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

Konzept und Entwicklung eines Schwachstellenprüfers für Java-Bibliotheken

Masterarbeit

im Studiengang Informatik

von

Leif Erik Wagner

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Joel Greenyer
Betreuer: M. Sc. Fabien Patrick Viertel

Hannover, 24.10.2017

Erklärung der Selbstständigkeit

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

Hannover, den 24.10.2017

_____

Leif Erik Wagner

Zusammenfassung

Die Masterarbeit beschäftigt sich mit der automatisierten Schwachstellensuche für Java-Bibliotheken.

Eines der zehn größten Risiken für Webprojekte entsteht laut OWASP, wenn Bibliotheken mit bekannten Sicherheitslücken eingesetzt werden. Deshalb sollte eine Schwachstellensuche fester Teil des Entwicklungsprozesses sein. Sie ist jedoch zeitintensiv, insbesondere, wenn sie regelmäßig und von Hand durchgeführt wird.

Zur Lösung wurde ein Schwachstellenprüfer für Java-Bibliotheken in Form eines Eclipse-Plugin entwickelt, das die Schwachstellensuche automatisiert. Es basiert auf der National Vulnerability Database (NVD), eine der größten Datenbanken für bekannte Schwachstellen. Für die Suche werden Metadaten der Bibliothek ausgewertet und mit einer lokalen Kopie der NVD verglichen. Bei bestehender Internetverbindung wird die lokale Datenbank mit der NVD synchronisiert. Neben der NVD können externe SQLite-Datenbanken hinzugefügt werden. Dadurch ist es zum Beispiel möglich, auch auf internen Schwachstellendatenbanken eines Unternehmens zu suchen. Die Schwachstellensuche ist vom Benutzer konfigurierbar. Sie kann hinsichtlich des Recalls oder der Precision optimiert werden. Der Benutzer kann Suchergebnisse dauerhaft entfernen und einen Mindestschweregrad für Schwachstellen setzen.

Die Schwachstellensuche wird mithilfe des Eclipse-Plugins in den Entwicklungsprozess integriert. Der Entwickler kann frühzeitig vor Schwachstellen gewarnt werden. Damit steht ihm mehr Zeit zur Verfügung nach Alternativen für die Nutzung der unsicheren Bibliothek zu Suchen.

Abstract

This master thesis deals with the automation of a vulnerability checking process for Java libraries.

Known vulnerabilities in software libraries are one of the ten most important risks for web applications, as stated by the OWASP. Therefor searching for vulnerabilities needs to be part of the development process. But this can be a time consuming effort, especially when performed manually on a regular basis.

As a solution, this thesis presents a Vulnerability Checker for Java Libraries implemented as an Eclipse plug-in. It is based on the National Vulnerability Database (NVD) which contains information on publicly known vulnerabilities.

The plug-in uses the library's meta-data to search for vulnerabilities on a local copy of the NVD. The database will be updated when connected to the Internet. Next to the NVD, the plug-in is able to use external SQLite databases. By this way it takes advantage of non public databases like a company's bug tracking database. The search engine can be adjusted for a high recall, high precision or a combination of both. Additionally vulnerabilities with a low or medium risk can be masked.

By using the Vulnerability Checker for Java Libraries, the developer is able to reduce the risk originating from vulnerable libraries. The implementation as Eclipse plug-in archives the goal of integrating the vulnerability risk analysis in the development process. Vulnerability warnings will be presented in an early stage of the development.

Inhaltsverzeichnis

Akronyme

BSI Bundesamt für Sicherheit in der Informationstechnik. 13

CNA CVE Numbering Authority. 15

CPE Common Plattform Enumeration. 18

CVE Common Vulnerabilities and Exposures. 15

CVSS Common Vulnerability Scoring System. 17

FN False Negative. 57, 59

FP False Positive. 57, 59

NIST National Institute of Standards and Technology. 6, 16

NVD National Vulnerability Database. 16

TN True Negative. 59

TP True Positive. 57, 59

Kapitel 1

Einleitung

Immer mehr Software-Systeme sind mit dem Internet verbunden, befeuert durch die großen Trends „Industrie 4.0“ (vgl. PlattformIndustire 4.0 [28]) und „Internet of Things (IoT)“ (vgl. IoT-Analyse der BCG [33]). Dies birgt neben neuen Chancen aber auch Risiken. Jedes System, das mit dem Internet verbunden ist, kann ein Ziel von Cyber-Kriminellen werden. So berichtet das Bundesamt für Sicherheit in der Informationstechnik (BSI): „Studien zufolge werden pro Tag weltweit mehrere Tausend neue Computer gekapert und für fremde Zwecke missbraucht. Ein neu ans Internet angeschlossener PC wird bereits nach wenigen Minuten erstmals angegriffen.“ [1].

Für die Software-Entwicklung bedeutet dies, dass - insbesondere für Projekte aus dem Bereich Web, Industrie 4.0 oder IoT - dem ISO/IEC 9126 Qualitätsmerkmal Sicherheit größere Bedeutung zufällt. Ein großes und vermeidbares Risiko entsteht bei der Verwendung von unsicheren Bibliotheken. Das Open Web Application Security Project (OWASP) pflegt eine Liste der 10 häufigsten Sicherheitsrisiken für Webanwendungen. An neunter Stelle ist die „Nutzung von Komponenten mit bekannten Schwachstellen“ [27] gelistet. Genauer heißt es:

> „Komponenten wie z.B. Bibliotheken, Frameworks oder andere Softwaremodule werden meistens mit vollen Berechtigungen ausgeführt. Wenn eine verwundbare Komponente ausgenutzt wird, kann ein solcher Angriff zu schwerwiegendem Datenverlust oder bis zu einer Serverübernahme führen. Applikationen, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so zahlreiche Angriffe und Auswirkungen ermöglichen.“ [27].

Software-Entwickler müssen also bei der Wahl der Bibliotheken, die sie in einem Projekt verwenden wollen, sehr sorgfältig sein. Mit dieser Masterarbeit soll der Entwickler ein Werkzeug an die Hand bekommen, das ihm hilft, bei Java-Projekten unsichere Bibliotheken zu vermeiden. Hierfür wird ein Schwachstellenprüfer für Java-Bibliotheken konzeptioniert und entwickelt.

In Abschnitt 1.1 wird erläutert, auf welche Problematiken der Entwickler trifft, wenn er Risiken durch unsichere Bibliotheken vermeiden möchte. Im Abschnitt 1.2

wird der Lösungsansatz für die vorherige Problemstellung beschrieben. Schließlich folgt ein kurzer Überblick auf die Struktur der Arbeit in Abschnitt 1.3.

1.1 Problemstellung

Um sicherzugehen, dass eine Bibliothek zumindest keine bekannten Sicherheitslücken enthält, muss der Entwickler mindestens eine Schwachstellendatenbank durchsuchen, wenn nicht mehrere. Eine der bekanntesten ist die National Vulnerability Database (NVD). Über die NVD-Webseite lässt sich die Datenbank durchsuchen. Je nach Suchwort, können sich die gelieferten Ergebnisse über mehrere Seiten erstrecken. Der Entwickler muss dabei für jede Sicherheitslücke einzeln prüfen, ob sie in seinem Fall relevant ist. Diese mühsame Tätigkeit muss regelmäßig wiederholt werden, um auch vor neu veröffentlichten Sicherheitslücken gewarnt zu werden.

Der zeitintensiven Suche stehen enge Projektpläne gegenüber. Der Entwickler kann geneigt sein, immer die gleichen Bibliotheken aus früheren Projekten wiederzuverwenden. Hierbei spart er sich die Einarbeitung und Integration durch veränderte APIs. Doch er läuft dabei in die Gefahr, unsichere Software zu verwenden.

1.2 Lösungsansatz

Die händische Suche nach Sicherheitslücken soll mithilfe des Schwachstellenprüfers für Java-Bibliotheken automatisiert werden. Als Plugin für die Eclipse-Programmierumgebung, kann er Java-Bibliotheken selbständig finden und den Benutzer vor Sicherheitslücken warnen. Die Realisierung als Plugin hat den Vorteil, dass die Schwachstellensuche fest in den Entwicklungsprozess integriert wird. Wenn eine Sicherheitslücke gefunden wird, bekommt der Entwickler unmittelbares Feedback. Er kann noch vor Benutzung der Bibliothek nach Alternativen suchen und investiert keine verlorene Zeit mit (veralteten) APIs einer unsicheren Bibliothek.

Als Quelle für Sicherheitslücken, wurde die NVD ausgewählt, deren Daten sich frei herunterladen lassen. Damit ist es möglich, eine lokale Datenbank für Sicherheitslücken im Plugin zu pflegen. Der Schwachstellenprüfer funktioniert so auch ohne Internetverbindung, die lediglich zum Updaten der Datenbank notwendig ist.

Um relevante Sicherheitslücken in der Datenbank zu identifizieren, werden die gefundenen Java-Bibliotheken zunächst analysiert. Hierbei werden Metadaten aus dem JAR-Manifest der Bibliothek zusammen mit dem Dateinamen verwertet. Diese Daten werden verwendet, um möglich viele potenzielle Sicherheitslücken bei der Datenbankabfrage zu erhalten. Jede potenzielle Sicherheitslücke durchläuft einen Validierungsprozess, um nicht relevante Sicherheitslücken wieder zu verwerfen. Mit dieser Vorgehensweise soll der Recall der Sicherheitslücken erhöht

werden. Dabei kann es auch zu False-Positives kommen. Durch die Art und Weise, wie Sicherheitslücken präsentiert werden, sollen dem Benutzer False-Positives jedoch unmittelbar auffallen. Sie können durch wenige Mausklicks schnell entfernt und für zukünftige Suchen ausgeschlossen werden. Hier zeigt sich auch erneut der Vorteil gegenüber einer händischen Suche. Der Benutzer hat nur einmal die Arbeit, nicht relevante Sicherheitslücken auszusortieren, während er dies bei einer manuellen Onlinesuche stets erneut durchführen muss.

Der Schwachstellenprüfer ist des Weiteren in der Lage, externe SQlite Datenbanken zu importieren. Somit können zum Beispiel im Unternehmen spezialisierte Datenbanken zum Beispiel für intern entwickelte Bibliotheken erstellt und genutzt werden.

1.3 Struktur der Arbeit

Diese Arbeit wird folgenderweise strukturiert: In Kapitel 2 wird detailliert auf die Anforderungen an den Schwachstellenprüfer und die Stakeholder eingegangen.

Das Kapitel 3 vermittelt die Grundlagen, wie Sicherheitslücken beschrieben werden und welche die wichtigsten Datenbanken hierfür sind. Des Weiteren werden die verfügbaren Metadaten einer Java-Bibliothek vorgestellt.

Anschließend wird in Kapitel 4 das Konzept des Schwachstellenprüfers beschrieben.

In Kapitel 5 werden Architektur und die implementierten Funktionen inklusive der Benutzeroberfläche des Plugins vorgestellt.

Eine Evaluation des Schwachstellenprüfers erfolgt in Kapitel 6.

Schließlich werden in Kapitel 7 die wichtigsten Inhalte und Ergebnisse dieser Arbeit zusammengefasst.

Kapitel 2

Anforderungen

Dieses Kapitel behandelt die Anforderungen an den Bibliotheks-Schwachstellenprüfer. Zunächst werden in dem Abschnitt 2.1 die Stakeholder vorgestellt. Es folgt ein Überblick der gesammelten Anforderungen in Abschnitt 2.2. Abschließend werden in Abschnitt 2.3 die Anforderungen priorisiert und in Abschnitt 2.4 die Arbeitspakete definiert.

2.1 Stakeholder

Die Zielgruppe des Schwachstellenprüfers sind Software-Entwickler. Diese sollen vor der Verwendung von unsicheren Java-Bibliotheken während der Implementierung geschützt werden. Für die Entwickler ist es wichtig, frühzeitig über Risiken der verwendeten Bibliotheken informiert zu werden. Sie haben dann bei Bedarf noch Zeit, nach sicheren Alternativen zu suchen. Für den Entwickler ist es wichtig, dass der Schwachstellenprüfer sie bei der eigentlichen Arbeit nicht behindert. Ständige falsche Warnungen stören den Arbeitsfluss und führen dazu, dass der Benutzer das Programm wieder entfernt. Auch darf der Schwachstellenprüfer die Hardwareressourcen nicht dermaßen beanspruchen, dass der Computer träge und langsam wird.

Eine weitere Nutzergruppe sind Menschen, die verantwortlich für Qualität und Sicherheit der Software sind. Dies können der Security-Tester oder Security Champion sein, wie Mathias Rohr in seinem Artikel „Sicherheit im Software-Entwicklungsprozess“ [30] beschreibt. In ihrer Tätigkeit müssen sie Risiken der Software identifizieren. Sie könnten den Schwachstellenprüfer benutzen, um alle Sicherheitslücken in den externen Komponenten aufzuspüren. Dabei werden falsche Warnungen akzeptiert, wenn diese schnell als solche zu erkennen sind und sie nicht durch ein zu häufiges Vorkommen die wahren Sicherheitslücken überlagern. Die Suche nach Sicherheitslücken mit dem Schwachstellenprüfer soll nicht mit der Suche der „Nadel im Heuhaufen“ vergleichbar sein. Sie profitieren von einer Präzisierung der Suche, zum Beispiel nach dem Schweregrad der, von der Sicherheitslücke ausgehenden, Gefahr.

Der dritte Stakeholder ist das Fachgebiet für Software Engineering. Sie möchten den Schwachstellenprüfer, ganz oder in Teilen, für weitere Forschungsprojekten verwenden. Für sie ist eine gut strukturierte und kommentierte Software wichtig, die von nicht am Projekt beteiligten Mitarbeitern oder Studenten um neue Funktionen erweitert werden kann.

2.2 Überblick

Im Folgenden werden die Anforderungen an den Schwachstellenprüfer, die von dem Fachgebiet Software Engineering gefordert wurden, vorgestellt. Funktionelle Anforderungen werden mit FR abgekürzt, qualitative Anforderungen mit QR.

[FR.1] Der Schwachstellenprüfer ist ein Eclipse-Plugin.

Die Suche nach Sicherheitslücken für verwendete Bibliotheken soll Teil des Entwicklungsprozesses werden. Deshalb ist es notwendig, diesen Prozess in die tägliche Arbeit der Software-Entwickler zu integrieren. Eclipse ist eine Entwicklungsumgebung, die häufig von Java-Entwicklern verwendet wird. Sie ist durch Plugins erweiterbar. Eclipse wird in der Version Neon verwendet.

[FR.2] Der Schwachstellenprüfer findet Java-Bibliotheken aus geöffneten Java-Projekten.

Der Schwachstellenprüfer findet selbständig alle Java-Bibliotheken aus geöffneten Java-Projekten. Dabei ist es unerheblich, ob das Java-Projekt schon vor Installation des Plugins vorhanden war, neu erstellt oder importiert wurde.

[FR.3] Der Schwachstellenprüfer durchsucht eine CVE-Datenbank nach Sicherheitslücken für gefundene Java-Bibliotheken.

Der Schwachstellenprüfer sollte entweder die „Common Vulnerability Databse“[1] der gemeinützigen MITRE Organisation oder die „National Vulnerability Database“[2] des National Institute of Standards and Technology (NIST) verwenden. Beide Datenbanken verwenden Common Vulnerability and Exposures (CVE) zur Referenzierung von Sicherheitslücken.

[FR.3.1] Die Genauigkeit der Schwachstellensuche (Precision) ist konfigurierbar.

Die Anzahl der gefundenen Sicherheitslücken und die Anzahl falscher Warnungen sind miteinander verbunden. Wenn weniger falsche Warnungen produziert werden sollen, sinkt auch die Anzahl der gefundenen Sicherheitslücken. Je nach Benutzer

[1] https://cve.mitre.org
[2] https://nvd.nist.gov

können hier unterschiedliche Präferenzen bestehen. Deshalb soll der Benutzer die Möglichkeit haben, zwischen verschiedenen Konfigurationen zu wählen.

[FR.3.2] Für die Schwachstellensuche kann ein Mindestschweregrad festgelegt werden.

Der Schweregrad beschreibt, ob von der Sicherheitslücke eine hohe, mittlere oder geringe Gefahr ausgeht. Ein Benutzer kann entsprechend seinen Anforderungen entscheiden, ob der Schwachstellenprüfer alle Sicherheitslücken anzeigen soll oder nur jene die mindestens einen mittleren oder hohen Schweregrad haben.

[FR.4] Der Schwachstellenprüfer nutzt, bei bestehender Internetverbindung, eine aktuelle Datenbank.

Die Suche nach Sicherheitslücken soll immer auf Basis der aktuellen CVE-Datenbank beruhen, damit der Benutzer auch vor neuesten Bedrohungen frühestmöglich informiert wird.

[FR.5] Der Schwachstellenprüfer warnt den Benutzer, wenn eine Sicherheitslücke gefunden wurde.

Wenn eine Sicherheitslücke gefunden wurde, soll der Benutzer eine Warnung für diese Bibliothek erhalten, um auf das Risiko, dass bei deren Verwendung auftritt, aufmerksam zu werden. Eine Markierung von unsicheren Bibliotheken im Package-Explorer von Eclipse ist gefordert.

[FR.5.1] Durch eine farbliche Kodierung im Package-Explorer werden unsichere Java-Bibliotheken in drei Schweregrade kategorisiert.

Ausgehend von dem höchsten Schweregrad aller Sicherheitslücken einer Bibliothek, wird die Bibliothek im Package-Explorer von Eclipse farblich hervorgehoben. Bei einem geringen Schweregrad wird der Hintergrund gelb, bei einem mittlerem Schweregrad orange und bei einem hohen Schweregrad rot eingefärbt, um den Benutzer zu warnen.

[FR.6] Der Schwachstellenprüfer listet eine Zusammenfassung der Sicherheitslücken auf.

Der Benutzer soll eine Übersicht über die gefundenen Sicherheitslücken und die zugehörigen Bibliotheken bekommen. Diese Übersicht wird stets aktualisiert, wenn Sicherheitslücken oder Bibliotheken hinzugefügt oder entfernt werden.

[FR.6.1] Die Zusammenfassung kann sich auf Sicherheitslücken mit einem Mindestschweregrad beschränken.

Der Benutzer kann einen Mindest-CVSS-Score für die Zusammenfassung auswählen. Alle Sicherheitslücken, die unter dem Score liegen, werden ausgeblendet.

[FR.6.2] Die Zusammenfassung kann sich auf neue Sicherheitslücken beschränken.

Die Zusammenfassung verfügt über einen Filter, um nur neue Sicherheitslücken anzuzeigen. Eine vorher unbekannte Sicherheitslücke wird bis zum Beenden von Eclipse als neu eingestuft.

[FR.6.3] Die Genauigkeit (Precision) der Zusammenfassung kann durch Filter angepasst werden.

Für die Bestimmung der Filter werden auf die gewonnenen Erkenntnisse aus [FR.3.1] zurückgegriffen. Der Unterschied zu [FR.3.1] besteht zum einen in der feineren Granularität der Filterung und darauf, dass sich der Filter nur auf die Zusammenfassung der Sicherheitslücken beschränkt.

[FR.7] Der Benutzer kann die Beschreibung der Sicherheitslücken im Schwachstellenprüfer einsehen.

CVEs enthalten eine kurze Beschreibung der Sicherheitslücke. Mithilfe des Schwachstellenprüfer soll der Benutzer die Beschreibung der Sicherheitslücke in der Eclipse-Entwicklungsumgebung lesen können um das von der Sicherheitslücke ausgehende Risiko einschätzen zu können.

[FR.8] Der Schwachstellenprüfer kann um externe SQLite-Datenbanken erweitert werden.

Es soll möglich sein, weitere externe Datenbanken über Sicherheitslücken hinzuzufügen. Dies können zum Beispiel unternehmenseigene Datenbanken über Bibliotheken sein, die in dem Unternehmen verwendet werden.

[FR.8.1] Optional: Der Benutzer muss für den Import der SQLite-Datenbank keine SQL-Statements schreiben.

Es wird gewünscht, für den Import der SQLite Datenbank, keine SQL-Statements schreiben zu müssen. Das Mapping des externen SQL-Schemas auf die interne Datenstruktur des Schwachstellenprüfers, soll über eine GUI mit Mausbedienung durchgeführt werden.

[QR.1] Der Schwachstellenprüfer darf die Eclipse-UI nicht blockieren.

Wenn der Schwachstellensucher Prozesse mit einem erhöhten Ressourcenverbrauch ausführt - erfassen der Java-Bibliotheken, updaten der Datenbank, suchen nach Sicherheitslücken - muss die UI voll funktionsfähig bleiben. Das heißt, sie muss auf Benutzereingaben reagieren. Der Software-Entwickler soll durch diese Prozesse nicht von seiner Arbeit abgehalten werden.

[QR.2] Das Updaten der Datenbank soll nicht länger als 15 Minuten benötigen.

Ein Updatevorgang soll in einem akzeptablen Zeitrahmen durchgeführt werden. Als akzeptabel wurde ein Limit von bis zu 15 Minuten gesetzt. Je schneller der Vorgang abgeschlossen ist, desto besser.

[QR.3] Die Suche nach Schwachstellen soll schnellstmöglich abgeschlossen sein.

Die Schwachstellensuche für eine Java-Bibliothek soll möglichst schnell abgeschlossen sein, damit der Benutzer frühzeitig gewarnt werden kann, wenn Sicherheitslücken gefunden wurden. Dadurch soll verhindert werden, dass der Benutzer keine unnötige Arbeit mit einer unsicheren Bibliothek verschwendet.

2.3 Priorisierung

Die ermittelten Anforderungen wurden mit Hilfe des Kano-Modells der Kundenzufriedenheit (vgl Kano-Theorie der Kundenzufriedenheitsmessung von Jörg A. Hölzing [8]) priorisiert. Das Kano-Modell der Kundenzufriedenheit beschreibt den Einfluss von Kundenanforderungen auf ihre Zufriedenheit. Dabei werden die Kundenanforderungen in die Kategorien in Basis-, Leistungs- und Begeisterungsanforderungen differenziert.

Basisanforderungen müssen von dem Produkt erfüllt werden, anderenfalls kann es keine Kundenzufriedenheit geben. Die Schwierigkeit besteht darin, Basisanforderungen zu identifizieren, weil der Kunde sie als selbstverständlich annehmen kann. Die volle Erfüllung von Basisanforderungen führt nicht zu Kundenzufriedenheit, wie man Abbildung 2.1 entnehmen kann.

Kundenzufriedenheit wird erst mit Erfüllung der Leistungsanforderungen erreicht. Dies wird im Kano-Modell als Gerade dargestellt. Je mehr Leistungsanforderungen erfüllt werden, desto zufriedener wird der Kunde. Im Gegensatz zu den Basisanforderungen, werden Leistungsanforderungen explizit von dem Kunden eingefordert.

Unter die dritte Kategorie fallen die Begeisterungsanforderungen. Dies sind Anforderungen, die dem Kunden noch gar nicht bewusst waren, dass er sie braucht. Begeisterungsanforderungen heben stark die Kundenzufriedenheit. Ein Produkt

kann sich bei Erfüllung dieser Anforderungen von der Konkurrenz hervorheben.

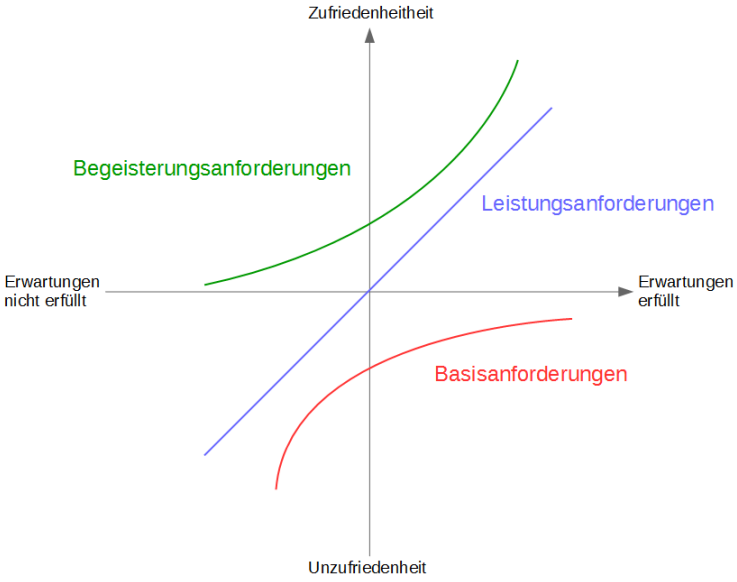

Abbildung 2.1: Kano-Modell der Kundenzufriedenheit (Quelle: vgl Kano-Theorie der Kundenzufriedenheitsmessung von Jörg A. Hölzing [8] S. 85)

Basisanforderungen des Schwachstellenprüfers bekommen eine hohe Priorität. Sie müssen im fertigen Programm erfüllt werden. Leistungsanforderungen bekommen je nach Wichtigkeit eine mittlere bis hohe Priorität. Begeisterungsanforderungen bekommen eine geringe bis mittlere Priorität. Sie werden zum Schluss realisiert, wenn die für den Auftraggeber wichtigen Anforderungen erfüllt sind.

