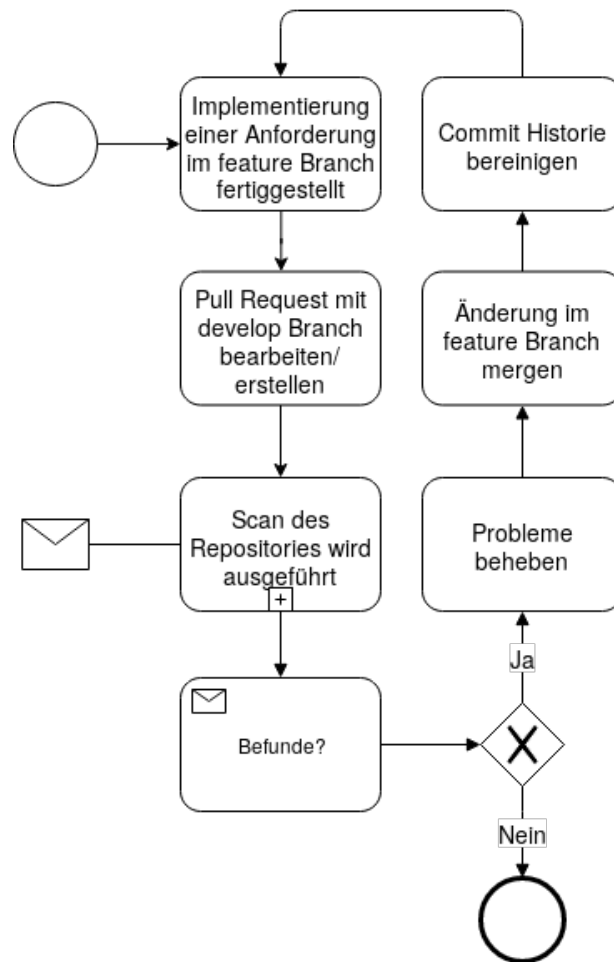


Abbildung 5.1: Integration des Systems in den agilen Entwicklungsprozess

Geheimnis sind und die Funktionalitätsausfall hätten und die Auswirkungen abwägen. Es muss in jedem Fall eine nachhaltige Lösung für die Nutzung von den Geheimnissen implementiert werden. Ein Beispiel folgt.

### Mit Umgebungsvariablen arbeiten

Eine Möglichkeit Geheimnisse zu nutzen, ohne dass sie im Quelltext auftauchen, ist es, mit Umgebungsvariablen zu arbeiten. Das sind konfigurierbare Variablen im Betriebssystem, die nur den Prozessen zur Verfügung stehen, die auf diesem laufen. Das heißt, ein/eine AngreiferIn müsste Zugriff auf die Maschine haben auf der das System läuft,

um Umgebungsvariablen auszulesen. Dies ist meistens eine größere Hürde für den Angreifer als das Auslesen eines Klartextgeheimnisses aus einem Softwarerepository. Diese Umgebungsvariablen können aus dem Prozess heraus referenziert werden. Somit ist es nicht nötig, die Geheimnisse im Quelltext zu hinterlegen. Nicht nur Betriebssysteme selber, sondern auch Servermanagement Werkzeuge bieten häufig Funktionen, diese Umgebungsvariablen zu verwalten. Des Weiteren bieten, die meisten Programmiersprachen auch von Haus aus interne Bibliotheken, die nützliche Funktionen für den Zugriff auf Umgebungsvariablen implementieren.

### Git Historie bereinigen

Wenn nun die Geheimnisse nicht mehr Teil des Quelltextes sind, dann sind sie trotzdem immer noch Teil der Commit Historie. Denn bei der Nutzung von Git wird jede Änderung dokumentiert. Wie in den Grundlagen erwähnt, errechnet sich jeder neue Commit Hash zum Teil auch aus der vorherigen Commit Historie. Das bedeutet, dass es nicht möglich ist, die Commit Historie zu editieren, ohne dass es zu Anomalien bei den Hashwerten kommt. Was jedoch möglich ist, ist Dateien aus der Commit Historie zu entfernen und die Historie neu zu erstellen. Dies ist mit Vorsicht durchzuführen. Alle lokalen Commits von EntwicklerInnen, müssen vorher gemerged werden, ansonsten kommt es zu Kollisionen. Des Weiteren müssen lokale Versionen im Nachhinein neu geklont werden, da es Diskrepanzen in den Hashwerten geben wird und existierende "Tags" werden überschrieben. Wenn also das gefundene Geheimnis beispielsweise ein einfacher API Token ist, der dann neu generiert wurde, sollte man sich überlegen, ob es Sinn macht die Commit Historie überhaupt zu bereinigen.

