

Fakultät für Informatik

Studiengang Informatik (M.Sc.)
Software- & Systems-Engineering

Erweiterung der IT-Sicherheit in einem mittelständischen
Unternehmen durch die Einführung von Secrets-Scanning im
Rahmen statischer Codeanalyse

Masterthesis

von

Christian Pritzl

Datum der Abgabe: 30.03.2023

Erstprüfer: Prof. Dr. Gerd Beneken
Zweitprüfer: Prof. Dr. Reiner Hüttl

Eigenständigkeitserklärung / Declaration of Originality

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Raubling, den 30.03.2023


Christian Pritzl

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Themenbereich des Secrets-Scanning aus dem Bereich der statischen Codeanalyse und der Herausforderung wie eine große Anzahl an Testergebnissen klassifiziert und verarbeitet werden kann.

Dabei wird im ersten Teil der Arbeit dargelegt, wie durch eine Risikobeurteilung im Informationssicherheitsmanagementsystem des mittelständischen Unternehmens IU Internationale Hochschule GmbH eine Erweiterung der statischen Codeanalyse um Secrets-Scanning nötig geworden ist. Das Ziel dieser Erweiterung ist die Quellcode-Repositories der IU zu analysieren und von sensiblen Informationen wie Passwörtern, API-Token oder API-Keys zu bereinigen.

Des Weiteren erfolgt eine Einordnung des Themas in den agilen Softwareentwicklungszyklus sowie in die Maßnahmen, welche durch das IT-Sicherheitsteam der IU ergriffen werden, um das Sicherheitsniveau im Unternehmen zu erhöhen.

Da bereits verschiedene Lösungen zur Suche nach sensiblen Informationen in unterschiedlichen Quellen wie Dateien oder Versionsverwaltungssystemen am Markt existieren, erfolgt im zweiten Teil der Arbeit eine Marktanalyse um potentielle Anwendungen zu ermitteln. Hierzu werden zunächst in Expertenrunden Anforderungen ermittelt und kategorisiert. Die drei vielversprechendsten Scanner Credential-Digger, TruffleHog und Gitleaks werden anschließend im Detail vorgestellt und einem praktischen Test unterzogen. Auf Basis dieser Analyse und des Testergebnisses wird abschließend die Auswahl für den Scanner Gitleaks für den weiteren Einsatz an der IU begründet.

Eine Herausforderung bei der manuellen Prüfung der Scan-Ergebnisse stellt die große Menge an zu prüfenden Daten dar. Während des Analysezeitraums werden etwa 500 Quellcode-Repositories an der IU eingesetzt, was zu mehreren tausend zu analysierenden Datensätzen pro Scan führt. Dabei wird nicht nur der aktuelle Stand eines Repositories geprüft, sondern auch der vollständige Verlauf der Entwicklung über alle Verzweigungen hinweg.

Da ein manuelles Prüfen bei der statischen Codeanalyse unabdingbar ist, wird im Rahmen dieser Arbeit eine webbasierte Anwendung entworfen und anschließend prototypisch implementiert, welche den Prüfer bei seiner Arbeit unterstützen soll.

Die ausgewählte Lösung Gitleaks verfügt über eine Möglichkeit, die Suchparameter anzupassen. Um eine Konfiguration zu erstellen, welche einerseits möglichst wenige wahr-positive Treffer unterschlägt und andererseits möglichst wenige falsch-positive Treffer zurückliefert, werden die

Repositories der IU im Rahmen dieser Arbeit mehrfach gescannt.

Der vierte Teil der Arbeit beschäftigt sich mit der Analyse dieser Scan-Läufe und dokumentiert, inwiefern sich die Änderungen an der Konfiguration auf die Scan-Ergebnisse auswirken. Abschließend wird ein detaillierter Einblick in den Aufbau der aktuell eingesetzten Konfiguration gegeben.

Die im Rahmen dieser Arbeit erstellten Auswertungen und Analysen bezüglich der gefundenen Secrets im Quellcode konnten dem IT-Security Team der IU maßgeblich helfen, die Sicherheit in der Entwicklung zu verbessern. Auch der Prototyp für das Review-Programm der Ergebnisse des Secret-Scanners Gitleaks wurde in den täglichen Gebrauch übernommen und wird stetig weiter verbessert.

Schlagworte: IT-Security, Secrets-Scanning, Gitleaks, ISMS, Datenanalyse, Anforderungsanalyse, SAST, SSDLC, Secure Development

Abkürzungsverzeichnis

AGPL Affero-General-Public-License	52
API Advanced-Programming-Interface	25
ASCII American-Standard-Code-for-Information-Interchange	30
AWS Amazon-Web-Services	7
BSI Bundesamt für Sicherheit in der Informationstechnik	1
CI/CD Continous-Integration/Continous-Delivery	5
CIS Center-for-Internet-Security	5
CLI Command-Line-Interface	52
CSS Cascading-Style-Sheets	82
CSV Comma-Separated-Value	42
CVE Common-Vulnerabilities-and-Exposures	8
CVSS Common-Vulnerability-Scoring-System	8
CWE Common-Weakness-Enumeration	11
DLP Data-Loss-Prevention	49
EMEA Europe, Middle-East and Africa	1
IEC Internationale Elektrotechnische Kommission	5
IG Implementierungsgruppe	18
IREB International-Requirements-Engineering-Board	36
ISMS Information-Security-Management-System	4
ISO Internationale Organisation für Normung	5
ISS Information-Security-Services	11
IU International-University	2
JSON JavaScript-Object-Notation	42
JWT JSON-Web-Token	72
MIT Massachusetts-Institute-of-Technology	53
MITRE MITRE Corporation	11

OWASP Open-Web-Application-Security-Project	11
PAM Privilege-Access-Management	6
PII Persönlich identifizierbare Informationen	1
Regex Regular-Expression	28
SaaS Software-as-a-Service	48
SARIF Static-Analysis-Results-Interchange-Format	42
SAST Static-Application-Security-Testing	4
SDLC Software-Development-Lifecycle	4
SHA Secure-Hash-Algorithm	35
SSH Secure-Shell	25
VCS Version-Control-System	14
YAML YAML-Ain't-Markup-Language	42

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Auflistungsverzeichnis	ix
1 Einleitung	1
1.1 IU - Internationale Hochschule	2
1.2 Motivation	2
1.3 Zielsetzung	4
1.4 Aufbau	5
1.5 Ausgewählte Cyberangriffe	6
1.5.1 Uber 2022	6
1.5.2 Twitch 2021	7
1.5.3 Confluence 2022	8
2 Einordnung	11
2.1 Sensible Informationen in Quellcode	11
2.2 ISMS-Analyse	12
2.2.1 Schwachstellenanalyse	14
2.2.2 Risikobeurteilung	15
2.3 Maßnahmen nach CIS	17
2.3.1 CIS-Control 5	19
2.3.2 CIS-Control 16	19
2.4 Agiler Softwareentwicklungszyklus	20
2.4.1 Shift-Left-Ansatz	21
2.4.2 DevSecOps	22
3 Grundlagen	25
3.1 Secrets	25
3.2 Secret-Sprawl	26
3.3 Secrets-Scanner	28
3.4 Statische Codeanalyse	28
3.5 Regulärer Ausdruck	30
3.6 Entropie und Entropiefilter	32

3.7 Klassifizierung von Ergebnissen	33
3.8 Versionsverwaltung am Beispiel von Git	34
3.9 Pre-Commit-Framework	35
3.10 Anforderungspriorisierung	36
4 Anforderungen Secrets-Scanner	39
4.1 Auswahlprozess	39
4.2 Definition von Auswahlkriterien	40
4.3 Priorisierung von Anforderungen	44
5 Evaluation Secrets-Scanner	47
5.1 SonarCloud für Secrets-Scanning	47
5.2 Marktübersicht	48
5.3 Vorstellung evaluierter Secrets-Scanner	49
5.3.1 Credential-Digger	50
5.3.2 TruffleHog	52
5.3.3 Gitleaks	53
5.4 Praktischer Testlauf	56
5.5 Entscheidungsbegründung für Gitleaks	58
6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse	61
6.1 Kernanforderungen	61
6.2 Prozessablauf zur Auswertung	62
6.3 Architektur- und Technikübersicht	63
6.4 Datenmodell	65
6.5 Frontend	66
6.6 API und Backend	72
6.7 Tests	75
6.8 Offene Punkte	75
7 Praktischer Einsatz von Gitleaks	77
7.1 Evolution der Konfiguration	77
7.2 Analyse der IU-Repositories	79
7.3 Vorstellung der aktuellen Konfiguration	80
8 Fazit und Zukunftsausblick	85
8.1 Fazit	85
8.2 Zusammenfassung	86
8.2.1 Einordnung des Secrets-Scannings	86
8.2.2 Anforderungsdefinition und Auswahlprozess	87
8.2.3 Prototyp für manuelles Review	87

8.2.4 Gitleaks Konfiguration	88
8.3 Zukunftsausblick	88
Literatur	93

Abbildungsverzeichnis

1.1	Übersicht potentieller Motivationsgründe von Cyberkriminellen [10]	2
1.2	Entwicklung der Anzahl Studierender an der IU seit 2016 [25]	3
1.3	Über 2022: Admin-Credentials aus Powershell-Skript [27]	7
1.4	Angreifer erbeuten unter Anderem vollständige Quellcode-Repositories von Twitch [1]	8
1.5	Fest einprogrammierte Zugangsdaten für Atlassian Confluence [34]	9
2.1	Risikomanagementprozess nach ISO/IEC 27001 (eigene, vereinfachte Darstellung nach[30])	13
2.2	Risikomatrix wie sie an der IU zum Einsatz kommt, eigene Darstellung auf Basis interner Dokumente	17
2.3	Beispiel für ein Safeguards aus CIS-Control 16[6]	18
2.4	Auszug aus der CIS-Control 5 - Account-Management [6]	20
2.5	Auszug aus der CIS-Control 16 -Application-Software-Security [6]	21
2.6	Shift-Left während des Software-Entwicklungsprozesses[28]	22
2.7	Einordnung von Secrets-Scanning durch GitLab in die CI/CD-Pipeline[18]	23
3.1	Aufteilung von in GitHub gefundenen Secrets auf verschiedene Zugriffskategorien (eigene Darstellung auf Basis von [11])	27
3.2	Übersicht potentieller Ergebnisse eines SAST-Scans, eigene Darstellung	34
3.3	Beispiel für die Ausführung von Gitleaks über einen Pre-Commit-Hook, eigene Darstellung	37
4.1	Überblick Evaluationsprozess zur Auswahl eines Secrets-Scanners, eigene Darstellung	41
6.1	Grafische Darstellung des Prozessablauf für das Ergebnis-Review, eigene Darstellung .	63
6.2	Architekturübersicht der prototypischen Implementierung, eigene Darstellung . . .	64
6.3	Ansicht der Repositories im Frontend des Review-Programms, eigene Darstellung . .	68
6.4	Ansicht der einzelnen Treffer pro Repository im Frontend des Review-Programms, eigene Darstellung	69
6.5	Ansicht des Ergebnisses von Gitleaks zu einem Treffer im Frontend des Review-Programms, eigene Darstellung	69
6.6	Pop-Up zur Klassifikation des Treffers im Frontend des Review-Programms, eigene Darstellung	70
6.7	Assistent zum Import neuer Scan-Ergebnisse im Frontend, eigene Darstellung . . .	71
6.8	Rückmeldung zum Status des Imports neuer Scan-Ergebnisse im Frontend, eigene Darstellung	71

6.9	Auszug aus der Swagger-Dokumentation zu den Treffer-Endpunkten der API, eigene Darstellung	73
6.10	Auszug aus der Swagger-Dokumentation zu weiteren relevanten Endpunkten der API, eigene Darstellung	73
6.11	Auszug aus der Definition der Schemas aus der Swagger-Dokumentation, eigene Darstellung	74
7.1	Grafische Übersicht der Auswertung der Gitleaks-Scans über alle Repositories der IU, eigene Darstellung	80
8.1	Anfrage an ChatGPT zur Identifikation sensibler Informationen im Quellcode, eigene Darstellung	89
8.2	Antwort von ChatGPT zur Identifikation sensibler Informationen in Quellcode, eigene Darstellung	90
8.3	Frage an ChatGPT die identifizierten Treffer zu validieren, eigene Darstellung	90

Tabellenverzeichnis

3.1 Vergleich der Shannon-Entropie ausgewählter Zeichenketten, eigene Berechnungen	32
5.1 Aggregierte Ergebnisse des praktischen Tests der untersuchten Secrets-Scanner	57
5.2 Auswertung der Secrets-Scanner hinsichtlich der Anforderungen aus Kapitel 4.2	58
6.1 Übersicht der Felder des Ergebnis-Datenmodells	67
7.1 Versionsvergleich der Regex-Konfigurationen von Gitleaks	79
7.2 Auswertung der Ergebnisse dreier Scans mit Gitleaks über alle Repositories der IU	80

Auflistungsverzeichnis

3.1 Passwortzuweisung	31
3.2 Passwortzuweisung Regex	31
3.3 Zeichenfolge zur Demonstration eines falsch-positiven Treffers bei der Suche nach Passwörtern	32
5.1 Auszug aus der Standardkonfiguration von Credential-Digger	51
5.2 Auszug aus einer Beispielkonfiguration von Gitleaks	54
5.3 Beispielhafter Fingerprint zur eindeutigen Identifikation eines Treffers von Gitleaks	55
5.4 JSON-Ergebnis eines Gitleaks-Scans in Version 8.15	56
6.1 Datenbankeintrag eines Scan-Treffers, angereichert mit Metadaten (Auszug)	66
7.1 Auszug aus der aktuellen Version 6 der IU-Gitleaks-Konfiguration	82

1 Einleitung

In einer immer stärker digitalisierten und vernetzten Welt nehmen auch die Gefahren, die von Cyberkriminellen ausgehen, stetig zu. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) verzeichnet in seinem Lagebericht zur IT-Sicherheit in Deutschland für das Jahr 2021 144 Millionen neue Schadprogramm-Varianten und einen deutlichen Anstieg von Erpressungsversuchen mittels Ransomware. Nicht nur Nachlässigkeiten bei der IT-Sicherheit sind hierbei ein großes Problem, auch die technische Qualität der Angriffe nimmt immer weiter zu.

Das Jahr 2021 stand dabei insbesondere unter dem Einfluss der COVID-19 Pandemie und den daraus resultierenden Auswirkungen auf die Gesellschaft, welche gezwungen wurde, Digitalisierungsprojekte kurzfristig zu realisieren.¹

Dies sorgte dafür, dass Hochschulen Lehrveranstaltungen online abhalten mussten, wodurch sie verstärkt in den Fokus von Cyberkriminellen gelangten. Dies zeigt sich auch in den Zahlen des US-Telekommunikationsanbieter Verizon, der in seinem „Data-Breach-Investigations-Report“ von 2021 zu dem Ergebnis kommt, dass 54% der Angriffe auf Unternehmen und Institutionen in der Europe, Middle-East and Africa (EMEA)-Region der Kategorie „Basic Web Application Attack“ zugerechnet werden können.²

Verizon nimmt dabei zusätzlich eine Aufschlüsselung nach Industrien vor, sodass für den Bereich „Educational Services“ separate Ergebnisse vorliegen. Dabei wurden insgesamt 1332 Vorfälle in diesem Industriezweig aufgezeichnet, bei welchen in 344 bestätigten Fällen Daten abgeflossen sind.

Als häufigster Einstiegspunkt in die IT-Systeme von Bildungseinrichtungen werden dabei explizit „Credential-Attacks“ genannt.³

Die große Menge an Persönlich identifizierbare Informationen (PII), welche im Bildungssektor verarbeitet werden, machen diesen Bereich zusätzlich attraktiv für Angreifer.

Diese werden dabei insbesondere von finanziellen Motiven geleitet, wie der Abbildung 1.1 aus dem „Data-Breach-Investigations-Report“ entnommen werden kann. Dies deckt sich mit der Zunahme von Angriffen mit Ransomware im Jahr 2021, bei der die Daten des Opfers verschlüsselt werden und anschließend gegen die Zahlung eines Lösegelds wieder entschlüsselt werden.

Prominentes Opfer einer solchen Attacke wurde im November 2022 die Universität Duisburg-Essen

1 [4]

2 [10]

3 [10]

1 Einleitung

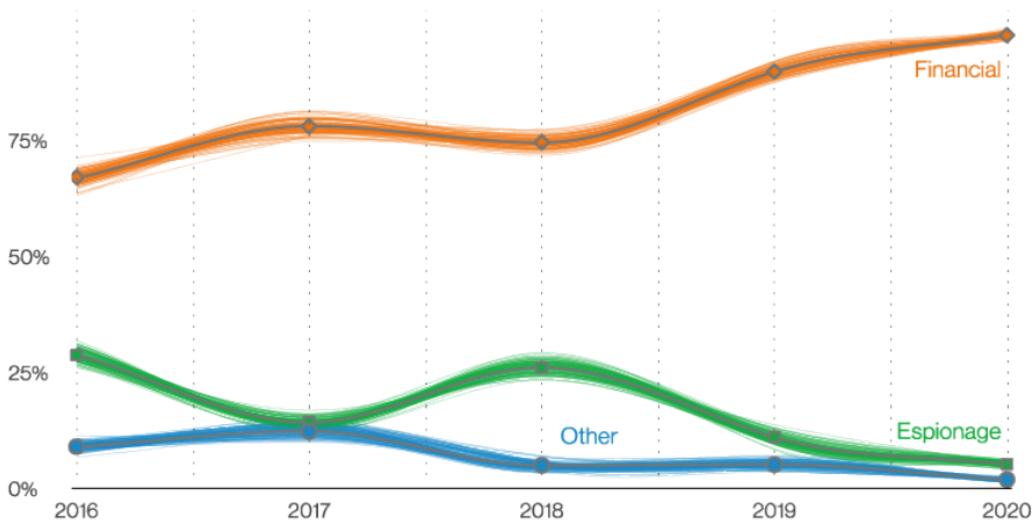


Abbildung 1.1 Zeitliche Übersicht häufigster Motivationsgründe für Cyberkriminelle[10]

mit mehr als 40.000 Studierenden. Nachdem Angreifer in die interne IT der Universität eindringen konnten, wurde anschließend mit der Verschlüsselung der Daten begonnen. Weitere Informationen zum Vorgehen der Angreifer und wie sie in die Systeme eindringen konnten, sind bis heute nicht bekannt geworden.