Tabelle 2.1 zeigt die Klassifizierung der Anforderungen nach dem Kano-Modell. Die Qualitätsanforderungen 1 bis 3 stellen Basisanforderungen dar. Wenn diese nicht eingehalten werden, ist der Schwachstellenprüfer oder die Entwicklungsumgebung nicht oder nur eingeschränkt zu verwenden. Damit alle Qualitätsanforderungen erfüllt werden, bekommen sie eine hohe Priorität. Die funktionale Anforderung [FR.1], der Realisierung des Schwachstellenprüfers als Eclipse-Plugin, ist eine geforderte Leistungsanforderung des Auftraggebers. Sie impliziert eine Einarbeitung in das Eclipse-Plugin-Framework und bekommt eine hohe Priorität, weil alle weiteren funktionalen Anforderungen auf diese aufbauen. Die Anforderungen [FR.2] bis [FR.5], sowie [FR.6] und [FR.7] bis [FR.8] sind ebenfalls Leistungsanforderungen. Es handelt sich um Kernfunktionalitäten, weshalb sie ebenfalls eine hohe Priorität bekommen. Die Leistungsanforderungen [FR.3.1] und [FR.3.2] erweitern diese Kernfunktionen durch eine Verbesserung

Anforderung	Klassifizierung	Priorisierung
QR.1	Basisanforderung	Hoch
QR.2	Basisanforderung	Hoch
QR.3	Basisanforderung	Hoch
FR.1	Leistungsanforderung	Hoch
FR.2	Leistungsanforderung	Hoch
FR.3	Leistungsanforderung	Hoch
FR.3.1	Leistungsanforderung	Mittel
FR.3.2	Leistungsanforderung	Mittel
FR.4	Leistungsanforderung	Hoch
FR.5	Leistungsanforderung	Hoch
FR.6	Leistungsanforderung	Hoch
FR.7	Leistungsanforderung	Hoch
FR.8	Leistungsanforderung	Hoch
FR.8.1	Begeisterungsanforderung	Mittel
FR.5.1	Begeisterungsanforderung	Niedrig
FR.6.1	Begeisterungsanforderung	Niedrig
FR.6.2	Begeisterungsanforderung	Niedrig
FR.6.3	Begeisterungsanforderung	Niedrig

Tabelle 2.1: Einordnung nach Kano-Modell und Priorisierung

der Usability. Sie werden nachfolgend realisiert und erhalten deshalb eine mittlere Priorität. Die übrigen Anforderungen gehören der Kategorie Begeisterungsanforderungen an. Anforderung [FR.8.1], der Import einer externen Datenbank via GUI, ist optional, führt aber zu sehr hoher Kundenzufriedenheit und deshalb erhält sie eine mittlere Priorität. Die verbleibenden Usability-Anforderungen [FR.5.1], [FR.6.1], [FR.6.2] und [FR.6.3] werden zum Schluss realisiert, wenn genügend Zeit zur Verfügung steht.

2.4 Arbeitspakete

Die priorisierten Anforderungen werden nachfolgend in Arbeitspakete eingeteilt. Diese Arbeitspakete bauen aufeinander auf und können deshalb nicht in beliebiger Reihenfolge bearbeitet werden. Sie sind sortiert nach Priorität. Das erste Arbeitspaket hat die höchste Priorität. So soll erreicht werden, dass die wichtigsten Anforderungen zuerst umgesetzt werden.

Eclipse-Grundlagen

Zur Erfüllung der Anforderungen [FR.1] und [QR.1] erfolgt eine Einarbeitung in das Eclipse-Plugin-Framework.

CVE-Datenbank

Eine lokale CVE-Datenbank wird für die Schwachstellensuche benötigt. Für die Erstellung und das Updaten der Datenbank müssen die CVE-Daten aus dem Internet heruntergeladen und verarbeitet werden. Dies betrifft die Anforderungen [FR.4] und [QR.2].

Schwachstellensuche

Es folgt die eigentliche Schwachstellensuche auf der lokalen CVE-Datenbank. Hierfür werden die Anforderungen [FR.2], [FR.3] und [QR.3] umgesetzt.

Ergebnis-Präsentation

Die Ergebnisse der Schwachstellensuche werden entsprechend der Anforderungen [FR.5], [FR.6] und [FR.7] dem Benutzer angezeigt.

Import externer Datenbank

Das importieren einer externen Datenbank [FR.8] ist die letzte verbleibende Kernfunktionalität. Die optionale Anforderung [FR.8.1] wird ebenfalls im Arbeitspaket umgesetzt, da sich der abgeschätzte Arbeitsaufwand im Rahmen hält.

Usability-Funktionen

Es folgt das Feintuning des Schwachstellenprüfers, um die Usability zu erhöhen. Hierfür werden die verbliebenen Anforderungen [FR.5.1], [FR.6.1], [FR.6.2] und [FR.6.3] umgesetzt.

Kapitel 3

Grundlagen

In diesem Kapitel sollen die Grundlagen vermittelt werden, die zum Verstehen der Arbeitsweise des Schwachstellenprüfers für Java-Bibliotheken benötigt werden. Hierfür wird zu Beginn im Abschnitt 3.1 der Begriff Schwachstelle definiert und der Lebenszyklus einer Schwachstelle erläutert. Es folgt die Vorstellung der Common Vulnerabilities and Exposures in Abschnitt 3.3.2, die eindeutige IDs für Schwachstellen vergibt, damit diese über verschiedene Datenbanken hinweg einheitlich referenziert werden können. Im Abschnitt 3.3 wird einer der wichtigsten Schwachstellendatenbanken, die National Vulnerability Database (NVD), vorgestellt. Die NVD wird von dem Schwachstellenprüfer, für die Suche nach Sicherheitslücken, verwendet. Anschließend befasst sich der Abschnitt 3.4 mit den verfügbaren Metadaten der Java-Bibliotheken.

3.1 Schwachstellen

Zu Beginn soll die Frage beantwortet werden, was in dieser Arbeit unter dem Begriff Schwachstelle verstanden wird. Hierfür möchte ich auf die Common Vulnerabilities and Exposures (CVE) der gemeinnützigen MITRE Corporation verweisen, die laut Bundesamt für Sicherheit in der Informationstechnik (BSI) „die einzige weltweit anerkannte Quelle für die Veröffentlichung neuer Schwachstellen“ ist [2] S. 9. CVE definiert eine Schwachstelle folgendermaßen:

> „A ’vulnerability’ is a weakness in the computational logic (e.g., code) found in software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact to confidentiality, integrity, OR availability. Mitigation of the vulnerabilities in this context typically involves coding changes, but could also include specification changes or even specification deprecations (e.g., removal of affected protocols or functionality in their entirety).“ [14].

In dieser Arbeit werden die Begriffe Schwachstelle und Sicherheitslücke synonym verwendet.

3.1.1 Lebenszyklus von Schwachstellen

Der Lebenszyklus von Schwachstellen kann nach Frei et al [6] in sechs Phasen eingeteilt werden: Creation, Discovery, Exploit, Disclosure, Patch-Available und Patch-Installed (siehe Abbildung 3.1. Von der Entdeckung (Discovery) bis zur Fehlerbehebung (Patch-Installed) ist die Schwachstelle von Angreifern ausnutzbar. Frei et al. unterteilt dabei das von der Schwachstelle ausgehende Risiko in das Pre-Disclosure-Risk, der Post-Disclosure-Risk und der Post-Patch-Risk. Das Pre-Disclosure-Risk beginnt ab dem Zeitpunkt, sobald die Schwachstelle von einer Person erkannt wurde, bis zur Veröffentlichung. In diese Risikogruppe fallen Zero-Day-Exploits. Dies sind Exploits, die bereits vor- oder am selben Tag der Veröffentlichung die Schwachstelle ausnutzen. Von dem Zeitpunkt der Veröffentlichung bis zu dem Zeitpunkt, in der ein Patch zur Beseitigung der Schwachstelle zur Verfügung gestellt wird, gilt das Post-Disclosure-Risk. Das Post-Patch-Risk gilt schließlich ab Verfügbarkeit einer Fehlerbehebung (Patch), bis zu dem Zeitpunkt, in dem der Patch installiert wurde.

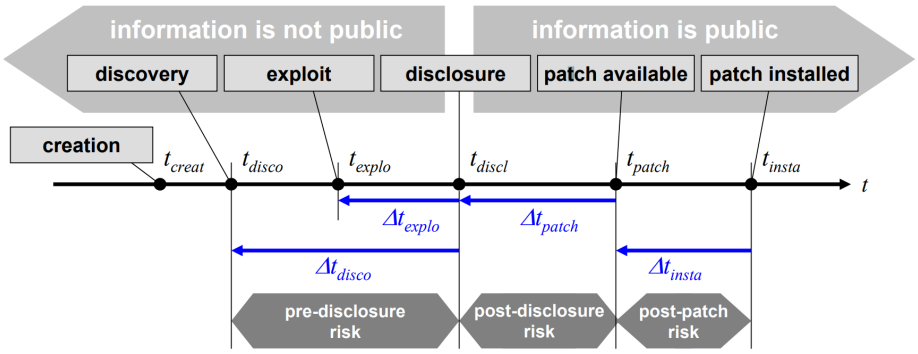

Abbildung 3.1: Lebenszyklus von Schwachstellen (Quelle: Frei et al. [6])

Der Schwachstellenprüfer basiert auf veröffentlichte Sicherheitslücken via CVE. Er schützt somit zum Einen vor Sicherheitslücken aus der Gruppe der Post-Disclosure-Risk. Der Benutzer erhält in diesem Fall eine Warnung und kann nach alternativen Bibliotheken Ausschau halten. Ebenso schützt er vor Post-Patch-Risk Sicherheitslücken. Hier kann der Benutzer auf eine fehlerbereinigte Version der Bibliothek zurückgreifen. Sicherheitslücken der Pre-Disclosure-Risk lassen sich nicht mittels der CVE-Datenbank beseitigen. Wenn ein Benutzer oder Unternehmen jedoch im Besitz von Informationen über nicht veröffentlichte Schwachstellen ist, kann dieses Wissen über den Import als SQLite-Datenbank dem Schwachstellenprüfer zur Verfügung gestellt werden.

3.2 Common Vulnerabilities and Exposures

Im Folgenden werden die Common Vulnerabilities and Exposures erläutert (vgl CVE-Webseite [12]).

Vor Entstehung der CVE herrschte die Problematik, wie man Sicherheitslücken, die in unterschiedlichen Datenbanken veröffentlicht wurden, miteinander vergleichen kann. So konnte es verschiedene Beschreibungen für ein und dieselbe Sicherheitslücke geben, ohne, dass dies offensichtlich war. Dies erschwert die Arbeit für Personen oder Werkzeuge, die alle Informationen zu einer Sicherheitslücke sammeln wollen.

Als Lösung hierfür wurden die Common Vulnerabilities and Exposures entwickelt. Die CVE ist eine Liste von veröffentlichten Sicherheitslücken, auf die Datenbanken für Sicherheitslücken oder Sicherheitswerkzeuge referenzieren können. Sie selbst bezeichnet sich als „dictionary rather than a database“. Aus dem Grund enthalten CVE-Einträge auch keine weitergehenden Analysen, wie eine Risikobewertung. Stattdessen verweist sie auf andere Datenbanken. Die CVE nimmt keine Sicherheitslücken in ihre Liste auf, bevor diese an anderer Stelle veröffentlicht wurde, wie sie in ihren Regeln zur Vergabe von CVE-IDs festlegt (vgl.CVE Numbering Authority (CNA) Rules [15] S. 7). Die CVE-Liste ist frei verfügbar und kann über die CVE-Webseite heruntergeladen werden[1].

Ein CVE-Eintrag besteht aus den in Tabelle 3.1 beschriebenen Datenfeldern. Der wichtigste Punkt ist die eindeutige CVE-ID, auch CVE-Name oder CVE-Number genannt. Sie referenziert genau eine Sicherheitslücke. Eine CVE-ID besteht aus dem Präfix CVE, gefolgt von dem Jahr der erstmaligen Veröffentlichung der Sicherheitslücke und einer fortlaufenden Nummer, zum Beispiel „CVE-2017-12611“. Das Datum der Veröffentlichung in der CVE wird im Feld Date Entry Created hinterlegt. Jeder CVE-Eintrag enthält zudem eine Beschreibung der Sicherheitslücke. MITRE stellt für die Beschreibung zwei Vorlagen zur Verfügung, die benutzt werden können:

> „**VULNTYPE** in **COMPONENT** in **VENDOR PRODUCT VERSION** allows **ATTACKER** to **IMPACT** via **VECTOR**.
>
> **COMPONENT** in **VENDOR PRODUCT VERSION ROOT CAUSE**, which allows **ATTACKER** to **IMPACT** via **VECTOR**.“, aus Key Details Phrasing [13] S.4.

Wenn eine autoritative Quelle die Gültigkeit der Schwachstelle bezweifelt, wird der Beschreibung „** DISPUTED **“ vorangestellt (vgl. CVE Numbering Authority (CNA) Rules [15] S. 22). Ein weiteres wichtiges Feld des CVE-Eintrages sind Referenzen auf Datenbanken, Bestätigungen der Sicherheitslücke und weitere Informationen. CVE-IDs für Sicherheitslücken werden durch eine CVE Numbering Authority (CNA) zugeteilt. CNA sind Organisationen, mit der Berechtigung, CVE-

[1] `https://cve.mitre.org/data/downloads/index.html`

Einträge zu erstellen. Das können große Projekte, Hersteller oder Sicherheitsforscher sein. Beispiele hierfür sind die Apache Software Foundation oder Apple Inc. Als sogenannte Primary CNA steht die MITRE Corporation an oberster Stelle aller CNAs. Sicherheitsforscher oder Unternehmen, die eine Sicherheitslücke melden wollen, müssen bei einer CNA einen Antrag für eine CVE-ID stellen. Welche CNA eine CVE-ID erstellt hat, ist unter dem Feld Assigning CNA des CVE-Eintrages dokumentiert. Die restlichen Felder der Tabelle 3.1 sind Legacy-Einträge und werden nicht mehr verwendet.

CVE-Datenfelder
CVE-ID
Description
References
Assigning CNA
Date Entry Created
Phase (Legacy)
Votes (Legacy)
Comments (Legacy)
Proposed (Legacy)

Tabelle 3.1: Bestandteile eines CVE-Eintrags

3.3 National Vulnerability Database

Eine der größten Datenbanken für Sicherheitslücken ist die National Vulnerability Database (NVD), wie Homaei und Shahriari in ihrem Artikel „Seven Years of Software Vulnerabilities“ [7] aussagen. Sie wird betrieben von der National Institute of Standards and Technology (NIST), einer Abteilung des U.S. Department of Commerce. Die NVD ist synchronisiert mit der CVE, das heißt, jeder CVE-Eintrag wird in die NVD aufgenommen. Wenn ein neuer CVE-Eintrag erstellt wurde, dann analysiert die NVD diesen innerhalb von zwei Werktagen mit Ausnahme von Feiertagen (vgl. NVD-FAQ [19]). Die NVD zeichnet aus, dass jeder Eintrag der Datenbank eine Risikoabschätzung nach Common Vulnerability Scoring System (CVSS) erhält (vgl. „General Information“ [17] der NVD). Eine Erläuterung der CVSS erfolgt in Abschnitt 3.3.1. Der zweite wichtige Punkt ist die Referenzierung der von der Schwachstelle verwundbaren Software, über Common Plattform Enumeration (CPE). Zusammen mit der freien Verfügbarkeit als Download[2] bietet sie eine gute Grundlage für die automatisierte Suche nach Schwachstellen. Aufgrund der großen Bedeutung der CPEs für den Schwachstellenprüfer für Java-Bibliotheken, werden diese detailliert in Abschnitt 3.3.2 vorgestellt. Die Webseite der NVD bietet eine Möglichkeit, die Datenbank zu durchsuchen und Einträge darzustellen.

[2] `https://nvd.nist.gov/vuln/data-feeds`

3.3.1 Common Vulnerability Scoring System

Das Common Vulnerability Scoring System (CVSS) ist ein offenes Framework, mit dem Charakteristiken und Auswirkungen von Schwachstellen bestimmt werden können. CVSS ist unterteilt in die Base-, Temporal- und Environmental-Metrik. Die Base-Metriken beschreibt die intrinsischen Merkmale der Schwachstelle, das heißt wie sie von einem Angreifer ausgenutzt werden kann und welche Auswirkungen sich ergeben. In die Temporal-Metriken fließen Risikofaktoren ein, die sich im Laufe des Lebenszyklus der Schwachstelle ändern. Darunter fallen verfügbare Exploits, Bestätigung der Schwachstelle und Bugfixes. Die optionalen Environmental-Metriken beschreiben die Auswirkungen der Schwachstelle auf die Umwelt. Für jede Gruppe wird ein Score von 0,0 bis 10,0 berechnet und ein zusammenfassender Vektor der Merkmale erstellt. Der Vorteil der CVSS ist, dass sie auf unterschiedliche Software- und Hardware-Plattformen angewendet werden können und damit ein einheitliches Bewertungsverfahren für das Risikomanagement besteht (vgl. CVSS-Guide [10]).

Die aktuelle Version von CVSS ist 3.0. Die XML-Datafeeds enthalten jedoch nur die Vorgängerversion 2.0, deren Scores nicht direkt mit denen aus Version 3.0 verglichen werden können. Im Folgenden werden die für den Schwachstellenprüfer relevanten Elemente des CVSS Version 2.0 vorgestellt.

Die verfügbaren XML-Datafeeds enthalten nur die Base-Metriken der CVSS. Diese beschreiben mittels Access Vector, Access Complexity und Authentication, wie ein Angreifer die Sicherheitslücke ausnutzen kann. Die Metriken Confidentiality Impact, Integrity Impact und Availability Impact beschreiben die Auswirkungen eines möglichen Exploits. Basierend auf den CVSS-Base-Score unterscheidet die NVD zwischen den drei Schweregraden aus Tabelle 3.2.

Schweregrad	Base-Score-Intervall
Gering	0,0 - 3,9
Mittel	4,0 - 6,9
Hoch	7,0 - 10,0

Tabelle 3.2: NVD-Schweregrade [21]

3.3.2 Common Platform Enumeration

Bei der Dokumentation von Sicherheitslücken, ist ein wesentlicher Bestandteil die Referenz auf das verwundbare System oder Software. In den CVEs ist dies Teil der Beschreibung. Für eine systematische Auswertung wird eine standardisierte und maschinenlesbare Referenzierung benötigt. Auf Grundlage der Standardisierung können zwei Objekte auf Gleichheit überprüft werden, zum Beispiel die referenzierte Bibliothek in der Schwachstellendatenbank, mit der verwendeten Bibliothek des Software-Projekts.

Die Common Plattform Enumeration (CPE) wurde genau für diesen Zweck entwickelt. CPE bietet ein standardisiertes und maschinenlesbares Format für Namen von IT-Produkten und Plattformen. Sie definiert Regeln, mit denen zwei Namen verglichen werden können (vgl. About CPE [11]).

Aufgrund der Vorteile verwendet die NVD CPEs, um verwundbare Produkte (Abbildung 3.6) und Konfiguration (Abbildung 3.7) zu referenzieren. Sie betreibt ein Verzeichnis über alle offiziellen CPE-Namen, das frei verfügbar ist (siehe CPE-Dictionary [20]). Durch das Verwenden der offiziellen CPEs können Unklarheiten, durch verschiedene Bezeichnungen des gleichen Produktes, vermieden werden.

CPEs bestehen aus den nachfolgend beschriebenen Attributen (vgl. CPE-Naming-Spezifikation [3] S. 12f) und erlauben so eine feingranulare Durchsuchung der NVD nach Sicherheitslücken für eine gegebene Bibliothek.

Part

CPEs dienen nicht nur der Bezeichnung von Software, auch Hardware-Geräte können eine eindeutige CPE erhalten. Über den Part wird beschrieben, um welche Art von Produkt es sich handelt. Das Attribut muss immer vorhanden sein und eines der folgenden Werte entsprechen:

- „*a*“ - für Anwendungen,
- „*o*“ - für Betriebssysteme (OS),
- „*h*“ - für Hardware-Geräte.

Vendor

Das Vendor-Attribut dient dazu, den Ersteller des Produktes zu identifizieren. Der Vendor kann eine Person oder Organisation sein. Wenn der Vendor schon in einem anderen CPE referenziert wurde, sollte diese Beschreibung übernommen werden.

Product

Der Produktname wird im Product-Attribut beschrieben. Hier sollte ein gängiger Name oder Titel für das Produkt gewählt werden. Wenn das Produkt schon in einem anderen CPE referenziert wurde, sollte diese Beschreibung übernommen werden.

Version

Die Release-Version des Produktes wird im Version-Attribut beschrieben. Der Attributswert sollte aus den verfügbaren Produktdaten kopiert werden.

Update

Update beschreibt den herstellerspezifischen Wert für Update, Service-Pack oder Vorabversion. Beispiele hierfür sind Alpha- oder Beta-Versionen des Produktes.

Edition

Dieses Attribut wurde bis CPE-Version 2.2 verwendet, um eine spezifische Produktedition zu beschreiben. In der aktuellen Version 2.3 wurde das Attribut Edition verfeinert in Software Edition, Target Software, Target Hardware und Other.

Software Edition

Das Attribut Software Edition beschreibt den jeweiligen Markt oder eine Klasse von Endnutzern. Hier kann zum Beispiel zwischen Client- und Server- oder Professional- und Community-Produkten unterschieden werden.

Target Software

Target Software beschreibt die Software-Umgebung, in der das Produkt eingesetzt wird. Dies können zum Beispiel das Betriebssystem sein (Android, Windows) oder Programme (Firefox) wenn es sich um Plugins oder Erweiterungen handelt.

Target Hardware

Im Attribut Target Hardware wird die Instruction-Set-Architektur beschrieben. Darunter fallen zum Beispiel die Instructions-Sets x86 und x64. Aber auch Java-Bytecode, für die Java Virtual Machine ist damit eingeschlossen.

Language

Hier kann die verwendete Sprache im User-Interface des Produkts über Language-Tags beschrieben werden. Das Attribut darf nur gültige Language-Tags nach RFC 5646 enthalten. Es sollten nur Language-Tags mit Sprache und Region gewählt werden.

Other

Im Attribut Other können sonstige herstellerspezifische Merkmale beschrieben werden, die helfen, das Produkt zu identifizieren. Dieses Attribut wird sehr selten verwendet, wie Abbildung 3.2 zeigt.

CPEs müssen nicht immer alle Attribute verwenden. Wenn ein Attribut nicht benutzt wird, erhält es standardmäßig den Wert „ANY“. Für Werte mit „ANY“ gelten keine Einschränkungen durch das Attribut, wenn zwei CPEs miteinander verglichen werden. Der Wert „NA“ („not applicable“ / „not used“) kann verwendet werden, wenn es keinen sinnvollen Wert für das Attribut gibt, oder dieser Wert „null“ ist. Vgl. CPE-Naming-Spezifikation [3] S. 9f

Die Abbildung 3.2 zeigt die Anzahl der einzelnen Attribute von CPEs, die innerhalb der NVD referenziert wurden. Es wurden nur Attribute ohne die Werte

„ANY“ und „NA“ betrachtet. Dies umfasst 216.636 CPEs. Die Abbildung zeigt, dass jeder CPE mindestens die Attribute Part, Vendor und Product enthält. Auch die Version ist mit 203.364 sehr häufig enthalten. Mit dem Faktor 10 deutlich seltener sind spezifische Updates. Noch vierstellige Werte sind für Edition, Target Software und Software Edition vorhanden. Die Attribute Target Hardware (289) Language (166) und Other (14) hingegen werden selten verwendet.