Das Bereinigen der git Historie ist mit von Haus aus mitgelieferten Mitteln der Software Git möglich. Es ist das Kommando, welches in Abbildung 5.2 zu sehen ist mit dem kompletten Pfad zu der Datei, welche das Geheimnis enthält, auszuführen.

```
1 $ git filter-branch --force --index-filter \  
2 "git rm --cached --ignore-unmatch PFAD-ZUR-DATEI-MIT-  
   SENSIBLEN-DATEN" \  
3 --prune-empty --tag-name-filter cat -- --all
```

Abbildung 5.2: Bereinigen der Commit Historie

### 5.3.2 Reaktion auf fehlerhaft entdeckte Geheimnisse

Wenn das System fehlerhaft Geheimnisse detektiert, also Zeichenketten, die keine Geheimnisse enthalten, ist es möglich die Dateien über die “.reposecignore” von den Reports auszuschließen (siehe Abschnitt 4.3.4). Natürlich birgt dies die Gefahr, dass zukünftig Geheimnisse, die in den ausgeschlossenen Dateien auftauchen können, nicht mehr entdeckt werden. Dies sollte bedacht werden, wenn Dateien der .reposecignore Datei hinzugefügt werden.

### 5.3.3 Reaktion auf gefundene Bibliotheken, die Sicherheitslücken enthalten

Wenn eine Bibliothek in dem Quelltext der aktuellen Version eines Systems gefunden wird, welche eine Schwachstelle enthält, dann sollte man sie austauschen oder aktualisieren. In vielen Fällen kann es schon eine neue Version geben, da es inzwischen häufig der Fall ist, dass sobald eine Schwachstelle in einer Bibliothek gefunden wird, der/die UrheberIn der Bibliothek zuerst kontaktiert wird und ihm/ihr Zeit gegeben wird, diese Schwachstelle zu beheben, bevor die Schwachstelle veröffentlicht wird (“Responsible Disclosure”).

## 5.4 Benutzung des Systems

In diesem Abschnitt wird die konkrete Anwendung des Systems aufgezeigt. Anleitungen für die Einrichtung des Systems in Bezug auf die Konfiguration bei Github sind in dem Hilfe Bereich des Systems vorhanden.

### 5.4.1 Autorisierung des Systems zum Scannen

Für die Autorisierung wird der “Public SSH Key” benötigt, welcher im “Settings Bereich des Systems zu finden ist” (siehe Abbildung 5.3). Dieser muss einem/einer Github NutzerIn hinzugefügt werden, der Zugriff auf das zu überprüfende Repository hat. Github hat die Einschränkung über Benutzerkonten hinaus, dass ein Public SSH Key auch nur einem/einer NutzerIn hinzugefügt werden kann. Daher bietet es sich an einen/einer NutzerIn bei der Plattform Github anzulegen, der als Service Account agiert. Dieser erhält

den SSH Key und wird den zu überprüfenden Repositories als sogenannter “Collaborator” hinzugefügt, also als Mitarbeiter.

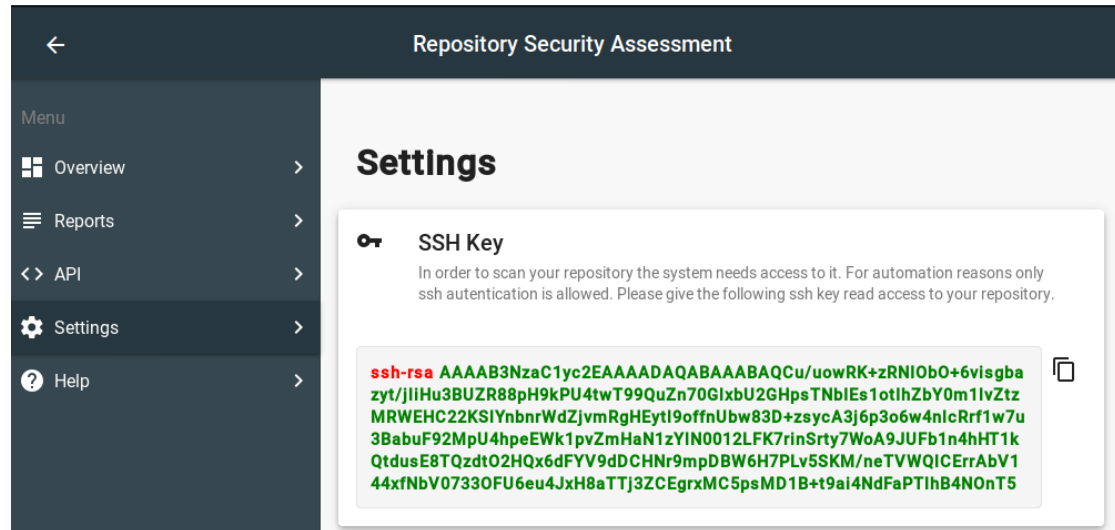


Abbildung 5.3: Der “Public SSH Key” des Systems

### 5.4.2 Manuelles Scannen von Repositories

Um ein Repository proaktiv zu überprüfen, muss man die “git SSH URL” dieses Repositories im Bereich “Overview” zuerst hinzufügen. HTTPS URLs zu Repositories werden nicht unterstützt. Hat man die git URL hinzugefügt, so taucht sie etwas weiter unten auf der Seite auf. Man kann nun den Scan konfigurieren und ausführen. Es gibt folgende Konfigurationsmöglichkeiten:

- Entropy: Aktiviert/Deaktiviert die Geheimniskandidatenerkennung mit Hilfe der Shannon Entropie
- RegEx: Aktiviert/Deaktiviert die Geheimniskandidatenerkennung mit Hilfe von regulären Ausdrücken
- Email Notification: Aktiviert/Deaktiviert die Benachrichtigung per E-Mail an die zuvor konfigurierten Adressen
- Branch Name: Der Branch, der auf Schwachstellen in Abhängigkeiten überprüft wird. Wird das Feld leer gelassen ist dies der ‘master’ Branch.