1.1 IU - Internationale Hochschule

Die International-University (IU) ist eine private staatlich anerkannte Fachhochschule, die 1999 gegründet wurde und ihren Sitz in Erfurt hat. Dank ihrem Fokus auf Onlinelehre und konstantem Wachstum, ist die IU seit 2021 mit über 85.000 Studierenden die größte Hochschule Deutschlands. Im November 2022 wurde dabei erstmals die Marke von 100.000 Studierenden überschritten.⁴ Die Entwicklung der Studierendenanzahl seit 2016 kann der Abbildung 1.2 entnommen werden. Aufgrund des starken Wachstums der IU, insbesondere während COVID-19, und des neuen Fokus von Cyberkriminellen auf den Bildungssektor, ist es Notwendigkeit geworden, die Anstrengungen im Bereich der IT-Sicherheit an die neue Bedrohungslage anzupassen.⁵

1.2 Motivation

Die IU bezeichnet sich selbst als Pionier und Visionär digitaler akademischer Bildung und setzt als Fernuniversität stark auf Digitalisierung und IT. Dies führt bei aktuell über 100.000 Studierenden

⁴ [25]

⁵ [26]

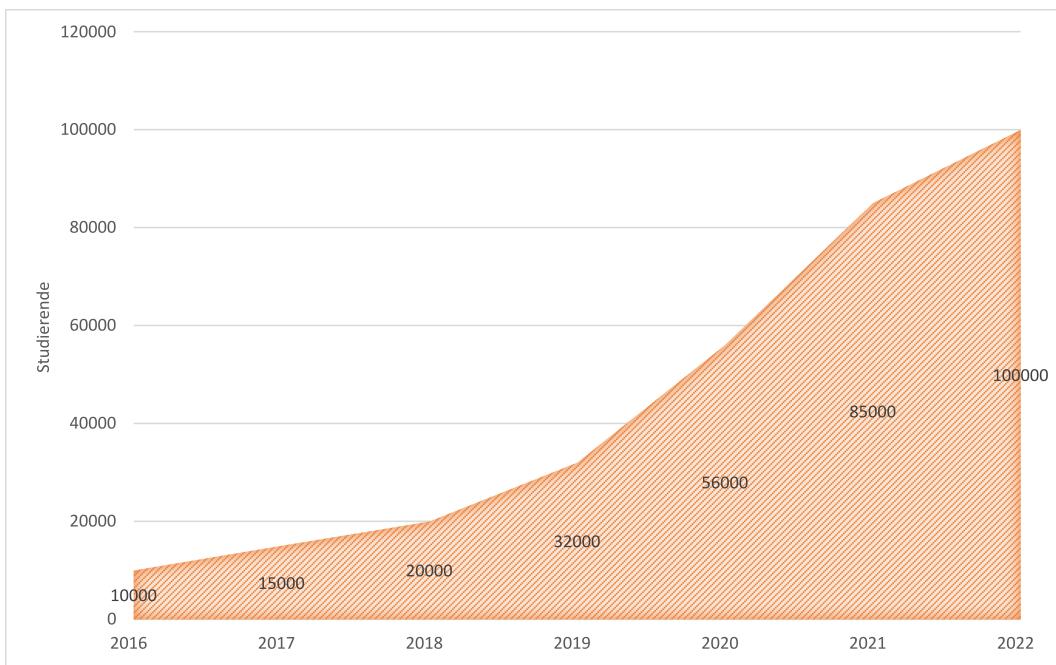


Abbildung 1.2 Entwicklung der Anzahl Studierender an der IU seit 2016 [25]

und mehr als 3000 Mitarbeitenden an 28 Standorten in Deutschland zu einer entsprechend ausgeprägten IT-Landschaft. Dabei setzt die IU ihren Fokus auf Modernität und Fortschritt und versucht ihre Vorreiterrolle bei digitaler Bildung immer weiter auszubauen.

Dieser Anspruch überträgt sich auch auf die interne Softwareentwicklung und IT, welche in den letzten Jahren ebenfalls stark angewachsen ist.⁶

Dies führte zu einer sehr heterogenen Softwarelandschaft, welche unter Anderem intern entwickelte Anwendungen in PHP-Laravel, NodeJS, Python, VueJS sowie verschiedene Android- und iOS-Applikationen umfasst. Hinzu kommen diverse Plattformen von Drittanbietern wie Wordpress, Moodle und Salesforce, die in unterschiedlichem Ausmaß durch die IU angepasst wurden.

Diese Anwendungen verteilen sich auf circa 500 Quellcode-Repositories. Gehostet werden diese aktuell auf der Plattform Atlassian Bitbucket, wobei ein Umzug auf GitLab aktuell in der Umsetzungsphase ist.

Hinzu kommt, dass es Teil der agilen Philosophie der IU ist, den einzelnen Entwicklerteams möglichst große Freiheiten bei ihrer Tätigkeit zu lassen.

Da Cyberkriminelle auch vor dem Bildungssektor keinen Halt machen und die IU eine Vorreiterrolle einnimmt, wird in letzter Zeit verstärkt in Maßnahmen zum Schutz der IT-Landschaft investiert. Dies wird auch durch weitere Zahlen aus dem „Data-Breach-Investigations-Report“ von 2021 unterstrichen, wonach 54% der Angriffe auf Unternehmen und Institutionen in der EMEA-Region

⁶ [26]

1 Einleitung

der Kategorie „Basic Web Application Attack“ zugerechnet werden.⁷ Die IU bietet hier mit ihren Webservices zur Onlinelehre naturgemäß eine große Angriffsfläche.

Darüber hinaus haben das Wachstum der letzten Jahre und der Erschluss neuer Geschäftsgebiete die Etablierung eines Risikomanagementprozesses nötig gemacht, um weitere verschiedene geschäftliche und regulatorische Vorgaben erfüllen zu können.

Der Fokus dieser Arbeit liegt dabei auf dem Themenkomplex „Secrets-Scanning“. Hierbei werden Quellcode-Repositories, beziehungsweise allgemein Textdateien, nach fest einprogrammierten Secrets durchsucht. Dies ist nötig, um die internen Richtlinien und Best-Practices bei der sicheren Entwicklung von Software durchzusetzen, um somit die Sicherheit der selbstentwickelten und -betriebenen IT-Landschaft von Beginn an abzusichern.

Eine erste manuelle Prüfung zeigte, dass bei der Menge an Daten in den Repositories nur eine automatisierte maschinelle Prüfung in Frage kommen kann. Auch belegen die bereits gefundenen Secrets der manuellen Prüfung deutlich, dass hier Handlungsbedarf besteht.

Die Einführung eines solchen Scanners soll dabei im Rahmen des Application-Security-Programms erfolgen. Secrets-Scanning ist Teil des Static-Application-Security-Testing (SAST) und dient folglich der Verbesserung der IT-Sicherheit des Software-Development-Lifecycle (SDLC).

1.3 Zielsetzung

Der Anstoß für diese Arbeit wurde aus einer Risikoanalyse des Information-Security-Management-System (ISMS) heraus gegeben. Die Erfassung des Secrets-Scanning-Prozesses und des damit einhergehenden Review-Prozesses im ISMS sind eine erste Zielsetzung für diese Arbeit.

Um einen Prozess für das Secrets-Scanning etablieren zu können, sollen zunächst verschiedene bereits am Markt existierende Secrets-Scanner anhand eines zu erarbeitenden Kriterienkatalogs evaluiert werden. Hierzu müssen Anforderungen erfasst werden, anhand derer der Auswahlprozess durchgeführt werden kann. Dies muss auch einen praktischen Test der Anwendung vor Einführung umfassen.

Nach Auswahl eines Scanners müssen die Quellcode-Repositories der IU nach sensiblen Informationen durchsucht werden. Hierzu soll eine Analyse der ermittelten Ergebnisse angefertigt werden. Von Interesse ist es auch, inwiefern sich die Ergebnismenge eines Scans über die Zeit entwickelt.

Da bei statischer Codeanalyse ein manueller Reviewprozess erforderlich ist, soll im Rahmen dieser Arbeit ein prototypischer Entwurf eines Programms erarbeitet werden, der diesen Prozess

⁷ [10]

unterstützt. Dieser Entwurf soll im Rahmen eines „Proof-of-Concept“ praktisch umgesetzt werden. Ein Schwerpunkt soll hierbei auf der Eliminierung von falsch-positiven Treffern liegen, da eine weitere Automatisierung nur möglich ist, wenn die Ergebnisse hinreichend genau sind. Hierzu soll das Programm eine Möglichkeit bereitstellen, über die falsch-positive Treffer im Rahmen eines manuellen Reviews als solche klassifiziert werden können. In zukünftigen Durchläufen sollen diese anschließend automatisch gefiltert werden. Um das manuelle Review zu unterstützen, soll das Programm eine entsprechende grafische Oberfläche bieten.

Im Sinne von DevSecOps soll die Codeanalyse so früh wie möglich in den SDLC etabliert werden, um die Kosten für eine Behebung so gering wie möglich zu halten. Aus diesem Grund soll evaluiert werden, inwiefern sich Secrets-Scanner in die entsprechenden Continous-Integration/Continous-Delivery (CI/CD)-Pipelines und über das Pre-Commit-Framework integrieren lassen.

1.4 Aufbau

Im ersten Kapitel erfolgt nach einer kurzen Einführung die Vorstellung der IU als Bildungseinrichtung und mittelständisches Unternehmen. Anschließend wird die Motivation für die Erarbeitung dieser Arbeit dargelegt. Hierzu werden einige Beispiele aufgeführt, welche das Risiko aufzeigen, das von Secrets im Quellcode ausgehen kann. Auch wird gezeigt, dass Bildungseinrichtungen zunehmend in den Fokus von Cyberkriminellen geraten.

In Kapitel 2 erfolgt anschließend eine Einordnung des Themenbereichs aus verschiedenen Blickwinkeln. Zunächst wird betrachtet, wie Secrets im Quellcode durch verschiedene Organisationen als Sicherheitsrisiko eingestuft werden. Anschließend erfolgt eine Einsortierung in den Risikomanagementprozess nach Internationale Organisation für Normung (ISO)/Internationale Elektrotechnische Kommission (IEC), in das Cybersecurity-Framework von Center-for-Internet-Security (CIS) sowie in den agilen Softwareentwicklungszyklus.

Um ein gemeinsames Verständnis für die nachfolgenden Kapitel zu schaffen, werden in Kapitel 3 wichtige Grundlagen erarbeitet und Begriffe definiert.

In Kapitel 4 wird der Auswahlprozess dargelegt, mit dessen Hilfe ein Secrets-Scanner für den Einsatz an der IU ermittelt wurde. In diesem Kapitel werden auch die Anforderungen vorgestellt, die für die Auswahl herangezogen wurden. Ebenfalls Teil dieses Kapitels ist die Priorisierung der ermittelten Anforderungen.

Das Kapitel 5 umfasst die Marktanalyse, um auf Basis der Anforderungen einen passenden Secrets-Scanner zu ermitteln. Hierzu werden die ausgewählten Anwendungen im Detail vorgestellt. Auch ein praktischer Test anhand eines dafür ausgewählten internen Repositories ist Teil dieses

1 Einleitung

Kapitels. Abgeschlossen wird Kapitel 5 mit einer konkreten Entscheidung für einen Secrets-Scanner.

Nach der erfolgten Auswahl wird in Kapitel 6 die Konzeptionierung des Prototypen vorgestellt, welcher für das manuelle Review angedacht ist. Hierzu wird zunächst der theoretische Reviewprozess vorgestellt. Anschließend wird die Architektur- und Technikübersicht vorgestellt. Bevor die Implementierung der Anwendung dargestellt wird, wird das zugrunde liegende Datenmodell detailliert besprochen. Dieses ist von hervorgehobener Wichtigkeit, da darauf der zukünftige Reviewprozess aufbaut. Zum Schluss werden einige bei der Implementierung offen gebliebene Punkte sowie potenzielle Verbesserungsvorschläge angesprochen.

Um die Menge an zu überprüfenden Daten einzuschränken, wurde auch an der Konfiguration der Suchmuster des Secrets-Scanners gearbeitet. Wie sich diese Ergebnisse über den untersuchten Zeitraum veränderten wird in Kapitel 7 vorgestellt.

Die Arbeit schließt in Kapitel 8 mit einem Fazit inwiefern die gesetzten Ziele erreicht werden konnten. Hier wird auch kurz auf eine mögliche zukünftige Weiterentwicklung des Themas Secrets-Scanning eingegangen.

1.5 Ausgewählte Cyberangriffe

In den folgenden Unterkapiteln werden drei aktuelle Beispiele vorgestellt, um verschiedene relevante Aspekte im Kontext von Secrets im Quellcode zu zeigen. Das Beispiel von Uber, Kapitel 1.5.1, zeigt ein prominentes Beispiel für fest einprogrammierte Secrets im Quellcode. Die trügerische Sicherheit, die private Repositories vermitteln können, zeigt das Beispiel von Twitch in Kapitel 1.5.2. Im letzten Beispiel wird anhand einer Schwachstelle in Atlassian Confluence in Kapitel 1.5.3 gezeigt, dass auch das schnelle Reagieren eines Serviceanbieters zu Problemen bei dessen Kunden führen kann.

1.5.1 Uber 2022

Bereits 2014 und 2016 konnten Angreifer Secrets aus Quellcodes, beziehungsweise Skripten, des US-Fahrdienstleisters Uber extrahieren. Im September 2022 bestätigte Uber erneut, dass Angreifer, nachdem sie sich Zugriff auf einen „Domain Admin Account“ verschaffen konnten, sich mittels lateraler Bewegung in der gesamten Unternehmensorganisation ausbreiten konnten.

Nachdem die Angreifer mittels Social-Engineering initialen Zugang zum Uber-Intranet erlangen konnten, fanden sie dort mehrere Powershell-Skripte. Abbildung 1.3 zeigt eine Nachricht der Angreifer, in der sie den Fund bestätigen. In einem dieser Skripte waren die Admin-Credentials für die Privilege-Access-Management (PAM)-Lösung „Thycotic“ von Uber fest hinterlegt. Innerhalb von Thycotic verwaltet Uber nicht nur Endbenutzer-Zugangsdaten, sondern auch Zugriff auf

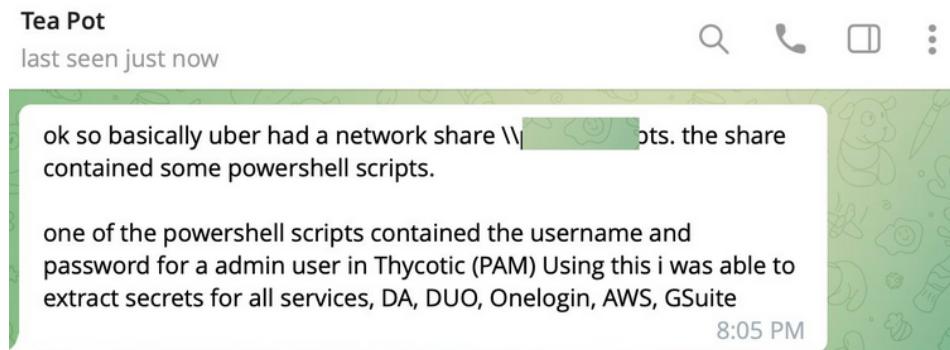


Abbildung 1.3 Uber 2022: Admin-Credentials aus Powershell-Skript [27]

Drittanbieterplattformen, Serviceaccounts und sensible Informationen der DevOps-Plattformen.

Privilegierter Zugriff ist grundsätzlich solcher, der über die Befugnisse eines Standardbenutzers hinaus reicht. Mit Hilfe von PAM lässt sich die Gewährung eines solchen Zugriffs steuern. Entsprechend kritisch sind diese Systeme einzuschätzen.

Die Angreifer waren im Folgenden in der Lage unter Anderem Zugriff auf Ubers Amazon-Web-Services (AWS), die Online-Geschäftsanwendungen des Google Workspace (GSuite) sowie diverse weitere kritische Services wie SentinelOne, Ubers „Endpoint Security Platform“, zu erlangen. Auch verschiedene Geräte von Mitarbeitern konnten übernommen werden.⁸

Da auch Zugriff auf Quellcode-Repositories erlangt wurde, kann nicht ausgeschlossen werden, dass die Angreifer Zugriff auf weitere darin fest einprogrammierte Credentials erlangen konnten. Für Uber selbst steht die kosten- und zeitintensive Prüfung praktisch aller Dienste und Plattformen bevor, um sicher gehen zu können, dass die Angreifer an keiner Stelle Persistenz erlangen konnten.

1.5.2 Twitch 2021

Das nachfolgende Beispiel zeigt die vermeintliche Vertraulichkeit von privaten Quellcode-Repositories.