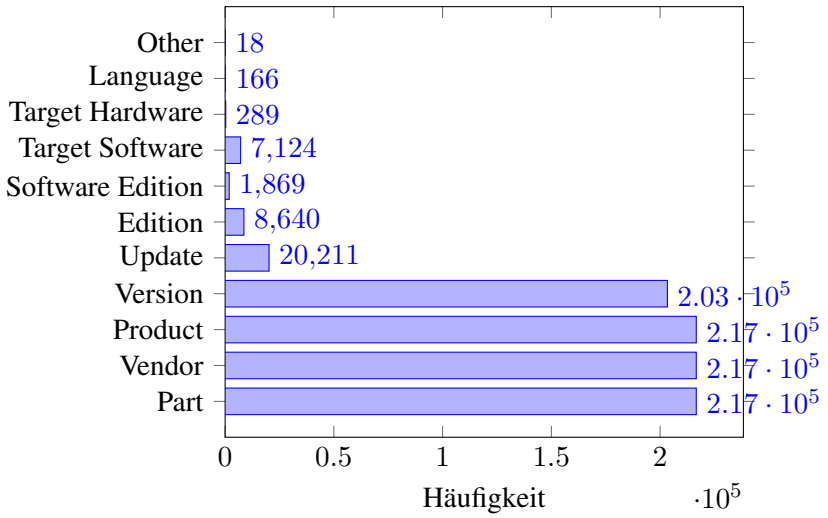

Abbildung 3.2: Häufigkeiten der CPE Attribute ohne die Werte „ANY“ und „NA“

Kodierungen

CPEs können auf zwei Arten kodiert werden. Die Uniform Resource Identifier (URI)-Kodierung ist abwärtskompatibel zur CPE Version 2.2. Sie hat eine der folgenden Formen:

- cpe:/*Part*:*Vendor*:*Product*:*Version*:*Update*:*Edition*:*Language*
- cpe:/*Part*:*Vendor*:*Product*:*Version*:*Update*:*Edition*
- cpe:/*Part*:*Vendor*:*Product*:*Version*:*Update*
- cpe:/*Part*:*Vendor*:*Product*:*Version*
- cpe:/*Part*:*Vendor*:*Product*
- cpe:/*Part*:*Vendor*
- cpe:/*Part*
- cpe:/

Wenn ein Attribut in der obigen Auflistung nicht enthalten ist, hat es den Wert „ANY“. Der Wert „NA“ wird durch das Symbol „-“ kodiert. Für Sonderzeichen wird die Prozentkodierung angewendet - %xx, mit xx als hexadezimalen ASCII-Wert des Sonderzeichens, zum Beispiel %24 für ein Dollarzeichen. Um die Abwärtskompatibilität zu erreichen werden, die in Version 2.3 neu eingeführten Attribute innerhalb des Edition Attribut geschrieben. Dies geschieht über ein „packing“ genanntes Verfahren, in dem die Attributwerte durch eine Tilde getrennt werden (vgl. CPE-Naming-Spezifikation [3] S. 22): „*~Edition~Software Edition~Target Software~Target Hardware~Other*“. Wenn ein Attribut nicht vorhanden ist (oder „ANY“), können auch zwei Tilden aufeinander folgen. Sollte die Edition ohne eines der neuen Attribute beschrieben werden, werden die Tilden weggelassen.

Die zweite mögliche Kodierung erfolgt als Formatted String. Im Gegensatz zur URI-Kodierung hat dieser immer die Form: cpe:2.3:*Part*:*Vendor*:*Product*:*Version*:*Update*:*Edition*:*Language*:*Software Edition*:*Target Software*:*Target Hardware*:*Other*. Wenn ein Attribut den Wert „ANY“ enthält wird dies als „*“, und für den Wert „NA“ als „-“ kodiert. Die Zeichen Punkt, Bindestrich und Unterstrich können ohne Maskierung verwendet werden. Alle weiteren Sonderzeichen werden durch ein führendes Backslash maskiert.

In Abbildung 3.3 wird anhand des Produktes „Microsoft Excel 2010 SP2 64-Bit“, welches in der Sicherheitslücke CVE-2017-0027 referenziert wird, die unterschiedlichen Kodierungen gezeigt.

```
Attribute
Part: application
Vendor: microsoft
Product: excel
Version: 2010
Update: sp2
Edition: ANY
Language: ANY
Software Edition: ANY
Target Software: ANY
Target Hardware: x64
Other: ANY

URI
cpe:/a:microsoft:excel:2010:sp2:~~~~x64~

Formatted String
cpe:2.3:a:microsoft:excel:2010:sp2:*:*:*:*:x64:*
```

Abbildung 3.3: Beispiel der möglichen Kodierungen

3.3.3 Datafeeds

Der Datenbestand der NVD lässt sich über sogenannte Datafeeds herunterladen. Diese stehen sowohl im JSON- als auch im XML-Format zur Verfügung. Die JSON-Feeds sind zum Zeitpunkt des 13.10.2017 noch im BETA-Status. Änderungen der Datenstruktur werden seitens der NVD nicht ausgeschlossen. Laut NVD sollen die JSON-Feeds die XML-Feeds mittelfristig ersetzen [18].

Die Datafeeds sind unterteilt nach Veröffentlichungsjahr der Sicherheitslücken. Ein Datafeed mit der Bezeichung CVE-xxxx enthält alle CVEs, deren CVE-ID mit CVE-xxxx beginnt. Zum Beispiel ist der CVE-Eintrag für CVE-2017-12611 im Datafeed CVE-2017 enthalten. Eine Ausnahme bildet der Datafeed CVE-2002, der zusätzlich alle CVEs für 1999, 2000 und 2001 enthält. Ein Datafeed wird aktualisiert, wenn der Eintrag einer Sicherheitslücke modifiziert, oder eine Sicherheitslücke hinzugefügt wurde. Um nicht jedes mal den kompletten Bestand herunterladen zu müssen, werden noch zwei zusätzliche Feeds angeboten. CVE-Recent enthält alle Sicherheitslücken, die innerhalb der letzten acht Tage veröffentlicht wurden. CVE-Modified enthält alle modifizierten und veröffentlichten Sicherheitslücken der letzten acht Tage. Für jeden Datafeed steht eine META-Datei zur Verfügung. Abbildung 3.4 zeigt beispielhaft die Meta-Datei für den Datafeed CVE-2017. Mithilfe des lastModifiedDate-Zeitstempels lässt sich automatisiert überprüfen, welcher Datafeed aktualisiert wurde. Zusätzlich wird der SHA-256 Hashwert des Feeds bereitgestellt, mit dem die heruntergeladene Datei auf Fehler überprüft werden kann.

```
1 lastModifiedDate:2017-10-13T03:02:45-04:00
2 size:28656778
3 zipSize:1801135
4 gzSize:1801001
5 sha256:71A8C725BA29B76ADE51DD780F24B190
6     0CE7F7DE37101FCDF8A63EBD342D56DD
```

Abbildung 3.4: Meta-Datei für CVE-2017

Die XML-Feeds werden in zwei Versionen (2.0 und 1.2.1) bereitgestellt. Die XML-Version 2.0 enthält einige Neuerungen gegenüber dem Vorgänger. In der aktuellen Version wird zu jedem Eintrag ein Common Weakness Enumeration (CWE) Typ zugeordnet. CWEs beschreiben die zugrundeliegende Ursache der Sicherheitslücke, wie zum Beispiel SQL-Injections. Eine detailierte CWE-Liste wird von der MITRE gepflegt, mit Unterstützung der National Cyber Security Division (DHS) [16].

Während in beiden Versionen von der Sicherheitslücke betroffene Software („Vulnerable-Software“) referenziert werden, geschieht dies ab Version 2.0 über CPEs. Vergleicht man die Abbildung 3.5 mit der Abbildung 3.6, dann fällt auf, dass in Version 1.2.1 ein zusätzliches Flag „prev“ mit dem Wert 1 enthalten ist. „Prev“ steht für „Previous Versions“ und bedeutet bei einem Wert von 1, dass

alle Vorgängerversionen ebenfalls von dieser Sicherheitslücke betroffen sind. Das

```
<vuln_soft>
  <prod name="perl" vendor="perl">
    <vers num="5.24.2" prev="1"/>
    <vers num="5.26.0"/>
  </prod>
</vuln_soft>
```

Abbildung 3.5: Vulnerable-Software in XML-Version 1.2.1 (CVE-2017-12814)

```
<vuln:vulnerable-software-list>
  <vuln:product>cpe:/a:perl:perl:5.24.2</vuln:product>
  <vuln:product>cpe:/a:perl:perl:5.26.0</vuln:product>
</vuln:vulnerable-software-list>
```

Abbildung 3.6: Vulnerable-Software in XML-Version 2.0 (CVE-2017-12814)

Previous-Version-Flag wird häufig in NVD-Einträgen verwendet und das Fehler dieses Flags ist somit ein großer Nachteil der Version 2.0. Betrachtet man die korrespondierende HTML Darstellung der Sicherheitslücke, findet man dieses Flag, siehe:

> cpe:2.3:a:perl:perl:5.24.2:*:*:*:*:*:*:* and previous versions

Ebenso ist das Previous-Version-Flag in den JSON-Datafeeds enthalten. Dort wird es in der Vulnerable-Configuration gesetzt.

In Version 2.0 wird zusätzlich der Vulnerable-Software eine „Vulnerable-Configuration“ eingeführt. Mit dieser ist es möglich, verschiedene Konfigurationen zu beschreiben. Abbildung 3.7 zeigt ein Beispiel, in der die Konfiguration aus einem Windows-Betriebssystem und der Programmiersprache Perl betroffen von der Sicherheitslücke ist. Zu beachten ist hierbei, dass die CPEs der Vulnerable-Software eine Teilmenge der CPEs aus allen Vulnerable-Configurations sind.

Auch bei den CVSS Feldern unterscheiden sich Version 2.0 und 1.2.1 voneinander. In der älteren Version werden zusätzlich zum CVSS-Base-Score noch ein Impact- und ein Exploitability-Subscore gelistet. Die XML-Version 2.0 enthält nur den Base-Score.

3.4 Verfügbare Metadaten

In diesem Abschnitt werden die relevanten Metadaten vorgestellt, die bei der Verwendung von Java-Bibliotheken zur Verfügung stehen. Dies sind im wesentlichen der Dateiname der Bibliothek, der Hashwert und das JAR-Manifest.

```
<vuln:vulnerable-configuration id="http://nvd.nist.gov/">
  <cpe-lang:logical-test operator="AND" negate="false">
    <cpe-lang:logical-test operator="OR" negate="false">
      <cpe-lang:fact-ref name="cpe:/a:perl:perl:5.24.2"/>
      <cpe-lang:fact-ref name="cpe:/a:perl:perl:5.26.0"/>
    </cpe-lang:logical-test>
    <cpe-lang:logical-test operator="OR" negate="false">
      <cpe-lang:fact-ref name="cpe:/o:microsoft:windows:-"/>
    </cpe-lang:logical-test>
  </cpe-lang:logical-test>
</vuln:vulnerable-configuration>
```

Abbildung 3.7: Vulnerable-Configuration in XML-Version 2.0 (CVE-2017-12814)

Dateiname

Der Dateiname der Java-Bibliothek ist eine der wichtigsten Metadaten. Für einen Software-Entwickler ist es von Vorteil, wenn er auf einen Blick erkennen kann, um welche Bibliothek es sich handelt. Die entscheidenden Merkmale sind zum einen der Name der ausgibt welche Software vorliegt und zum anderen die Version, weil sich Funktionsumfang und API je nach Version unterscheiden können. Da Software-Bibliotheken von und für Entwickler geschrieben werden, nehme ich an, dass diese Anforderungen erfüllt werden. Ein Vergleich der 20 populärsten Java-Bibliotheken der Webseite Maven Repository[3] hat diese Annahme bestätigt. Die Dateinamen entsprachen der Form *Name-Version*.jar. Wenn der Name aus mehreren Wörtern besteht, werden diese mit einem Bindestrich verbunden, wie in diesem Beispiel: „slf4j-api-1.7.25.jar“. Preston-Werner beschreibt in „Semantic Versioning 2.0.0“ [29] eine Unterteilung der Versionsnummern nach Major-, Minor- und Patch-Version. Diese Unterteilung zeigte sich auch in den meisten Dateinamen der untersuchten Bibliotheken.

Hashwert

Durch den Vergleich von Hashwerten kann schnell überprüft werden, ob es sich bei zwei Bibliotheken um die gleichen handelt. Hashes lassen sich schnell berechnen. Sie wären damit ein hervorragendes Suchkriterium für Schwachstellendatenbanken. Die größten Datenbanken - NVD, OSVDB und Exploit Database - bieten jedoch keine Suche über Hashwerte an.

Manifest

Das JAR-Manifest ist eine Metadatei, die Informationen über die Java-Bibliothek enthält. Sie beschreibt die enthaltenen Dateien und kann zum Beispiel für die

[3] https://mvnrepository.com/popular

Versionierung verwendet werden (vgl. „Working with Manifest Files: The Basics“ [26]). Jede JAR enthält höchstens ein Manifest und ist unter dem Pfad META-INF/MANIFEST.MF zu finden (vgl. „Understanding the Default Manifest“ [25]). Das Manifest besitzt einen Hauptabschnitt, in dem Attribute für die gesamte JAR deklariert werden. Jeweils durch Leerzeilen getrennt können eine oder mehrere File-Entries mit eigenen Attributen gelistet werden (vgl. „JAR File Specification“ [23]). In dem Manifest aus Abbildung 3.8 beginnt der Hauptteil bei Zeile 1 und endet in Zeile 11. Darüber hinaus werden drei Entries gelistet (Zeile 13 bis 15, 17 bis 18 und 20 bis 21).

```
1  Manifest-Version: 1.0
2  Ant-Version: Apache Ant 1.6.5
3  Created-By: 1.5.0_08-b03 (Sun Microsystems Inc.)
4  Sealed: true
5  Built-By: gregw
6  Specification-Version: 4.2
7  Implementation-Version: 4.2.27
8  Package-Title: org.mortbay.http
9  Implementation-Vendor: Mort Bay Consulting, Pty. Ltd.
10 Implementation-URL: http://jetty.mortbay.org
11 Main-Class: org.mortbay.http.HttpServer
12
13 Name: org/mortbay/jetty/
14 Specification-Title: Servlet/2.3
15 Implementation-Title: Jetty/4.2
16
17 Name: org/mortbay/http/
18 Implementation-Title: Mort Bay HTTP Server
19
20 Name: org/mortbay/util/
21 Implementation-Title: Mort Bay Misc. Utilities
```

Abbildung 3.8: Manifest der Jetty 4.2.27 Bibliothek

Zur Identifizierung der Bibliothek können die Attribute der „Package Version Information“ aus Tabelle 3.3 verwendet werden. Sie können im Hauptabschnitt des Manifests stehen und gelten damit für die gesamte JAR. Ein Entry kann ebenfalls Package Version Information enthalten und überschreibt einen eventuell vorhandenen globalen Wert innerhalb des Entrys. Das Attribut „Name“ ist die erste Zeile eines Entry-Abschnitts und enthält den Pfad zum Entry. Implementation-Title, -Version und -Vendor beschreiben den Namen, die Versionsnummer und den Hersteller der Implementierung. Dies sind die Attribute, die für die Schwachstellensuche verwendet werden. Alle Attribute sind optional und dementsprechend nicht in jeder Bibliothek enthalten.

Attribute
Name
Specification-Title
Specification-Version
Specification-Vendor
Implementation-Title
Implementation-Version
Implementation-Vendor

Tabelle 3.3: Package Version Information (Quelle: Oracle [24])

Kapitel 4

Konzept des Schwachstellenprüfers

In diesem Kapitel wird das Konzept für den Schwachstellenprüfer für Java-Bibliotheken vorgestellt. Die Grundlage des Schwachstellenprüfers bildet die National Vulnerability Database. Der Abschnitt 4.1 beschreibt, wie eine lokale Datenbank mit den Daten der NVD erstellt wird. Für die Suche nach Sicherheitslücken in dieser Datenbank werden die verfügbaren Metadaten der Java-Bibliotheken den CPEs der NVD zugeordnet. Wenn eine passende Relation gefunden wurde, werden alle Sicherheitslücken, die diese CPE referenzieren, von der Datenbank abgefragt. Der genaue Vorgang wird im Abschnitt 4.2 erläutert. Neben der NVD sollen zusätzlich externe SQLite-Datenbanken hinzugefügt werden können. Hierfür muss ein Mapping zwischen Metadaten und dem externen Datenbankschema stattfinden, der in Abschnitt 4.3 erläutert wird. Im letzten Abschnitt 4.4 werden die verschiedenen Konfigurationsmöglichkeiten des Schwachstellenprüfers erklärt.

4.1 Lokale Schwachstellendatenbank

Der Schwachstellenprüfer verwendet eine lokale Schwachstellendatenbank, um unabhängig von der Internetverbindung nach Sicherheitslücken suchen zu können. Anforderung [FR.3] fordert, dass eine CVE-Datenbank verwendet werden soll. Die Entscheidung fiel zugunsten der NVD, weil sie eine programmatische Suche über CPE erlaubt und zusätzlich eine Risikoanalyse nach CVSS Version 2.0 anbietet.

Zum Updaten der lokalen Schwachstellendatenbank wird über die Meta-Datei der NVD-Feeds überprüft, welcher Feed modifiziert wurde und heruntergeladen werden muss. Interessanterweise referenziert das „lastModifiedDate“ in der Meta-Datei nicht das neueste „lastModifiedDate“ eines CVE-Eintrags des entsprechenden Feeds. Stattdessen ist das Datum der Meta-Datei neuer. Zum Beispiel enthält die Meta-Datei für den Feed für 2014 das Datum 2017-10-18T03:11:48-04:00. Das neueste Veröffentlichungsdatum des Feeds ist jedoch 2017-10-17T12:29:00.377-04:00 und die neueste Modifikation 2017-10-

17T21:29:00.467-04:00. Beide sind älter als das Datum der Meta-Datei. Um zu überprüfen, ob ein Feed heruntergeladen werden muss, wird deshalb ein zusätzlicher Zeitstempel benötigt. Dafür speichert die Datenbank den Zeitpunkt nach der letzten Suche nach Aktualisierungen. Ist das Datum in der Meta-Datei neuer als dieser Zeitstempel, dann handelt es sich um einen neuen Feed. Mit diesem Verfahren muss der Schwachstellenprüfer immer nur die modifizierten Datensätze herunterladen. Dadurch kann Zeit eingespart werden und das Netzwerk wird nicht unnötig belastet.

Die XML-Feeds müssen immer in Version 2.0 und 1.2.1 heruntergeladen werden. Leider bietet die aktuelle Version 2.0 nicht das wichtige Merkmal der Sicherheitslücken, ob auch frühere CPE-Versionen von der Sicherheitslücke betroffen sind. Die NVD enthielt bis zum 16.10.2017 30.662 CVE-Einträge, die auf CPEs mit dem Merkmal für frühere Versionen referenzierten. Damit muss dieses Kriterium in der Schwachstellensuche berücksichtigt werden. Nur die Version 1.2.1 zu verwenden, würde allerdings auch zu Einschränkungen führen. Diese Version enthält keine Informationen über den CWE-Typ und beschreibt keine unsicheren Konfigurationen. Der Schwachstellenprüfer verwendet als Basis die Version 2.0. Aus der Version 1.2.1 werden lediglich die Informationen über frühere CPEs früherer Informationen extrahiert. Version 1.2.1 verwendet zur Referenzierung zwar nicht die URI-Kodierung der CPEs, aber es sind die gleichen Daten vorhanden. So kann die Produktbeschreibung (siehe Abbildung 3.5) in eine CPE transformiert werden.

Der Schwachstellenprüfer lädt die XML-Dateien komprimiert als ZIP-Archiv herunter. Die größte Datei, der Feed CVE-2011, ist komprimiert 6,01 MB groß. Wird er entpackt, benötigt er jedoch 108 MB und enthält 4594 CVE-Einträge. Bei dieser Größe muss beim Parsen der XML-Datei auf die Performance geachtet werden. Java bietet verschiedene APIs für das Parsen von XML-Dateien an. Das Document Object Model (DOM), das von der W3C im Web-Kontext entworfen wurde, repräsentiert die gesamte XML-Struktur im Speicher (vgl. Abschnitt 18.3 „Die Java-APIs für XML“ [31]). Dies ist bei den großen Data-Feeds nicht praktikabel, aber auch nicht notwendig. Der Schwachstellenprüfer muss nur die Liste der CVE-Einträge nacheinander auslesen. Ein Random-Access der einzelnen Knoten innerhalb des XML-Baums wird nicht benötigt. Um die größtmögliche Performance zu erreichen, verwendet der Schwachstellenprüfer daher ein serielles Verfahren zum Parsen des Data-Feeds. Es kommt die StAX-API zum Einsatz, die mit einem Cursor-Verfahren den Data-Feed komponentenweise durchläuft (vgl. Abschnitt 18.5 „Serielle Verarbeitung mit StAX“ [32]). Mithilfe spezieller Klassen für das Parsen der beiden Data-Feed-Versionen können Java-Objekte für die jeweiligen CVE-Einträge instanziiert werden.

Nachdem ein Data-Feed heruntergeladen und geparst wurde, wird nach neuen oder modifizierten CVE-Einträgen gesucht. CVEs deren, „lastModifiedDate“ neuer ist als das neuste „lastModifiedDate“ des entsprechenden Feeds der Datenbank, werden einer Update-Liste hinzugefügt. Der weitere Vorgang ist nun davon abhängig, wie viele Einträge die Update-Liste enthält. Zum Updaten der Datenbank

stehen aus Performancegründen zwei verschiedene Methoden zur Verfügung. Messungen haben gezeigt, dass der Updateprozess für ein einzelnes CVE ungefähr 0,8 Sekunden benötigt. Wenn nur wenige (bis zu zehn) CVEs in einem Data-Feed modifiziert wurden, dann ist das schnellste Verfahren, die CVEs einzeln zu aktualisieren. Je nach Alter des Datenbestandes kann es jedoch auch vorkommen, dass hunderte bis tausende CVEs aktualisiert werden müssen. In diesen Fällen ist es deutlich schneller, den kompletten veralteten Data-Feed aus der Datenbank zu löschen und neu einzufügen. Das hat allerdings zur Folge, dass die Recent- und Modified-Data-Feeds nicht verwendet werden können. Die Modified-Liste enthält viele Einträge, deren einzelne Verarbeitung zu viel Zeit benötigt. Weil sie nur neue/modifizierte CVEs enthalten, kann nicht einfach in der Datenbank der alte Feed durch den Modified-Feed ersetzt werden. Unabhängig davon, welche Methode zum Einsatz kam, erfolgen alle INSERT-, UPDATE- oder DELETE-Operationen pro Data-Feed in einer Transaktion, um die Datenbankoperationen zu beschleunigen.Die benötigten Zeiten für das Updaten eines ganzen Feeds, können der Tabelle 6.13 der Evaluation entnommen werden.

Das Datenbankschema wurde so konzipiert, dass alle Informationen eines CVE-Eintrages abgebildet werden. Auch wenn derzeit nicht alle Informationen (der Übersichtlichkeit halber) in der GUI angezeigt werden, können in der Zukunft neu konzipierte Ansichten oder Einsatzmöglichkeiten auf die Datenbankstruktur zurückgreifen. Abbildung 4.1 zeigt die verwendete Datenbankstruktur der Schwachstellendatenbank. Die wichtigste Tabelle ist „cve“, deren Primärschlüssel „id“ der CVE-ID entspricht. Über die CVE-ID können referenzierte Webseiten, CVSS-Ergebnisse, Vulnerable-Software und Vulnerable-Configurations in den entsprechenden Tabellen gefunden werden. Für CPEs ist eine eigene Tabelle „cpe“ vorhanden, die für jedes Attribute eine eigene Spalte besitzt. Sie listet alle CPEs, die von einem CVE-Eintrag der NVD referenziert werden. Die „configuration_items“ Tabelle wurde so konstruiert, dass mit einer einzigen Datenbankabfrage alle relevanten Information – CVE-ID, CPE-URI und Previous-Version-Flag – abgefragt werden können. Um dies zu erreichen, wurden zusätzlich die CPE-Attribute Part, Vendor und Product in die Tabelle mit aufgenommen. So kann die Schwachstellensuche eine Datenbankabfrage auf die „cpe“-Tabelle einsparen, was die Performance des Schwachstellenprüfers erhöht. Die logischen AND und OR Verknüpfungen der Vulnerable-Configuration werden über die „operations“-Tabelle abgebildet. Die „meta“-Tabelle enthält den Zeitstempel der letzten Datenbankaktualisierung.