Die oben beschriebene Ansicht ist in Abbildung 5.4 zu sehen. Nachdem der Scan ausgeführt wurde, wird man zum neuen Report automatisch weitergeleitet.

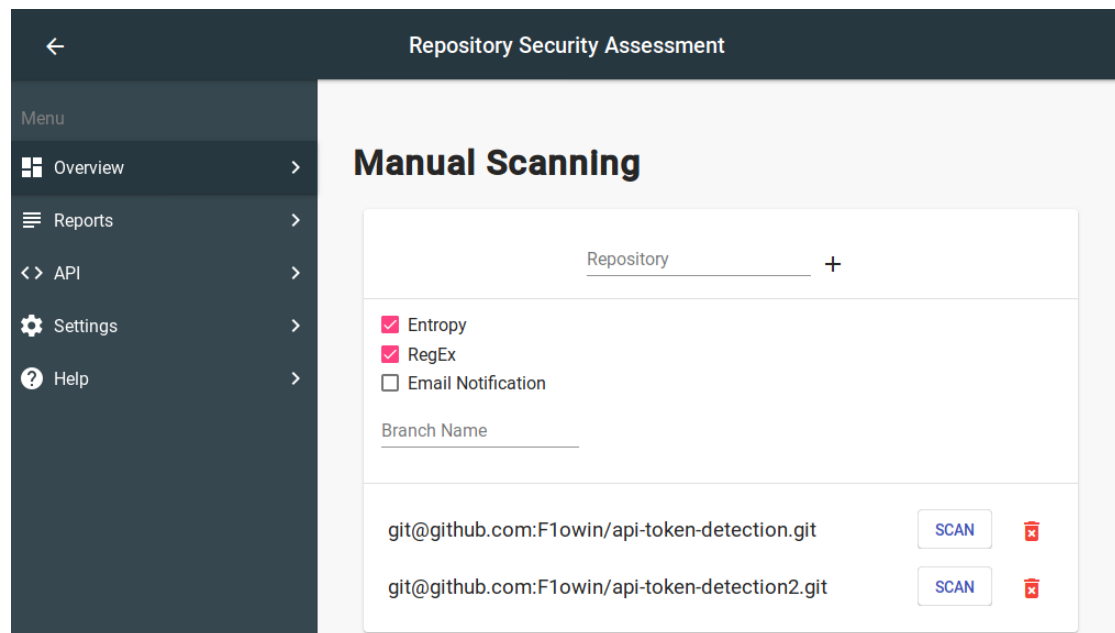


Abbildung 5.4: Manuelles Scannen

### 5.4.3 Konfigurieren der E-Mail Empfängerliste

Die E-Mail Empfängerliste lässt sich im Bereich “Settings” unterhalb des SSH Keys bearbeiten (Bereich “Email Notifications”). Über das Eingabefeld lassen sich neue E-Mail Adressen hinzufügen, welche dann unterhalb des Eingabefeldes in einer Liste auftauchen, woraus sie auch wieder über das Mülleimer Symbol entfernt werden können.

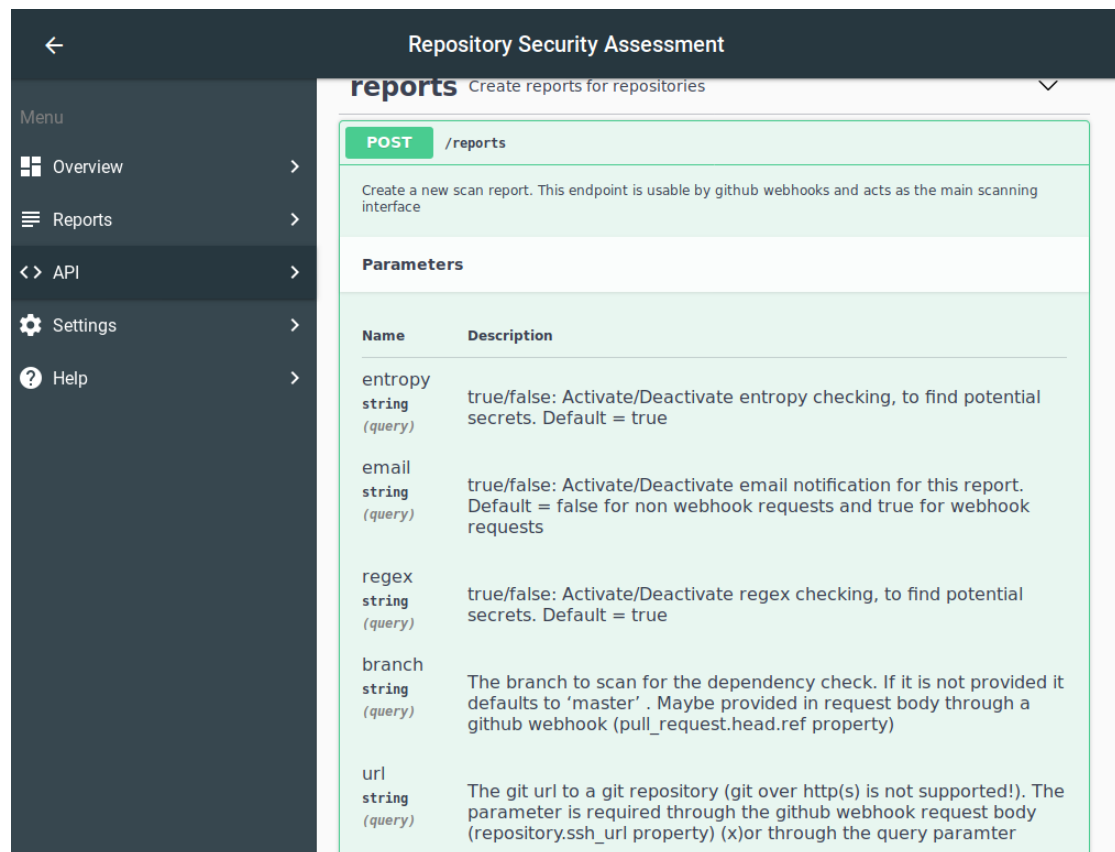
### 5.4.4 Einrichten des Webhooks der Plattform Github für automatisiertes Scannen