Im Oktober 2021 konnten Angreifer Zugriff auf umfangreiche Datensätze das Live-Streaming-Videoportals Twitch.tv erlangen. Abbildung 1.4 zeigt einen Screenshot des Postings der Angreifer.

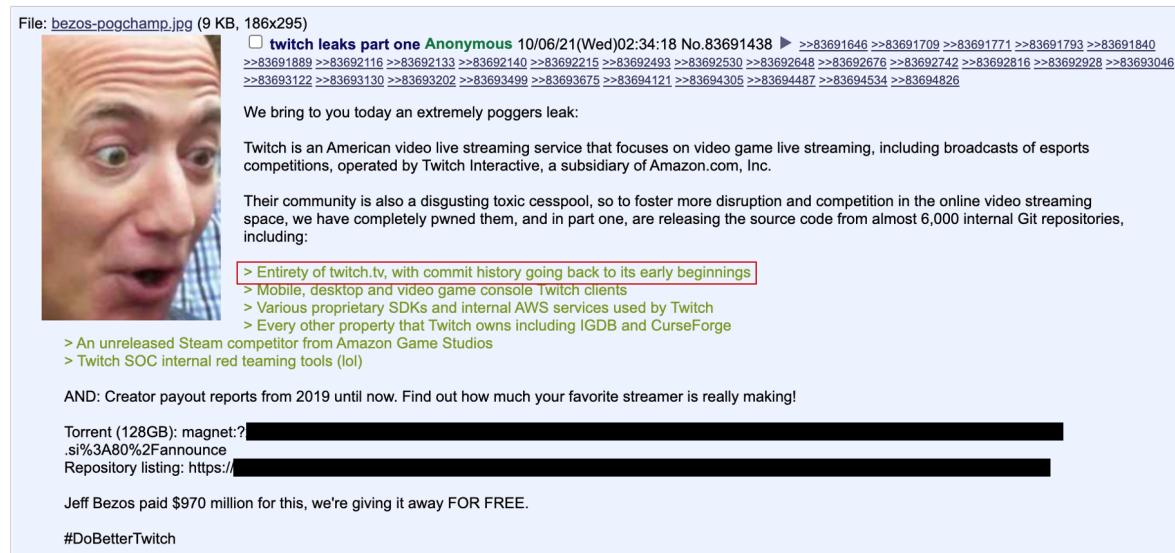
Ursächlich verantwortlich war dabei ein Konfigurationsfehler an einer Serverkonfiguration durch Twitch selbst.⁹ Der erste Teil dieser Daten im Umfang von 125 GB wurde anschließend im Internet frei zum Download angeboten.

Diese Daten umfassten, neben Abrechnungsdaten der Streamer, vor allem auch fast 6000

⁸ [27]

⁹ [20]

1 Einleitung



File: bezos-pogchamp.jpg (9 KB, 186x295)

□ twitch leaks part one **Anonymous** 10/06/21(Wed)02:34:18 No.83691438 ► >>83691646 >>83691709 >>83691771 >>83691793 >>83691840
>>83691889 >>83692116 >>83692133 >>83692140 >>8369215 >>83692493 >>83692530 >>83692648 >>83692676 >>83692742 >>83692816 >>83692928 >>83693046
>>83693122 >>83693130 >>83693202 >>83693499 >>83693675 >>83694121 >>83694305 >>83694487 >>83694534 >>83694826

We bring to you today an extremely poggers leak:

Twitch is an American video live streaming service that focuses on video game live streaming, including broadcasts of esports competitions, operated by Twitch Interactive, a subsidiary of Amazon.com, Inc.

Their community is also a disgusting toxic cesspool, so to foster more disruption and competition in the online video streaming space, we have completely pwned them, and in part one, are releasing the source code from almost 6,000 internal Git repositories, including:

> Entirety of twitch.tv, with commit history going back to its early beginnings
> Mobile, desktop and video game console Twitch clients
> Various proprietary SDKs and internal AWS services used by Twitch
> Every other property that Twitch owns including IGDB and CurseForge

> An unreleased Steam competitor from Amazon Game Studios
> Twitch SOC Internal red teaming tools (lol)

AND: Creator payout reports from 2019 until now. Find out how much your favorite streamer is really making!

Torrent (128GB): magnet:?xt=

.s!%3A80%2fannounce

Repository listing: https://

Jeff Bezos paid \$970 million for this, we're giving it away FOR FREE.

#DoBetterTwitch

Abbildung 1.4 Angreifer erbeuten unter Anderem vollständige Quellcode-Repositories von Twitch[1]

private Quellcode-Repositories von Twitch. Diese umfassen nicht nur den aktuellen Stand der Entwicklung der Mobile-, Desktop und Konsolen-Applikationen, sondern auch die Commit-History der Repositories bis hin zu den Anfängen der Programmierung.¹⁰ Weitere Informationen bezüglich Commit-History können Kapitel 3.8 entnommen werden.

Bei der Menge an veröffentlichten Daten zeigt die Erfahrung, dass es unwahrscheinlich ist, dass Twitch es schafft alle potentiellen Secrets zu finden und anschließend zu invalidieren bevor Angreifer sie zur Authentifizierung nutzen. Dies belegt die Wichtigkeit, den Quellcode präventiv von sensiblen Informationen frei zu halten.

1.5.3 Confluence 2022

Die Schwachstelle in der Confluence-Software des US-Softwareentwicklers Atlassian aus dem Jahr 2022, welcher unter der Common-Vulnerabilities-and-Exposures (CVE)-2022-26138¹¹ erfasst wurde, ist ähnlich wie der Angriff auf Uber ein Beispiel für das Risiko, das mit fest einprogrammierten Zugangsdaten verbunden ist. In diesem Fall kommt erschwerend hinzu, dass viele Kunden von Confluence nicht in der Lage waren, die bereitgestellten Patches zeitnah auf ihren On-Premise-Systemen einzuspielen, sodass Angreifer leichtes Spiel hatten. Aus diesem Grund wurde die Schwere der Schwachstelle mit einem Wert von 9.8 im Common-Vulnerability-Scoring-System (cvss)-3 als „Critical“ angegeben.

Die extrahierten Zugangsdaten umfassten Benutzername, E-Mail Adresse und Passwort eines automatisch generierten Accounts, der für die Migration der Anwendung in die Cloud benötigt

10[31]

11[39]

```
Barricade→ confluence-questions-3.0.2 cat default.properties
build.year=2022
predefined.user.username=disabledsystemuser
predefined.user.fullName=disabledsystemuser
predefined.user.email=dontdeletethisuser@email.com
predefined.user.password=disabled1system1user6708
Barricade→ confluence-questions-3.0.2 []
```

Abbildung 1.5 Fest einprogrammierte Zugangsdaten für Atlassian Confluence [34]

wird. Dabei wurde für alle Instanzen von Confluence der selbe Datensatz generiert. Abbildung 1.5 zeigt einen Screenshot der Daten wie sie in der entsprechenden Datei „*default.properties*“ abgelegt sind.

Diese Zugangsdaten waren Teil der Anwendung „Questions For Confluence“ und ermöglichen Jedem, der sie kannte, sich in einem entsprechenden Confluence-System anzumelden. Der Angreifer erlangte anschließend Zugriff auf die Daten der „confluence-user-group“.¹²

¹²[2]

2 Einordnung

Um den Themenbereich Secrets-Scanning erfassen zu können, erfolgt in diesem Kapitel eine entsprechende Einordnung.

Zunächst wird das Risiko eingeschätzt, das von Secrets im Quellcode ausgeht. Hierzu wird auf die Einschätzung verschiedener relevanter externer Organisationen wie Open-Web-Application-Security-Project (OWASP) und MITRE Corporation (MITRE) zurückgegriffen.

Anschließend erfolgt eine Einordnung des Secrets-Scanning-Prozesses in bestehende Standards zum Risikomanagement auf der einen Seite und Cybersecurity-Frameworks zur Umsetzung konkreter Maßnahmen auf der anderen Seite.

Das Information-Security-Services (ISS)-Team orientiert sich dabei an den Standards ISO/IEC 27001 und ISO/IEC 27005. Für die konkrete Umsetzung erfolgt eine Orientierung nach den Critical-Security-Controls des CIS in der Version 8 vom Mai 2021.

Abschließend erfolgt eine Einordnung von Secrets-Scanning als Teilbereich statischer Codeanalyse in den SDLC, damit auch eine Perspektive aus Sicht der Softwareentwicklung geschaffen wird.

2.1 Sensible Informationen in Quellcode

Unter der Common-Weakness-Enumeration (CWE)-798 beschreibt die US-Non-Profit-Organisation MITRE das Problem von „Hard-coded Credentials“. MITRE unterscheidet dabei die zwei Fälle „Inbound“ und „Outbound“. Bei Inbound wird ein eingehender Datensatz gegen ein fest einprogrammiertes Secret im Quellcode geprüft. Bei Outbound verbindet sich das System über ein fest programmiertes Secret mit einem anderen System.

Nähere Informationen, was unter dem Begriff „Secret“ zusammengefasst werden kann, lässt sich Kapitel 3.1 entnehmen.

In beiden Fällen kann durch Offenlegung des Quellcodes das Secret kompromittiert werden. Eine Änderung des Secrets zur Mitigation kann anschließend nur durch einen Patch der Software durchgeführt werden. Ein Beispiel für dieses Problem in der Praxis liefert der, in Kapitel 1.5.3 beschriebene, Angriff auf Atlassian Confluence in 2021.

Eine weitere Möglichkeit den Quellcode offen zu legen und daraus sensible Informationen zu extrahieren besteht, falls die Software dem Anwender übergeben wird. Dieser kann anschließend

2 Einordnung

mittels Dekompilierung das Secret aus der Binary der Anwendung extrahieren.¹

Die CWE-Top-25 Liste für das Jahr 2022 listet die CWE-798 an Position 15. In dieser Liste werden die 25 gefährlichsten und gängigsten Software-Schwachstellen des Jahres aufgeführt, um verantwortlichen Stellen eine Orientierung zur Priorisierung zu bieten.

Interessanterweise handelt es sich bei den 24 anderen Schwachstellen auf der Liste um dynamische Schwachstellen. Dass heißt, diese lassen sich nur ausnutzen, während die Software ausgeführt wird. Lediglich CWE-798 ist eine statische Schwachstelle, womit sie auch vorhanden und „nutzbar“ ist, wenn das Programm nur im Quellcode vorliegt.²

Die OWASP Foundation klassifiziert die Schwere der Veröffentlichung eines Secrets als „high“. Dies entspricht dem höchsten Schweregrad dieser Klassifizierung. Die Wahrscheinlichkeit eines anschließenden „Exploits“ wird von OWASP mit „very high“ angegeben.³ Dies deckt sich mit dem Beispiel des Confluence-Hacks, welcher in Kapitel 1.5.3 vorgestellt wird.

MITRE liefert über die ATT&CK-Matrix noch eine weitere Einordnung. Bei der ATT&CK-Matrix handelt es sich um eine Wissensdatenbank, in welcher Taktiken und Techniken von Angreifern, basierend auf tatsächlich beobachteten Vorgehensweisen, gesammelt werden. Aus diesen können Angriffsmodelle für die Etablierung einer Verteidigungsstrategie entwickelt werden.⁴

In dieser Matrix wird die Subtechnik „Unsecured Credentials: Credentials in Files“ seit Februar 2020 unter der ID 1552.001 als Bestandteil der Technik „Credentials Access“ geführt. Dabei wird als Vorgehen beschrieben, dass Angreifer lokale Dateisysteme und Netzwerkshares nach Dateien durchsuchen, welche sensible Informationen enthalten. Für ein praktisches Beispiel kann der Angriff auf Uber in 2022, beschrieben in Kapitel 1.5.1, herangezogen werden.

Dies deckt sich mit den Gründen für das Phänomen des „Secret-Sprawls“. Weitere Informationen hierzu können dem Kapitel 3.2 entnommen werden.

Zur Mitigation nennt MITRE unter Anderem Audits zur präventiven Suche nach Dateien, die Secrets beinhalten sowie Zugriffsbeschränkungen auf Dateien und Verzeichnisse.⁵

2.2 ISMS-Analyse

In der Informationssicherheit werden allgemein die drei primären Schutzziele Verfügbarkeit, Integrität und Vertraulichkeit definiert, um die Sicherheit schützenswerter Güter sicherzustellen.⁶ Bei der Anwendung dieser drei Schutzziele auf Informationen, beziehungsweise Daten, kann die

1 [7]

2 [49]

3 [42]

4 [50]

5 [29]

6 [28]

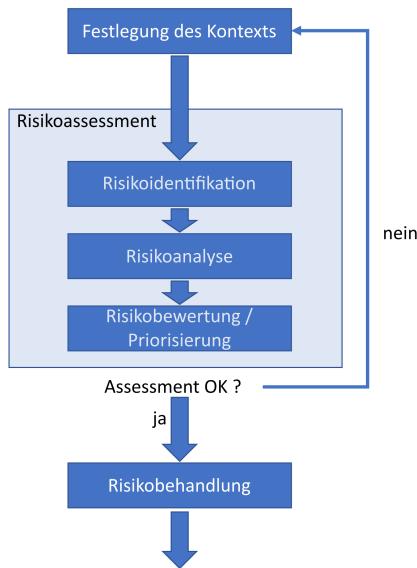


Abbildung 2.1 Risikomanagementprozess nach ISO/IEC 27001 (eigene vereinfachte Darstellung nach[30])

Einführung eines ISMS in einer Organisation helfen.

Um die Anforderungen an ein solches ISMS festzulegen, wurde die Norm ISO/IEC 27001 geschaffen.⁷

Da der Bildungsbereich viele PII verarbeitet und Cyberkriminelle diesen Bereich zunehmend in den Fokus nehmen, wird an der IU intensiv am Ausbau eines ISMS gearbeitet.

Über die Zunahme der Angriffe auf den Bildungsbereich kann Kapitel 1 eingesehen werden.

Während ISO/IEC 27001 beschreibt, wie ein ISMS auszusehen hat, wird in ISO/IEC 27005 genauer auf den Bereich des Risikomanagementprozesses, insbesondere auf Abnahme- und Akzeptanzkriterien, eingegangen. Dabei ist das Risikomanagement ein integraler Bestandteil eines ISMS.⁸

Der Abbildung 2.1 kann eine vereinfachte Darstellung des Risikomanagementprozesses entnommen werden. Die Definition des Kontexts, also die Klärung von Fragen nach dem Anwendungsbereich des Risikomanagementprozesses oder welche Rollen und Verantwortlichkeiten es gibt, ist nicht Teil dieser Arbeit.

Die Schnittstelle des Risikomanagementprozesses zum ISMS beginnt dabei ab dem Prozessschritt der Risikobehandlung.⁹

7 [9]

8 [30]

9 [30]

2 Einordnung

2.2.1 Schwachstellenanalyse

Die Identifikation potentieller Risiken ist der erste Schritt des Risikoassessments. Die im Folgenden beschriebene Schwachstellenanalyse stellt keine vollständige Analyse im Sinne des Risikomanagements dar, sondern soll vielmehr das Vorgehen an der IU exemplarisch an einem Beispiel zeigen.

Quellcode wird an der IU als Teil des Assets „GitLab und Atlassian Bitbucket“ eingeordnet, welches wiederum der Kategorie „Softwarepflege und Repositories“ zugeordnet ist.

Bei Atlassian Bitbucket handelt es sich um die aktuelle Plattform, auf der die Repositories der IU gehostet sind. Während der Erarbeitung dieser Arbeit wurde eine Migration auf die neue Hostingplattform GitLab vorbereitet.

Zusammen mit dem Assetbeauftragten wurde anschließend eine Analyse potentieller Schwachstellen durchgeführt. Hierbei orientiert sich die IU an den Vorgaben aus dem Katalog des BSI-Grundschutzes.

Als Ergebnis dieser Analyse wurde unter Anderem folgende Bedrohung des Assets festgestellt: „*Fehlende Sicherheit in der Entwicklung (Fehlende Sicherheit im Entwicklungs- und Unterstützungsprozess)*“.

Der Assetbeauftragte begründete seine Entscheidung mit dem signifikanten Risiko, welches durch potentiell hoch privilegierten Zugriff auf produktive Firmen- und Kundendaten einhergeht, das von Credentials im Quellcode, beziehungsweise in der History des Version-Control-System (vcs) ausgeht, da diese Credentials so auch nach Jahren noch abrufbar sind.

Das weitere Vorgehen innerhalb der IU umfasste anschließend die Abwägung, ob ein neues Risiko für diese Schwachstelle erfasst werden muss. Falls nämlich ein sogenannter „Quickwin“, also eine umgehende Beseitigung der Schwachstelle möglich wäre, hätte keine Notwendigkeit bestanden, ein neues Risiko anzulegen.

Im Falle sensibler Informationen im Quellcode wäre die umgehende Entfernung und Invalidierung dieser Daten ein solcher Quickwin. Die Erfahrung zeigte aber, dass dies oftmals aus unterschiedlichen Gründen nicht möglich ist. So mussten beispielsweise im Fall von Service-Credentials erstmals die betroffenen Services identifiziert werden, da die Credentials oft zwischen einzelnen Services geteilt wurden. Weiteres kann hierzu Kapitel 3.2 entnommen werden.

Bevor die Zugangsdaten zu produktiven Systemen getauscht werden konnten, mussten zunächst die Systemverantwortlichen identifiziert werden, um den Austausch mit diesen zu koordinieren. Auch ist das Bereinigen der Repositories und deren History oftmals zeitaufwendig, siehe hierzu auch Kapitel 3.8.

Dies zeigt, dass ein Quickwin in diesem Fall nicht möglich ist.