4.2 Schwachstellensuche

Die Grundidee der Schwachstellensuche ist es, mithilfe der vorhandenen Metadaten der Java-Bibliothek, die zugehörige CPE und über die CPE schließlich die Sicherheitslücke zu finden. Betrachtet man die CPE, dann sind die wichtigsten Attribute Part, Vendor, Product, Version und Update, mit denen sich jede Bi-

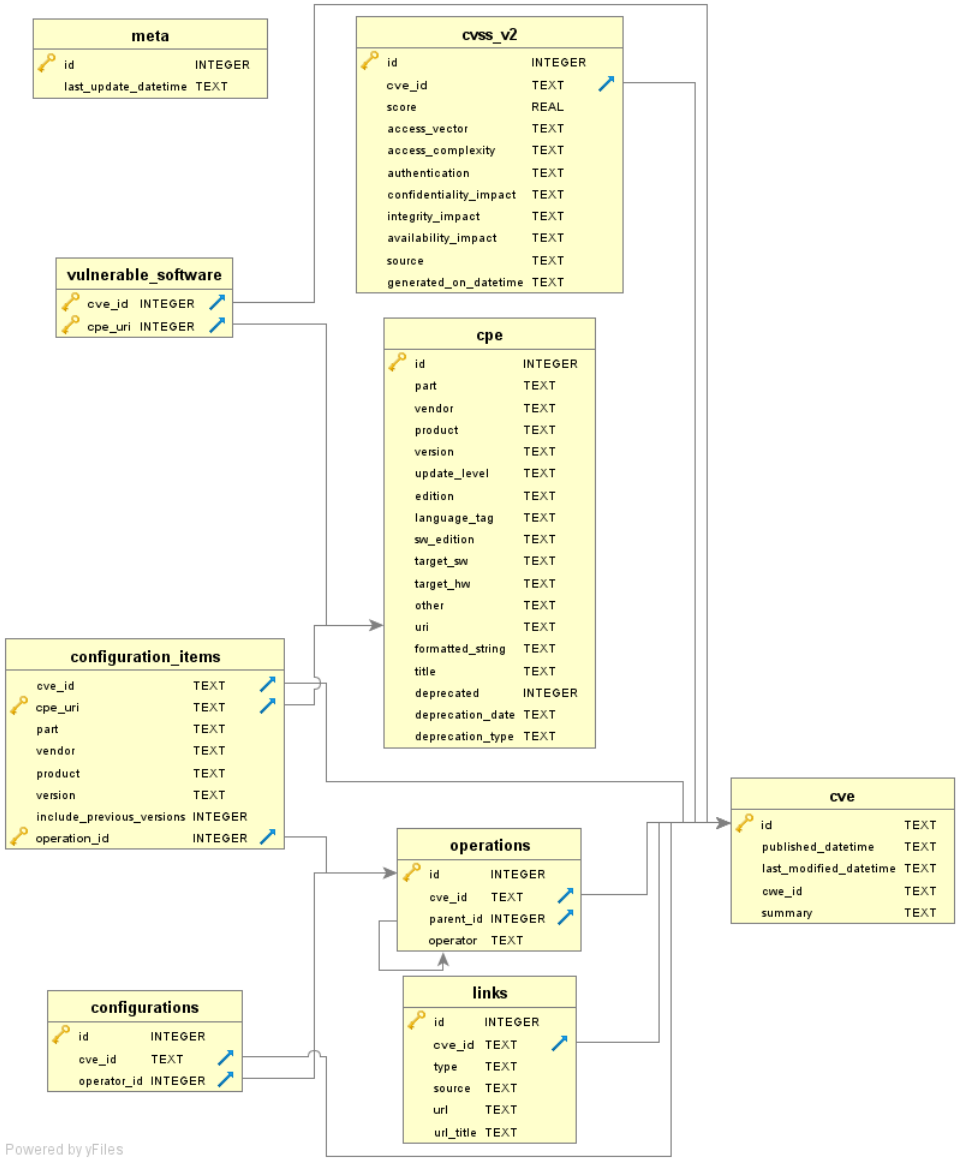

Abbildung 4.1: Schwachstellendatenbank

bliothek beschreiben lässt. Der Part hat für alle Bibliotheken immer den Wert „a“ für Application. Die meisten Bibliotheken sind über Vendor, Product und Version eindeutig beschreibbar. Lediglich für Vorabversionen, wie Alpha oder Beta, ist das Update-Attribut von Bedeutung. Die Tabelle 4.1 zeigt, wie sich die Attribute aus dem JAR-Manifest semantisch den CPE-Attributen zuordnen lassen. In der Praxis ergeben sich jedoch Schwierigkeiten bei der Zuordnung. Werden die

JAR-Manifest	CPE
Implementation-Vendor	Vendor
Implementation-Title	Product
Implementation-Version	Version

Tabelle 4.1: Semantische Zuordnung der Attribute aus Manifest und CPE

Implementation-Vendor der drei Apache Bibliotheken Struts, Commons HttpClient und Axis betrachtet, werden drei verschiedene Beschreibungen geliefert:

- The Apache Software Foundation (Struts)
- Apache Sofware Foundation (Commons HttpClient)
- Apache Web Services (Axis)

Dem gegenüber steht das CPE-Vendor-Attribut: apache. Während der Hersteller im CPE kurz und prägnant beschrieben wird, handelt es sich im Manifest um eine ausführliche Beschreibung. Ein einfacher Stringvergleich der beiden Attribute ist so nicht möglich. Auch ein passendes CPE-Product zu finden, ist schwerer als es zunächst scheint, wie Tabelle 4.2 zeigt. Betrachtet man den Dateinamen der Axis2-Bibliothek, ergeben sich die Token „axis2“ und „kernel“ als mögliche Produktnamen. Der zur Bibliothek gehörende CPE beinhaltet aber nur den Wert „axis2“. Die gleiche Problematik tritt beim Implementation-Title der Struts-Bibliothek auf, mit „Struts Framgework“ gegenüber dem Product-Attribut „struts“. Es kann aber auch der Fall eintreten, dass der CPE Token enthält, die nicht in den Metadaten zu finden sind, wie das Beispiel der Xalan-Bibliothek zeigt. Im CPE wird das Produkt als „xalan-java“ beschrieben. Über die Metadaten ist aber nur das Wort „xalan“ enthalten. Im letzten Beispiel der Commons FileUpload Bibliothek von Apache sind nur minimale Unterschiede zum CPE-Produkt vorhanden. Lediglich die Groß- und Kleinschreibung und die Substitution von Leerzeichen mit einem Unterstrich, sind die Unterschiede.

Um die oben genannten Bibliotheken ihren CPEs zuzuordnen, sind einfache String-Operationen nicht ausreichend. Die Überprüfung auf Gleichheit zweier Strings ist zu strikt. Keine der vorherigen Bibliotheken wäre so der richtigen CPE zugeordnet worden. Möglich wäre auch eine Abfrage, ob die Token aus den Metadaten im CPE-Produkt enthalten sind und umgekehrt. Diese Bedingung ist jedoch nicht konkret genug und so werden zu viele CPEs falsch zugeordnet. Außerdem würde so auch die Commons FileUpload Bibliothek nicht der richtigen

Metadaten-Typ	Wert	CPE-Product
Dateiname	axis2-kernel-1.4.1	axis2
Implementation-Title	Struts Framework	struts
Implementation-Title	Commons FileUpload	commons_fileupload
Dateiname	xalan-2.7.0	xalan-java

Tabelle 4.2: Vergleich Metadaten und CPE-Product

CPE zugewiesen werden, wegen des Unterstrichs im CPE-Product. Notwendig ist ein Ähnlichkeitsmaß zum Vergleichen zweier Strings. Der Schwachstellenprüfer verwendet hierfür die Levenshtein-Distanz.

Die Levenshtein-Distanz gibt an, wie viele Einfüge-, Substitutions- oder Löschoperationen notwendig sind, um einen String s1 in einen anderen String s2 zu transformieren. Bei Gleichheit beträgt die Levenshtein-Distanz 0 (vgl. Abschnitt 3.3.3 „Edit distance“ aus dem Buch „Introduction to Information Retrieval“ [9]).

Aber auch die Levenshtein-Distanz alleine ist nicht ausreichend. Berechnet man die Distanz zwischen „struts framework“ und „struts“, ergibt dies einen Wert von 10. Würde man jedoch als Höchstgrenze für die Levenshtein-Distanz den Wert auf 10 setzen, käme es zu vielen falsch zugeordneten CPEs.

Der Schwachstellenprüfer löst das Problem in mehreren Schritten. Zunächst werden alle Metadaten einer Bibliothek gesammelt und verarbeitet. Dies wird in Abschnitt 4.2.1 beschrieben. Auf Grundlage dieser Metadaten werden potenzielle CPE-Kandidaten aus der Datenbank abgefragt. Diese Kandidaten werden anschließend genauer untersucht. Wenn sie den Kriterien nicht genügen, werden sie verworfen. Der Prozess, wie CPEs ausgewählt werden, wird in Abschnitt 4.2.2 erläutert. Für die verbliebenen CPEs werden die referenzierten Sicherheitslücken abgefragt, womit sich der Abschnitt 4.2.3 befasst.

4.2.1 Metadaten

Der Schwachstellenprüfer sammelt zu Beginn alle Metadaten, die im Manifest und im Dateinamen vorhanden sind und kategorisiert sie nach Hersteller, Produkt und Version.

Manifest

Dafür wird das JAR-Manifest nach den Attributen Implementation-Title, Implementation-Vendor und Implementation-Version durchsucht. Dabei muss zusätzlich darauf geachtet werden, dass das Attribut keine leeren Werte enthält. Dies trifft zum Beispiel auf die Bibliothek „servlet-api-2.5.jar“ zu, die einen leeren Implementation-Title enthält.

Die Angabe aller Attribute ist freiwillig. Von den 38 verwendeten Test-Bibliotheken, enthielten 14 keines der drei Attribute, wie Abbildung 4.2 zeigt. Die restlichen 24 Bibliotheken enthielten mindestens eines der drei Attribute. Sie

können in einer beliebigen Kombination enthalten sein, nur eins von drei, zwei von drei oder alle Attribute. Am häufigsten ist die Implementation-Version mit 23 Bibliotheken enthalten. Gefolgt von 22 Bibliotheken mit Implementation-Title und 19 mit Implementation-Vendor.

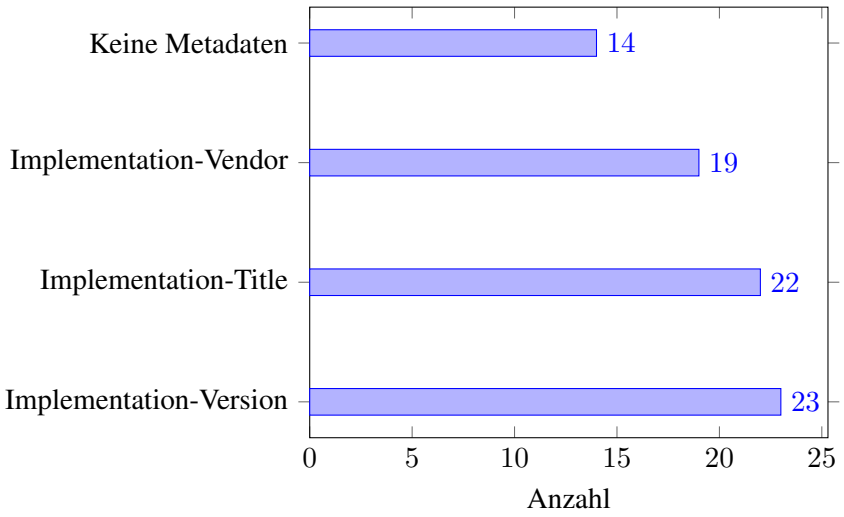

Abbildung 4.2: Anzahl der Bibliotheken mit den jeweiligen Manifest-Attributen

Das Attribut Implementation-Vendor ist die einzige Quelle, mithilfe der Schwachstellenprüfer selbstständig den Hersteller der Bibliothek ermitteln kann. Für Produktnamen ist oftmals der Dateiname präziser als der ausführlichere Implementation-Title.

Dateiname

Der Dateiname enthält oftmals Produkt und Versionsinformationen. Um diese zu unterscheiden, wird zunächst der Name in einzelne Token eingeteilt. Über den regulären Ausdruck aus Abbildung 4.3 wird überprüft, ob das Token Ziffern enthält. Wenn ja, dann wird das Token als mögliche Versionsnummer betrachtet und in die Liste aller Versionsnummern aufgenommen. Anderenfalls ist das Token ein möglicher Produktname und wird der Produktliste hinzugefügt. Zusätzlich

```
1 .*\d+.*
2
3 Durch .* kann eine beliebige Zeichenfolge vor der Ziffer stehen.
4 \d+ bestimmt, dass mindestens eine Ziffer (0−9) folgt.
5 Nach einer oder mehreren Ziffern kann wieder eine beliebige Zeichenfolge aufgrund .* stehen.
```

Abbildung 4.3: Regular Expression für mögliche Versionsnummern

wird das Token noch auf weitere Sonderfälle untersucht. Endet zum Beispiel

die Versionsnummer auf „.0“ wie bei dem Dateinamen „jetty-6.1.0“, wird ein zusätzliches Token mit „6.1“ erzeugt und in die Versionsliste aufgenommen. So kann die CPE

cpe:/a:mortbay_jetty:jetty:6.1

gefunden werden. Ein Token, bestehend aus Buchstaben und Ziffern, kann aber auch ein Produktname sein, wie es bei der Apache Bibliothek Axis2 der Fall ist. Deshalb wird das Token mit einem weiteren regulären Ausdruck (Abbildung 4.4) auf Buchstaben geprüft. Wenn Buchstaben vorhanden sind, dann wird das Token jeweils einmal mit (zum Beispiel „axis2“) und einmal ohne Ziffern („axis“) in die Produktliste aufgenommen.

```
1 .*\p{L}.*
2
3 Durch .* kann das Token mit einer beliebigen Zeichenfolge beginnen.
4 \p{L} bestimmt, dass ein Unicode−Buchstabe enthalten ist.
5 Zum Schluss kann wieder eine beliebige Zeichenfolge aufgrund .* stehen.
```

Abbildung 4.4: Regular Expression für mögliche Produktnamen

Metadaten-Analyse

Die Listen mit den Token für Produkt, Version und Hersteller werden zunächst von redundanten Einträgen bereinigt. Danach wird für jede Liste die Potenzmenge über ihre Token berechnet und in einer separaten Liste gespeichert. Aus einer hypothetischen Tokenliste (apache, commons, fileupload) wird die Potenzmenge, ohne die leeren Menge, erstellt: (apache, commons, fileupload, apachecommons, apachefilupload, commonsfileupload, apachecommonsfileupload). Diese generierte Menge kann nun als Basis für einen CPE-Vergleich verwendet werden, in dem ein Eintrag der Potenzmenge dem CPE-Attribut gegenübergestellt wird. Das Element der Potenzmenge commonsfileupload und das CPE-Product commons_fileupload sind sich zum Beispiel, mit einer Levenshtein-Distanz von 1, am ähnlichsten. Da die Potenzmenge exponentiell wächst, wird sie nur für Listen mit maximal 8 Einträgen berechnet.

Bearbeiten von Metadaten

Die Qualität der Suchergebnisse ist abhängig von den verfügbaren Metadaten. Wenn im Manifest keine Produktinfomationen bereitgestellt werden und kein sinnvoller Dateiname gewählt wurde, können keine relevanten CPEs zugeordnet werden. Um dieses Problem zu beheben, hat der Benutzer die Möglichkeit, die Produkt-, Versions- und Herstellerliste zu bearbeiten. Er kann neue Einträge hinzufügen oder irrelevante Einträge entfernen. Nachdem eine Liste bearbeitet wurde, wird eine aktualisierte Potenzmenge berechnet. Ein Beispiel für die

Folgen von irrelevanten Einträgen zeigt die Apache Commons IO Bibliothek aus Tabelle 4.3. Sie enthält sowohl in der Vendor- als auch in der Produktliste das Token „apache“. Dadurch entsteht die falsche Zuordnung mit „cpe:/a:apache:apache“.

Vendor	Product	Version
the apache software foundation	apache commons io commons io apache	2.5

Tabelle 4.3: Metadaten der Apache Commons IO Bibliothek

Der Benutzer kann dieses verhindern, indem er den Eintrag „apache“ aus der Produktliste entfernt.

Im vorherigen Beispiel waren die Token aus den Metadaten und die Attribute im CPE identisch. Es können aber auch Fehler durch die Ähnlichkeit von Token und Attributen entstehen. Deshalb erhält jede Bibliothek zusätzliche Blacklisten für Produkt, Hersteller und Version, mit dem der Benutzer Suchbegriffe ausschließen kann. Wenn ein CPE-Attribut in der Blacklist enthalten ist, wird es verworfen. Für die mockito-Bibliothek wird aufgrund der Produktmetadaten „mockito“ und „core“ die CPE „cpe:/a:m-core:m-core“ gefunden. Der Benutzer kann nun „m-core“ auf die Blacklist für Produkte oder für Hersteller setzen, um diesen CPE von der Suche auszuschließen.

4.2.2 CPE-Auswahl

Nachdem die Metadaten zur Verfügung stehen, wird die Datenbank nach möglichen CPE-Kandidaten durchsucht. Dies geschieht auf der „configuration_items“-Tabelle. Als Kriterien werden Part, der immer den Wert „a“ für Application enthalten muss, Product und Version verwendet. Das Product-Attribut muss einen Eintrag aus der Produkt-Potenzmenge enthalten. Dies nach dem Schema „WHERE product like %Wert1% OR ...“, sodass jeder Eintrag der Potenzmenge über die OR-Verknüpfung abgefragt wird. Um zu überprüfen, ob der Wert im Attribut enthalten ist, kommen die Wildcards (%, für beliebige Zeichenfolgen) zum Einsatz. Bei der Version sind keine Wildcards erwünscht. Die Bedingung lautet hier, dass die Version „größer gleich“ (im lexikographischem Sinne) einer Version der Potenzmenge ist. Die Abfrage lautet hier auf g „größer gleich“, weil auch frühere Versionen mit eingeschlossen werdenn können, wenn das Previous-Version-Flag gesetzt wurde. Die genaue Versionsprüfung erfolgt ebenso wie die Überprüfung des Vendors bei der Analyse der resultierenden CPEs. Bei einem Treffer werden CVE-ID, CPE in URI-Form und ein Include-Previous-Versions-Flag zurückgeliefert.

Die Kandidaten werden nun auf ihre Ähnlichkeit zu den Metadaten überprüft. Als erstes wird überprüft, ob das Vendor-, Product- oder Versionsattribut in einer Blacklist enthalten ist und falls ja, direkt verworfen. Im nächsten Schritt werden die Attribute Vendor und Product mit den Potenzmengen der Metadaten über die

Levenshtein-Distanz verglichen. Aufgrund der %-Wildards der SQlite-Abfrage, kann das CPE-Product stark von den Produkt-Metadaten divergieren. Am Beispiel des Vergleichs der Produktnamen aus Abbildung 4.5 wird der Prozess erläutert. Das Attribut Product wird mit den Einträgen der Produkt-Potenzmenge verglichen. Wenn das CPE-Product Ziffern enthält, nicht jedoch das Token der Produktmenge, dann ähneln sich das Attribut und das Token nicht und es wird das nächste Token betrachtet. Diese Überprüfung geschieht in Zeile 3 bis 5. Anderenfalls würde die Bibliothek Apache Axis die gleichen CPEs wie Apache Axis2 erhalten. Der umgekehrte Fall wird nicht geprüft, weil nicht ausgeschlossen werden kann, dass eine Ziffer im Produktnamen, im CPE nicht in einem anderen Attribut, wie zum Beispiel Version, beschrieben wird. Als nächstes wird die Levenshtein-Distanz zwischen Token und Attribut berechnet. Die maximale Distanz ist dabei abhängig von der größten Wortlänge zwischen Attribut und Token. Bei Wörtern, die aus mehr als 8 Zeichen bestehen, beträgt sie höchstens 4. Bei kürzeren Wörtern kann eine akzeptierte Levenshtein-Distanz von 4 jedoch schon zu groß sein, weshalb sie hier die abgerundete halbe Wortlänge beträgt. Wenn ein Token gefunden wurde, das akzeptiert wird, dann gilt das CPE-Product als korrekt. Analog wird auch das Vendor-Attribut überprüft, mit der Ausnahme, dass die Bedingung von Zeile 3 bis 5 nicht gilt. Der Vendor wird nicht nach enthaltenen Ziffern überprüft. Sollten die Metadaten keine Informationen über den Hersteller enthalten, wird jeder CPE-Vendor akzeptiert. Die CPE-Version wird immer akzeptiert, wenn sie den Wert „ANY“, „NA“ enthält oder das Previous Version Flag für die CPE gesetzt wurde. Das Flag signalisiert, dass die CPE auch für frühere Versionen gültig ist. Aufgrund der SQL-Abfrage wissen wir bereits, dass die CPE-Version größer oder gleich der Bibliotheksversion ist. In allen anderen Fällen wird die CPE-Version akzeptiert, wenn sie mit einem Eintrag aus der Versionspotenzmenge übereinstimmt. Wenn die Attribute Vendor, Product und Version der CPE akzeptiert wurden, dann wird auch die CPE als gültig betrachtet.

4.2.3 CVE-Suche

Nachdem nun die CPEs feststehen, werden die referenzierten CVEs aus der Datenbank abgefragt. Die dafür benötigten CVE-IDs sind schon durch die Abfrage der „configuration_items“-Tabelle bekannt. Da verschiedene CPEs auf die gleiche CVE-ID verweisen können, wird darauf geachtet, dass ein CVE nicht mehrmals aus der Datenbank abgefragt wird.

CVE-Blacklist

Nicht jede Sicherheitslücke muss für einen Benutzer relevant sein. Wenn er nach einer kritischen Prüfung der CVE zu der Auffassung gelangt, die Bibliothek trotzdem verwenden zu wollen, kann er die CVE auf eine Blacklist setzen. Die Blacklist kann, je nach Wunsch des Benutzers, nur für das aktuelle Projekt oder für alle Projekte gelten. Der Schwachstellenprüfer warnt den Benutzer nicht weiter

```
1  private boolean isProductMatch(AnalyzedLib lib, String candidate) {
2    for (String title : lib.getTitleSuperset()) {
3      if (candidate.matches(".*\\d+.*")
4          && !title.matches(".*\\d+.*")) {
5        continue;
6      }
7
8      int dist = StringUtils.getLevenshteinDistance(title, candidate);
9      int max = Math.min(title.length(), candidate.length());
10     if (max >= 2) {
11       max /= 2;
12     }
13     max = Math.min(max, 4);
14     if (dist <= max) {
15       return true;
16     }
17   }
18
19   return false;
20 }
```

Abbildung 4.5: Vergleich von Metadaten mit CPE-Product

vor CVEs, die auf einer Blacklist stehen.

4.3 Verwendung externer Schwachstellendatenbank

Der Schwachstellenprüfer soll in der Lage sein, externe SQLite-Datenbanken für die Schwachstellensuche zu verwenden. Die Schwierigkeit ist hier, ein unbekanntes Datenbankschema mit den Schnittstellen des Schwachstellenprüfers zu verknüpfen.

Zur Lösung des Problems, soll dem Benutzer des Schwachstellenprüfers, ein Schema der externen Datenbank angezeigt werden. Über die GUI verknüpft der Benutzer dann Spalten der Datenbank mit den Schnittstellen des Schwachstellenprüfers. Dieser Prozess wird im Abschnitt 4.3.1 beschrieben. Darauf folgt im Abschnitt 4.3.2 eine Beschreibung der Schwachstellensuche auf der externen SQLite-Datenbank.

4.3.1 Mapping

Um den Benutzer das Datenbankschema anzuzeigen, muss dieses zunächst ermittelt werden. Das Schema kann durch die Statements aus Abbildung 4.6 abgefragt werden.

Die externe Datenbank muss nun zwei Schnittstellen zugeordnet werden. Zum einen muss festgelegt werden, welche der verfügbaren Metadaten wie verwendet werden. Zum anderen müssen die Suchergebnisse der Datenbank auf

```
/*
 * Namen aller Tabellen
 */
SELECT name FROM sqlite_master WHERE type='table';

/*
 * Beschreibung der Tabelle cve.
 * Eine Reihe beschreibt je eine Spalte der Tabelle:
 * Spaltennummer, Name, Type, Not Null Boolean, Defaultwert, Primary Key Boolean
 */
PRAGMA table_info (cve);

/*
 * Beschreibung der Foreign Keys der Tabelle vulnerable_software
 * id, seq, table, from, to, on_update, on_delete, match
 */
PRAGMA foreign_key_list (vulnerable_software);
```

Abbildung 4.6: Statements zum Abfragen des Datenbankschemas

die Vulnerability-Schnittstelle abgebildet werden.

Eingabe-Mapping

Zuerst wählt der Benutzer aus, welche Metadaten er verwenden möchte. Zur Auswahl stehen:

- Dateiname
- SHA-256
- Vendor
- Product Name
- Version

Der Benutzer klickt in der GUI eine Tabellenspalte an und wählt dann aus, auf welche Metadaten die Spalte zugeordnet werden soll.