Wichtig für diesen Schritt ist, dass das System von der Plattform Github aus erreichbar ist. Das bedeutet, dass es öffentlich erreichbar ist. Im Rahmen dieser Arbeit wird eine öffentliche Instanz des Systems und ein dazugehöriger Github Service Account, der bei zu scannenden Repositories hinzugefügt werden kann sowie eine Reihe von Test Repositories über den beigelegten Datenträger gestellt bzw. referenziert.



Die Schnittstelle für den Webhook ist dem Bereich “API” des Systems zu entnehmen. Standardmäßig ist dies “POST http(s)://<domain-name-des-systems>/reports”. Die Schnittstelle lässt sich über Queryparameter analog zu dem manuellen Scannen konfigurieren, welche auch im Bereich “API” dokumentiert aufgeführt sind (siehe Abbildung 5.5).

Auf der Plattform Github öffnet man das zu scannende Repository und öffnet dort den “Settings” Bereich. Im Settings Bereich gibt es einen “Webhooks” Unterpunkt, den man besuchen muss. Hier kann man einen neuen Webhook hinzufügen. Beim Hinzufügen ist zu beachten, dass die zuvor erhaltene Webhook Schnittstelle bei dem Feld “Payload URL” eingetragen wird, bei dem Feld “Content type” “application/json” ausgewählt wird und bei “Which events would you like to trigger this webhook?” der Punkt “Let me select individual events.” ausgewählt wird. Bei den nun aufgetauchten Eventtypen wählt man “Pushes” ab und wählt “Pull requests” aus. Eine Anleitung mit Bildern ist in dem reposer System im Bereich “Help” integriert.



The screenshot shows the API documentation for the endpoint `POST /reports`. The page title is "Repository Security Assessment" and the sub-header is "reports Create reports for repositories". The endpoint description states: "Create a new scan report. This endpoint is usable by github webhooks and acts as the main scanning interface".

The parameters section is as follows:

Name	Description
entropy string (query)	true/false: Activate/Deactivate entropy checking, to find potential secrets. Default = true
email string (query)	true/false: Activate/Deactivate email notification for this report. Default = false for non webhook requests and true for webhook requests
regex string (query)	true/false: Activate/Deactivate regex checking, to find potential secrets. Default = true
branch string (query)	The branch to scan for the dependency check. If it is not provided it defaults to 'master' . Maybe provided in request body through a github webhook (pull_request.head.ref property)
url string (query)	The git url to a git repository (git over http(s) is not supported!). The parameter is required through the github webhook request body (repository.ssh_url property) (x)or through the query paramter

Abbildung 5.5: Bereich API - Schnittstelle für den Github Webhook

### 5.4.5 Abrufen von Reports

Das Abrufen von Reports findet im Bereich "Reports" statt. Dort befindet sich eine Liste der zuletzt erstellten Reports. Klickt man einen Listeneintrag an, so landet man bei diesem Report (Siehe Abbildung 5.6).