2.2.2 Risikobeurteilung

Ausgehend von der Schwachstellenanalyse aus Kapitel 2.2.1 wird im Allgemeinen eines oder mehrere Risiken aus der Analyse abgeleitet. Jedem Risiko wird anschließend, als erster Schritt der Risikobeurteilung an der IU, ein Risikobeauftragter und ein Risikoverantwortlicher zugeordnet. So stehen konkrete Ansprechpartner für den weiteren Verlauf zur Verfügung. Anschließend werden die Szenarien und Konsequenzen, die mit diesem Risiko einhergehen, aufgezeigt und anhand derer eine Analyse des neuen Risikos hinsichtlich Eintrittswahrscheinlichkeit und Auswirkung vorgenommen.

Konkrete Konsequenzen, die sich aus sensiblen Informationen im Quellcode ergeben können, wurden beispielsweise in Kapitel 1.5 aufgezeigt. Das Spektrum an potentiellen Szenarien und Konsequenzen ist allerdings deutlich größer, sodass eine Erfassung den Rahmen dieser Arbeit überschreiten würde.

In der allgemeinen Definition eines Risikos spricht man statt Auswirkungen von Schadenshöhe, sodass gilt:

$$\text{Risiko} = \text{Eintrittswahrscheinlichkeit} * \text{Schadenshöhe}$$

.¹⁰

Die Eintrittswahrscheinlichkeit für den unbefugten Zugriff Dritter auf Systeme der IU durch das Abgreifen sensibler Informationen im Quellcode wurde auf „Mittel“ geschätzt, da bisher keine konkreten Eintrittsfälle bekannt geworden sind. Mittel bedeutet in diesem Zusammenhang, einmal oder mehr als einmal alle zehn Jahre, jedoch weniger als einmal pro Jahr.

Die weiteren verfügbaren Kategorien für die Eintrittswahrscheinlichkeit sind „gering“, was weniger als einmal in zehn Jahren entspricht und „hoch“, was mindestens einmal pro Jahr bedeutet.

Auf Grundlage der aktuell verfügbaren Informationen lässt sich somit eine Eintrittswahrscheinlichkeit von „Mittel“ gut vertreten.

Die Schadenshöhe, beziehungsweise Auswirkungen, wurden nach Diskussion mit dem zuständigen Assetbeauftragten ebenfalls auf „Mittel“ festgelegt. Die als Mittel definierten Auswirkungen bedeuten, dass das zu erfassende Risiko die drei Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit des betroffenen Assets in „normalem Maße“ bedroht.

Die exakte Begründung für diese Einschätzung kann an dieser Stelle aus Gründen der Geheimhaltung nicht wiedergegeben werden.

Für die vollständige Berechnung des Risikos muss an der IU zusätzlich abgewogen werden, ob das Risiko sowohl unmittelbare Auswirkungen auf das operative Geschäft hat als auch einen Reputationsschaden für die IU nach sich ziehen kann.

Der Abbildung 2.2 kann die Risikomatrix entnommen werden, wie sie an der IU zum Einsatz

10[30]

2 Einordnung

kommt. Dabei lässt sich erkennen, dass die Kombination aus mittlerer Eintrittswahrscheinlichkeit und mittlerer Auswirkung normalerweise zu einem moderaten Risiko der Stufe 2 führt.

Da die Optionen für unmittelbare Auswirkungen auf das operative Geschäft sowie den Reputationsschaden beide als zutreffend erkannt wurden, ergibt sich nach Abschluss der Risikobeurteilung insgesamt aber eine Verschiebung des Risikos in den Bereich eines „signifikanten Risikos“.

Unmittelbare Auswirkungen auf das operative Geschäft könnten beispielsweise dadurch entstehen, dass unbefugte Dritte Zugriff auf Zugangsdaten für administrative Anwendungen erlangen, die für den Betrieb der internen IT der IU notwendig sind. Falls es hierdurch zu Ausfällen von Diensten und Anwendungen kommt, würde sich dies folglich auch negativ auf die Reputation des Unternehmens auswirken.

Grundsätzlich macht ein signifikantes Risiko nicht zwingend eine sofortige Maßnahme an der IU erforderlich. Mögliche Maßnahmen umfassen in diesem Fall die Akzeptanz des Risikos, eine vorübergehende Akzeptanz sowie die Behandlung des Risikos.

Dies liegt darin begründet, da Risiken der Kategorien 4 und vor allem 5 eine sofortige Behandlung nach sich ziehen, sodass ein Risiko der Stufe 3 gegebenenfalls zunächst vorübergehend akzeptiert werden muss.

Eine vollständige Akzeptanz eines signifikanten Risikos, welches Auswirkungen auf das operative Geschäft sowie die Reputation des Unternehmens haben kann, muss aber generell kritisch gesehen werden.

Eine weitere Möglichkeit im Umgang mit Risiken stellt eine Übertragung des Risikos, beispielsweise über eine Versicherung, dar. Dies wurde im Rahmen dieser Arbeit für den vorliegenden Fall allerdings nicht weiter untersucht.

Risiken der Stufe 3 bieten des Weiteren die Möglichkeit einer wirtschaftlichen Abwägung bei welcher die Kosten für eine Abmilderung, beziehungsweise Behebung, dem erwartbaren Schaden gegenübergestellt werden.

Da die wirtschaftlichen Kosten für eine Abmilderung des Risikos in diesem Fall mutmaßlich geringer ausfallen als der erwartbare Schaden, wurde entschieden, eine Maßnahme zur Risikobehandlung zu formulieren.

Diese Maßnahme stellt eine Grundlage für die Erstellung der vorliegenden Arbeit dar.

Um nach Durchführung der Maßnahme eine erneute Risikobeurteilung durchzuführen, wurde entschieden, die Maßnahme nach Abschluss dieser Arbeit auf Wiedervorlage zu legen.

	Eintrittswahrscheinlichkeit	Geringe bis Mittlere Auswirkung	Hohe Auswirkung	Kritische Auswirkung
Gering	Minderes Risiko (Stufe 1)	Moderates Risiko (Stufe 2)	Signifikantes Risiko (Stufe 3)	
Mittel	Moderates Risiko (Stufe 2)	Signifikantes Risiko (Stufe 3)	Erhebliches Risiko (Stufe 4)	
Hoch	Signifikantes Risiko (Stufe 3)	Erhebliches Risiko (Stufe 4)	Kritisches Risiko (Stufe 5)	

Abbildung 2.2 Risikomatrix wie sie an der IU zum Einsatz kommt, eigene Darstellung auf Basis interner Dokumente

2.3 Maßnahmen nach CIS

Da die Beschreibungen der Maßnahmen im ISMS, beziehungsweise des Risikomanagementprozesses nach den Normen ISO/IEC 27001 und 27005 allgemein formuliert sind, eignen sie sich nur bedingt, um daraus konkrete Maßnahmen zur Verbesserung der IT-Sicherheit innerhalb einer Organisation abzuleiten. Aus diesem Grund wird innerhalb der IU auch auf die Critical-Security-Controls nach CIS zurückgegriffen.

Diese Maßnahmen werden von CIS in Expertenrunden erarbeitet, um Empfehlungen auszusprechen, welche Schritte nötig sind, um ein Unternehmen gegen Cyberangriffe zu wappnen. Da diese Expertenrunden Vertreter aus den unterschiedlichen Bereichen der IT-Sicherheit, wie Angriff und Verteidigung (Red und Blue Teams), Analysten und Entwicklern sowie Endnutzern zusammenbringen, ergeben sich unterschiedliche Perspektiven und somit ein ganzheitlicher Lösungsansatz.

CIS legt großen Wert darauf, dass diese Vertreter aus unterschiedlichen Branchen wie Beratung, Wissenschaft, Wirtschaft und Regierungsbehörden kommen, um den branchenübergreifenden Ansatz der CIS-Controls zu unterstreichen.¹¹

Die Liste der Controls umfasst in 18 Kapiteln konkrete Maßnahmen, die implementiert werden können, um die IT-Sicherheit in einer Organisation zu verbessern. Dabei wird versucht, die einzelnen Maßnahmen so zu gestalten, dass eine Übertragung auf die Anforderungen aus ISO/IEC möglich ist.

Jede Maßnahme ist hierzu in weitere „Safeguards“ unterteilt. Jedes Safeguard entspricht dabei einer konkret umzusetzenden Tätigkeit, wie beispielhaft in Abbildung 2.3 anhand des Safeguard

¹¹[6]

2 Einordnung

NUMBER	TITLE/DESCRIPTION	ASSET TYPE	SECURITY FUNCTION	IG1	IG2	IG3
16.1	Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.	Applications	Protect		●	●

Abbildung 2.3 Beispiel für ein Safeguards aus CIS-Control 16[6]

16.1 gezeigt wird.

In diesem Safeguard wird die Einführung und Aufrechterhaltung eines Prozesses für sichere Anwendungsentwicklung empfohlen. Dabei werden in der Beschreibung Beispiele genannt, welche Punkte abgedeckt werden müssen, um die Maßnahme zu erfüllen. Diese wären im konkreten Fall beispielsweise die Einführung von Secure-Coding-Guidelines, Training für Entwickler sowie Schwachstellenmanagement. Der Abbildung lässt sich des Weiteren entnehmen, dass Safeguard 16.1 Teil des Asset-Types „Applications“ ist und die Security-Function „Protect“ erfüllt. Zugeordnet ist das Safeguard der Implementierungsgruppe (IG)-2 und der IG-3.

Zur Unterstützung einer schrittweisen Umsetzung sind die Safeguards in drei IG unterteilt. Diese Gruppen orientieren sich an Unternehmensgrößen und der Expertise der Mitarbeiter.

Die IG-1 richtet sich dabei an kleine bis mittlere Unternehmen mit dem Ziel den Geschäftsbetrieb im Ernstfall aufrecht erhalten zu können. Das Ziel dieser Maßnahmen ist eine grundsätzliche „Basic Cyber Hygiene“ und stellt die Mindestaktionen dar, die ein Unternehmen zum Selbstschutz ergreifen sollte.

Die IG-2 und IG-3 bauen anschließend jeweils auf der vorhergehenden Gruppe auf.

Abbildung 2.1 lässt sich über die farblichen Markierungen entnehmen, dass Safeguard 16.1 beispielsweise Teil der IG-2 und der IG-3 ist.

Der „Data-Breach-Investigations-Report“ von Verizon aus dem Jahr 2021 nimmt ebenfalls Bezug auf die CIS-Controls. Für den Bildungsbereich werden aus der IG-1 die Controls „Security Awareness and Skills Training (14)“, „Access-Control-Management (6)“ und „Security Configuration of Enterprise Assets and Software“ zur Umsetzung empfohlen.¹²

IG-2 richtet sich an Unternehmen, die bereits dediziertes Personal für die Verwaltung und den Schutz ihrer IT-Infrastruktur angestellt haben. Solche Unternehmen weisen komplexere Strukturen, beispielsweise mehrere Standorte mit unterschiedlichen Risikoprofilen, auf. Unternehmen in dieser Kategorie verarbeiten oftmals auch sensible Kundendaten. Öffentliches Vertrauen und Reputation sind für diese Unternehmen ein wichtiges Anliegen.

¹²[10]

Unternehmen, an die sich IG-3 richtet, beschäftigen Spezialisten für dedizierte Bereiche wie Risikomanagement, Penetrationstests und Application-Security. Assets, die durch Maßnahmen aus IG-3 geschützt werden sollen, beinhalten sensible Daten, die regulatorischen Maßnahmen unterliegen. Im Falle eines erfolgreichen Angriffs können sich Auswirkungen auf das Gemeinwohl ergeben. Die hier beschriebenen Maßnahmen richten sich folglich gegen fortgeschrittene und gezielte Angriffe.

Die IU beschäftigt dediziert Personal für den Bereich der IT-Sicherheit und des Datenschutzes und verarbeitet als Hochschule personenbezogenen Daten von Studierenden, sodass sich grundsätzlich die Umsetzung von Maßnahmen bis einschließlich IG-3 empfiehlt.

In den folgenden Unterkapiteln werden die CIS-Control 5 und CIS-Control 16 detaillierter vorgestellt, da sich aus diesen beiden die Notwendigkeit für Secrets-Scanning ableiten lässt.

2.3.1 CIS-Control 5

Diese Maßnahme umfasst sechs Safeguards, welche Teil der IG-1 und IG-2 sind und beschreibt dabei das mögliche Vorgehen zur Verwaltung von Accounts.¹³

Ein Überblick über die Maßnahme kann Abbildung 2.4 entnommen werden.

Hierzu sollen Prozesse und Werkzeuge eingeführt werden, welche die Zuweisung und Autorisierung von Zugangsdaten für Benutzer- und Serviceaccounts sowie Administratoraccounts für Software und sonstige Assets regeln. Secrets sind dabei als eigene Assets zu betrachten, welche, wie alle Gegenstände von Wert für die Organisation, inventarisiert und nachverfolgt werden müssen. Die IU hat hierzu, gemäß Control 5.1 und 5.5, unternehmensweit einen Passwortmanager eingeführt. Wie in Kapitel 3.2 genauer beschrieben, kann es allerdings passieren, dass Secrets über verschiedene Kanäle so verteilen werden, dass keine Nachvollziehbarkeit mehr möglich ist. Eines der Beispiele, die CIS in der Beschreibung des Controls 5 liefert, handelt dabei passenderweise von Credentials von Serviceaccounts, die fest in Quellcode einprogrammiert wurden.

Daraus lässt sich die Notwendigkeit einer weiteren Lösung ableiten, die ein gezieltes Monitoring der Quellcode-Repositories auf fest einprogrammierte Secrets ermöglicht.

2.3.2 CIS-Control 16

Da die IU eine ganze Reihe an Anwendungen selbst entwickelt und betreibt, ist die Maßnahme 16 „Application Software Security“ von besonderer Bedeutung. Ziel dieser Maßnahme ist es, Schwachstellen in Software präventiv zu finden und gegebenenfalls zu beheben. Ein Auszug aus dieser Maßnahme kann der Abbildung 2.3 entnommen werden.

Alle 14 Safeguards sind Teil der IG-2 und IG-3 und gehen damit über den Anspruch an grundsätzliche

¹³[6]



Overview

Use processes and tools to assign and manage authorization to credentials for user accounts, including administrator accounts, as well as service accounts, to enterprise assets and software.

Abbildung 2.4 Auszug aus der CIS-Control 5 - Account-Management [6]

IT-Sicherheit hinaus.

Zur Einführung eines Application-Security-Programms greift das ISS-Team auf die CIS-Control 16 als Leitfaden zur Einführung an der IU zurück. Ein Application-Security-Programm ist dabei als ein formaler Überbau für alle IT-Sicherheitsbezogenen Einzelaktivitäten des SDLC zu verstehen und umfasst grundsätzlich noch weitere Punkte als in CIS-Control 16 beschrieben werden.

Zur Sicherstellung von Codequalität und dem Auffinden potentieller Schwachstellen im Quellcode schlägt CIS in Safeguard 16.12 den Einsatz von statischer und dynamischer Codeanalyse vor. Diese beiden Formen der Analyse sind bereits Teil der IG-3.¹⁴

Für dynamische Codeanalyse, also dem Testen des Codes während der Ausführung zur Laufzeit, gibt es aktuell keine einheitliche Lösung an der IU. Für statische Codeanalyse wird teamübergreifend auf SonarCloud gesetzt. Da Secrets im Quellcode durch statische Codeanalyse gefunden werden können, wurde analysiert, inwiefern SonarCloud eine Lösung sein kann. Die Analyse und die Begründung, warum SonarCloud nicht geeignet ist, kann Kapitel 5.1 entnommen werden. So bleibt die Notwendigkeit, eine Lösung zu ermitteln, um Quellcode frei von sensiblen Informationen zu halten.

2.4 Agiler Softwareentwicklungszyklus

In diesem Kapitel wird untersucht wie sich Secrets-Scanning in den SDLC integrieren lässt. Hierzu werden die Begriffe „shift-left“ und „DevSecOps“ eingeführt sowie analysiert inwiefern die Einführung eines Secret-Scanning-Prozesses die damit verbundenen Prinzipien unterstützen kann.

¹⁴[6]

CONTROL 16 Application Software Security



Overview

Manage the security life cycle of in-house developed, hosted, or acquired software to prevent, detect, and remediate security weaknesses before they can impact the enterprise.

Abbildung 2.5 Auszug aus der CIS-Control 16 -Application-Software-Security [6]

2.4.1 Shift-Left-Ansatz

Der SDLC beschreibt in fünf Schritten das Vorgehen beim Entwickeln von Software. Die einzelnen Schritte können Abbildung 2.6 entnommen werden.

Die Schritte und ihre Reihenfolge sind dabei unabhängig vom jeweiligen Vorgehensmodell gültig. So beginnt der Prozess mit der Erhebung von Anforderungen (Requirements) an das Produkt. Anschließend erfolgt die Designphase, in der die Architektur der Anwendung festgelegt wird. Das Ergebnis dieses Prozessschrittes wird anschließend in Code umgesetzt und getestet. Abschließend kann die Anwendung veröffentlicht werden.

Bei traditionellen Vorgehensmodellen, wie dem Wasserfallmodell, wird das Produkt erst nach der Entwicklung, mit Beginn der Testphase, verschiedenen Sicherheitstests unterzogen. Auch während der Releasephase wird oftmals noch versucht, Sicherheitsmaßnahmen umzusetzen. Da die Software zu diesem Zeitpunkt bereits sehr weit in ihrer Entwicklung fortgeschritten ist, sind die Kosten, welche für die Behebung von Schwachstellen anfallen, deutlich höher als wenn bereits während des Entwicklungsschrittes auf potentielle Schwachstellen geprüft worden wäre. Noch kostenintensiver wird es, falls grundsätzliche Architekturentscheidungen abgeändert werden müssen, weil beispielsweise Logikfehler im Design in Schwachstellen resultieren.