Ausgabe-Mapping

Als nächstes muss der Benutzer auswählen, welche Ergebnisse zurückgeliefert werden. Es erfolgt dabei eine Zuordnung der Tabellenspalten auf das Vulnerability-Interface.

- Vulnerability ID
- Beschreibung

- Veröffentlichungsdatum
- Letzte Modifizierung
- Weblinks

Zu beachten ist hier, dass die Vulnerability ID ein Pflichtfeld ist. Sie wird benötigt, um später verschiedene Sicherheitslücken unterscheiden zu können. Eine Sicherheitslücke darf höchstens je eine Beschreibung, Veröffentlichungsdatum oder letzte Modifikation enthalten. Diese Einschränkung gilt jedoch nicht für Weblinks. Die Sicherheitslücke kann eine Liste von Weblinks enthalten.

Die Benutzereingaben für Eingabe- und Ausgabe-Mapping verwendet der Schwachstellenprüfer nun, um ein allgemeines Mapping-Modell zu erstellen. Für jede externe Datenbank existiert genau ein Mapping-Modell, das die Datenbank beschreibt. Dieses Modell wird später für die Schwachstellensuche auf der Datenbank verwendet. Für die Erstellung des Mapping-Modells wird eine Generator Klasse verwendet. So können Logik und Modell voneinander getrennt werden. Dieser „MappingGenerator“ erstellt das Grundgerüst für eine SQL-Query: die Teile SELECT und FROM.

SELECT enthält alle vom Benutzer ausgewählten Ein- und Ausgabespalten. Beginnend mit den Tabellen vom Eingabe-Mapping werden alle Tabellen über ein Left-Join miteinander verbunden. Wenn die Felder des Eingabe-Mappings null sind, bedeutet dies, dass keine Sicherheitslücke für die Bibliothek gefunden wurde. Die Felder des Ausgabe-Mappings können aufgrund des Left-Joins jedoch null-Werte enthalten. So können auch Sicherheitslücken gefunden werden, die nicht vollständig beschrieben sind. Der Left-Join wird automatisch durch eine Auswertung der Foreign-Key-Constraints konstruiert. Daher ist es zwingend notwendig, dass die externe SQLite-Datenbank Foreign-Keys definiert, wenn Spalten verschiedener Tabellen verwendet werden.

Nachdem das Mapping-Modell erstellt wurde, wird der Mapping-Generator nicht mehr benötigt. Das Modell hingegen wird in der Datenbank des Schwachstellenprüfers persistiert.

4.3.2 Schwachstellensuche

Bevor eine Schwachstellensuche auf einer externen Datenbank gestartet wird, stellt der Schwachstellenprüfer sicher, dass die zu überprüfende Bibliothek die notwendigen Metadaten enthält. Hierfür wird im Mapping-Modell nachgeschaut, welche Metadaten die Datenbank referenziert und ob diese für die aktuelle Bibliothek enthalten sind. Wenn dies der Fall ist, wird die noch fehlende WHERE-Bedingung mit den notwendigen Metadaten ergänzt. Es folgt die Datenbankabfrage mit dem konstruierten Statement. Über das Mapping-Modell können die zurückgelieferten Spalten den Feldern der Vulnerability-Schnittstelle zugeordnet werden. Wenn das Mapping keine Relation mit Weblinks enthält, dann entspricht jede Ergebnisreihe

der Datenbank einer Schwachstelle. Anderenfalls müssen die Spalten über die ID differenziert werden.

4.4 Konfiguration

Der Schwachstellenprüfer lässt sich im Hinblick auf die Treffergenauigkeit der präsentierten Schwachstellen und eines Mindestschweregrades konfigurieren. Je nach gewählter Konfiguration, werden dem Benutzer nur ein Teil oder alle gefundenen Schwachstellen präsentiert. Wenn die Konfiguration geändert wird, hat dies nur Einfluss auf den Filterprozess. Eine erneute Schwachstellensuche ist deshalb nicht erforderlich.

Die Precision der Schwachstellensuche ist in die drei Stufen

- High Recall
- Best Combined Precision and Recall
- High Precision

einstellbar und erfüllt somit die Anforderung [FR.3.1]. Als Standard ist die „High Recall“-Konfiguration ausgewählt, in der die meisten Sicherheitslücken gefunden werden. Allerdings werden dabei auch mehr falsche Warnungen ausgegeben. Die Konfiguration „High Precision“ hingegen ist darauf optimiert, möglichst wenig falsche Warnungen zu produzieren, wodurch aber auch wichtige Sicherheitslücken unentdeckt bleiben können. Die mittlere Konfiguration „Best Combined Precision and Recall“ findet mehr CVEs als in High Precision und ist treffsicherer als High Recall Konfiguration. Die Performance der jeweiligen Einstellung wird detailliert in Abschnitt 6.1.2 betrachtet.

Für den Schwachstellenprüfer kann ein Mindestschweregrad für Sicherheitslücken angegeben werden. Es werden drei Möglichkeiten angeboten, die den NVD-Schweregraden aus Tabelle 3.2 entsprechen. Der Benutzer hat die Wahl zwischen:

- Show All
- Show Medium and High
- Show High

Kapitel 5

Implementierung

In diesem Kapitel wird zunächst in Abschnitt 5.1 die Architektur des Schwachstellenprüfers vorgestellt. Darauf folgt im Abschnitt 5.2 die Präsentation der verschiedenen Ansichten und die Interaktionsmöglichkeiten des Benutzers.

5.1 Architektur

Die Daten werden in zwei separaten SQLite-Datenbanken persistiert. Eine enthält alle Daten der CVE, die andere Einstellungen, Metadaten und gefundene Schwachstellen in den Projekten. Durch die Trennung sollen die wertvollen Metadaten, die eventuell von Hand bearbeitet und verfeinert wurden besonders geschützt werden. Das Updaten der Schwachstellendatenbank läuft in einem längeren Prozess. Es ist wahrscheinlicher, dass eine mögliche Beschädigung der Datenbank, bei diesem Prozess auftritt. Die Datenbanken werden im Ordner „configuration“ des Eclipse-Verzeichnisses gespeichert. Sie stehen dadurch unabhängig vom aktiven Eclipse-Workspace zur Verfügung.

Der Schwachstellenprüfer basiert auf einer Model-View-Controller Architektur. Der Controller steuert den Programmablauf. Er ruft Komponenten wie NVD- und externe Datenbank auf, verarbeitet die Ergebnisse und schickt sie ans Model. Änderungen im Model werden ebenso nur von dem Controller durchgeführt. Die View-Komponente enthält Ansichten, die zusätzlich eine eigene MVC-Architektur aufweisen. Sie verwenden die Daten des globalen Modells in einem separaten View-Model. Durch diese Struktur verwenden alle Views ein synchronisiertes Model. Die Fachlogik im Model und die Prozesse im Controller werden so besser von der Präsentation der Daten in der View getrennt. Neue Views können schneller hinzugefügt werden und Modelle und Controller in anderen Projekten wiederverwendet werden. Nachfolgend werden die Komponenten der Bausteinsicht aus Abbildung 5.1 einzeln vorgestellt.

5.1.1 NVD-Datenbank

Diese Komponente kapselt die fachlichen Modelle und Controller für die NVD-Datenbank. Sie enthält einen Update-Controller, der für die erstmalige Einrichtung der lokalen NVD-Datenbank und für das Aktualisieren der Daten zuständig ist. Die NVD-Data-Feeds werden in einer separaten SQLite-Datenbank importiert. Sollte die Schwachstellendatenbank jemals beschädigt werden, zum Beispiel, weil während eines Schreibvorgangs die Stromversorgung ausfällt, kann sie ohne Bedenken gelöscht werden. Die Data-Feeds werden einfach erneut heruntergeladen und importiert. Die wertvolleren Daten, wie von Benutzer erstellte Ignore-Lists und weitere Metadaten, werden durch die separaten Datenbanken so zusätzlich geschützt.

Der zweite Controller ist für die Suche auf dieser Datenbank zuständig. Er wird mit einem AnalyzedLib-Objekt aufgerufen und gibt eine Liste von Sicherheitslücken zurück. Abhängigkeiten zu anderen Komponenten bestehen nur bei der Suche, welche die analysierten Metadaten benötigt. Weil die Metadaten auch für die Suche in der externen Datenbank benötigt werden, ist die Verarbeitung der Roh-Metadaten nicht Teil dieser Komponente. Durch die Kapselung lässt sich die NVD-Komponente mit wenig Aufwand in anderen Projekten integrieren, die eine Schwachstellendatenbank verwenden.

5.1.2 Controller

Die Controller-Komponente steuert die zentralen Prozesse des Schwachstellenprüfers und reagiert auf Ereignisse die durch den Nutzer ausgelöst werden, wie das Hinzufügen oder Entfernen von Bibliotheken.

Eclipse bietet eine Jobs-API an, mit der ein Hintergrundprozess gestartet und der Fortschritt des Prozesses dem Benutzer angezeigt werden kann. Das Updaten der Datenbank erfolgt in solch einem Job. Er wird von dem Controller, beim Starten von Eclipse oder wenn der Benutzer ein Update-Command auslöst, erstellt. Die NVD-Komponente teilt dem Job mit, wie viele Data-Feeds vorhanden sind und wie viele davon bereits aktualisiert wurden. Dadurch kennt Eclipse den Status des laufenden Jobs und zeigt diesen Prozentual oder als Fortschrittsbalken in der Progress-View an. Der Benutzer kann Jobs manuell beenden. Die Datenbankaktualisierung wird dann nach dem Import des aktuellen Data-Feeds unterbrochen.

Der Controller wird durch einen Ressourcen-Listener über Änderungen im Eclipse-Workspace informiert. So kann er feststellen, wenn in einem der Projekte eine Datei hinzugefügt oder gelöscht wird. Bei neuen Java-Bibliotheken werden die Metadaten in einem Hintergrundprozess analysiert. Als Resultat entsteht eine AnalyzedLib-Objekt, das in der Datenbank und im Modell gespeichert wird. Wenn eine Bibliothek schon bereits einmal analysiert wurde, werden die Metadaten aus der Datenbank ausgelesen. Ebenso werden schon einmal gefundene Schwachstellen für die Bibliothek in das Schwachstellenmodell geladen. Im Anschluss wird eine neue Schwachstellensuche ausgelöst.

Auch die Schwachstellensuche wird von dem Controller in einem Job gestartet. Die Prozesse für Update und Suche dürfen nicht gleichzeitig ausgeführt werden, weil sie beide dieselbe SQLite Datenbank der NVD-Komponente nutzen. Um dies sicherzustellen verwendet der Controller einen Mutex. Update oder Suche werden erst dann gestartet, wenn der laufende Prozess beendet wurde. Die Schwachstellensuche empfängt alle AnalyzedLib-Objecte aus dem Modell. Die Anzahl der zu untersuchenden Bibliotheken, wird für den Job vermerkt, damit dieser den Benutzer über den Status der Suche informieren kann. Jedes AnalyzedLib enthält einen Zeitstempel von der zuletzt durchgeführten Suche. Wenn dieser länger zurückliegt als das letzte Datenbankupdate der lokalen NVD-Datenbank, wird eine neue Suche gestartet. Dafür wird die NVD-Komponente und die externe Datenbankskomponente aufgerufen, die eine Liste von Sicherheitslücken zurückliefern. Die Sicherheitslücken werden im Controller dahingehend überprüft, ob sie auf einer Blacklist stehen und falls nicht dem Schwachstellenmodell hinzugefügt. Wenn eine Sicherheitslücke mit gleicher ID und für die gleiche Bibliothek schon in dem Schwachstellenmodell enthalten ist, werden alte und neue Sicherheitslücken auf Gleichheit überpfrüft. Sollten sie sich unterscheiden, wird die alte Schwachstelle aus dem Modell entfernt und die neue eingefügt. So wird sichergestellt, dass nach einem Datenbankupdate auch immer die neuesten Beschreibungen dem Benutzer angezeigt werden. Sicherheitslücken, die dem Modell hinzugefügt wurden, werden für die Dauer der Eclipsesitzung als neu gekennzeichnet, so dass nach diesen Sicherheitslücken gefiltert werden kann.

5.1.3 Model

Diese Komponente enthält zwei Modelle, eins für verarbeitete Metadaten und eins für gefundene Schwachstellen. Diese sind nach dem Singleton-Pattern konstruiert, so dass es von jedem Modell nur eine Instanz gibt und es global von anderen Komponenten verwendet werden kann. Bei Änderungen in den Modellen, wie zum Beispiel nach dem Hinzufügen oder Entfernen von Schwachstellen, werden registrierte Listener über die Modifikation benachrichtigt.

In dem Metadatenmodell befinden sich die AnalyzedLib-Objekte, von allen Bibliotheken, die momentan in einem geöffnneten Projekt enthalten sind. Der Schwachstellenprüfer weiß so, welche Bibliotheken durchsucht werden müssen.

Das Schwachstellenmodell filtert abgerufene Sicherheitslücken je nach aktiver Konfiguration (High Recall, High Precision, Best Combined Precision and Recall) und Mindestschweregrad (Show All, Show Medium and High, Show High). Damit wird erreicht, dass sich die gewählte Einstellung auf alle Views für Schwachstellen bezieht. Außerdem kann so zwischen den Konfigurationen gewechselt werden, ohne dass eine neue Schwachstellensuche notwendig ist.

Für die Persistierung von Einstellungen, Metadaten und gefundenen Schwachstellen verwendet das Modell eine eigene SQLite-Datenbank.

5.1.4 View

Die View-Komponente enthält die Ansichten des Schwachstellenprüfers. Diese wurden unter Zuhilfenahme des JFace UI Toolkits von Eclipse entwickelt. JFace erleichtert die Entwicklung von GUIs nach der MVC-Architektur.

Wenn ein JFace-Viewer erstellt wird, erzeugt es eine neue Instanz eines Contentproviders, der das Datenmodell des Viewers bildet. Um die Objekte des Contentproviders darzustellen, wird ein Labelprovider verwendet, der für ein gegebenes Objekt, Texte und Bilder ausgibt. In dem View Model Paket befinden sich die Content- und Labelprovider, sowie weitere Modellklassen, die zur Darstellung der Daten in der View benötit werden.

Die Contentprovider werden mit den Daten aus der Modell-Komponente befüllt. Wenn der Benutzer Aktionen durchführt die sich auf das globale Datenmodell auswirken, dann ruft der View-Controller Methoden der Controller-Komponente auf.

5.1.5 Externe Datenbank

Diese Komponente enthält Modelle und Controller, um eine externe Datenbank zu beschreiben und sich mit ihr zu verbinden. Der Controller verwendet die Komponente um nach Schwachstellen zu suchen.

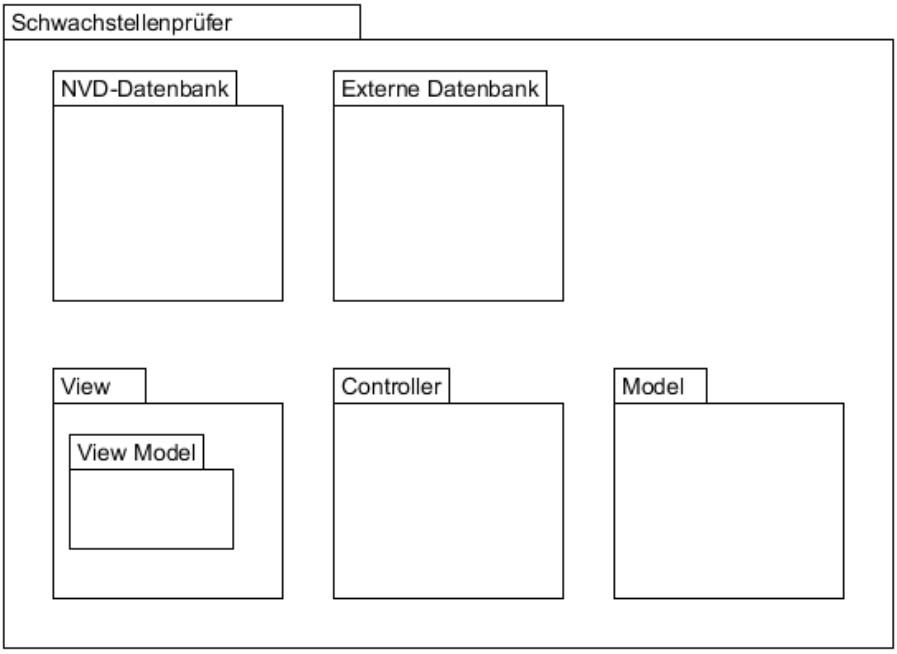

Abbildung 5.1: Bausteinsicht

5.1.6 Erweiterbarkeit

Der Zugriff auf alle Datenbanken erfolgt nach dem Data Access Object (DAO)-Pattern. DAOs bieten eine Abstraktionsschicht zwischen der Anwendung und verschiedenen Datanbankimplementierungen. Der Datenaustausch zwischen Anwendung und Datenbank erfolgt über Objekte. Die Anwendung verwendet dabei nicht direkt die Datenbank, sondern nutzt das DAO, das eine definierte Schnittstelle für die Datenbank anbietet (vgl. „Core J2EE Patterns - Data Access Object“ [22]). Der Schwachstellenprüfer zum Beispiel, ruft eine DAO auf, um eine Liste von Sicherheitslücken zu bekommen. Dadurch entsteht eine bessere Kapselung der Daten. Eine Anwendung benötigt keine Informationen über das verwendete Datenbankschema oder Art der Datenbank. Dies ermöglicht auch einen einfachen Umstieg von SQLite auf einer anderen Datenbank. Hierfür muss nur die DAO-Schnittstelle für die neue Datenbank implementiert werden.

Die Schwachstellensuche verwendet AnalyzedLib-Objekte, welche die gesammelten Metadaten und eine Referenz auf die Bibliothek enthalten. Durch die Abstraktion der Metadaten von einer Java-Bibliothek auf eine allgemeine Beschreibung, kann sie auch für weitere Programmiersprachen verwendet werden. Hierzu muss ein neuer Scanner erstellt werden, der die Bibliothek erkennt und die Metadaten ausliest.

Die NVD-Komponente kann durch ihre hohe Kohäsion und geringe Kopplung mit wenig Aufwand in Software-Projekte integriert werden, die eine Schwachstellendatenbank verwenden.

Zusätzliche Ansichten für den Schwachstellenprüfer können einfach integriert werden. Eclipse verwendet eine XML-Beschreibung der Abhängigkeiten in einer „plugin.xml“ genannten Datei. Eine neue Ansicht muss lediglich mit Beschreibung des vollständigen Klassennamens, ID, Namen und Kategorie, in die XML-Datei eingetragen werden. Sie ist fortan in der Liste aller Ansichten (Show Views) für die eingetragene Kategorie und Namen aufrufbar.

5.2 GUI

Die Ansichten des Schwachstellenprüfers wurden, unter Berücksichtigung der „Eclipse User Interface Guidelines“ [5] (Guideline 7.1 bis 7.21), erstellt. Demnach sollten lieber mehrere spezialisierte Views erstellt werden, als eine umfassende View. Dies hat den Vorteil, dass der Benutzer mehrere Views öffnen und frei in der Eclipse-Umgebung positionieren kann. So kann der Benutzer zum Beispiel die Ansichten über die Metadaten und Schwachstellen nebeneinander positionieren. Wenn eine View modifiziert wurde, sollten die Änderungen unmittelbar gespeichert werden. Views werden in Eclipse über das „Show View“-Menü aufgerufen. Eine Liste aller Views findet man unter „Window“ -> „Show View“ -> „Others...“. Die Views des Schwachstellenprüfers sind in der Kategorie „Vulnerability“ zu finden.

5.2.1 Vulnerability View

Die „Vulnerability View“ (Abbildung 5.2) ist eine Zusammenfassung der Schwachstellensuche, die in [FR.6] gefordert wurde. Sie listet listet alle unsicheren Java-Bibliotheken zusammen mit den gefundenen Sicherheitslücken. Um die wichtigsten Informationen übersichtlich darzustellen, wurde sie als Tree-View konzipiert.

Der oberste Eintrag enthält den Dateinamen und zusätzlich - falls vorhanden - den größten CVSS Score der gefundenen Sicherheitslücken. Damit sieht der Benutzer immer den Schweregrad, auch wenn der Knoten noch nicht expandiert wurde.

Eine Ebene tiefer wird die Quelle der Sicherheitslücke genannt. Sofern keine externe Datenbank hinzugefügt wurde, ist dies immer die NVD. Anderenfalls kann hier auch der Name der externen Datenbank stehen. Nach dem Datenbanksnamen folgt wieder der größte CVSS Score.

Die nächste Ebene zeigt die CPE, mit der die Sicherheitslücke gefunden wurde. Anstelle der URI werden hier die Attribute mit den jeweiligen Werten genannt. Die Attribute Vendor, Product und Version werden immer angezeigt, alle anderen nur, wenn sie einen von „ANY“ verschiedenen Wert haben. Wenn die CPE auch für frühere Versionen gilt, steht „<=“ vor der eigentlichen Versionsnummer. Im Falle einer externen Datenbank stehen hier die verwendeten Metadaten, mit der die Schwachstelle gefunden wurde. Zum Schluss folgt wieder der größte CVSS Score aller folgenden Schwachstellen. Diese Ebene wurde eingeführt, damit der Benutzer sofort erkennen kann, ob es sich um eine korrekte oder falsche CPE handelt. Bei einer falschen CPE, kann der Benutzer durch Rechtsklick auf den Knoten ein Kontextmenü öffnen und hat die Wahl entweder den Vendor, das Product oder die Version auf die jeweilige Blacklist für diese Bibliothek zu setzen. In dem Fall wird die Vulnerability View umgehend aktualisiert. Die CPE Blacklisten können in der Library View angesehen und editiert werden. Handelt es sich um eine Referenz einer externen Datenbank ist dies auch für die Vendor, Product oder Version möglich, falls diese vorhanden sind. Der Dateiname und Hash wurde nicht als Option hinzugefügt, weil hier eine falsche Zuordnung nicht erwartet wird.

Die nächst tiefere Ebene zeigt die Vulnerability ID. Für NVD-Einträge sind dies immer CVE-IDs. Nach der ID steht, falls vorhanden, der CVSS Score dieser Schwachstelle. Wenn mehrere Schwachstellen für diese CPE gefunden wurden, dann sind sie nach CVSS Score sortiert, wobei der höchste Eintrag an erster Stelle steht. Wenn der Benutzer nicht mehr vor dieser Schwachstelle gewarnt werden möchte, kann er sie auf eine Blacklist setzen, entweder für alle aktuellen und zukünftigen Projekte oder nur für das Projekt in dem die Schwachstelle gefunden wurde. Dies erfolgt wieder über ein Kontextmenü, dass durch einen Rechtsklick auf den Knoten aufgerufen werden kann.

In der nächsten Ebene folgen Information über die Sicherheitslücke. Der erste Knoten enthält die Beschreibung. Weil die Beschreibung ein langer Text sein kann, ist sie über den Description-Knoten ein und ausblendbar. Unter Windows

gilt die Beschränkung, dass nur die ersten 260 Zeichen eines Knotens des Tree-Viewers angezeigt werden. CVE-Beschreibungen können allerdings länger als 260 Zeichen sein. Deswegen und um eine bessere Lesbarkeit zu erreichen, wird der Text auf mehrere Knoten verteilt. Nach der Beschreibung folgen Knoten mit dem Veröffentlichungsdatum des CVE-Eintrages und dem Datum letzten Änderung des CVEs. Zum Schluss folgt ein Link, der auf die ausführliche CVE-Beschreibung der NVD-Webseite verweist. Durch klicken auf den Knoten wird der Standardbrowser des Betriebssystems geöffnet und die Webseite geladen. Es wurde zugunsten einer besseren Übersichtlichkeit darauf verzichtet weitere Informationen in dieser Ansicht darzustellen. Wenn der Benutzer weitere Details über die Schwachstelle lesen möchte, ist die Darstellung über die NVD-Webseite die bessere Wahl, weshalb der Link hinzugefügt wurde. Welche Informationen der externen Datenbank in dieser Ebene angezeigt werden ist abhängig von den Feldern, die der Benutzer beim Hinzufügen der Datenbank ausgewählt hat. Im Gegensatz zu den CVEs der NVD können hier noch weitere Links gelistet werden.

Um die Bedienung zu erleichtern, kann ein Knoten durch Doppelklick inklusive aller Kinderknoten geöffnet werden. Ein weiterer Doppelklick schließt den Knoten wieder. Des Weiteren können mit [STRG+A] alle Knoten markiert und mit [STRG+C] der Text von markierten Knoten kopiert werden.