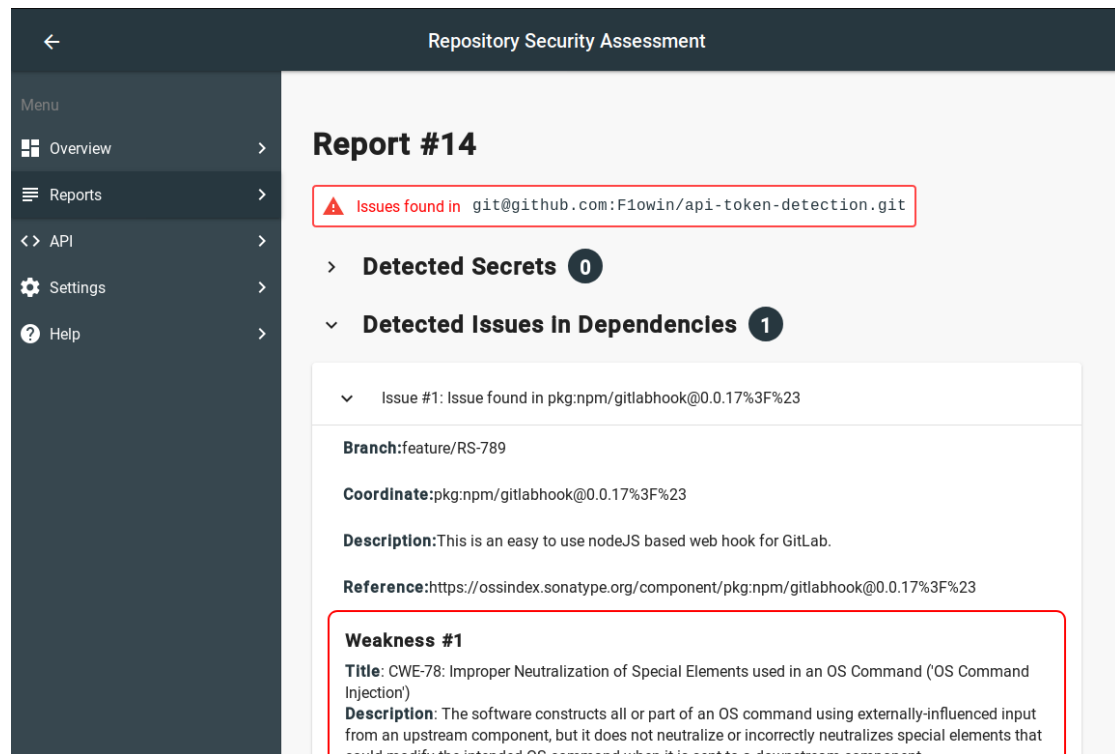


Abbildung 5.6: Bereich Reports - Beispiel eines Reports

## 6 Fazit

In diesem Kapitel wird zurückgeschaut auf das entstandene Ergebnis und bewertet, ob die ursprüngliche Zielsetzung erfüllt wurde. Des Weiteren wird betrachtet, wie es mit dem entstandenen Ergebnis weitergehen könnte.

### 6.1 Zusammenfassung der Ergebnisse

Es wurde anhand von theoretischen Konzepten, die teilweise auch in der Praxis schon Anwendung gefunden haben, ein System entwickelt, welches bestimmte Sicherheitsprobleme innerhalb des Softwareentwicklungsprozesses entdecken kann. Bei diesen Sicherheitsproblemen handelt es sich zum Einen um das versehentliche Veröffentlichen von Geheimnissen in Quelltexten und zum Anderen um das Verwenden von externen Bibliotheken, die bekannte Schwachstellen enthalten. Diese beiden Sicherheitsprobleme haben gemeinsam, dass man sie in statischen Quelltexten analysieren kann.

Bei der Konzeption und der Auswahl der Kernfunktionalitäten sind Prozesse und Herausforderungen aus aktuell verbreiteten Softwareentwicklungsprozessen mit bedacht worden, um eine Nutzung des Systems so realistisch wie möglich zu gestalten. Das Endprodukt ist ein Prototyp einer Webapplikation mit dem Arbeitstitel reposec.

### 6.2 Bewertung des Ergebnisses

Die Einfachheit der Einsetzbarkeit wurde in manchen Aspekten nur bedingt erreicht. Dies könnte durch eine vorgeschaltete Benutzerverwaltung verbessert werden, da so das Authentifizierungsproblem mit mehreren SSH Schlüsselpaaren pro Benutzer gelöst werden könnte. Des Weiteren ist eine Authentifizierungshürde sinnvoll, wenn das System im öffentlichen Internet verfügbar ist, was eine Voraussetzung für die Nutzung von Webhooks von öffentlichen Git Repository Anbietern wie Github ist.

Andere Aspekte der Einfachheit wurden wiederum erreicht durch die Integration von Github Webhooks und E-Mail Benachrichtigungen, die den modernen Softwareentwicklungsprozess nicht stören und trotzdem die Möglichkeit bieten, das Sicherheitslevel zu erhöhen.

Außerdem wäre technisch eine Auslieferung der Schnittstelle über HTTPS sowie ein Skalierbares Datenbanksystem hinter dem SQLAlchemy Adapter notwendig, um das System mit anderen Daten als Testdaten zu nutzen. Hierrauf wurde bei dem Prototypen verzichtet, da er nur in einem kontrollierten Umfeld oder mit Testdaten verwendet wird.

Über die Qualität der Erkennung der Geheimnisse kann keine Aussage getroffen werden, da keine Tests, mit angemessen großen Datensätzen durchgeführt wurden. In den Tests die durchgeführt wurden, wurden Geheimnisse immer erkannt. Es waren jedoch auch "False Positives" mit dabei.

Die Erkennung von bekannten Schwachstellen in externen npm Bibliotheken liefert sehr gute Ergebnisse.