Aus diesem Grund wird angestrebt, Sicherheit früher in den SDLC zu integrieren. Dieser Ansatz wird auch als „shift left“ bezeichnet. Woher diese Bezeichnung kommt, lässt sich Abbildung 2.6 entnehmen. Die Maßnahmen zur Umsetzung von Sicherheit in der Anwendung werden, ausgehend von der Release- und Testingphase, in die links davor angesiedelten Prozessschritte bewegt. Das Ziel ist es dabei, die passenden Sicherheitsprüfungen in jedem Prozessschritt zu etablieren. Konkret bezogen auf Secrets-Scanning bedeutet das, dass mit der Entwicklung der ersten Zeilen an Quellcode auch ein Scan-Prozess gestartet werden muss.

2 Einordnung

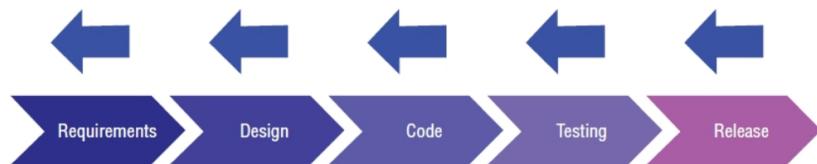


Abbildung 2.6 Shift-Left während des Software-Entwicklungsprozesses[28]

2.4.2 DevSecOps

Die zunehmende Umstellung auf agile Prozesse und Vorgehensweisen bei der Softwareentwicklung führt dazu, dass die Grenzen zwischen Entwicklung und IT-Betrieb zunehmend verwischen. Klassischerweise ist der IT-Betrieb nach dem Release, siehe Abbildung 2.6, im SDLC angesiedelt und kümmert sich um die Bereitstellung und Verwaltung von Ressourcen, um die Anwendung am laufen zu halten.

Unter dem Begriff „DevOps“ hat sich in den letzten Jahren eine neue Kultur der Kooperation zwischen Entwicklung und IT-Betrieb entwickelt. Diese ist nötig geworden, da der IT-Betrieb nicht mehr als eigener sequentieller Schritt betrachtet werden kann, da Software im Rahmen agiler Vorgehensmodelle schneller und häufiger freigegeben wird.

Im Laufe der letzten Jahre hat sich ebenfalls gezeigt, dass auch der Bereich der IT-Sicherheit nicht mehr losgelöst betrachtet werden kann. Sicherheit wird im SDLC nach links geschoben, siehe Kapitel 2.4.1, und verteilt sich somit im optimalen Fall als integraler Teil über jeden Prozessschritt. Mit der frühen Integration von Sicherheitstests verschwimmt folglich auch die Grenze zum IT-Sicherheitsbereich, sodass zunehmend von „DevSecOps“ gesprochen werden kann. Dieser Begriff beschreibt damit die neue Kooperation zwischen DevOps und IT-Sicherheit.¹⁵

Microsoft hat für DevSecOps acht praktische Erfahrungspunkte erarbeitet, welche Unternehmen bei der Umsetzung helfen sollen.

Dabei wird unter dem Punkt „Keep Secrets Safe“ empfohlen, keine sensiblen Informationen im Quellcode zu hinterlegen, da dort kein Schutz gegen Angreifer besteht. Da Sicherheit zudem so früh wie möglich integriert werden soll, empfiehlt es sich laut Microsoft, bereits beim Commit auf dem Rechner des Entwicklers anzusetzen.¹⁶

Im Hinblick auf den SDLC wäre somit der frühest mögliche Prozessschritt für Secrets-Scanning der Schritt „Code“ aus Abbildung 2.6.

Der selbe Ansatz wird analog auch von GitLab empfohlen. Da GitLab als Quellcode-Hoster erst über die CI/CD-Pipeline eingreifen kann, lässt sich ein statischer Test in GitLab automatisch mit

¹⁵[28]

¹⁶[36]

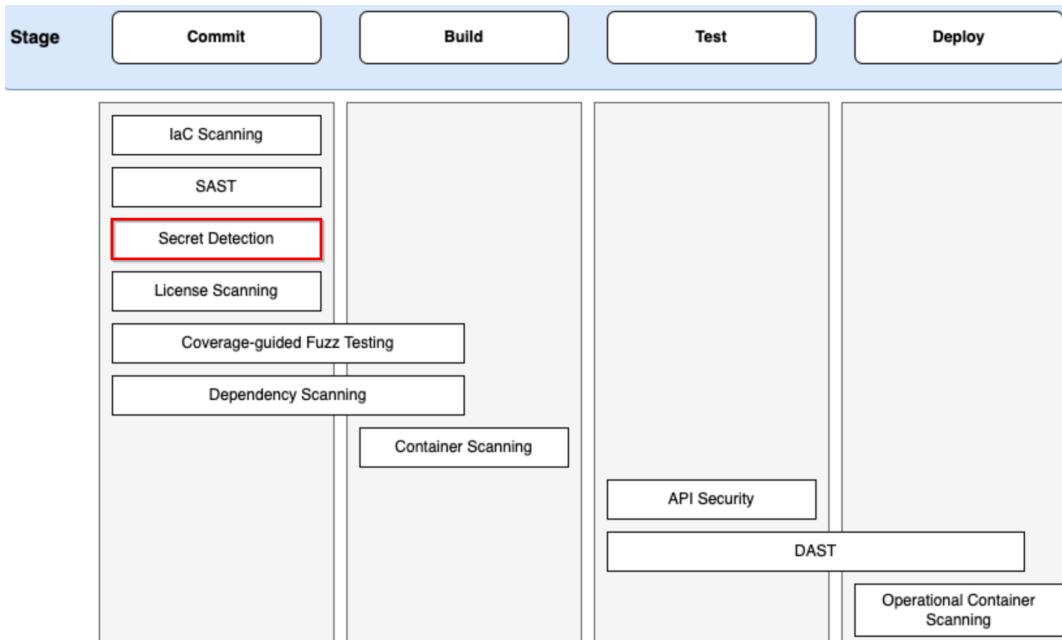


Abbildung 2.7 Einordnung von Secrets-Scanning durch GitLab in die CI/CD-Pipeline[18]

jedem Commit über die Pipeline anstoßen.¹⁷

Die CI/CD-Pipeline läuft nach einem Commit mit dem Ziel, die Codeänderungen zu integrieren, beziehungsweise zu „deployen“. Dies entspricht den Schritten „Testing“ und „Release“ des SDLC aus Abbildung 2.6.

Abbildung 2.7 kann die Einordnung von Secrets-Scanning als „Secret Detection“ von Seiten GitLabs innerhalb einer CI/CD-Pipeline entnommen werden. Interessanterweise fasst GitLab das Secret-Scanning nicht allgemein unter dem Punkt SAST zusammen, sondern nennt es explizit als eigenen Schritt.

Zum Zeitpunkt der Erstellung dieser Arbeit wird der Ansatz über die CI/CD-Pipeline auch als teamübergreifende Initiative der Entwickler an der IU evaluiert. Der Vorteil gegenüber dem Ansatz von Microsoft liegt für die IU darin, dass es nicht nötig ist, das Secrets-Scanning lokal auszuführen. Wie in Kapitel 1.2 dargelegt, ist es Teil der agilen Philosophie der IU den Entwicklern große Freiheit in ihrer Arbeitsweise zu lassen. Somit kann keine Garantie gegeben werden, dass ein Entwickler auch wirklich immer nach Secrets scannt bevor ein Commit ausgeführt wird.

¹⁷[18]

3 Grundlagen

Im folgenden Kapitel werden für das weitere Verständnis wichtige Begriffe und Vorgehensweisen eingeführt und definiert.

Ziel ist es, eine gemeinsame Grundlage an Verständnis für die weitere Erarbeitung des Themas Secrets-Scanning zu schaffen.

3.1 Secrets

Unter „Secrets“ lassen sich allgemein schützenswerte sensible Informationen wie zum Beispiel Advanced-Programming-Interface (API)-Token und API-Keys, Access-Keys, Passwörter, private kryptographische Schlüssel oder Ähnliches verstehen, welche aus Sicherheitsgründen geheim gehalten werden müssen.

Mit Hilfe eines Secrets ist es möglich, sich digital zu Authentifizieren, um Zugriff auf beispielsweise einen Service, ein System oder allgemein Daten zu erlangen.

Ein Secret kann dabei Teil eines Credential-Sets sein. Bei der Kombination von Benutzername und Passwort handelt es sich um ein solches Credential-Set. Hierbei entspricht das Passwort dem Secret. Bei Secure-Shell (SSH)-Schlüsseln wäre der private Schlüssel das Secret und die Kombination aus privatem und öffentlichem Schlüssel das Credential-Set.^{1 2 3}

Umgangssprachlich ist hier oftmals auch nur von „Credentials“ oder „Secrets“ die Rede.

Mithilfe eines Secrets sollten sich immer nur berechtigte Entitäten, also eine bestimmte Person oder ein bestimmter Service, authentifizieren können. Folglich gilt, wer sich mit einem validen Secret authentifizieren kann, muss als ein legitimer Benutzer gelten. Wenn nun allerdings, zum Beispiel aufgrund von Secret-Sprawl, nicht mehr nachvollziehbar ist, wer alles im Besitz eines Secrets ist, führt dies zu einem Vertrauensverlust innerhalb der Organisation.

1 [11]

2 [35]

3 [32]

3 Grundlagen

3.2 Secret-Sprawl

Sowohl in der modernen als auch in der klassischen Softwareentwicklung sind Secrets ein essentieller Bestandteil, um einzelne Komponenten einer Anwendung zu verbinden. Daraus ergibt sich das Problem, dass diese eigentlich besonders schützenswerten Informationen sowohl von Entwicklern als auch von Services und der Infrastruktur regelmäßig genutzt werden.

Das Phänomen, dass sich schützenswerte Informationen über eine Organisation hinweg ausbreiten, wird als Secret-Sprawl bezeichnet.⁴

In einer Webanwendung interagieren Frontend und Backend heutzutage oftmals über APIs, welche beispielsweise API-Keys zur Zugriffsteuerung verwenden. Diese API-Keys werden bereits während der Entwicklung benötigt und können so unvorsichtigerweise fest einprogrammiert im Quellcode der Anwendung landen oder werden in Konfigurationsdateien unverschlüsselt abgelegt.

Im Cloud- und Containerumfeld können Secrets als Aufrufparameter an Funktionen oder in „Connection-Strings“ für beispielsweise Datenbankverbindungen fest einprogrammiert in entsprechenden Dateien hinterlegt werden. Von dort aus werden sie in Backups übernommen oder in Log-Dateien geschrieben und verbreiten sich so über verschiedene Anwendungen und Abteilungen hinweg im Unternehmen.

MITRE liefert in der Beschreibung der Subtechnik T1552.001 „Unsecured Credentials: Credentials in Files“⁵ einige Beispiele, in welchem Kontext Dateien, welche sensible Informationen enthalten können, erstellt werden. Den klassischen Fall stellt dabei die private Passwortliste eines Benutzers dar, welche eventuell auch innerhalb eines Teams geteilt wird. Auch das direkte Teilen von Secrets in Dokumentationen oder Chatprogrammen ist ein häufiges Problem.

Auch VCS, wie zum Beispiel Git, die jegliche Änderung am Quellcode dokumentieren und über eine History nachvollziehbar machen, tragen zur Verbreitung von Secrets bei. Weiteres kann hierzu Kapitel 3.8 entnommen werden.

Der Jahresbericht 2021 des französischen IT-Sicherheitsunternehmens GitGuardian zum Thema Secret-Sprawl zeigt, dass das Unternehmen über 6 Millionen fest einprogrammierte Secrets in den mehr als 45 Millionen öffentlichen Repositories von GitHub gefunden hat. Dies entspricht einer Verdopplung der Werte von 2020. Auf DockerHub wurden in den mehr als 10.000 geprüften Images über 1.200 Secrets gefunden.

Die Auswertung zeigt, dass in privaten Repositories Secrets mit einer vierfach höheren Wahrscheinlichkeit enthalten sind als in Öffentlichen. Vermutlich vermitteln diese den Entwicklern ein falsches Gefühl von Sicherheit. Genauere Zahlen veröffentlichte GitGuardian bisher allerdings

⁴ [11]

⁵ [50]

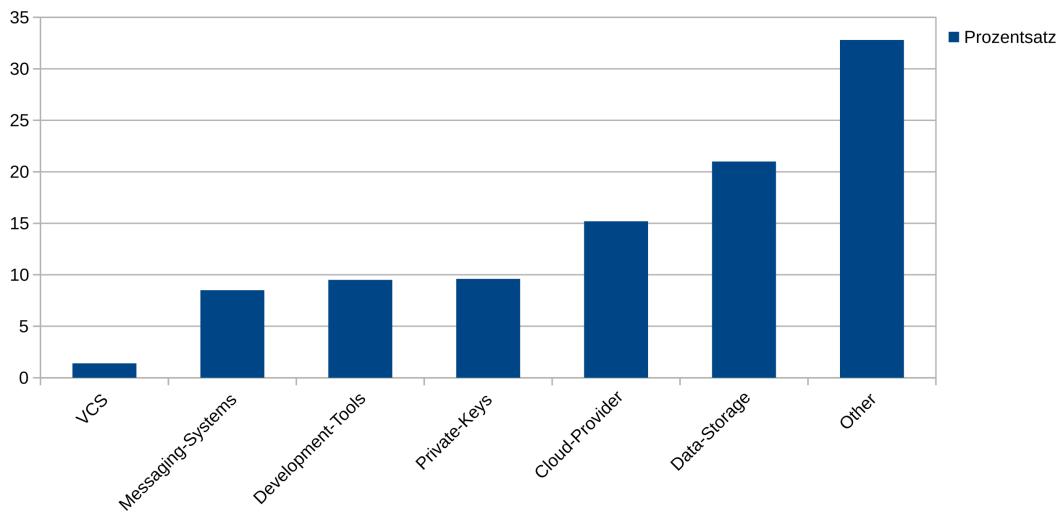


Abbildung 3.1 Aufteilung von in GitHub gefundenen Secrets auf verschiedene Zugriffskategorien (eigene Darstellung auf Basis von [11])

nicht.⁶

Ein Beispiel wie sich in GitHub gefundene Secrets über verschiedene Kategorien hinweg verteilen, kann Abbildung 3.1 entnommen werden. Dabei zeigt sich, dass etwa 20% der gefundenen Secrets Zugriff auf Datenspeicher ermöglichen sowie 15% Zugriff auf Cloud-Provider und bei immerhin 10% der gefundenen Secrets handelt es sich um private Schlüssel.

Aus dem Secret-Sprawl ergeben sich nun eine Reihe von Problemen für die IT-Sicherheit. Zunächst fehlt es an Sichtbarkeit, da nicht bekannt ist, ob und in welchen Systemen Secrets abgelegt wurden. Auch kann nicht mehr nachvollzogen werden, welche Secrets auf welchem Weg verteilt wurden. Repositories, Wikis und Chatprogramme dienen dem Teilen und Verbreiten von Informationen und bieten folglich keine oder nur eingeschränkte Möglichkeiten die Ausbreitung von Informationen über beispielsweise Audit-Logs nachzuvollziehen.

Eine Studie aus dem Jahr 2019 kam bei einer Untersuchung von mehr als 200.000 gefundenen validen Secrets in öffentlichen Repositories auf GitHub zu dem Ergebnis, dass 89% davon Zugriff auf sensible Daten oder Services ermöglichten.⁷

Folglich verwundert es kaum, dass Unternehmen regelmäßig Opfer ihrer mangelhaften oder nicht vorhandenen Quellcodeanalyse werden. Das Kapitel 1.5 zeigt hierzu einige aktuelle Beispiele.

⁶ [11]

⁷ [35]

3 Grundlagen

3.3 Secrets-Scanner

Bei Secrets-Scannern handelt es sich um Anwendungen aus dem Bereich der statischen Codeanalyse, die dazu dienen Textdateien nach sensiblen Informationen zu durchsuchen. Bei diesen Textdateien handelt es sich in der Praxis oftmals um den Quellcode von Anwendungen.

Die Mehrzahl der aktuell verfügbaren Scanner basiert dabei auf Regular-Expression (Regex) welche genutzt werden, um Muster in Text zu identifizieren. Ein Vorteil dieser Methode ist die Unabhängigkeit von einer spezifischen Programm- oder Auszeichnungssprache. Selbst Dokumentationen können nach sensiblen Informationen durchsucht werden. Mehr zu regulären Ausdrücken, insbesondere deren Vor- und Nachteile, kann Kapitel 3.5 entnommen werden.

Eine kurze Übersicht aktueller am Markt verfügbarer Scanner wird in Kapitel 5.2 gegeben. Dabei zeigt sich, dass neben den erwähnten Scannern auf Basis von regulären Ausdrücken auch zunehmend weitere Anwendungen zum Einsatz kommen, die mithilfe von Methoden der künstlichen Intelligenz arbeiten. Diese versprechen ein deutlich besseres Verhältnis von falsch-positiven zu wahr-positiven Ergebnissen. Weitere Informationen zur Klassifizierung von Ergebnissen kann Kapitel 3.7 entnommen werden.

Secrets-Scanner lassen sich dabei sowohl lokal, also auf einer Entwicklungsmaschine als auch in einer CI/CD-Pipeline einsetzen. Der Vorteil des lokalen Einsatzes liegt darin, dass bereits im Ansatz, also beim Commit in ein VCS-Repository, verhindert werden kann, dass sensible Informationen die Maschine verlassen.