Die Einträge der Vulnerability View lassen sich filtern, um nur neue Schwachstellen [FR.6.2] oder welche mit einem bestimmten CVSS-Score [FR.6.1] anzuzeigen. Zusätzlich sind Filter für bestimmte Versionseigenschaften vorhanden. Da die Schwachstellensuche hauptsächlich auf den CPE-Attributen Product und Version beruht, werden mehr falsche Ergebnisse angezeigt, wenn die Kriterien für eine akzeptierte Versionsnummer abgeschwächt werden, was zum Beispiel bei einem Wert von „ANY“ oder bei gesetztem Previous-Version-Flag der Fall ist. Mit den folgenden Filtern lässt sich deshalb die Precision der Ergebnisse erhöhen [FR.6.3]:

- Filter All Previous Versions
- Filter Previous Major Versions
- Filter Versions of Type ANY
- Filter Versions of Type NA

Für das Previous-Version-Flag sind zwei Filter vorhanden. Entweder können alle Schwachstellen mit Previous-Version-Flag gefiltert werden oder nur solche, deren Hauptversionsnummer nicht mit der Bibliotheksversion übereinstimmt. Alle Filter sind über das „View Menu“ (das Dreiecksymbol, oben rechts in der Toolbar) ein- und ausschaltbar. Das View Menu ist ein Standardelement von Eclipse Views, das für Einstellungen, Filter und Sortieroptionen verwendet wird. Dadurch ist es für erfahrene Eclipsenutzer schnell aufzufinden.

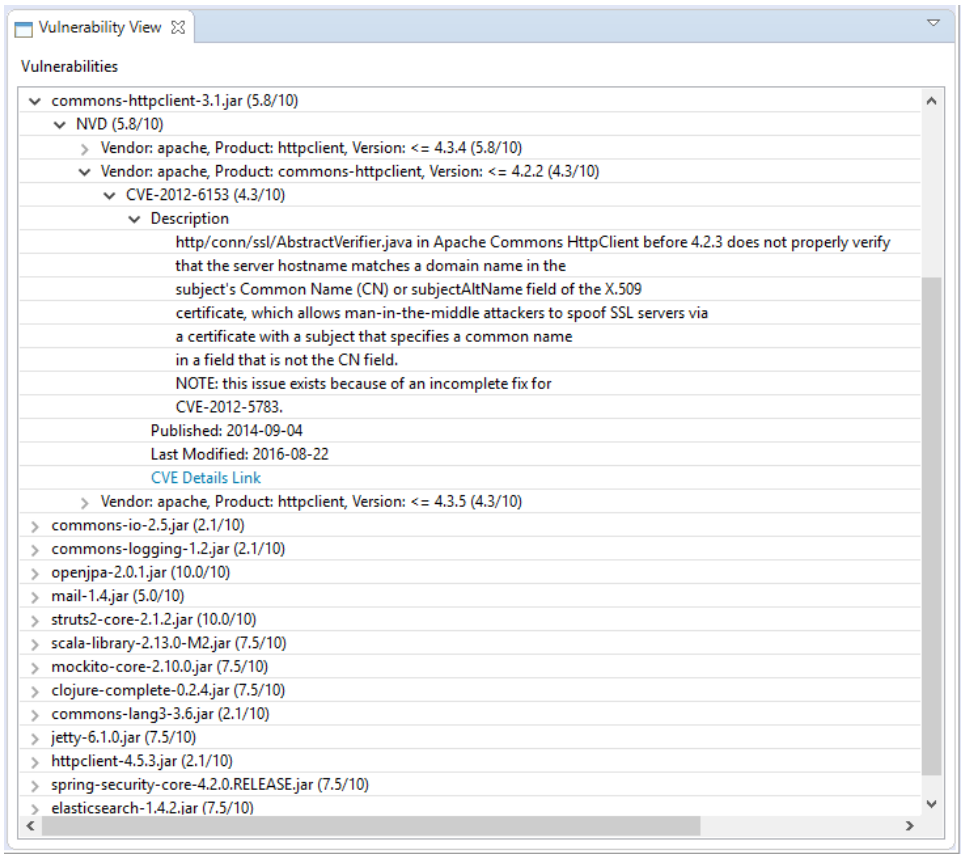

Abbildung 5.2: Vulnerability View

5.2.2 Library View

Über die „Library View“ (Abbildung 5.3) können alle gesammelten Metadaten zu einer Bibliothek angesehen und bearbeitet werden. Die Bibliothek wird über eine Combobox ausgewählt. Sie listet alphabetisch sortiert die Dateinamen von allen gefundenen Bibliotheken. Unter der Combobox befindet sich eine Checkbox mit der Beschriftung: „Ignore this library“. Wenn sie aktiviert ist, wird nicht mehr nach Sicherheitslücken für diese Bibliothek gesucht.

Die Metadaten über Vendor, Product und Version werden in separaten Listen angezeigt. Jede Zeile entspricht einem Token, so wurde aus dem Implementation-Title „Apache Software Foundation“ die Zeilen „apache“, „software“ und „foundation“. Diese Token werden für die Generierung der Potenzmenge (siehe Abschnitt 4.2.1) verwendet. Der Benutzer kann beliebig viele Zeilen hinzufügen oder editieren. Ein Token kann gelöscht werden, in dem der Text der Zeile gelöscht wird. Mithilfe dieser Tabellen ist es so möglich fehlerhafte Metadaten zu entfernen oder fehlende hinzuzufügen.

Zusätzlich zu den Vendor-, Product- und Version-Listen ist für jede eine

zusätzliche Blacklist vorhanden. So können mit der „Ignored Product Candidates“-Liste CPEs ausgeschlossen werden, deren Product-Attribut in der Liste enthalten ist. Standardmäßig sind die Listen leer. Sie können entweder in der View bearbeitet, oder über das Kontextmenü des CPE/Referenz-Knotens der Vulnerability View gesetzt werden.

Wenn eine Liste geändert wird, werden Schwachstellen auf Basis der alten Metadaten gelöscht. Die neuen Metadaten werden analysiert und anschließend eine Schwachstellensuche für diese Bibliothek gestartet.

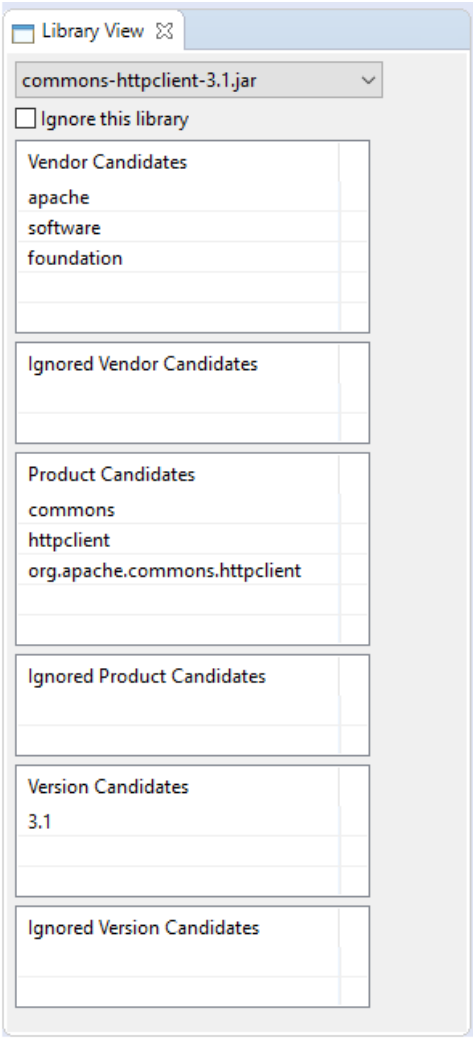

Abbildung 5.3: Library View

5.2.3 Warnung im Package-Explorer

Unsichere Bibliothek werden im Package-Explorer je nach Schweregrad des CVSS-Scores farblich hervorgehoben (siehe Abbildung 5.4). Entscheidend ist der größte Score aller Sicherheitslücken, die für diese Bibliothek gefunden wurde. Dieser Score wird zusätzlich hinter den Dateinamen angezeigt. Bei einem hohen Schweregrad mit einem Score von mindestens 7,0 oder höher, bekommt der Dateiname einen roten Hintergrund. Gleiches gilt, falls kein CVSS-Score vorhanden ist. In dem Fall wird immer von einem höchstmöglichen Schweregrad ausgegangen. Bei einem mittlerem Schweregrad mit einem Score von mindestens 4,0 wird der Hintergrund orange. Bei einem geringen Schweregrad dessen Score kleiner als 4,0 ist, wird der Hintergrund gelb. Damit wird die Anforderung [FR.5.1] erfüllt.

Ordner die verwundbare Bibliotheken enthalten werden expandiert, damit die Warnung für den Benutzer sichtbar wird.

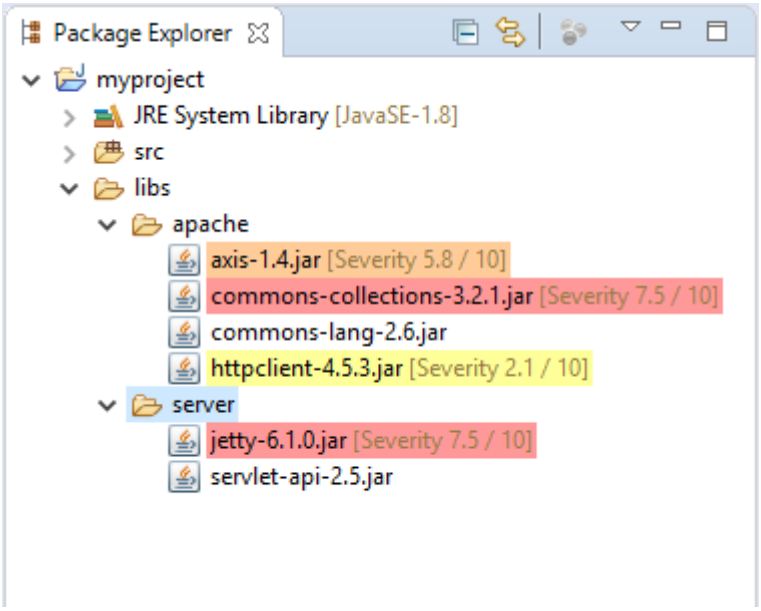

Abbildung 5.4: Package-Explorer

5.2.4 Warndialog

Wenn Schwachstellen für eine Bibliothek gefunden wurden, wird ein Warndialog (Abbildung 5.5) geöffnet, der die unsicheren Bibliotheken nennt. Der Warndialog soll sicherstellen, dass neu gefundene Schwachstellen von dem Benutzer registriert werden. Angenommen für eine Bibliothek wurde bereits in der Vergangenheit eine Schwachstelle gefunden. Wenn nun eine zusätzliche Schwachstelle entdeckt wird,

bemerkt es der Benutzer unter Umständen nicht, wenn er nur in den Package-Explorer und nicht in die Schwachstellenzusammenfassung schaut.

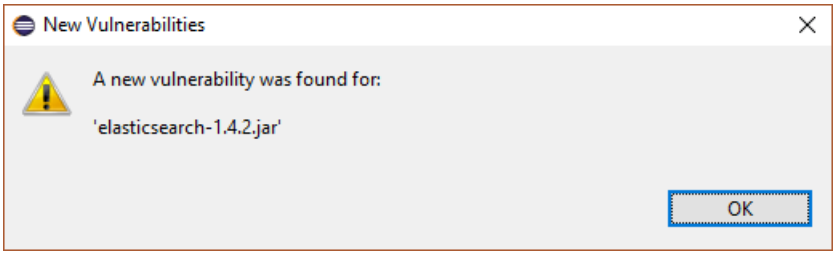

Abbildung 5.5: Warnbenachrichtigung

5.2.5 Vulnerability Blacklist

Die „Vulnerability Blacklist“ (Abbildung 5.6) enthält alle Schwachstellen, die vom Benutzer in der Vulnerability View aussortiert wurden. Über diese Ansicht können die Schwachstellen wieder von der Liste entfernt werden. Die Blacklist gilt für das Dreier-Tupel aus Projekt, Datenbank und Schwachstellen-ID. Jeder Wert wird in einer eigenen Spalte in der Ansicht ausgegeben. Wenn die Blacklist für alle Projekte gilt, steht in der Spalte der Wert „ALL“. Wenn es sich bei der Schwachstellen-ID um eine CVE-ID handelt, was für alle NVD-Einträge gilt, dann kann über das Kontextmenü die NVD-Webseite für diese Schwachstelle geöffnet werden. So ist für den Benutzer schnell nachzuvollziehen, welche Schwachstelle hier referenziert wurde. Über das Kontextmenü kann der Eintrag auch wieder aus der Blacklist entfernt werden. In diesem Fall wird eine neue Schwachstellensuche gestartet.

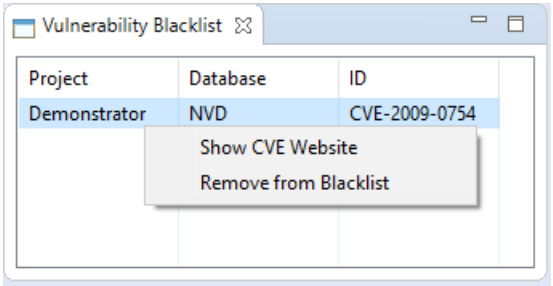

Abbildung 5.6: Vulnerability Blacklist

5.2.6 Externe SQLite-Datenbank

Das Hinzufügen einer externen SQLite-Datenbank für die Schwachstellensuche erfordert mehrere Schritte. Aus diesem Grund wurde hierfür ein Wizard implementiert, der den Benutzer durch die einzelnen Schritte führt. Dieser Vorgang wird

in Abschnitt 5.2.6 beschrieben. Wo eine Auflistung aller externen Datenbanken zu finden ist, und wie diese gelöscht werden, beschreibt der Abschnitt 5.2.6.

Hinzufügen einer SQLite-Datenbank

Der Wizard wird über das Menü des Schwachstellenprüfers unter „Vulnerability Checker“ -> „Add External Vulnerability DB“ aufgerufen. Auf der ersten Seite des Wizards wird der Benutzer aufgefordert, einen Namen für die Datenbank und den Dateipfad einzugeben. Der Name wird verwendet, um zwischen mehreren Datenbanken zu unterscheiden, zum Beispiel in der Vulnerability View. Der Dateipfad wird über einen File-Dialog ausgewählt. Durch einen Mausklick auf Connect, versucht sich der Schwachstellenprüfer mit der Datenbank zu verbinden. Wenn dies erfolgreich war, wird die nächste Seite freigeschaltet und der Benutzer kann auf Next klicken.

Auf der zweiten Seite (Abbildung 5.7) muss der Benutzer auswählen, welche Metadaten auf welche Tabellenspalte der externen Datenbank abgebildet werden sollen. Die Ansicht zeigt die Tabellen der externen Datenbank an. Der Benutzer kann durch einen Rechtsklick auf eine Tabellenspalte auswählen, ob die Spalte dem Dateinamen, einem SHA-256-Hash, dem Vendor, Product Name oder Version zugeordnet werden soll. Das gewählte Mapping der Metadaten auf die Tabelle wird in einer Liste dargestellt. Durch einen Klick auf „Map input“ wird das Mapping vom Schwachstellenprüfer im Mapping Generator verarbeitet. Wenn mindestens eine Spalte zugewiesen wurde, kann auf die nächste Seite geblättert werden.

Zum Schluss werden auf der letzten Seite (Abbildung 5.8) die Tabellenspalten den Attributen einer Schwachstelle, nach dem vorherigen Prinzip, zugeordnet. Dabei muss die Vulnerability ID immer vergeben werden, um Schwachstellen eindeutig referenzieren zu können. Nach einem Klick auf „Map output“, wird das Mapping abgeschlossen. Der Wizard kann nun durch Klicken auf Finish beendet werden. Die externe Datenbank wurde dann erfolgreich hinzugefügt. Es folgt eine automatische Schwachstellensuche.

Verwalten der externen Datenbanken

Über die Ansicht „External Database List“ (Abbildung 5.9) können alle hinzugefügten Datenbanken überblickt werden. Jede Datenbank wird mit ihrem Namen und Pfad gelistet. Durch einen Rechtsklick auf einen Listeneintrag wird ein Kontextmenü geöffnet, mit dem eine Datenbank wieder entfernt werden kann.

5.2.7 Vulnerability Checker Menü

Über das „Vulnerability Checker“-Menü kann manuell der Update-Prozess und die Schwachstellensuche gestartet werden. Dies ist im Normalfall nicht notwendig, weil zu Beginn nach Updates gesucht und die Schwachstellensuche automatisch nach Events (unter anderem: abgeschlossene Aktualisierung oder neue Bibliothek)

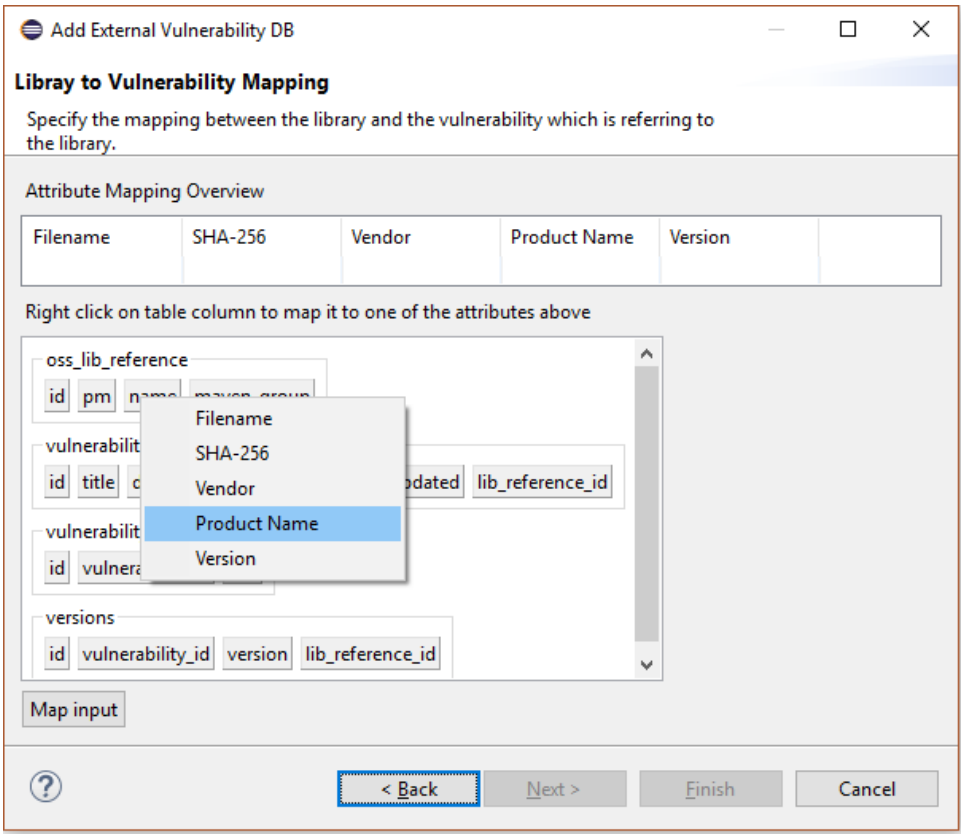

Abbildung 5.7: Wizard Seite 2

gestartet wird. Allerdings ist der Benutzer in der Lage, die jeweiligen Prozesse zu unterbrechen und benötigt deswegen auch die Möglichkeit sie wieder zu starten.

Über das Menü erfolgt zudem die Konfiguration des Schwachstellenprüfers (Abbildung 5.10). Im Untermenü „Severity Filter“ kann ausgewählt werden, ob vor allen Schwachstellen oder nur vor solchen mit einem mindestens mittleren oder hohen CVSS-Schweregrad gewarnt wird.

Im Untermenü „Search Engine“ wird eine der drei Konfigurationen „High Recall“, „Best Combined Recall and Precision“, „High Precision“ ausgewählt. Standardmäßig ist die Einstellung High Recall aktiv.

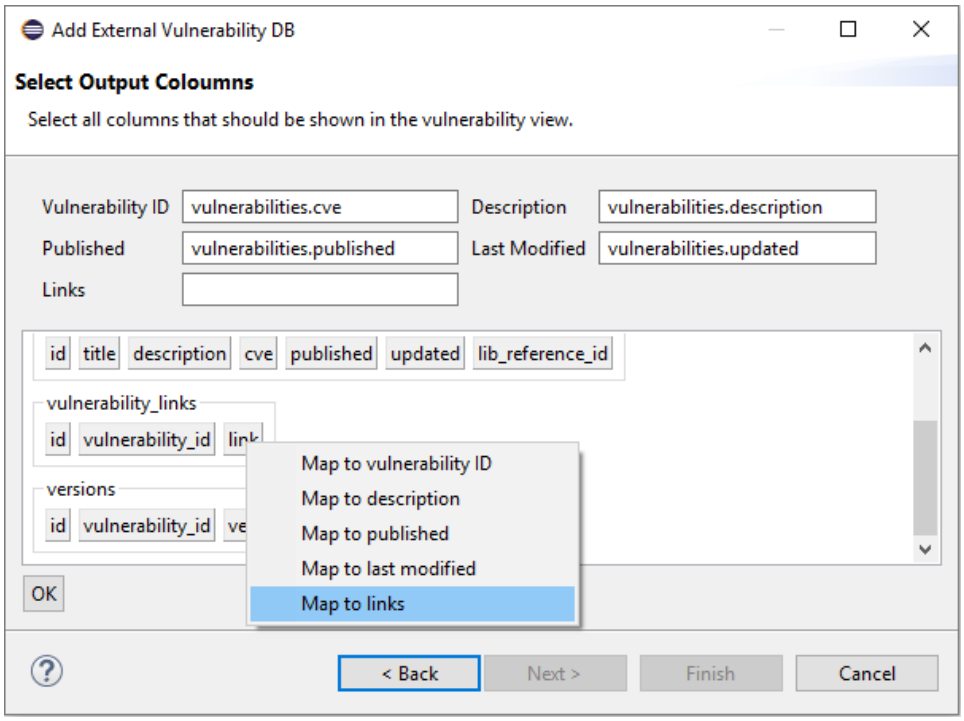

Abbildung 5.8: Wizard Seite 3

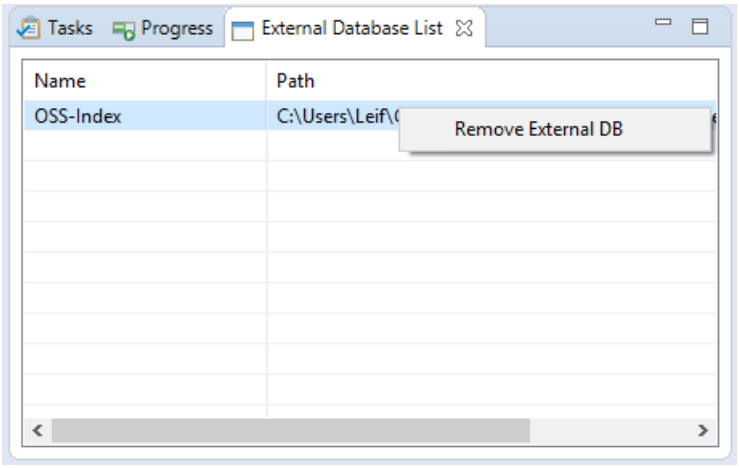

Abbildung 5.9: External Database List

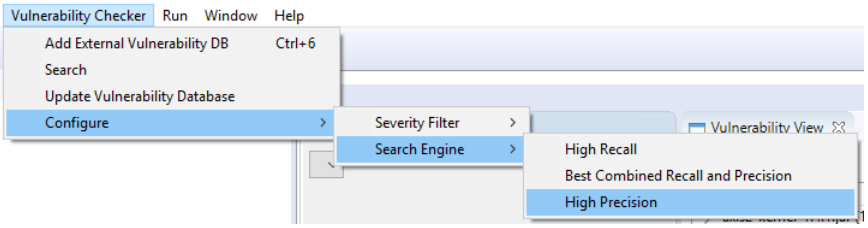

Abbildung 5.10: Konfiguration des Schwachstellenprüfers

Kapitel 6

Evaluation

In diesem Kapitel wird die Performance des Schwachstellenprüfers im Hinblick auf die gefundenen Sicherheitslücken 6.1 und die benötigte Zeit für das Suchen, Importieren und Aktualisieren der Daten 6.2, untersucht.

6.1 Retrieval

In diesem Abschnitt wird untersucht, wie viele Sicherheitslücken der Schwachstellenprüfer für eine gegebene Menge von Java-Bibliotheken findet und wie viele Falschmeldungen produziert werden. Dafür wird zunächst die verwendete Methodik vorgestellt 6.1.1, anschließend die Ergebnisse ausgewertet 6.1.1 und ein Fazit gezogen 6.1.3.