### 6.3 Ausblick

In Zukunft könnte es an mehreren Punkten mit dieser Arbeit weiter gehen. Ein Ansatz wäre es, weitere Methoden zu entwickeln, um die Geheimniserkennung zu verbessern. Es könnten dann auch Studien mit großen Datensätzen durchgeführt werden und mit anderen Diensten für die Geheimniserkennung verglichen werden. Generell wäre es auch denkbar, weitere Sicherheitsaspekte zu prüfen, die sich im statischen Quelltext analysieren lassen, um somit die Software als Werkzeug für die Sicherheit im Softwareentwicklungsprozess zu erweitern. Die technischen Aspekte, die in der Bewertung der Ergebnisse angesprochen wurden, könnten implementiert werden, um somit aus dem Prototypen eine Plattform zu entwickeln, die in der Realität verwendbar ist. Hierfür müssten weitere Anbieter von Software Repositories und weitere Anbieter von Sammlungen von Softwarebibliotheken angebunden werden.

# Literaturverzeichnis

- [Ayrey 2018] AYREY, Dylan: *trufflehog*. November 2018. – URL <https://github.com/dxa4481/truffleHog>. – Zugriffsdatum: 2019-09-06
- [Bayer 2019] BAYER, Michael: *SQLAlchemy*. Oktober 2019. – URL <https://www.sqlalchemy.org/>
- [Bicking 2019] BICKING, Ian: *virtualenv*. Juli 2019. – URL <https://virtualenv.pypa.io/en/latest/>
- [Chacon und Straub 2019a] CHACON, Scott ; STRAUB, Ben: Getting Started - A Short History of Git. In: *Pro Git*. 2. Oktober 2019, S. 13–24
- [Chacon und Straub 2019b] CHACON, Scott ; STRAUB, Ben: Getting Started - What is Git. In: *Pro Git*. 2. Oktober 2019, S. 15
- [Docker 2019a] DOCKER, Inc: *Docker*. Juli 2019. – URL <https://www.docker.com/>
- [Docker 2019b] DOCKER, Inc: *Docker Compose*. Juli 2019. – URL <https://docs.docker.com/compose/>
- [Doll 2013] DOLL, Brian: *Secrets in the code*. Januar 2013. – URL <https://github.blog/2013-01-25-secrets-in-the-code/>. – Zugriffsdatum: 2019-07-08
- [Driessen 2010] DRIESSEN, Vincent: *A successful Git branching model*. Januar 2010. – URL <https://nvie.com/posts/a-successful-git-branching-model/>. – Zugriffsdatum: 2019-11-08
- [FIRST.Org 2019] FIRST.ORG, Inc: *Scoring System v3.0: Specification Document*. August 2019. – URL [https://www.first.org/cvss/v3.0/cvss-v30-examples\\_v1.5.pdf](https://www.first.org/cvss/v3.0/cvss-v30-examples_v1.5.pdf). – Zugriffsdatum: 2019-11-30

- [Fischer und Hofer 2010] FISCHER, Peter ; HOFER, Peter: *Lexikon der Informatik*. August 2010. – URL <https://link.springer.com/content/pdf/10.1007%2F978-3-642-15126-2.pdf>
- [Frochte 2019] FROCHTE, Jörg: Entscheidungsbäume. In: *Maschinelles Lernen Grundlagen und Algorithmen in Python*. 2. Januar 2019, S. 127–132. – ISBN 978-3-446-45996-0
- [Github 2019] GITHUB, Inc: *About pull requests*. 2019. – URL <https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests>. – Zugriffsdatum: 2019-12-01
- [Google 2019a] GOOGLE, LLC: *Angular*. Juli 2019. – URL <https://angular.io/>
- [Google 2019b] GOOGLE, LLC: *Angular Material*. Juli 2019. – URL <https://material.angular.io/>
- [Google 2019c] GOOGLE, LLC: *Google Mail*. 2019. – URL <https://www.google.com/intl/de/gmail/about/>
- [Google 2019d] GOOGLE, LLC: *Ich bin mir nicht sicher, ob ich einen API-Schlüssel für die Google Maps Plattform verwende. Wie kann ich das überprüfen?* 2019. – URL <https://cloud.google.com/maps-platform/user-guide/account-changes/?hl=de>. – Zugriffsdatum: 2019-11-30
- [Haustant 2019] HAUSTANT, Axel: *Flask-RESTPlus*. August 2019. – URL <https://flask-restplus.readthedocs.io/en/stable/>
- [Hipp 2019] HIPPE, Dwayne R.: *SQLite*. April 2019. – URL <https://www.sqlite.org/index.html>
- [Kashcha 2018] KASHCHA, Andrei: *Common words*. 2018. – URL <https://github.com/anvaka/common-words>. – Zugriffsdatum: 2019-10-08
- [Meli et al. 2019] MELI, Michael ; MCNIECE, Matthew R. ; REAVES, Bradley: How Bad Can It Get? Characterizing Secret Leakage in Public GitHub Repositories. San Diego, CA, USA, Februar 2019. – URL [https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019\\_04B-3\\_Meli\\_paper.pdf](https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf). – Zugriffsdatum: 2019-07-08
- [Microsoft 2019] MICROSOFT, Corp.: *Visual Studio Code*. Oktober 2019. – URL <https://code.visualstudio.com/>