Hierfür kann auf das Pre-Commit-Framework zurückgegriffen werden, welches in Kapitel 3.9 am Beispiel des Secrets-Scanners Gitleaks genauer vorgestellt wird.

Falls dies, wie an der IU, flächendeckend nicht möglich ist, beziehungsweise als ergänzende Maßnahme neben dem lokalen Einsatz, kann ein Secrets-Scanner auch in die CI/CD-Pipeline integriert werden, sodass der Scan automatisch angestoßen wird. Hier ist somit auch eine zentrale Verwaltung der eingesetzten Scan-Konfiguration, beziehungsweise eine zentrale Auswertung der Ergebnisse möglich.

Abschließend lässt sich festhalten, dass Secrets-Scanner als essentieller Bestandteil einer aktuellen SAST-Suite im Kontext von DevSecOps betrachtet werden muss.

3.4 Statische Codeanalyse

Im SDLC, wie er der Abbildung 2.6 entnommen werden kann, wird im Schritt „Testing“ unter Anderem auf die Werte Vertraulichkeit, Integrität und Verfügbarkeit der Anwendung geprüft. Die Tests lassen sich dabei in die zwei großen Kategorien statischer und dynamischer Tests einteilen.

Der Unterschied liegt darin, dass bei dynamischen Tests die Anwendung, also der Quellcode, ausgeführt und zur Laufzeit getestet wird, bei statischen Tests wird der Quellcode im Klartext Zeile für Zeile geprüft.

Statische Tests werden unter dem Begriff SAST zusammengefasst und umfassen neben automatisierten Tests auch manuelle Reviews des Quellcodes. Ein manuelles Review auf Secrets wäre zwar grundsätzlich möglich und würde deutlich weniger falsch-positive Treffer produzieren als ein Regex-basierter Test, ist aber aufgrund der schieren Menge an Quellcode in der Regel nicht praktikabel. Aus diesem Grund greift man für SAST häufig auf automatisierbare Lösungen zurück.

Eine Auswahl gängiger SAST-Lösungen wird beispielsweise von OWASP bereitgestellt.⁸

Solche Lösungen parsen den Quellcode nach potentiellen Schwachstellen, wie zum Beispiel dem Aufruf unsicherer Funktionen, Speicherlecks oder generell geringer Codequalität. Diese Suchvorgaben lassen sich verhältnismäßig gut abstrahieren und automatisieren, sodass SAST-Lösungen hier im Schnitt bei vergleichbarem Aufwand bessere Ergebnisse erreichen als manuelle Reviews.⁹

Der Nachteil dieser Lösungen zeigt sich am Beispiel des Secrets-Scanning in Form einer großen Anzahl an Ergebnissen, unter denen die größte Gruppe falsch-positive sind. Zur Klassifizierung von Ergebnissen siehe Kapitel 3.7.

Der Grund dafür ist, dass die meisten Scans im SAST „ungenau“ konfiguriert werden, da die Gefahr eine Schwachstelle zu übersehen ein größeres Problem darstellt als die vielen falsch-positiven Resultate. Aus diesem Grund ist beispielsweise der Einsatz von Entropiefiltern, wie er in Kapitel 3.6 gezeigt wird, nur bedingt praktikabel.

Die Ergebnisse von SAST-Scans sollten vielmehr als eine Liste von Hinweisen auf potentielle Schwachstellen angesehen werden, die im Nachgang manuell ausgewertet werden muss. Die Empfehlung lautet hier generell die Ausgabe eines jeden SAST-Scans manuell zu validieren.¹⁰

Möglicherweise kann die Einbeziehung des Kontexts durch den Einsatz von maschinellem Lernen, wie er beispielsweise von der SAP mit dem Secrets-Scanner Credential-Digger angewendet wird, in Zukunft ein besseres Verhältnis von falsch-positiven zu wahr-positiven Ergebnissen liefern. Weiteres hierzu kann Kapitel 5.3.1 entnommen werden.

8 [41]

9 [28]

10[28]

3.5 Regulärer Ausdruck

Unter einem regulären Ausdruck, englisch „Regular-Expression“ und kurz „Regex“, versteht man eine Zeichenkette, bei der normale Zeichen und Metazeichen miteinander kombiniert werden. Bei normalen Zeichen handelt es sich zum Beispiel um die druckbaren Zeichen aus der American-Standard-Code-for-Information-Interchange (ASCII)-Tabelle. Die Metazeichen wiederum werden beispielsweise genutzt, um die Menge, Art oder Lage der normalen Zeichen zu spezifizieren. Über dieses Konzept lassen sich „Muster“, englisch „Pattern“, formulieren. Diese Muster können anschließend über das sogenannte „Pattern-Matching“ genutzt werden, um passende Abschnitte in einem Text zu finden.¹¹

Pattern-Matching lässt sich somit nutzen, um in Dateien nach Secrets zu suchen.

Bekannte Secrets-Scanner wie Gitleaks oder TruffleHog arbeiten mit Pattern-Matching, um sensible Informationen in Dateien zu finden. Dazu müssen die zu suchenden Secrets zunächst durch ein Regex-Muster abstrahiert werden.

Im folgenden Abschnitt wird das Prinzip der Abstraktion am Beispiel eines Passwortes demonstriert.

Eine typische Passwortzuweisung im Quellcode könnte ähnlich zu der in Auflistung 3.1 aussehen. Diese Codezeile lässt sich in zwei Abschnitte um den Zuweisungsoperator herum unterteilen.

Linker Hand des Zuweisungsoperators „=“ steht eine Zeichenkette, die typischerweise „Password“, „pw“ oder Ähnliches beinhaltet. Rechter Hand des Zuweisungsoperators steht das eigentliche Passwort. Dieses besteht in der Regel aus einer Zeichenkette mit hoher Entropie und ist begrenzt durch Anführungszeichen. Weitere Informationen zu Entropie können dem Kapitel 3.6 entnommen werden.

Abhängig von der Programmiersprache endet die Zeile mit einem speziellen Zeichen oder einem Zeilenumbruch. In manchen Fällen, wie einem minifizierten Javascript-Code kann die Zeile auch ohne Zeilenumbruch fortgeführt werden. In diesem Fall endet die Anweisung mit einem Semikolon oder einer schließenden Klammer.

Weitere Beispiele um eine Anweisung zu beenden, können der jeweiligen Programmiersprache entnommen werden.

In Auflistung 3.2 wird ein regulärer Ausdruck für die Regex-Engine der Programmiersprache Go gezeigt, der genutzt werden kann, um die Passwortzuweisung aus Auflistung 3.1 zu finden.

Im Folgenden wird dieses Muster aus Auflistung 3.2 im Detail vorgestellt.

Zunächst wird mit (?i) dafür gesorgt, dass das folgende Muster Groß- und Kleinschreibung gleich behandelt. Anschließend folgt mit (?i:secret|password|pw) eine Auflistung von Ausdrücken, die linker

¹¹[48]

Hand des Zuweisungsoperators stehen können und mit einem logischen ODER verknüpft sind. In einem komplexeren Fall müsste auch berücksichtigt werden, dass dieser Ausdruck in einen umfassenden Ausdruck eingebettet sein könnte. Ein Beispiel für einen solchen umfassenden Ausdruck wäre LOGIN_PASSWORD_ADMIN.

In den meisten Programmiersprachen verlangt die Konvention ein Leerzeichen um den Zuweisungsoperator. Dieser Fall wird mit `(?:\\s?)` behandelt.

Der Zuweisungsoperator selbst ist wiederum in der Liste `(?:|=|=|=>|:)` enthalten. Da es eine ganze Reihe gültiger Zuweisungsoperatoren gibt, sind diese mittels logischen ODER verknüpft.

Rechts der Liste an Zuweisungsoperatoren folgt der generische Ausdruck `[\" '](.) [\"']`, der jedes beliebige Zeichen, außer einem Zeilenumbruch, umfasst. Dieser Ausdruck muss in einfache oder doppelte Anführungszeichen gesetzt sein, da nach Zeichenketten gesucht werden soll.

Hier zeigt sich das Problem des Regex-basierten Ansatzes. Je generischer dieser Ausdruck auf der rechten Seite ist, desto größer ist die Anzahl an Ergebnissen, die zurückgeliefert werden. Andererseits muss dieser Ausdruck aber auch bis zu einem gewissen Grad generisch formuliert sein, da Passwörter keinem spezifischen Muster folgen sollten.

So wird das Muster aus Auflistung 3.2 neben dem Passwort aus Auflistung 3.1 auch auf die Zeile PHP-Code aus Auflistung 3.3 passen.

Dabei wird ein falsch-positiver Treffer für folgendes Muster zurückgeliefert:

```
 Password'); ?></span> <?php echo \$result['password'
```

Das gefundene „Password“ entspricht dabei dem Ausdruck :

```
''); ?></span> <?php echo \$result['password'
```

Dieses Beispiel unterstreicht die Aussage aus Kapitel 3.4, wonach bei statischer Codeanalyse aufgrund der hohen Anzahl an falsch-positiven Treffern immer manuelle Nacharbeitung notwendig ist.

Die Herausforderung besteht darin, das generische Muster expliziter zu formulieren ohne dabei falsch-negative zu generieren, also valide Treffer auszuschließen.

```
1 LOGIN_PASSWORD = 'hunter2'
```

Listing 3.1 Passwortzuweisung

```
1 (?i) (?:secret|password|pw) (?:\\s?) (?:|=|=|=>|:) {1} (?:\\s?) [\" '] (. ) [\"']
```

Listing 3.2 Passwortzuweisung Regex

3 Grundlagen

Tabelle 3.1 Vergleich der Shannon-Entropie ausgewählter Zeichenketten, eigene Berechnungen

	Wert	Shannon-Entropie (H)
Falsch-Positiver Treffer	<p><?php _e('Password:'); ?> <?php echo \$result['password']; ?></p>	4.4
Zufällige Zeichenfolge	v8z,X)\%mKS/ZnHg5wpuV	4.3
GitLab SHA-1 Commit-Hash	ded97f4b0a8c153244c1736c98e1e30cca99ad45	3.8
AWS-API-Key	AKIAI44QH8DHBEXAMPLE	3.6
beliebtes Passwort	Passwort123!	3.4
Zeichenfolge	Hello, World!	3.1

1 <p><?php _e('Password: '); ?> <?php echo \\$result['password']; ?></p>

Listing 3.3 Zeichenfolge zur Demonstration eines falsch-positiven Treffers bei der Suche nach Passwörtern

3.6 Entropie und Entropiefilter

In der Informationstheorie gilt Entropie als das Maß für den mittleren Informationsgehalt eines Zeichens einer Zeichenfolge. Begründet wurde diese Einheit 1948 von Claude Shannon.

Enthält eine Zeichenfolge viele Redundanzen oder statistische Regelmäßigkeiten, so sinkt der Informationsgehalt und der Wert der Entropie wird kleiner.¹² Eine zufällige Zeichenfolge, wie ein API-Key, wird folglich eine hohe Entropie aufweisen. Der Tabelle 3.1 kann ein Vergleich der Werte der Shannon-Entropie für eine Reihe ausgewählter Zeichenketten entnommen werden. Die zugrundeliegende Formel zur Berechnung ¹³ lautet:

$$H(X) = - \sum_{i=1}^n p(x_i) \log_b p(x_i)$$

Um den Wert der falsch-positiv-Rate von Secrets-Scannern zu reduzieren, kann ein unterer Schwellwert für die Entropie des gefundenen Secrets formuliert werden. Für einen generischen API-Key liegt dieser Wert in der Standardkonfiguration von Gitleaks beispielsweise bei $H = 3.5$.¹⁴

Problematischer wird es, einen solchen Schwellwert für Passwörter zu formulieren. So zeigt Tabelle 3.1 für die Zeichenfolge Passwort123! einen Wert von $H = 3.4$, für die zufällige Zeichenfolge v8z,X)\%mKS/ZnHg5wpuV einen Wert von $H = 4.3$ und für den falsch-positiven Wert aus dem PHP-Code in Auflistung 3.3 eine Entropie von $H = 4.4$.

12[35]

13[5]

14[14]

Auf Grundlage dieser Werte lässt sich kein eindeutiger Filter festlegen, bei dem nicht das Risiko besteht, valide Secrets zu ignorieren. Aus diesem Grund sind Entropie-Filter mit Vorsicht einzusetzen.

3.7 Klassifizierung von Ergebnissen

Um die Ergebnisse der statischen Codeanalyse verarbeiten zu können, lassen sie sich in vier Kategorien einteilen, welche der Abbildung 3.2 entnommen werden können.

Bei einem validem Secret handelt es sich um die Kategorien wahr-positiv falls der Scanner ein valides Secret identifizieren konnte und um falsch-negativ falls der Scanner ein valides Secret nicht identifizieren konnte.

Falls ein Scanner einen Wert als Secret identifiziert hat, sich beim manuellen Review allerdings herausstellt, dass es sich hierbei nicht um einen sensiblen Wert handelt, spricht man von einem falsch-positivem Wert. Falls der Scanner einen solchen Wert korrekterweise nicht ermittelt, spricht man von einem wahr-negativem Ergebnis.

Die Farbgebung der Abbildung 3.2 dient dabei der Einordnung. Rot hinterlegt sind die beiden Kategorien falsch-positiv und falsch-negativ, da sie einen unerwünschten Zustand im Sinne der statischen Codeanalyse entsprechen. Grün wiederum entspricht den beiden Kategorien, die dem Wunschzustand entsprechen.

Die Ursachen für falsch-positive Treffer sind vielfältig und können beispielsweise in ungenauer Abstraktion der zu findenden Ausdrücke liegen. Vor allem bei SAST-Anwendungen wird diese Ungenauigkeit allerdings oftmals in Kauf genommen, da falsch-negative Ergebnisse minimiert werden sollen. Die Gefahr liegt hierbei darin, dass Informationen, die als falsch-negativ eingestuft werden, gar nicht erst beim manuellen Review erscheinen und somit nicht behoben werden können.

15

Je nach Herangehensweise beim Review gilt jedes Ergebnis des Scan zunächst als wahr-positiv oder als falsch-positiv. Im Rahmen des manuellen Reviews werden diese Ergebnisse nun bestätigt beziehungsweise als falsch-positiv vermerkt.

Für die Implementierung der Review-Anwendung in Kapitel 7 wurde der Ansatz gewählt, jeden Treffer zunächst als falsch-positiv zu klassifizieren. Der Grund dafür ist, wie in Kapitel 6.4 genauer erklärt, dass es, aufgrund der großen Datenmenge, weniger Aufwand bedeutet, nur die tatsächlich validierten Treffer als wahr-positiv zu klassifizieren als alle unerwünschten Treffer als falsch-positiv.

Die Dimension dieses Problems zeigt sich auch darin, dass OWASP das Verhältnis von falsch-positiven zu falsch-negativen Ergebnissen als wichtiges Kriterium bei der Auswahl von Quellcodeanalysewerkzeugen nennt.¹⁶

15[28]

16[8]

3 Grundlagen

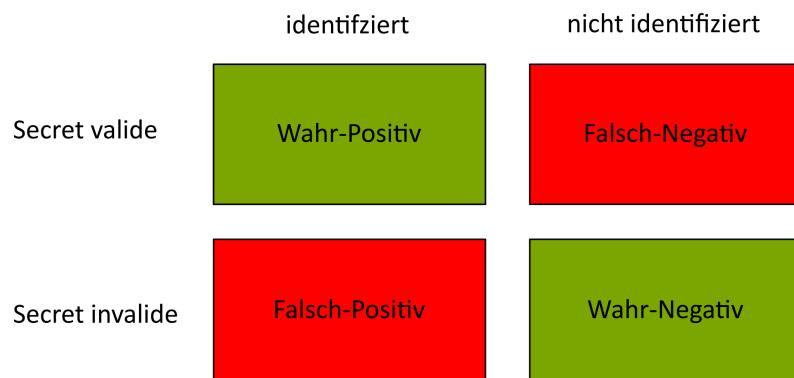


Abbildung 3.2 Übersicht potentieller Ergebnisse eines SAST-Scans, eigene Darstellung

3.8 Versionsverwaltung am Beispiel von Git

Git wurde ab 2005 als freies Werkzeug zur verteilten Versionsverwaltung von Dateien entwickelt. Das Projekt erfreut sich dabei seit vielen Jahren großer Beliebtheit in der IT-Branche, sowohl im kommerziellen als auch im open-source Bereich. Bekannte Unternehmen wie Google und Microsoft sowie große Open-Source-Projekte wie der Linux-Kernel setzen auf Git für die Versionsverwaltung ihrer Projekte. Auch innerhalb der IU findet Git Anwendung.

Als VCS dient Git unter Anderem dazu, jegliche Änderungen an den Dateien in einem Projekt aufzuzeichnen. Hierzu erzeugt Git mit jedem Commit faktisch einen Snapshot des Zustandes aller Dateien im Projekt und speichert anschließend eine Referenz auf diesen Snapshot. Das bedeutet, dass jede lokale Kopie eines Repositories auf dem Computer eines Entwicklers den gesamten Verlauf, auch bezeichnet als History, des Projektes beinhaltet.

Diese Dezentralität ist einer der Kernaspekte mit denen Git im Hinterkopf entwickelt wurde. Bezogen auf sensible Information im Quellcode trägt sie aber auch zum Secret-Sprawl, vergleiche Kapitel 3.2 bei, da ein Angreifer von jedem kompromittierten Computer aus eine lokale Suche über das Repository ausführen kann.