6.1.1 Methodik

Im Abschnitt Methodik wird zunächst die Auswahl der Test-Bibliotheken erläutert, darauffolgend wird beschrieben, wie die Referenzwerte bestimmt wurden und schließlich werden die verwendeten Metriken beleuchtet.

Test-Bibliotheken

Für die Evaluation wurden insgesamt 38 Java-Bibliotheken unter verschiedenen Gesichtspunkten ausgewählt. Zum einen unterteilt sich die Menge in 17 Bibliotheken mit Sicherheitslücken und in 21 sicheren Bibliotheken (siehe Tabelle 6.1. Ausschlaggebend waren die Daten der NVD-Data-Feeds[1] vom 18.09.2017. So lassen sich zum einen die Anzahl der gefundenen CVEs, als auch die Anzahl der falschen Warnungen auswerten. 20 Bibliotheken entstammen der Top 20 der Popular-Kategorie von MVN Repository[2]. So kann der Schwachstellenprüfer mit

[1] `https://nvd.nist.gov/vuln/data-feeds#CVE_FEED`
[2] `https://mvnrepository.com/popular`

Mit CVE-Einträgen	Ohne CVE-Einträge
Apache Axis 1.4	JUnit 4.12
Apache Axis2 1.4.1	Scala Library 2.13.0-M2
Apache Commons Fileupload 1.2.1	SLF4J API Module 1.7.25
Apache Commons HttpClient 3.1	Guava 23.0
Jetty 4.2.27	Apache Log4j 1.2.17
Jetty 6.1.0	Apache Commons IO 2.5
javax.mail 1.4	javax.mailapi 1.5.6
Apache OpenJPA 2.0.1	Apache Commons Logging 1.2
Apache Struts 1.2.7	Logback Classic Module 1.2.3
Apache Struts2 Core 2.1.2	Clojure 1.8.0
Apache Xalan-Java 2.7.0	SLF4J LOG4J 12 Binding 1.7.25
XStream 1.4.8	Mockito All 1.10.19
OpenSymphony XWork 2.1.1	Mockito Core 2.10.0
Spring Security Core 4.2.0	Servlet API 2.5
Elasticsearch 1.4.2	Apache Commons Lang 2.6
Apache Commons Collection 3.2.1	Apache Commons Lang3 3.6
Jackson Databind 2.9.1	javax.servlet-api 4.0.0
	Clojure Tools.nrepl 0.2.13
	Apache HttpClient 4.5.3
	Daytrader-EJB 2.1.7
	Clojure Complete 0.2.4

Tabelle 6.1: Test-Bibliotheken

in der Praxis häufig verwendeten Bibliotheken getestet werden. Von diesen 20 Bibliotheken ist für eine (Jackson Databind) ein CVE hinterlegt.

Referenzwerte

Für jede Java-Bibliothek wurde händisch eine Liste von zutreffenden CPEs erstellt. Ein CPE gilt dann als zutreffend, wenn seine Komponenten *Vendor*, *Product*, *Version* und *Update* mit der Bibliothek korrespondieren. Die Version muss hierfür entweder gleich der Bibliotheksversion sein oder der CPE muss für alle vorherigen Softwareversionen gelten. CPEs für Vorgänger der Bibliotheken - zum Beispiel Alpha-, Beta- und Relase-Candidate-Versionen - wurden nicht berücksichtigt. Für jede CPE wurden die zugehörigen CVEs ermittelt. Diese Daten bilden den Referenzwert.

Metriken

Für die Evaluierung der Sicherheitslücken, die der Schwachstellenprüfer ausgibt, werden Methoden des Information-Retrievals verwendet, wie sie von Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze in Introduction to Infor-

mation Retrieval [9] beschrieben wurden. Sie sollen im Folgenden kurz mit Bezug zum Schwachstellenprüfer vorgestellt werden.

Die Auswertung basiert auf den Werten der Wahrheitsmatrix, auch Konfusionsmatrix genannt. Anhand der Werte aus Tabelle 6.2 der Java-Bibliothek Struts 2, soll diese beispielhaft erläutert werden. Struts 2 enthält 33 bekannte Sicherheitslücken.

	CVE vorhanden	CVE nicht vorhanden	Summe
CVE gefunden	33 TP	2 FP	35
CVE nicht gefunden	0 FN	94750 TN	94750
Summe	33	94752	94785

Tabelle 6.2: Beispiel Konfusionsmatrix für Struts 2

Sicherheitslücken, die von dem Schwachstellenprüfer gefunden wurden, werden als True Positive (TP) und nicht gefundene Sicherheitslücken als False Negative (FN) bezeichnet. In dem Beispiel wurden alle Sicherheitslücken gefunden. Somit gibt es 33 TP und 0 FN. Im Beispiel wurden fälschlicherweise zwei weitere Sicherheitslücken gefunden. Diese werden False Positive (FP) genannt. Übrig bleiben von den insgesamt 94785 Sicherheitslücken der Datenbank, 94750 acrfulltn, also Sicherheitslücken die irrelevant für die Struts 2-Bibliothek sind.

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

Um den Nutzen des Schwachstellenprüfers einschätzen zu können, ist es wichtig zu wissen, wie viel Prozent aller Sicherheitslücken tatsächlich gefunden wurden. Dies lässt sich mit Hilfe des Recall berechnen. Im schlechtesten Fall, bei einem Recall von null, wurde keine relevante Sicherheitslücke gefunden. Umgekehrt besagt ein Recall von eins, dass alle relevanten Sicherheitslücken gefunden wurden, wie im Falle von Tabelle 6.2. Dies lässt sich allerdings auch trivial erreichen, in dem man immer alle Sicherheitslücken der Datenbank ausgibt. In diesem Fall hätte der Schwachstellenprüfer zwar alle relevanten Sicherheitslücken ausgegeben, der Nutzen für den Benutzer wäre aber offensichtlich gleich null.

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

Dieses Beispiel zeigt, dass der Recall alleine nicht aussagekräftig genug ist. Zusätzlich muss noch überprüft werden, ob die Sicherheitslücke, die das Programm der Java-Bibliothek zugeordnet hat, auch wirklich für diese Bibliothek relevant ist. Der Anteil, wie viel Prozent der gelieferten Sicherheitslücken relevant sind, wird mit der Precision ausgedrückt. Bei einer Precision von eins sind alle Sicherheitslücken relevant, bei einem Wert von null keine. Eine höhere Precision wird meist mit einem schlechteren Recall erkauft. Werden zum Beispiel mehr Sicherheitslücken herausgefiltert, die weniger zuverlässig einer Bibliothek zugeordnet werden können, dann steigt die Precision, weil weniger FP auftreten. Gleichzeitig sinkt aber auch der Recall, weil die Anzahl der FN steigt. Im Extremfall, wenn nur eine

von insgesamt 100 relevanten Sicherheitslücken ausgegeben wird, ist die Precision bei 100%, der Recall allerdings nur 1%. Ein weiter Sonderfall besteht, wenn der Schwachstellenprüfer keine Sicherheitslücke findet, obwohl welche vorhanden wären. In diesem Fall wären Zähler und Nenner der Precision null. Weil in der leeren Menge keine nicht relevante Sicherheitslücke enthalten ist, wird hier eine Precision von 1 angegeben. Im Beispiel aus 6.2 beträgt die Precision 33/35 = 94,29%.

Welche Balance zwischen Recall und Precision sollte der Schwachstellenprüfer einnehmen? Auf der einen Seite steht das Ziel, möglichst alle Sicherheitslücken zu finden (hoher Recall). Wenn der Benutzer dabei allerdings mit zu vielen Falschmeldungen konfrontiert wird (niedrige Precision), dann verwendet er im schlechtesten Fall dieses Sicherheitswerkzeug gar nicht mehr. Auch der umgekehrte Fall, hohe Precision und niedriger Recall, ist nicht akzeptabel. Bei einem Recall unter, zum Beispiel, 50% kann der Benutzer genötigt sein, selbst nach Sicherheitslücken zu suchen. Eine Arbeit, die dem Benutzer eigentlich abgenommen werden sollte. In dieser Evaluation werden Recall und Precision gleich gewichtet.

Wie kann die Performance des Schwachstellenprüfers, unter Berücksichtigung von Recall und Precision, in einer Zahl ausgedrückt werden? Die einfachste Möglichkeit wäre, zum Beispiel, das arithmetische Mittel zu bilden. Dies hätte aber den Nachteil, dass sehr weit auseinanderklaffende Werte im Durchschnitt besser dastehen, als sie eigentlich sind. Ein Beispiel: Angenommen der Schwachstellenprüfer findet keine einzige Sicherheitslücke. In diesem Fall wäre der Recall bei 0% und die Precision bei 100%, also im Mittel 50%.

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Besser eignet sich das harmonische Mittel zwischen Recall und Precision. Hier liegt das Mittel näher am kleineren Wert. Im oben genannten Beispiel wären es sogar 0%. Im Idealfall, wenn Recall und Precision beide bei 100% liegen, wäre auch das harmonische Mittel bei 100%. Das gewichtete harmonische Mittel aus Recall und Precision wird auch F-measure genannt. Wenn Recall und Precision gleich gewichtet werden, wird genauer von F1 gesprochen.

Die bisher vorgestellten Metriken bezogen sich auf die positiven Fälle, wie viele Sicherheitslücken wurden gefunden (Recall) und wie viele von den Sicherheitslücken sind korrekt (Precision). Diese lassen sich auch für die negativen Fälle bestimmen und heißen dann True Negative Rate (TNR) und Negative Prediction Value (NPV). Diese Metriken sind jedoch, ebenso wie die Accuracy, stark von den TN abhängig. In Information-Retrieval-Systemen sind die TN jedoch überproportional vorhanden. Das führt bei TNR, NPV und Accuracy zu Werten nahe 1 respektive 100%.

Betrachtet man das Ergebnis des Schwachstellenprüfers aus einer anderen Perspektive, sind diese Metriken dennoch hilfreich. Hierfür wird ein neues Beispiel benötigt. Der Schwachstellenprüfer markiert unsichere Java-Bibliotheken im Package-Explorer von Eclipse. Eine Bibliothek wird als unsicher eingeschätzt,

	Unsichere JAR	Sichere JAR	Summe
Unsicher eingeschätzt	17	8	25
Sicher eingeschätzt	0	13	13
Summe	17	21	38

Tabelle 6.3: Beispiel Klassifikation

wenn mindestens eine Sicherheitslücke gefunden wurde. Anderenfalls wird angenommen, dass die Bibliothek sicher ist. Tabelle 6.3 zeigt beispielhafte Werte für solch eine Einschätzung. Hier wurden alle 17 unsichere Bibliotheken entdeckt. Von den 21 sicheren Bibliotheken wurden allerdings 8 fälschlicherweise als unsicher deklariert. Anhand dieses Beispiels sollen nun die restlichen Metriken erläutert werden.

$$\text{True Negative Rate} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Die True Negative Rate beschreibt, wie viele der sicheren Bibliotheken (TN) korrekt klassifiziert wurden. In diesem Beispiel wurden 13/21 = 61,9% der unsicheren Bibliotheken gefunden. Analog zum Recall lässt sich diese Metrik trivial optimieren, indem einfach alle Bibliotheken als unsicher eingeschätzt werden. Deswegen gibt es auch hier ein Pendant zur Precision: der Negative Predictive Value (NPV).

$$\text{Negative Predictive Value} = \frac{\text{TN}}{\text{TN} + \text{FN}}$$

Der Negative Predictive Value besagt, mit welcher Wahrscheinlichkeit eine als sicher eingeschätzte Bibliothek (TN) tatsächlich sicher ist. Im Beispiel 6.2 gibt es keine unsichere Bibliothek (FN), die nicht gefunden wurde. Damit beträgt der NPV 100% und alle als sicher eingestufte Bibliotheken sind auch wirklich sicher.

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Mit Hilfe der Accuracy kann nun ein abschließendes Urteil über die Richtigkeit der Klassifikation gefällt werden. Die Accuracy beschreibt, den Anteil aller korrekten Klassifikatoren, also den Anteil aller als sicher und unsicher eingeschätzten Bibliotheken. Für die Matrix 6.2 ergibt sich somit eine Accuracy von 78,95%.

Um die Problematik bei überproportional vielen TN noch einmal zu verdeutlichen, soll die Acurracy für die Zuordnung von Sicherheitslücken berechnet werden. Gegeben sei die Tabelle 6.4. Der Schwachstellenprüfer hätte hier seine Aufgabe nicht erfüllt, da die Sicherheitslücke nicht gefunden wurde. Trotzdem beträgt die Accuracy hier 99,9989%.

	CVE vorhanden	CVE nicht vorhanden	Summe
CVE gefunden	0 TP	0 FP	0
CVE nicht gefunden	1 FN	94784 TN	94785
Summe	1	94784	94785

Tabelle 6.4: Fiktives Beispiel einer Konfusionsmatrix mit nur einem CVE-Eintrag

6.1.2 Auswertung

Im Folgenden werden die Ergebnisse für jede Konfiguration des Schwachstellenprüfers - *High Recall*, *Combined Recall and Precision* sowie *High Precision* - ausgewertet. Die Daten werden hierfür aus drei Sichten ausgewertet: Zum einen werden Precision und Recall für gefundene CPEs berechnet. Dies eignet sich für eine Abschätzung der potenziellen Leistung, weil nur über korrekt zugeordneten CPEs Sicherheitslücken gefunden werden. Bei einem schlechten Recall der CPEs ist es möglich, dass auch viele CVEs nicht gefunden werden. Allerdings verweisen die verschiedenen CPEs mitunter auf die gleichen CVEs. Deshalb werden die CVEs noch einmal separat betrachtet und geben einen Einblick auf die tatsächliche Performance. Schließlich wird die Klassifizierung in sichere und unsichere Bibliotheken untersucht, wie sie im Eclipse-Package-Explorer vorzufinden ist. Stand 18.09.2017 enthält die NVD 94.785 CVE-Einträge und 214.955 CPEs

Recall-Konfiguration

In der Recall-konfiguration soll der Schwachstellenprüfer möglichst alle Sicherheitslücken für eine Konfiguration finden. Dies gelingt auch bei 96% der CPEs und 95% der CVEs. Die CPEs

- cpe:/a:mortbay_jetty:jetty:6.1.6rc0
- cpe:/a:eclipse:jetty:9.4.6:20170531

für die Jetty 4 Bibliothek wurden nicht gefunden. Dies liegt daran, dass im JAR-Manifest als Implementation-Vendor „Mort Bay Consulting, Pty. Ltd.“ angegeben wurde. Die CPE-Vendor „mortbay_jetty“ und „eclipse“ liegen jedoch nicht mehr in der Fehlertoleranz und wurden deshalb ausgefiltert.

Die Precision der gefundenen CPEs beträgt nur 47,57%. Das hat den Grund, weil zugunsten des Recalls laxere Filterkriterien für die CPE-Komponenten Version und Update gelten. Die Precision der CVEs ist mit 54,44% etwas besser. Mit insgesamt 21 falschen CPEs ist die häufigste Ursache ein unpassender CPE-Update. Hier handelt es sich um eine sehr strenge Auslegung von dem, was eine korrekte CPE ist und was nicht. In 20 der 21 Fällen beschreibt die Update-Komponente, ob es sich um eine Alpha, Beta oder einen Release-Candidate handelt. Betroffen waren hier die Bibliotheken Axis 1 und 2, sowie Jetty 6.

Streng genommen sind dies Vorgänger der verwendeten Bibliotheksversion und wurden dementsprechend als inkorrekt gewertet. Allerdings verweisen sie in diesem Test auf die gleichen CVEs, die mit korrekten CPEs gefunden wurden. Mit 19 inkorrekten Fällen sind am zweithäufigsten CPEs betroffen, die mit der Version „ANY“ ausgewiesen sind. Dem Gegenüber stehen drei CPEs mit Version „ANY“ die korrekt der Bibliothek zugewiesen wurden. Ebenso problematisch sind Versionen der Kategorie „NA“ mit 13 False-Positives und nur drei True-Positives. Der Grund für die vielen False-Positives mit „ANY“ und „NA“ ist, dass das Filterkriterium Version entfällt. Wenn der Bibliothek keine Herstellerinformation entnommen werden konnte, dann beruht die Zurodnung der CPE nur auf einem (Teil-)Match zwischen dem Dateinamen und / oder dem Implementation-Title. Die dritte und letzte Ursache für False-Positive sind CPEs, die vorherige Versionen mit einschließen. Auch hier besteht die Problematik darin, dass das Kriterium der passenden Version abgeschwächt wurde. Für ein Match muss die CPE-Version nur noch größer gleich der Bibliotheksversion sein. Dies führte zu neun inkorrekten CPEs. Schaut man sich diese genauer an, fällt auf, dass bei allen die Hauptversionsnummer größer ist als bei der Bibliothek. Die Hauptversionsnummer ist die führende Zahl, die mit einem Punkt von der Nebenversionsnummer und nach einem weiteren Punkt von der Revisionsnummer getrennt ist. Zum Beispiel ist 5 die Hauptversionsnummer von 5.21.02. Dem gegenüber stehen allerdings 51 CPEs (19 mit gleicher Hauptversionsnummer), die als korrekt eingestuft wurden.

	Unsichere JAR	Sichere JAR	Summe
Unsicher eingeschätzt	17	8	25
Sicher eingeschätzt	0	13	13
Summe	17	21	38

Tabelle 6.5: Konfusionsmatrix für Recall-Konfiguration

Unter den Top-20 der Mavenrepositories sind viele Apache-Bibliotheken enthalten. Gemein haben viele dieser Bibliotheken, dass sowohl beim Implementation-Vendor als auch beim Implementation-Title das Wort „Apache“ enthalten ist. Exemplarisch hierfür steht Tabelle 6.6. Dieser Umstand führt dazu,

Implementation-Title	Apache Commons IO
Implementation-Vendor	The Apache Software Foundation

Tabelle 6.6: Manifest Auszug

dass es zu einem Match mit der CPE cpe:/a:apache:apache (Version = ANY) kommt und damit vier der acht False-Positives aus Tabelle 6.5 nur deshalb falsch klassifiziert wurden. Betroffen sind:

- Apache Commons IO 2.5
- Apache Commons Lang3 3.6

- Apache HttpClient 4.5.3
- Apache Commons Logging 1.2

Die restlichen vier Bibliotheken, die zu unrecht als unsicher klassifiziert wurden sind

- Mockito Core 2.10.0
- Scala Library 2.13.0-M2
- SLF4J API Module 1.7.25
- Clojure Complete 0.2.4

Bei diesen Bibliotheken kam es zu einem Teilmatch dem CPE-Product und dem Dateinamen (Core, Library, M2, API und Complete). Auch hier sind die CPE-Versionen entweder „ANY“, „NA“ oder „<=“.

Konfiguration	Recall	Precision	F1	TNR	NPV	Accuracy
Recall	**100**	68	80,95	61,90	**100**	78,95
Kombiniert	94	84,21	88,89	85,71	94,74	89,47
Precision	82,35	**100**	**90,32**	**100**	87,5	**92,11**

Tabelle 6.7: Klassifikation im Package-Explorer

Als nächstes sollen die Warnungen im Package-Explorer untersucht werden, der unsichere Bibliotheken farblich hervorhebt. Die folgenden Werte beziehen sich dabei auf die Tabelle 6.7. In der Recall-Konfiguration wurden alle unsicheren Bibliotheken gefunden. Bei einer Precision von 68% ergibt das eine gute Gesamtperformance bezüglich der F1 mit 81%. Die acht False-Positive Bibliotheken senken die True-Negatvie-Rate auf 62%. Ansonsten sind alle als sicher eingeschätzte Bibliotheken tatsächlich sicher, wie es sich aus dem hundertprozentigen Negative-Predictive-Value ablesen lässt. Schlussendlich hat die Klassifizierung nach sichere und unsichere Bibliotheken und damit die Warnungen aus dem Package-Explorer eine gute Accuracy von 78,95%.

Kombinierte Konfiguration

Um die Precision zu verbessern, unter Beibehaltung eines möglichst hohen Recalls, wurde eine weitere Konfiguration eingeführt, die hier als „Kombiniert“ bezeichnet wird, weil es sich um eine Kombination aus Recall und Precision handelt. Die höhere Precision werden durch zusätzliche Filterregeln bei den CPE-Komponenten Version und Update erreicht. Diese wurden so gewählt, dass deutlich mehr False-Positives als True-Positives entfernt werden. Hierfür sei ein Blick auf die Tabelle 6.8 angebracht. Das Update-Feld der CPE führte sowohl zu vielen richtigen als auch falschen Ergebnissen. Bei den richtigen korrekten Klassifizierungen

CPE-Komponente	TP	FP
Update	16	21
Version = ANY	5	19
Version = NA	3	13
Version <=	51	9

Tabelle 6.8: Auswirkungen von Version und Update auf die Wahrheitsmatrix

handelt es sich um CPEs, die vorherige Versionen mit einschließen, zum Beispiel für die Jetty 4 Bibliothek die CPE „cpe:/a:mortbay:jetty:8.1.0:rc2“. Für alle anderen wird nun zusätzlich überprüft, ob das Update im Dateinamen enthalten ist. Vorversionen wie, Alpha und Beta sind üblicherweise Teil des Dateinamens. Die referenzierte Bibliothek von der vorherigen CPE hat im Mavenrepository den Dateinamen „jetty-maven-plugin-8.1.0.RC2.jar“ und hätte so die Bedingung erfüllt. Eine weitere Regel lautet, dass Versionen mit „ANY“ ausgeschlossen werden. Diese Versionen produzieren deutlich mehr False-Positives als True-Negatives. Dies gilt zwar auch für Versionen mit „NA“, jedoch konnten diese schon durch den vorherigen Update-Filter ausgeschlossen werden.

Tabelle 6.10 zeigt die Auswirkungen der neuen Filteregeln auf die Perfomance. Mit der kombinierten Konfiguration steigt die Precision für CPE von 48% auf 75%. Dies geht einher mit einem Verlust von 23 Prozentpunkten beim Recall auf schließlich 73%. Der F1-Wert aus den mittleren Werten von Recall und Pecision verbesserte sich um 10 Prozentpunkte auf 74%, den besten Wert aller Konfiguration. Die Perfomance der gefundenen CVEs entspricht in dieser Konfiguration fast der Werte der CPEs, mit 73% Recall, 76% Precision und 74% F1.

	Unsichere JAR	Sichere JAR	Summe
Unsicher eingeschätzt	16	3	19
Sicher eingeschätzt	1	18	19
Summe	17	21	38

Tabelle 6.9: Konfusionsmatrix für kombinierte Konfiguration

Betrachtet man die Klassifizierung aus Tabelle 6.9 des Package-Explorers, fällt auf, dass nun statt acht nur noch drei sichere Bibliotheken falsch eingeschätzt wurden. Damit steigt die Precision in Tabelle 6.7 um 16 Prozentpunkte, gegenüber der Recall-Konfiguration, auf 84%. Namentlich wurden die folgenden sicheren Bibliotheken falsch klassifiziert:

- Scala Library 2.13.0-M2
- SLF4J API Module 1.7.25
- Clojure Complete 0.2.4

Alle drei haben gemein, dass die CPEs sich auch auf vorherige Versionen beziehen und dass die Hauptversionsnummer nicht mit der Hauptversionsnummer der Bibliotheken übereinstimmt. Des weiteren wurde für die Bibliothek

- javax.mail 1.4

der einzig vorhandene CVE nicht gefunden, weil dieser mit „ANY“ deklariert ist und deshalb ausgefiltert wurde. Somit wurde diese Bibliothek fälschlich als sicher eingeschätzt. Damit fällt der Recall von 100 auf 94% und der Negative-Predictive-Value auf 95%. Im Gegenzug verbessert sich aufgrund der wenigen False-Positives sowohl die True-Negative-Rate auf 86% als auch die Gesamtperformance bezüglich der F1 und der Accuracy auf 89%.

Konfiguration	Ziel	Mittlerer Recall	Mittlere Precision	F1
Recall	CPE	**96,27**	47,57	63,67
Recall	CVE	**95,1**	54,44	69,24
Precision	CPE	57,74	**100**	73,21
Precision	CVE	65,54	**100**	**79,18**
Kombiniert	CPE	72,96	74,91	**73,92**
Kombiniert	CVE	72,57	76,44	74,46

Tabelle 6.10: Konfigurationen im Vergleich

Precision-Konfiguration

Mit der dritten und letzten Konfiguration soll die Precision ihre maximale Größe erreichen. Hierfür müssen die Filterregeln aus der Recall- und der kombinierten Konfiguration um zusätzliche Regeln ergänzt werden. Erinnern wir uns zurück an Tabelle 6.8, so blieben CPEs mit einer „NA“-Version und solche die vorherige Versionen mit einschließen, aus Rücksicht auf einen hohen Recall, ungefiltert, obwohl sie eine Fehlerquelle waren. Folglich wurden die fehlenden Filter hinzugefügt. Genauer werden die CPE ignoriert, die zwar vorherige Versionen einschließen, aber eine größere Hauptversionsnummer haben als die zu untersuchende Bibliothek.