- [MITRE Corporation 2019a] MITRE CORPORATION, The: *About CVE*. November 2019. – URL <https://cve.mitre.org/about/index.html>. – Zugriffsdatum: 2019-11-30
- [MITRE Corporation 2019b] MITRE CORPORATION, The: *About CWE*. April 2019. – URL <http://cwe.mitre.org/about/index.html>. – Zugriffsdatum: 2019-11-30
- [MITRE Corporation 2019c] MITRE CORPORATION, The: *CVE Numbering Authorities*. November 2019. – URL <https://cve.mitre.org/cve/cna.html>. – Zugriffsdatum: 2019-11-30
- [MITRE Corporation 2019d] MITRE CORPORATION, The: *CWE Community*. Mai 2019. – URL <https://cwe.mitre.org/community/index.html>
- [Moolenaar 2016] MOOLENAAR, Bram: *Vim*. September 2016. – URL <https://www.vim.org/>
- [NIST ] NIST, Nat. Inst. of Stand. and T.: *Common Vulnerability Scoring System Version 3.1 Calculator*. – URL <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>. – Zugriffsdatum: 2019-11-30
- [npm Inc. 2019] NPM INC.: *About npm*. 2019. – URL <https://docs.npmjs.com/about-npm/>. – Zugriffsdatum: 2019-10-13
- [OWASP Foundation 2019] OWASP FOUNDATION: *OWASP Top 10 -2017 Die 10 kritischsten Sicherheitsrisiken für Webanwendungen*. November 2019. – URL [https://www.owasp.org/images/9/90/OWASP\\_Top\\_10-2017\\_de\\_V1.0.pdf](https://www.owasp.org/images/9/90/OWASP_Top_10-2017_de_V1.0.pdf). – Zugriffsdatum: 2019-07-30
- [Pohl und Hof 2015] POHL, Christoph ; HOF, Hans-Joachim: *Secure Scrum: Development of Secure Software with Scrum*, URL [www.thinkmind.org%2Fdownload.php%3Farticleid%3Dsecureware\\_2015\\_1\\_50\\_30099&usg=AOvVaw1XSbli8SDcrhHA0HcfoRNq](http://www.thinkmind.org%2Fdownload.php%3Farticleid%3Dsecureware_2015_1_50_30099&usg=AOvVaw1XSbli8SDcrhHA0HcfoRNq), 2015
- [Python Software Foundation ] PYTHON SOFTWARE FOUNDATION, NPO: *smtplib*. – URL <https://docs.python.org/3/library/smtplib.html>
- [Rohr 2018] ROHR, Matthias: *Security Dependency Scanner (Software Composition Analysis)*. In: *Sicherheit von Webanwendungen in der Praxis: Wie sich Unternehmen schützen können – Hintergründe, Maßnahmen, Prüfverfahren und Prozesse*. URL <https://link.springer.com/content/pdf/10.1007%2F978-3-658-20145-6.pdf>, 2018, S. 389–390

- [Ronacher et al. 2019] RONACHER, Armin ; BRANDL, Georg ; LORD, David: *Flask*. 2019. – URL <https://www.palletsprojects.com/p/flask/>
- [Schirmacher 2018] SCHIRRMACHER, Dennis: Sicherheitslücke in jQuery-File-Upload Plug-in macht unzählige Server verwundbar. (2018), Oktober. – URL <https://www.heise.de/security/meldung/Sicherheitsluecke-in-jQuery-File-Upload-Plug-in-macht-unzaehlige-Server-verwundbar-4196771.html>
- [Schoen 2018] SCHOEN, Mbula: *Hype Cycle for Project and Portfolio Management, 2018*. Juli 2018. – URL <https://www.gartner.com/en/documents/3882878/hype-cycle-for-project-and-portfolio-management-2018>. – Zugriffsdatum: 2019-07-30
- [Schuberth et al. 2019] SCHUBERTH, Sebastian ; OMBREDANNE, Philippe ; SPRINGETT, Steve: *Package-URL Specification*. November 2019. – URL <https://github.com/package-url/purl-spec>. – Zugriffsdatum: 2019-12-01
- [Schwalbe 2015] SCHWALBE, Kathy: Project Constraints. In: *Information Technology Project Management*. 7-9. Cengage Learning, Oktober 2015, S. 38
- [Shannon 1951] SHANNON, Claude E.: Prediction and Entropy of Printed English. 30 (1951), Nr. 1, S. 50–64. – URL [https://www.princeton.edu/~wbialek/rome/refs/shannon\\_51.pdf](https://www.princeton.edu/~wbialek/rome/refs/shannon_51.pdf)
- [Sonatype Inc. 2019] SONATYPE INC.: *About sonatype OSS Index*. 2019. – URL <https://ossindex.sonatype.org/about>. – Zugriffsdatum: 2019-10-12
- [Stellman und Greene 2019] STELLMAN, Andrew ; GREENE, Jennifer: Was ist den dann nun Agile? In: *Agile Methoden von Kopf bis Fuß*. 1. 2019, S. 10. – ISBN 978-3-96009-079-3
- [Toth 2019] TOTH, Stefan: Abgleich mit der Realität. In: *Vorgehensmuster für Softwarearchitektur*. 3. 2019, S. 210
- [w3schools 2019] W3SCHOOLS: *Python RegEx*. September 2019. – URL [https://www.w3schools.com/python/python\\_regex.asp](https://www.w3schools.com/python/python_regex.asp). – Zugriffsdatum: 2019-09-24