Im Sinne der Nachvollziehbarkeit von Änderungen an einem Projekt ist die History von enormer Bedeutung. Die goldene Regel lautet, dass Git dem Entwickler alle Freiheiten bietet solange die Änderungen sich noch lokal auf dem Computer befinden. Sobald Code einmal in das Repository übertragen wurde, ist dieser aber als „endgültig“ zu betrachten.

Ein Nachteil dieser Funktionalität zeigt sich, sobald sensible Daten mit dem Code in das Repository geschrieben wurden. Diese lassen sich, falls überhaupt, nur unter großem Aufwand wieder entfernen. Hierzu muss die History des Repositories abgeändert werden, was wiederum dem

Grundsatz von Git widerspricht.

Dass es sich hierbei um ein weit verbreitetes Problem handelt, belegen auch die Zahlen von GitGuardian, welche in Kapitel 3.2 besprochen wurden. Über die Gründe für das Zustandekommen dieser Zahlen kann nur spekuliert werden, vermutlich ließe sich hier mit entsprechenden Schulungen der Entwickler sowie Maßnahmen, wie dem Scannen nach sensiblen Informationen bereits beim Commit eine Verbesserung der Situation erreichen.

Eine Änderung der History ist dabei prinzipiell über die Option „filter-branch“, beziehungsweise über deren Nachfolger „filter-repo“ möglich. Sobald ein Projekt allerdings veröffentlicht wurde oder andere Entwickler ihrerseits bereits auf diesen Commits gearbeitet haben, wird davon abgeraten, diese Option anzuwenden, um das Repository nicht zu beschädigen.

Auch lässt sich über diese Funktionen die lokale History auf den Computern anderer Entwickler nicht direkt ändern. Erst wenn diese die Änderungen synchronisiert haben, wird auch deren lokale History angepasst.¹⁷

Erschwerend kommt hinzu, dass ein direkter Zugriff auf einen eigentlich entfernten Commit über dessen Secure-Hash-Algorithm (SHA)-1 Hash oftmals weiter möglich ist. Dies liegt daran, dass Plattformen wie GitHub, GitLab oder Bitbucket Commits cachen. Falls die Commit-ID bekannt ist, oder erraten werden kann, ist auf diesen Plattformen weiter direkter Zugriff auf den eigentlich gelöschten Commit möglich. Erst ein serverseitiges Leeren der Caches kann dieses Problem beseitigen.

Die einzige verlässliche Lösung für Secrets, die einmal in ein Repository übertragen wurden, ist folglich diese zu invalidieren und neu zu erstellen.

3.9 Pre-Commit-Framework

In Kapitel 3.8 wurde dargestellt, wie groß der Aufwand sein kann, um sensible Informationen aus einem Repository zu entfernen. Folglich empfiehlt es sich, präventiv anzusetzen und damit zu verhindern, dass überhaupt ein Commit mit sensiblen Informationen abgesetzt werden kann.

Eine Möglichkeit, die Git standardmäßig mitbringt und die hierfür genutzt werden kann, sind die sogenannten „Hooks“. Diese bieten eine Möglichkeit, definierte Funktionen beim Eintritt bestimmter Ereignisse auszuführen.

Ein mögliches Framework, welches dabei hilft die Verwaltung von Git-Hooks einfacher zu gestalten, ist das Pre-Commit-Framework, welches, trotz seines Namens, mittlerweile nicht mehr nur für

¹⁷[43]

3 Grundlagen

Commits nutzbar ist.

Das Framework zieht für seine Funktionen dabei ausschließlich Dateien in Betracht, die sich bereits in der Staging-Area befinden. Auf diese Weise kann die Anzahl falsch-positiver Treffer reduziert werden.¹⁸

In der Abbildung 3.3 wird am Beispiel von Gitleaks demonstriert, wie ein solcher Hook über das Pre-Commit-Framework funktioniert.

Hierzu muss in jedem zu überwachendem Repository eine entsprechende *.pre-commit-config.yaml*-Datei angelegt werden. Diese beinhaltet eine Referenz auf eine *.pre-commit-hooks.yaml*-Datei im Repository der Anwendung, die über diesen Hook angestoßen werden soll. Weitere Konfigurationen sind an dieser Stelle nicht nötig.

Im Beispiel wird nun versucht, eine Datei mit einem AWS-Access-Token in das Repository zu übertragen. Beim Ausführen des Commits wird Gitleaks automatisch angestoßen, welches einen Treffer zurückmeldet. Der Commit wird daraufhin unterbunden.

Zusätzlich erhält der Commiter die Ausgabe von Gitleaks, welche genaue Informationen über Art und Lage des potentiellen Secrets enthält.

Eine der definierten Anforderungen aus Kapitel 4.2 ist die Möglichkeit, den Scanner im Rahmen eines solchen Hooks einzusetzen zu können. Aktuell findet das Pre-Commit-Framework, beziehungsweise Git-Hooks, noch keine flächendeckende Verbreitung an der IU. Im Zuge der Umsetzung eines mehrstufigen SAST-Prozesses können die Git-Hooks als ein erster Schritt verstanden werden.

Da die Ausführung lokal erfolgt, liegt sie immer in Verantwortung der Entwickler, sodass nicht garantiert werden kann, dass die Scans auch ausgeführt werden.

3.10 Anforderungspriorisierung

Gemäß der Definition der non-profit Organisation International-Requirements-Engineering-Board (IREB) ist das Anforderungsmanagement ein Teil des Requirements-Engineering und umfasst unter Anderem die Ermittlung und Erhebung von Anforderungen.

Da nicht alle Anforderungen von gleicher Wichtigkeit sind, ist es notwendig, diese zu priorisieren. Um eine Priorisierung durchführen zu können, ist zunächst eine Bewertung der einzelnen Anforderungen durch entsprechend qualifizierte Personen notwendig.

Anschließend kann auf Basis einer Priorisierungstechnik eine Sortierung der Anforderungen erfolgen.

¹⁸[37]

```

~/R/iss.nessus_vlan_setup ➜ 🐧 dev 12 ➤ echo 'aws = "AIDACKCEVSQ6C2EXAMPLE"' > password.txt
~/R/iss.nessus_vlan_setup ➜ 🐧 dev 12 !1 ➤ git add .
~/R/iss.nessus_vlan_setup ➜ 🐧 dev 12 +1 ➤ git commit -m "added AWS-Access-Token"
Detect hardcoded secrets..... Failed
- hook id: gitleaks
- exit code: 1

o
  |
  o
    gitleaks

Finding: aws = "REDACTED"
Secret: REDACTED
RuleID: aws-access-token
Entropy: 3.746439
File: password.txt
Line: 1
Fingerprint: password.txt:aws-access-token:1

8:54AM INF 1 commits scanned.
8:54AM INF scan completed in 63.7ms
8:54AM WRN leaks found: 1

```

Abbildung 3.3 Beispiel für die Ausführung von Gitleaks über einen Pre-Commit-Hook, eigene Darstellung

Grundsätzlich lassen sich hierbei zwei Techniken zur Priorisierung von Anforderungen unterscheiden. Bei der Ad-Hoc-Technik geschieht die Priorisierung auf Basis von Erfahrungswerten. Dabei weist eine entsprechend qualifizierte Person jeder Anforderung einen Wert zu. Bekannte Verfahren aus diesem Bereich sind die Requirements-Triage, die Top-Ten-Technik und die Kano-Klassifikation.

Vorteil der Ad-Hoc-Technik ist, dass sie sich vergleichsweise schnell durchführen lässt. Andererseits ist sie aber fehleranfälliger als der zweite Ansatz über die analytische Priorisierungstechnik.

Bei der analytischen Priorisierungstechnik werden die Anforderungen systematisch gegeneinander verglichen. Oftmals kommen hier auch verschiedene Experten zum Einsatz, sodass ein Ergebnis sorgfältiger und zuverlässiger ermittelt werden kann. Dieser Vorgang nimmt allerdings deutlich mehr Zeit in Anspruch als die Ad-Hoc-Technik.

Bekannte Verfahren aus diesem Bereich sind die Wiegers'sche Priorisierungsmatrix und der sogenannte „Analytical Hierarchy Process“.¹⁹

Bei der Auswahl des Secrets-Scanners wurde von Seiten der IU die überschaubare Anzahl von 13 Anforderungen, vergleiche Kapitel 4.2, definiert.

Aus diesem Grund wurde entschieden, den Auswahlprozess zweistufig aufzubauen. Zunächst soll anhand der Requirements-Triage eine Vorauswahl getroffen werden. Anschließend soll eine praktische Evaluation der verbliebenen Kandidaten anhand eines Quellcode-Repositories ein Ergebnis liefern.

¹⁹[47]

3 Grundlagen

Bei der Requirements-Triage handelt es sich um eine sogenannte „Ein-Kriterien-Klassifikation“. Dabei wird jede Anforderung in eine von drei Kategorien eingeteilt:²⁰

- **Muss-Anforderungen**

Hier eingeordnete Anforderungen müssen von der Anwendung erfüllt werden, sonst werden sie vom weiteren Auswahlprozess ausgeschlossen.

- **Soll-Anforderungen**

Hier eingeordnete Anforderungen sollen im besten Fall von der Anwendung erfüllt werden. Falls eine solche Anforderung bereits mit einem festen Umsetzungsdatum auf der Roadmap der untersuchten Anwendung aufgeführt ist, wird sie als erfüllt betrachtet.

- **Kann-Anforderungen**

Hier handelt es sich um Anforderungen, deren Erfüllung nicht zwingend vorausgesetzt wird. Solche Anforderungen können aber beispielsweise die Integration in bestehende Prozesse oder Infrastruktur erleichtern.

Das ISS-Team hat sich hierbei bewusst für eine schnelle Entscheidungsfindung im Einklang mit dem agilen Ansatz der IU entschieden.

²⁰[47]

4 Anforderungen Secrets-Scanner

Im folgenden Kapitel wird der Auswahlprozess beschrieben, der durchgeführt wurde, um einen Secrets-Scanner für den zukünftigen Einsatz an der IU zu finden. Hierzu werden zunächst ein Überblick über den Auswahlprozess gegeben und anschließend die einzelnen Schritte des Prozesses im Detail erläutert.

Grundsätzliche Informationen worum es sich bei einem Secrets-Scanner handelt, kann Kapitel 3.3 entnommen werden.

Nach der Vorauswahl werden drei geeignete Scanner im Detail vorgestellt. Anschließend werden diese drei einem praktischen Test an einem Repository der IU unterzogen.

Da es sich im Bereich statischer Codeanalyse empfiehlt, nicht nur mit einem Programm zu arbeiten, sondern die selbe Analyse mit mehreren Programmen durchzuführen und deren Ergebnisse anschließend zusammenzuführen, wird der praktische Test an allen drei Kandidaten durchgeführt.¹ Folglich müsste eine Entscheidung nicht auf eine Anwendung hinauslaufen, der Einsatz von mehr als einem Scanner würde allerdings den Rahmen dieser Arbeit überschreiten.

Abschließend wird in diesem Kapitel eine konkrete Auswahl für den Scanner Gitleaks vorgenommen und die Entscheidung begründet.

4.1 Auswahlprozess

Ziel des Auswahlprozesses ist es, eine konkrete Entscheidung zu treffen, welcher am Markt verfügbare Secrets-Scanner durch das ISS-Team der IU zukünftig eingesetzt werden soll. Dieser Scanner liefert auch die Basis für den, im Rahmen dieser Arbeit, angefertigten Prototypen einer Anwendung zum klassifizieren von Ergebnissen.

Der Abbildung 4.1 lässt sich der genaue Ablaufprozess grafisch entnehmen.

Entsprechend dem ersten Schritt „Anforderungen“ wurden diese zunächst erarbeitet. Dies geschah auf Basis von praktischen Erfahrungen des ISS-Teams und kann in Kapitel 4.2 eingesehen werden. Diese Anforderungen wurden anschließend nach dem Prinzip der Requirements-Triage, vergleiche Kapitel 3.10, geordnet. Weitere Informationen zu diesem Prozessschritt können auch Kapitel 4.3

¹ [28]

4 Anforderungen Secrets-Scanner

entnommen werden.

Um neben dem bereits einmalig eingesetzten Gitleaks, vergleiche Kapitel 1.2, eine möglichst breite Palette an Produkten für den praktischen Test zur Verfügung zu haben, wurde im nächsten Prozessschritt zunächst eine Recherche zur Marktevaluation durchgeführt. Weitere Details können hierzu dem Kapitel 5.2 entnommen werden. Bei dieser Recherche wurden bereits erste Scanner auf Basis von Experteneinschätzungen durch das ISS-Team wieder verworfen.

Die drei interessantesten Kandidaten Credential-Scanner, TruffleHog und Gitleaks wurden anschließend einer detaillierten Prüfung unterzogen. Für weitere Informationen zu diesen drei Anwendungen kann Kapitel 5.3 eingesehen werden.

Im Prozessschritt „Praktischer Test“ wurde ein konkreter Testfall auf Basis eines realen Quellcode-Repositories der IU generiert. Dieser wurde um die Anforderungen herum aufgebaut, um möglichst viele davon praktisch evaluieren zu können. Weitere Informationen lassen sich dem Kapitel 5.4 entnehmen.

Darüber hinaus konnte diese Prüfung auch Erkenntnisse über das Setup und die Handhabung der Scanner liefern.

Zur Entscheidungsfindung wurden abschließend die Testergebnisse der Scans sowie ihre Übereinstimmung mit den Anforderungen analysiert. Da für den Prototypen nur ein Scanner unterstützt werden kann, musste das Ergebnis eindeutig ausfallen. Die detaillierte Entscheidungsfindung kann Kapitel 5.5 entnommen werden.

4.2 Definition von Auswahlkriterien

Der Rahmen eines Secrets-Scan an der IU wird, entsprechend dem Kapitel 1.3, auf Git-Repositories sowie Textdateien eingeschränkt.

Die Liste an Anforderungen ist aus den spezifischen Bedürfnissen der IU abgeleitet worden. Hierbei sind die Erfahrungen eingeflossen, die im Rahmen der in Kapitel 1.2 erwähnten manuellen Prüfung gesammelt werden konnten. Zusätzlich berücksichtigt wurden die Empfehlungen der OWASP an Quellcodeanalysewerkzeuge.²

Das ISS-Team hat folgende Anforderungen definiert, die in den Auswahlprozess einfließen müssen:

History-Scanning

Da ein Git-Repository über eine History verfügt, ist es für das ISS-Team essentiell, dass auch diese durchsucht werden kann. Gerade in der History eines Repositories können

² [8]

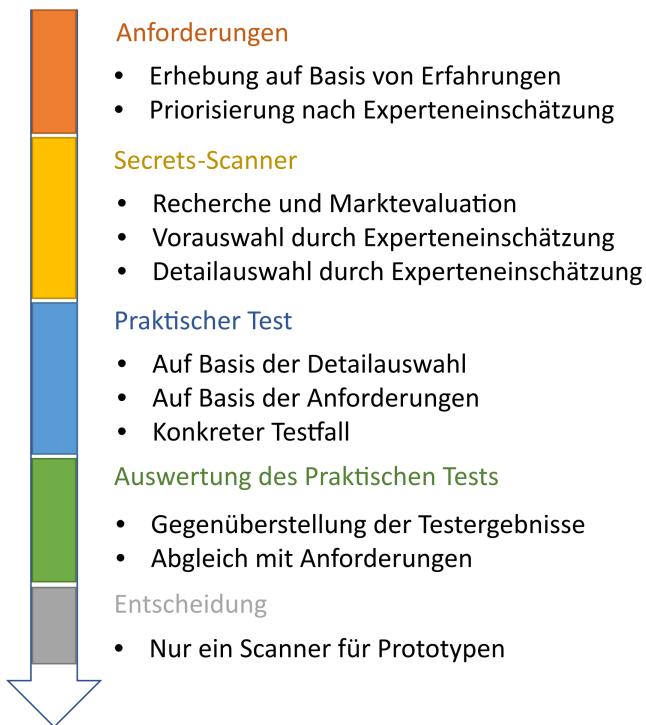


Abbildung 4.1 Überblick Evaluationsprozess zur Auswahl eines Secrets-Scanners, eigene Darstellung

sich im Laufe des Lebenszyklus einer Software viele sensible Informationen ansammeln, siehe hierzu auch Kapitel 3.8.

Einschränkungen des Scanumfangs

Um Scans feingranularer steuern zu können, muss die Möglichkeit existieren, Scans hinsichtlich verschiedener Kriterien, wie zum Beispiel dem Zeitpunkt des letzten Commits oder einem spezifischen Commiter, einzuschränken.

Anpassungen der Suchmuster

Viele Scanner verfügen über fest hinterlegte Regeln um beispielsweise API-Token zu erkennen. Dies ist möglich, da manche Anbieter ihre API-Tokens nach einem festen Schema aufbauen. Nicht erfasst werden in diesem Fall generische Secrets oder Schemata, die nur intern verwendet werden. Hier ist die Erwartung, durch anpassbare Suchmuster die Ergebnismenge zu erweitern. Für weitere Details zu Suchmustern können die Kapitel 3.5 und 3.7 herangezogen werden.

Es soll dabei auch möglich sein, gezielt nach bekannten Bezeichnern zu suchen. Beispielsweise haben Konfigurationsdateien aus Konventionsgründen oftmals eindeutige Indikatoren, wie zum Beispiel den Namen `.env`.

Anpassungen von Ausschlussmuster (Anti-Pattern)

Die Erfahrung zeigt, dass gerade Testdateien oftmals viele falsch-positive Treffer produzieren. Um die falsch-positiv-Rate zu verbessern muss es möglich sein, gewisse Dateien, Dateitypen oder ganze Verzeichnisse von der Suche auszuschließen.