Mit diesen Maßnahmen konnte die Precision für die CPE - und somit auch für die CVE - hundert Prozent erreichen. Der Recall fällt dafür auf einen Tiefstwert. Für CPEs beträgt er nur noch 58%. Bei den tatsächlichen Sicherheitslücken werden noch 66% gefunden. Betrachtet man den Wert für F1, zeigen sich bei CPEs gute 73%, nur knapp schlechter als die kombinierte Konfiguration. Im Hinblick auf die CPEs jedoch liegt hier ein deutlicher Bestwert mit 79%.

Betrachtet man das Klassifizierungsergebnis aus dem Package-Explorer in Tabelle 6.11, dann spiegelt sich zum einen die hundertprozentige Precision in null False-Positives und der verringerte Recall in drei False-Negatives, namentlich:

- javax.mail 1.4
- Apache Commons HttpClient 3.1

	Unsichere JAR	Sichere JAR	Summe
Unsicher eingeschätzt	14	0	14
Sicher eingeschätzt	3	21	24
Summe	17	21	38

Tabelle 6.11: Konfusionsmatrix für Precision-Konfiguration

- Jackson Databind 2.9.1

Die Mail-Bibliothek wurde schon durch den „ANY“-Filter aus der kombinierten Konfiguration ausgeschlossen. Neu ist der HttpClient, dessen CPE mit Versionen kleiner gleich 4.2.2 nicht mehr dem strengeren Filter entspricht, weil sich die Hauptversionsnummern nicht mehr gleichen. Ebenso wurde die Jackson Databind-Bibliothek ausgeschlossen, deren CPE die Version „NA“ enthält. Durch die drei False-Negatives fallen sowohl Recall als auch Negative-Predictive-Value auf den niedrigsten Wert aller Konfigurationen, erreichen aber noch gute 82% (Recall) respektive 88% (NPV). Dafür erreicht die Konfiguration hunderprozentige Precision und True-Negative-rate und ebenfalls Bestwerte für die Gesamtperformance in F1 mit 90% und Accuracy mit 92%.

6.1.3 Fazit

Jede Konfiguration hat ihre eigene Vorzüge. Die Recall-Konfiguration findet fast alle Sicherheitslücken, büßt aber in der Precision ein. Für die Precision-Konfiguration gilt das Gleiche umgekehrt. Dazwischen ist es gelungen, eine kombinierte Konfiugration zu finden, die den besten F1 Wert für die gefundenen CPEs liefert. Der Benutzer hat die Möglichkeit, nach eigenen Interessen eine passende Konfiguration zu wählen. Von den objektiven Zahlen her schneidet die Precision-Konfiguration am Besten ab. Als Standard ist jedoch die Recall-Konfiguration gewählt, um möglichst alle Sicherheitslücken zu finden. Zwar werden hierbei auch einige False-Positives angezeigt, diese sind aber so offensichtlich, dass sie der Benutzer innerhalb kürzester Zeit aussortieren kann. So wird dann mit wenig Arbeit eine ebenfalls hundertprozentige Precision erreicht. Je nach Anzahl der gefundenen Sicherheitslücken kann aber auch eine strengere Konfiguration gewählt werden, um sich auf die wahrscheinlicheren Sicherheitslücken zu konzentrieren.

6.2 Zeitliche Performance

In diesem Abschnitt wird gemessen, wie lange im Durchschnitt die erstmalige Einrichtung, das Updaten der Datenbank und die Suche für eine Bibliothek dauert. Die Messungen wurden auf einem Desktop PC mit den Spezifikationen aus Tabelle 6.12 durchgeführt.

Betriebssystem	Windows 10 64-Bit
Prozessor	Intel Core i5-3450
Arbeitsspeicher	8 GB
Festplatte	SAMSUNG SSD 830

Tabelle 6.12: Testumgebung

Setup

Für die erstmalige Einrichtung des Plugins müssen alle CVE-Listen als XML Datei heruntergeladen und in die Datenbank importiert werden. Jeder Datafeed muss zweimal heruntergeladen werden: in der alten 1.2.1 Version und in der aktuellen Version 2.0. Insgesamt sind dies 55,81 MB die als ZIP komprimiert sind. Im Folgenden wird die benötigte Zeit gemessen. Um Schwankungen festzustellen und auszugleichen, wurde die Messung dreimal wiederholt und Mittelwerte gebildet.

Die komplette Einrichtung inklusive Download dauerte im Schnitt 262,64 Sekunden bei einer 50 Mbit Internetverbindung. Da dieser Wert jedoch stark abhängig von den NVD-Servern und dem Internetanschluss des Benutzers ist, wird zusätzlich gemessen, wie lange der Import der Daten in die Datenbank benötigt. Der Tabelle 6.13 können aus der Spalte Setup die einzelnen Zeiten entnommen werden. Die starken Schwankungen pro Data-Feed erklären sich durch die unterschiedliche Dateigrößen. So ist der Data-Feed für 2011 zum Beispiel 111.184 KB groß, während der Feed für 2007 nur eine Größe von 25.294 KB hat. Im Durchschnitt dauert der Import während des Setups 126,64 Sekunden um insgesamt 556.471 KB Daten von 95.552 CVEs zu importieren.

Update

Der Updateprozess der Datenbank ist abhängig von der Anzahl der zu aktualisierenden CVEs. Bei bis zu 10 zu aktualisierender CVEs werden pro CVE ca 0,8 Sekunden benötigt. Sollten mehr als 10 CVEs aktualisiert werden, wird zunächst der veraltete Feed aus der Datenbank gelöscht und der aktuelle eingefügt. Die Zeiten für diesen Vorgang können aus der Spalte Update-Zeit von Tabelle 6.13 entnommen werden. Sollten alle Feeds aktualisiert werden müssen, werden hierfür 163,66 Sekunden benötigt.

Suche

Die benötigte Zeit für das Suchen von Sicherheitslücken ist abhängig von der Anzahl möglicher CPE-Produktnamen, die aus den Metadaten gewonnen wurden, der Anzahl gefundener CPEs und schließlich die Anzahl der referenzierten Sicherheitslücken. Um zu messen, wie viel Zeit die Suche benötigt, wurden die 38 Testbibliotheken aufgeteilt in 25 Bibliotheken mit Sicherheitslücken, egal ob True-Positive oder False-Positive und 13 ohne Treffer.

Data-Feed	Setup-Zeit	Update-Zeit
2002	2,7293486663	6,5287906543
2003	0,7522529693	1,7103470203
2004	1,6622132023	2,827146263
2005	2,8103062577	4,052647501
2006	3,8064258783	5,2515990503
2007	3,5809461693	4,9730743453
2008	5,0796498587	6,7724235827
2009	6,567241683	8,59559286
2010	11,700451747	15,2520989547
2011	38,166562324	49,295397662
2012	11,205608021	13,4662710403
2013	11,2007294613	13,3762934823
2014	9,137334664	10,7508762157
2015	6,2434145173	7,2615984563
2016	7,16468581	8,1481546953
2017	4,8309914987	5,3949336367
Gesamt	126,6381627283	163,6572454203

Tabelle 6.13: Zeiten für Setup und Update (kompletter Feed) in Sekunden

Da sich die durchschnittlichen Zeiten für Median und arithmetischem Mittel aufgrund von Einzelfällen mit längerer Suchzeit unterscheiden, werden die Ergebnisse als Boxplot dargestellt. Der Kasten (Box) der Abbildung enthält die mittleren 50% der gemessenen Zeiten. Der Anfang des Kastens stellt dabei das 25% Quartil und das Ende das 75% Quartil dar. Der Median wird als senkrechter Strich in der Box eingezeichnet. Die Zäune links und rechts der Box, auch Whisker genannt, stellen das Minimum und Maximum dar. Werte außerhalb der Zäune werden als Ausreißer interpretiert. Um Minimum und Maximum zu bestimmen, muss zunächst der Interquartilabstand, der Abstand zwischen dem 25%-Quartil und dem 75%-Quartil berechnet werden. Das Minimum ist dann der kleinste Wert, der nicht mehr als das 1,5-fache des Interquartilabstandes vom 25%-Quartil entfernt ist. Analog ist das Maximum der größte Wert innerahlb des 1,5-fachen Interquartilabstandes vom 75%-Quartil. Weitere Informationen können im Buch Statistik Grundlagen - Methoden - Beispiele von Eckey, Kosfeld und Dreger [4] auf Seite 66 nachgelesen werden.

Zunächst werden die Suchzeiten für Bibliotheken mit Treffer aus Abbildung 6.1 betrachtet. Das Minimum liegt hier bei 0,6 Sekunden, das 1. Quartil bei 1,06 Sekunden, der Median bei 1,79 Sekunden, das 3. Quartil bei 3,53 Sekunden und das Maximum bei 6,05 Sekunden. Auffallend sind die vier Ausreißer von den insgesamt 25 gemessenen Zeiten, die den Durchschnitt auf 4,28 Sekunden erhöht haben.

- javax.mail 1.4 (7,6 Sekunden)

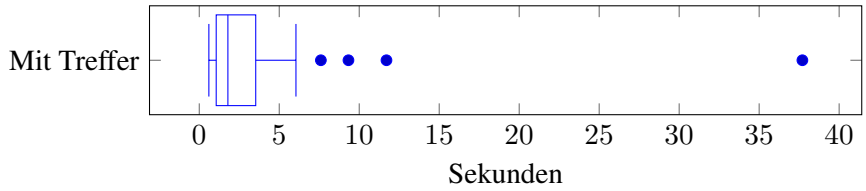

Abbildung 6.1: Suchzeiten mit Treffer

- Spring Security Core 4.2.0 (9,33 Sekudnen)
- Scala Library 2.13.0-M2 (11,7 Sekunden)
- Jetty 4.2.27 (37,7 Sekunden)

Der längste Teil der Suche ist die Datenbankabfrage nach möglichen CPEs. Die obigen Bibliotheken haben gemein, dass viele Produktnamen aus den Metadaten gewonnen wurden. Je mehr Einträge für Produktnamen vorhanden sind, desto länger benötigt die Datenbankabfrage, weil mehr Vergleiche durchgeführt werden müssen und desto mehr Ergebnisse werden geliefert. Exemplarisch soll dies am Beispiel der Jetty 4 Bibliothek gezeigt werden. Der Bibliothek wurden aus den Metadaten die möglichen Produktnamen jetty, mort, may, misc., utilities, http, server und jetty/4.2 entnommen. Mit acht Produktnamen ist das die größte Anzahl unter den Testbibliotheken. Für diese Produktnamen wurden mit 21.729 gefundenen CPEs überdurchschnittlich viele von der Datenbank geliefert. Das deutet darauf hin, dass Namen wie http und server häufiger Bestandteil des CPE-Produktes sind. Die CPE-Abfrage benötigte dabei 22,75 Sekunden, den Großteil der insgesamt 37,7 Sekunden. Von den 21.729 CPEs blieben nach einem Filterprozess noch 17 CPEs übrig, was 14,86 Sekunden benötigte. Die Abfrage der von den 17 CPEs referenzierten Sicherheitslücken war nach 0,09 Sekunden beendet.

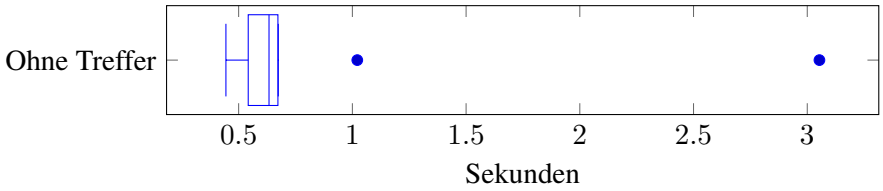

Abbildung 6.2: Suchzeiten ohne Treffer

Für Bibliotheken ohne gefundene Sicherheitslücken werden geringere Suchzeiten erwartet, was in Abbildung 6.2 bestätigt wird. Das Minimum liegt bei 0,44 Sekunden, das 1. Quartil bei 0,54 Sekunden, der Median bei 0,63 Sekunden und das 3. Quartil sowie das Maximum bei 0,67 Sekunden. Auch hier sind zwei Ausreißer mit 1,02 und 3,05 Sekunden enthalten. Dies hat jedoch nur geringen Einfluss auf den Durchschnitt der mit 0,81 Sekunden nahe am Median liegt. Die Gründe für die

Ausreißer sind erneut eine größere Anzahl von extrahierten Produktnamen aus den Metadaten.

Alles in allem wird das Ziel einer schnellen Warnung vor Sicherheitslücken erreicht. In der Regel geschieht dies innerhalb weniger Sekunden. Über alle Bibliotheken hinweg liegt der Median bei 1,09 und im Durchschnitt bei 3,1 Sekunden. Das ist schnell genug, um den Benutzer zu warnen, bevor er mit der Implementierung beginnt.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Der Schwachstellenprüfer für Java-Bibliotheken wurde entwickelt, um den Entwickler vor der Verwendung von unsicheren Bibliotheken in Software-Projekten zu warnen. Dies ist ein großes Problem und wird von der OWASP in der Top 10 der häufigsten Sicherheitsrisiken für Webanwendungen gelistet. Durch die Realisierung als Eclipse-Plugin wird die automatisierte Schwachstellensuche fest in den Entwicklungsprozess integriert. Der Entwickler kann so zu einem frühen Zeitpunkt vor Schwachstellen gewarnt werden und hat noch Zeit, sich nach Alternativen für die Bibliothek umzusehen.

Der Schwachstellenprüfer findet selbständig Java-Bibliotheken aus geöffneten Projekten im Eclipse-Workspace. Aus dem Dateinamen und Attributen des JAR-Manifest (Implementation-Title, -Vendor und -Version) werden mögliche Hersteller, Produktnamen und Versionen ermittelt. Diese Informationen werden verwendet, um eine CPE-Referenzierung zu der Bibliothek zu finden. CPEs sind eine standardisierte und maschinenlesbare Beschreibung von Produktnamen. Sie werden von der NVD zur Referenzierung von Sicherheitslücken auf die betroffene Software verwendet. Über die CPEs können die Sicherheitslücken der NVD gefunden werden. Die NVD ist eine der größten und wichtigsten Datenbanken für öffentliche Schwachstellen. Der Schwachstellenprüfer betreibt eine lokale Kopie der Datenbank, die bei bestehender Internetverbindung aktualisiert wird.

Ein weiteres Hauptfeature des Schwachstellenprüfers ist es, externe SQLite-Datenbanken in die Schwachstellensuche zu integrieren. Damit wird es möglich, auch bisher unveröffentlichte Schwachstellen für eine Bibliothek zu finden, wenn eine entsprechende Datenbank vorliegt. Das kann zum Beispiel eine unternehmensinterne Datenbank sein, die Schwachstellen aus gemeldeten Bugreports, enthält. Für die Integration ist ein Mapping von den Metadaten auf das unbekannte Datenbankschema notwendig. Dies wird von dem Entwickler durchgeführt, in dem er über eine GUI die Spalten einer Datenbanktabelle den Metadaten zuordnet. Hierfür sind keine SQL-Kenntnisse erforderlich.

Den Schwachstellenprüfer zeichnet seine konfigurierbare Suche aus. Je nach Einstellung konnte bei 38 Testbibliotheken ein mittlerer Recall / Precision von 96,27% / 47,57% (High-Recall) oder eine mittlerer Recall / Precision von 57,74% / 100% (High-Precision) erreicht werden. Auf die unsicheren Bibliotheken wird der Entwickler durch eine Markierung der Bibliotheken in den Farben gelb, orange oder rot, je nach Schweregrad des Sicherheitsrisikos, unmittelbar gewarnt. Zusätzlich ist eine filterbare Zusammenfassung der Sicherheitslücken verfügbar. Diese wurde so gestaltet, dass der Entwickler falsche Ergebnisse schnell erkennen und durch wenige Mausklicks dauerhaft auf eine Blacklist für die Bibliothek setzen kann. Ebenso ist es möglich, einzelne Sicherheitslücken zu sperren, wenn der Entwickler das Risiko eingehen möchte.

7.2 Ausblick

Die Precision der gefundenen CPEs ist abhängig von den verfügbaren Metadaten, insbesondere von den vom Benutzer stammenden Listen, was kein akzeptierter Wert für Vendor, Product und Version ist. Die Nutzer des Schwachstellenprüfers, würden davon profitieren, wenn sie ihre gesammelten Metadaten untereinander teilen könnten. Dies könnte zum Beispiel über einen zentralen Webservice realisiert werden, an dem Nutzer ihre Metadaten schicken und im Gegenzug die gesammelten Daten des Services verwenden können. Über den Hash einer Bibliothek, könnten die relevanten Metadaten gefunden werden. In einem nächsten Schritt könnte der Webservice, um CPEs ergänzt werden, die der Schwachstellenprüfer gefunden und Nutzer bestätigt haben. Damit würden die Suchergebnisse präziser werden, je länger die Schwachstelle veröffentlicht wurde. Auch eine Markierung von akzeptierten CPEs in der Schwachstellenzusammenfassung wäre denkbar.

Der Schwachstellenprüfer könnte für weitere Programmiersprachen erweitert werden. Hierfür wäre eine zusätzliche Komponente notwendig, die Bibliotheken identifiziert und von dieser Metadaten extrahiert. Die Schwachstellensuche würde dann auf Basis dieser Metadaten durchgeführt.

Der Schwachstellenprüfer könnte um zusätzliche Heuristiken erweitert werden, die auf ein höheres Risiko für Schwachstellen hinweisen. Fehler in einer Bibliothek werden zum Beispiel oft erst in der nächsten Softwareversion beseitigt. Wenn es eine neue Version zu einer verwendeten Bibliothek gibt, könnte der Entwickler darüber informiert werden. Die Information, ob eine neue Version verfügbar ist, könnte von Maven stammen, wenn die Bibliothek aus dem Maven-Repository stammt. Ein weiterer interessanter Aspekt wären Informationen, wie aktiv an dem Projekt gearbeitet wird. Ein altes Projekt ohne neuere Releases, oder Gitprojekte mit älterem letzten Commit können darauf hindeuten, dass die Software nicht mehr weiterentwickelt und damit keine Fehler behoben werden.

Literaturverzeichnis

[1] BSI: *Botnetze*. https://www.bsi-fuer-buerger.de/BSIFB/DE/Risiken/BotNetze/botnetze_node.html, . – [Online; abgerufen am 11.10.2017]

[2] BSI: *Die Lage der IT-Sicherheit in Deutschland 2016*. https://www.bsi.bund.de/DE/Publikationen/Lageberichte/lageberichte_node.html, 2016. – [Stand Oktober 2016]

[3] CHEIKES, Brant ; WALTERMIRE, David ; SCARFONE, Karen: *Common Platform Enumeration: Naming Specification Version 2.3*. https://csrc.nist.gov/publications/detail/nistir/7695/final, August 2011. – [Abgerufen am 15.10.2017]

[4] ECKEY, Hans-Friedrich ; KOSFELD, Reinhold ; DREGER, Christian: *Statistik, Grundlagen - Methoden - Beispiele*. Gabler Verlag, Wiesbaden, 2000

[5] EDGAR, Nick ; HAALAND, Kevin ; LI, Jin ; PETER, Kimberley: *Eclipse User Interface Guidelines Version 2.1*. https://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html, Februar 2004. – [Online; abgerufen am 20.10.2017]

[6] FREI, Stefan ; SCHATZMANN, Dominik ; PLATTNER, Bernhard ; TRAMMELL, Brian: *Modelling the Security Ecosystem - The Dynamics of (In)Security*. 2009. – [The Eighth Workshop on the Economics of Information Security (WEIS 2009)]

[7] HOMAEI, H. ; SHAHRIARI, H. R.: Seven Years of Software Vulnerabilities: The Ebb and Flow. In: *IEEE Security Privacy* 15 (2017), Jan, Nr. 1, S. 58–65. http://dx.doi.org/10.1109/MSP.2017.15. – DOI 10.1109/MSP.2017.15. – ISSN 1540–7993

[8] HÖLZING, Jörg A.: *Die Kano-Theorie der Kundenzufriedenheitsmessung*. 1. Auflage. Gabler, 2008

[9] MANNING, Christopher D. ; RAGHAVAN, Prabhakar ; SCHÜTZE, Hinrich: *Introduction to Information Retrieval*. Cambridge University Press., 2008

[10] MELL, Peter ; SCARFONE, Karen ; ROMANOSKY, Sasha: *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. https://www.first.org/cvss/cvss-v2-guide.pdf, Juni 2007. – [Abgerufen am 14.10.2017]

[11] MITRE: *About CPE*. http://cpe.mitre.org/about/, . – [Online; abgerufen am 15.10.2017]

[12] MITRE: *Common Vulnerabilities and Exposures*. https://cve.mitre.org/about/, . – [Online; abgerufen am 12.10.2017]

[13] MITRE: *Key Details Phrasing*. https://cveproject.github.io/docs/content/key-details-phrasing.pdf, . – [Abgerufen am 12.10.2017]

[14] MITRE: *Terminology*. https://cve.mitre.org/about/terminology.html, . – [Online; abgerufen am 12.10.2017]

[15] MITRE: *Common Vulnerabilities and Exposures (CVE) Numbering Authority (CNA) Rules*. http://cve.mitre.org/cve/cna/CNA_Rules_v1.1.pdf, September 2016. – [Version 1.1]

[16] NVD: *Common Weakness Enumeration Specification*. https://nvd.nist.gov/vuln/categories, . – [Online; abgerufen am 13.10.2017]

[17] NVD: *General Information*. https://nvd.nist.gov/general, . – [Online; abgerufen am 13.10.2017]

[18] NVD: *NVD Data Feeds*. https://nvd.nist.gov/vuln/data-feeds, . – [Online; abgerufen am 13.10.2017]

[19] NVD: *NVD FAQ Answers*. https://nvd.nist.gov/general/faq, . – [Online; abgerufen am 13.10.2017]

[20] NVD: *Official Common Platform Enumeration (CPE) Dictionary*. https://nvd.nist.gov/products/cpe, . – [Online; abgerufen am 15.10.2017]

[21] NVD: *Vulnerability Metrics*. https://nvd.nist.gov/vuln-metrics/cvss, . – [Online; abgerufen am 13.10.2017]

[22] ORACLE: *Core J2EE Patterns - Data Access Object*. http://www.oracle.com/technetwork/java/dataaccessobject-138824.html, . – [Online; abgerufen am 21.10.2017]

[23] ORACLE: *JAR File Specification*. https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html#JARManifest, . – [Online; abgerufen am 14.10.2017]

[24] ORACLE: *Setting Package Version Information*. https://docs.oracle.com/javase/tutorial/deployment/jar/packageman.html, . – [Online; abgerufen am 14.10.2017]

[25] ORACLE: *Understanding the Default Manifest*. https://docs.oracle.com/javase/tutorial/deployment/jar/defman.html, . – [Online; abgerufen am 14.10.2017]

[26] ORACLE: *Working with Manifest Files: The Basics*. https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html, . – [Online; abgerufen am 14.10.2017]

[27] OWASP: *OWASP Top 10 - 2013*. https://www.owasp.org/images/4/42/OWASP_Top_10_2013_DE_Version_1_0.pdf, 2013. – [Deutsche Übersetzung Version 1.0]

[28] PLATTFORM-INDUSTRIE-4.0: *Was ist Industrie 4.0?* http://www.plattform-i40.de/I40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html, 2017. – [Online; abgerufen am 11.10.2017]

[29] PRESTON-WERNER, Tom: *Semantic Versioning 2.0.0*. http://semver.org/, . – [Online; abgerufen am 14.10.2017]

[30] ROHR, Matthias: *Sicherheit im Software-Entwicklungsprozess*. https://www.informatik-aktuell.de/betrieb/sicherheit/sicherheit-im-software-entwicklungsprozess.html, 2015. – [Online; abgerufen am 10.10.2017]

[31] ULLENBOOM, Christian: *Die Java-APIs für XML*. http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_18_003.htm, . – [Online; abgerufen am 16.10.2017]

[32] ULLENBOOM, Christian: *Serielle Verarbeitung mit StAX*. http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_18_005.htm, . – [Online; abgerufen am 16.10.2017]

[33] YUSUF, Zia ; BHATIA, Akash ; KALRA, Nipun ; HUNKE, Nicolas ; RÜSSMANN, Michael ; SCHMIEG, Florian: *Winning in IoT: It's All About the Business Processes*. https://www.bcgperspectives.com/content/articles/hardware-software-energy-environment-winning-in-iot-all-about-winning-processes/, 2017. – [Online; abgerufen am 11.10.2017]