# A Anhang

## A.1 Liste der verwendeten regulären Ausdrücke für Geheimniskandidatenerkennung

```

{
  "Slack Token": "(xox[p|b|o|a]-[0-9]{12}-[0-9]{12}-[0-9]{12}-[a-z0-9]{32})",
  "RSA private key": "-----BEGIN RSA PRIVATE KEY-----",
  "SSH (DSA) private key": "-----BEGIN DSA PRIVATE KEY-----",
  "SSH (EC) private key": "-----BEGIN EC PRIVATE KEY-----",
  "PGP private key block": "-----BEGIN PGP PRIVATE KEY BLOCK-----",
  "Amazon AWS Access Key ID": "AKIA[0-9A-Z]{16}",
  "Amazon MWS Auth Token": "amzn\\.mws\\. [0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}",
  "AWS API Key": "AKIA[0-9A-Z]{16}",
  "Facebook Access Token": "EAACEdEose0cBA[0-9A-Za-z]+",
  "Facebook OAuth": "[f|F][a|A][c|C][e|E][b|B][o|O][o|O][k|K].*['|\\"] [0-9a-f]{32} ['|\\"]",
  "GitHub": "[g|G][i|I][t|T][h|H][u|U][b|B].*['|\\"] [0-9a-zA-Z]{35,40} ['|\\"]",
  "Generic API Key": "[a|A][p|P][i|I][_]?[k|K][e|E][y|Y].*['|\\"] [0-9a-zA-Z]{32,45} ['|\\"]",
  "Generic Secret": "[s|S][e|E][c|C][r|R][e|E][t|T].*['|\\"] [0-9a-zA-Z]{32,45} ['|\\"]",
  "Google API Key": "AIza[0-9A-Za-z\\-]{35}",
  "Google Cloud Platform API Key": "AIza[0-9A-Za-z\\-]{35}",
  "Google Cloud Platform OAuth": "[0-9]+-[0-9A-Za-z]{32}\\\\.apps\\.googleusercontent\\.com",
  "Google Drive API Key": "AIza[0-9A-Za-z\\-]{35}",
  "Google Drive OAuth": "[0-9]+-[0-9A-Za-z]{32}\\\\.apps\\.googleusercontent\\.com",
  "Google (GCP) Service-account": "\"type\": \"service_account\"",
  "Google Gmail API Key": "AIza[0-9A-Za-z\\-]{35}",
  "Google Gmail OAuth": "[0-9]+-[0-9A-Za-z]{32}\\\\.apps\\.googleusercontent\\.com",
  "Google OAuth Access Token": "ya29\\.[0-9A-Za-z\\-]{35}+",
  "Google YouTube API Key": "AIza[0-9A-Za-z\\-]{35}",
  "Google YouTube OAuth": "[0-9]+-[0-9A-Za-z]{32}\\\\.apps\\.googleusercontent\\.com",
  "Heroku API Key": "[h|H][e|E][r|R][o|O][k|K][u|U].*[0-9A-F]{8}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{12}",
  "MailChimp API Key": "[0-9a-f]{32}-us[0-9]{1,2}",
  "Mailgun API Key": "key-[0-9a-zA-Z]{32}",
  "Password in URL": "[a-zA-Z]{3,10}://[^\\s:@]{3,20}:[^\\s:@]{3,20}@.{1,100}[\\'\\s]",
  "PayPal Braintree Access Token": "access_token\\$production\\$[0-9a-z]{16}\\$[0-9a-f]{32}",
  "Picatic API Key": "sk_live_[0-9a-z]{32}",
  "Slack Webhook": "https://hooks.slack.com/services/T[a-zA-Z0-9]{8}/B[a-zA-Z0-9]{8}/[a-zA-Z0-9]{24}",
  "Stripe API Key": "sk_live_[0-9a-zA-Z]{24}",
  "Stripe Restricted API Key": "rk_live_[0-9a-zA-Z]{24}",
  "Square Access Token": "sq0atp-[0-9A-Za-z\\-]{22}",
  "Square OAuth Secret": "sq0csp-[0-9A-Za-z\\-]{43}",
  "Twilio API Key": "SK[0-9a-fA-F]{32}",
  "Twitter Access Token": "[t|T][w|W][i|I][t|T][t|T][e|E][r|R].*[1-9][0-9]+-[0-9a-zA-Z]{40}",
  "Twitter OAuth": "[t|T][w|W][i|I][t|T][t|T][e|E][r|R].*['|\\"] [0-9a-zA-Z]{35,44} ['|\\"]"
}

```

Abbildung A.1: Benutzte Reguläre Ausdrücke im reposec System) <https://github.com/dxa4481/truffleHogRegexes/blob/master/truffleHogRegexes/regexes.json>

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

*Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI*

## Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: \_\_\_\_\_

Vorname: \_\_\_\_\_

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

### **Entwicklung eines Systems zur automatisierten statischen Sicherheitsüberprüfung von Software Repositories zur Integration in agile Softwareentwicklungsprozesse**

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

---

Ort

Datum

Unterschrift im Original