Darüber hinaus wäre es wünschenswert, wenn diese Anpassungen über eine gewisse Flexibilität, beispielsweise über Regex, verfügen würden.

Lokale Ausführung

Da der Quellcode der IU-Anwendungen proprietär ist, muss jeder Scan lokal auf Systemen der IU erfolgen können. Auch der Zugriff eines Drittanbieter-Services auf die Repositories der IU ist ohne weitere Überprüfung des Anbieters sowie einer entsprechenden vertraglichen Vereinbarung nicht ohne weiteres möglich. Siehe hierzu auch Kapitel 5.2.

Auch für die Einbindung in die CI/CD-Pipeline ist eine lokale Ausführung nötig, da hier keine API-Aufrufe ausgeführt werden um die Ausführungszeit möglichst gering zu halten.

Maschinellweiterarbeitbares Ausgabeformat

Im Rahmen des Risikomanagementprozesses ist es erforderlich, die identifizierten und als wahr-positiv klassifizierten Treffer weiter zu verarbeiten. Hierzu müssen die Ergebnisse in einem Format geliefert werden können, welches von nachfolgenden Services aufgegriffen und weiterverarbeitet werden kann. Wünschenswert ist ein gängiges Format wie JavaScript-Object-Notation (JSON), YAML-Ain't-Markup-Language (YAML) oder Comma-Separated-Value (CSV). Interessant wäre darüber hinaus eine Ausgabe im Static-Analysis-Results-Interchange-Format (SARIF), um den Datenaustausch mit weiteren Anwendungen zur statischen Codeanalyse zu erleichtern.³

Umfang der Ausgabedaten

Um ein Ergebnis verifizieren und gegebenenfalls weiterverarbeiten zu können, müssen ausreichend Daten zurückgeliefert werden. Dies muss mindestens den SHA-1 Hash des Commits, die Datei, in der das potentielle Secret gefunden wurde, sowie die gefundene Zeichenfolge an sich beinhalten. Falls es sich nicht um ein Git-Repository handelt, kann die Information bezüglich des Commits entfallen.

Weitere Informationen, die geliefert werden können, umfassen die Zeile innerhalb der Datei sowie eine Information, welche der Suchmuster zu einer Übereinstimmung geführt haben.

Einbindung in CI/CD-Pipelines

Das Quellcode-Repository ist die Stelle, an der alle Softwareentwicklungen zusammenlaufen. Um im Einklang mit dem SDLC und dem DevSecOps-Ansatz eine

³ [33]

Sicherheitsprüfung in jedem Entwicklungsschritt vorzuhalten, soll der Scanner eine Einbindung in die entsprechenden Pipelines ermöglichen. Dies bietet zusätzlich die Möglichkeit eines automatisierten Scannings. Für weitere Möglichkeiten siehe Kapitel 2.4.2.

Pre-Commit-Hooks

Um das Hochladen von sensiblen Informationen durch den Entwickler in ein Repository bereits im Ansatz zu verhindern, gibt es in Git die Möglichkeit sogenannter „Hooks“. Hier kann ein Scanner, der lokal auf dem PC eines Entwicklers installiert ist, bereits beim Commit ausgeführt werden und gegebenenfalls eine Warnung oder einen Block generieren. Für weitere Details hierzu siehe Kapitel 2.4.2, 3.8 und Kapitel 3.9.

Um die Integration zu erleichtern, soll ein Scanner bereits eine vordefinierte Konfiguration für das Pre-Commit-Framework mitliefern.

Repository-Aktivität und Support

Bei der Verwendung von Drittanbieter-Software spielt der Support, entweder durch die Entwicklerfirma oder durch die Entwicklergemeinschaft, eine zentrale Rolle. Wichtig ist hierbei, ob es eine Möglichkeit gibt, mit den Entwicklern im Problemfall in Kontakt treten zu können. Dies betrifft auch die Weiterentwicklung sowie das Bugfixing des Projekts sowie die Frage, ob regelmäßige Aktualisierungen eingespielt werden. Diese Anforderung lässt sich nicht immer mit einem Ja oder Nein beantworten, beziehungsweise in Zahlen messen, sodass hier eine Experteneinschätzung im Umgang mit Open-Source-Anwendungen zugrunde gelegt wird. Sofern möglich, wird versucht diese Entscheidung durch Zahlen zu untermauern.

Ein Scanner muss allerdings eine gewisse Aktivität in seinem Repository in den letzten Monaten aufweisen, um überhaupt in Betracht gezogen zu werden.

Kommerzielle Einschränkungen

Kostenpflichtige Lösungen werden von vornherein ausgeschlossen. Grundsätzlich wäre es denkbar, für Supportleistungen oder einen Vollzugriff auf eine Anwendung zu bezahlen, sobald sich eine Lösung als nützlich erwiesen hat.

Anwendungen, die aufgrund ihrer Lizenzierung generell nicht für kommerzielle Zwecke eingesetzt werden dürfen, werden für die Evaluation nicht berücksichtigt.

Dies gilt nicht für Anwendungen, deren Lizenzierung Copy-Left Klauseln beinhalten, da, nach aktuellem Stand, keine internen Anpassungen an den zu evaluierenden Scannern geplant sind.

Validierung von Secrets

Manche Anbieter von API-Keys und Access-Tokens bieten APIs an, über die sich prüfen lässt, ob ein Key oder Token gültig ist, beziehungsweise welche Zugriffe mit diesem Secret möglich sind. Eine solche Funktionalität kann helfen, die Qualität der zurückgelieferten

4 Anforderungen Secrets-Scanner

Ergebnisse zu verbessern, indem die Anzahl an falsch-positiven Ergebnissen gesenkt wird.

Art der Bereitstellung

Neben der Bereitstellung des Quellcodes wäre es wünschenswert, dass die Anwendung in einem direkt nutzbaren Format, wie einem vorkompiliertem Binary zur Verfügung steht. Falls dies nicht möglich ist, muss mindestens eine Anleitung verfügbar sein, ein solches Binary zu erstellen. Falls die Anwendung als Bibliothek verfügbar ist, muss eine Anleitung vorliegen, wie diese Bibliothek zu nutzen ist. Alternativ denkbar wäre auch die Bereitstellung über ein Container-Image.

4.3 Priorisierung von Anforderungen

Wie in Kapitel 3.10 beschrieben, wurde durch das ISS-Team entschieden, die erhobenen Anforderungen zunächst nach dem Prinzip der Requirements-Triage in die Kategorien „Muss“, „Soll“ und „Kann“ zu unterteilen.

In den Rahmen der Muss-Kategorie wurden folgende Anforderungen eingesortiert, die durch das ISS-Team als absolut grundsätzlich angesehen wurden, um einen Scanner überhaupt in Betracht zu ziehen:

- History-Scanning
- Lokale Ausführung
- Repository-Aktivität und Support
- Maschinellweiterverarbeitbares Ausgabeformat
- Anpassung von Ausschlussmustern (Anti-Pattern)
- Umfang der Ausgabedaten

In die Kategorie „Soll“ wurden folgende Anforderungen eingesortiert, die nach Abdeckung der Muss-Anforderungen immer noch von großer Wichtigkeit sind:

- Einbindung in CI/CD-Pipelines
- Pre-Commit-Hooks
- Anpassung der Suchmuster (hier eingesortiert, um den Fall eines bereits perfekten Suchmusters abzudecken)
- Einschränkung des Scanumfangs

4.3 Priorisierung von Anforderungen

Und abschließend wurden in die Kategorie „Kann“ folgende Anforderungen eingesortiert, die im Falle eines Gleichstandes zwischen den Scannern einen positiven Unterschied in der Handhabung machen können:

- Validierung von Secrets
- Art der Bereitstellung
- Kommerzielle Einschränkungen

Diese Priorisierung wurde auf Basis der Erfahrungen aus dem manuellen Scan sowie auf Basis der bisherigen Erfahrungen im Umgang mit Anwendungen und Services aus den Bereichen der IT-Sicherheit und der Softwareentwicklung getroffen. Es ist durchaus möglich, dass sich diese Priorisierung zukünftig durch neue Erfahrungen und Anforderungen ändert.

5 Evaluation Secrets-Scanner

Im folgenden Kapitel wird beschrieben, wie ein Secrets-Scanner für die IU ausgewählt wurde. Hierzu wird zunächst dargelegt, warum das aktuell zur statischen Codeanalyse eingesetzte SonarCloud die Anforderungen an einen Secrets-Scanner nicht erfüllt, die von Seiten des ISS-Teams gestellt werden. Die entsprechenden Anforderungen können Kapitel 4.2 entnommen werden.

Im ersten Schritt des Auswahlprozesses wurde zunächst eine Recherche durchgeführt, welche Scanner aktuell am Markt verfügbar sind. Hierbei konnte bereits eine erste Vorauswahl anhand der Requirements-Triage, vorgestellt in Kapitel 3.10, durchgeführt werden.

Die drei Kandidaten, welche die Anforderungen am besten abdecken konnten, werden anschließend im Detail vorgestellt. Hier wird auch geklärt, inwiefern sie die einzelnen Anforderungen erfüllen, beziehungsweise nicht erfüllen können.

Zur weiteren Evaluation wurden diese anschließend einem praktischen Test unterzogen. Der Aufbau dieses Tests sowie die daraus resultierenden Ergebnisse werden zusammenfassend vorgestellt, da aus Gründen der Geheimhaltung keine detaillierten Werte genannt werden können.

Das Kapitel schließt mit einer Zusammenfassung des Prozesses und der Begründung der Entscheidung für den Scanner Gitleaks.

5.1 SonarCloud für Secrets-Scanning

SonarCloud ist ein cloudbasiertes Produkt des Unternehmens SonarSource S.A., welches zur statischen Codeanalyse mit Fokus auf Codequalität eingesetzt werden kann. Die Produkte der Firma SonarSource sollen Entwicklern dabei helfen, saubereren Code zu schreiben. Der Fokus bei SonarCloud liegt dabei auf der nahtlosen Integration in die cloudbasierten Entwicklungsprozesse der Kunden.¹

SonarCloud wird von Seiten der IU für statische Codeanalyse eingesetzt. Als zentrale Sammelstelle für alles, was den Quellcode betrifft, wurde zunächst evaluiert, inwiefern SonarCloud bei der Suche nach Secrets im Quellcode genutzt werden kann.

Nach einer Analyse des ISS-Teams verfügt diese Anwendung nur über eine rudimentäre Funktion zur Suche nach sensiblen Informationen im Quellcode.

¹ [45]

5 Evaluation Secrets-Scanner

Die einzelnen Funktionen, über die SonarCloud die Qualität des Quellcodes prüft, sind als Regeln formuliert. Die Suche nach Secrets ist hierbei über die „Rule S2068: Credentials should not be hard-coded“ abgebildet. Dabei handelt es sich um eine simple Schlüsselwortsuche, welche keine komplexeren Suchmuster erlaubt.

Das bedeutet, dass SonarCloud nur Ausdrücke findet, die exakt so im Quellcode vorkommen, wie sie in der übergebenen Auflistung definiert wurden.

Die Entwickler von SonarCloud rechtfertigen dies mit der potentiell hohen Anzahl an falsch-positiven Ergebnissen, welche einen hohen manuellen Aufwand zur Validierung nach sich ziehen würde.²

Da es sich bei SonarCloud um eine Software-as-a-Service (SaaS)-Lösung handelt, gibt es aktuell keine Möglichkeit eine solche Funktionalität über Erweiterungen einzubinden, wie es bei der On-Premise-Lösung SonarQube möglich wäre.

Innerhalb der IU ist für die nächste Zeit allerdings kein Wechsel auf SonarQube angestrebt, sodass das Secrets-Scanning über eine zusätzliche Lösung erfolgen muss.

5.2 Marktübersicht

Um eine Übersicht über die am Markt verfügbaren Secrets-Scanner zu erhalten, wurde eine Online-Recherche durchgeführt. Dabei wurde die Suchmaschine Google mit verschiedenen Stichwörtern ausgeführt, welche im Folgenden aufgeführt sind:

- secrets scanner
- credential scanner
- secret sprawl
- secret leak
- secret detection

Darüber hinaus wurde OWASP³ nach dort gelisteten Programmen durchsucht und vergleichende Studien⁴ herangezogen.

Hierbei lässt sich zunächst festhalten, dass es vergleichsweise wenige Anwendungen gibt, die auf das Aufspüren von Secrets in Quellcode-Repositories oder allgemein Dateien spezialisiert sind. Dies gilt sowohl für Open-Source-Anwendungen als auch für kommerzielle Anbieter. OWASP listet zwar 115 Einträge für Anwendungen zur statischen Codeanalyse, einschlägige Anwendungen wie

2 [46]

3 [8]

4 [32]

Gitleaks oder TruffelHog fehlen in dieser Übersicht aber.

Ausgeschlossen wurden Open-Source-Anwendungen, die, ausgehend von Hosting-Plattformen wie GitHub oder GitLab, aufgrund fehlender Aktivitäten offensichtlich nicht mehr weiterentwickelt werden. Auch Lösungen, die nur im Rahmen von Data-Loss-Prevention (DLP) nach Secrets suchen wurden ausgeschlossen, da von Seiten des ISS-Teams keine DLP-Lösung gesucht wird.

Eine Reihe weiterer Produkte, die auf Basis von künstlicher Intelligenz versprechen, die falsch-positiv-Rate zu verbessern, wurde ausgeschlossen, da sie keine Möglichkeit bieten, ein Repository lokal zu scannen. Dies ist zunächst unbedingt erforderlich, da es nicht vertretbar ist, Quellcode, der potentiell sensible Daten beinhaltet, auf eine nicht vertrauenswürdige Drittanbieterplattform hochzuladen. Es wäre denkbar, zukünftig eine solche Anwendung zu evaluieren, nachdem der Drittanbieter einer entsprechenden Prüfung unterzogen wurde.

5.3 Vorstellung evaluierter Secrets-Scanner

In den folgenden Kapiteln erfolgt eine Vorstellung der drei Anwendungen Credential-Digger, TruffleHog und Gitleaks, die für eine weitere Analyse nach der Marktevaluation in Betracht gezogen wurden. Dabei werden die Features der Anwendungen den in Kapitel 4.2 erhobenen Anforderungen gegenübergestellt.

Bei allen drei untersuchten Kandidaten basiert die Suche zunächst auf regulären Ausdrücken, genauer hierzu in Kapitel 3.5. Bei der Einschränkung der Suchmenge auf gültige Secrets unterscheiden sich die Herangehensweisen und reichen von maschinellem Lernen über Entropiefilter, vergleiche hierzu Kapitel 3.6, bis hin zu einer Validierung gegen bekannte API-Endpunkte.

Die Treffer, welche die drei Scanner im praktischen Test geliefert haben, sowie eine Interpretation des Ergebnisses kann Kapitel 5.4 entnommen werden.

Um zu prüfen, inwiefern die Kandidaten aktiv weiterentwickelt werden, wurde mittels des Open-Source-Tools „IsItMaintained“ die Anzahl offener Issues und die Bearbeitungszeit dieser untersucht. Hierbei werden der Median der Bearbeitungszeit bis ein Issue geschlossen wird und der prozentuale Anteil offener Issues an der Gesamtmenge aller Issues ausgewertet.

Entsprechend der Vorgaben durch IsItMaintained werden Issues, die älter als sechs Monate sind, bei der Auswertung der Bearbeitungszeit nicht berücksichtigt. Unberücksichtigt bleiben auch Issues, die als „feature“, „enhancement“ oder „duplicate“ gelabelt sind.⁵

⁵ [21]

5.3.1 Credential-Digger

Wie in einem Blogbeitrag⁶ erläutert, stand das Cybersecurity-Team der SAP vor der Aufgabe, die umfangreichen Quellcode-Repositories des größten europäischen Softwareentwicklers nach sensiblen Informationen zu durchsuchen.

Dabei kamen sie im Rahmen ihrer Evaluation zu dem Ergebnis, dass die verfügbaren Secrets-Scanner eine, für ihre Zwecke, zu hohe Rate an falsch-positiven Treffern zurück lieferten. Ihrer Analyse nach lag dies zum Einen an der Regex-basierten Mustererkennung von sensiblen Informationen abseits von wohldefinierten Token. Zum Anderen konnten die evaluierten Scanner nicht zwischen potentiell validen Daten und generell invaliden Testdaten unterscheiden.

Aus diesen Gründen entschied sich das Team dazu, einen eigenen Secrets-Scanner zu entwickeln, welcher neben dem Regex-Ansatz auch einen Ansatz auf Basis von maschinellem Lernen integriert.⁷ Nach Stand der Recherche ist der Credential-Digger der einzige Scanner mit maschinellem Lernen, der unter einer Open-Source-Lizenz verfügbar ist.

Die Anwendung wurde unter dem Namen „Credential-Digger“, mit der Apache-2.0-Lizenz versehen, im Juni 2020 auf GitHub veröffentlicht. Dort hat das Projekt im Oktober 2022 über 220 Stars und zehn Beobachter, sowie 12 verschiedene Beitragende.

IsItMaintained gibt dabei an, das 24% aller Issues offen sind und der Median der Bearbeitungszeit überdurchschnittlich hohe 50 Tage beträgt.⁸

Diese Zahlen belegen, dass der Credential-Digger außerhalb der SAP keine nennenswerte Verbreitung erlangen konnte. Das Produkt steht und fällt folglich mit der Unterstützung durch Mitarbeiter der SAP.

Nach einer schnellen und unkomplizierten Inbetriebnahme über das bereitgestellte Docker-Image konnte der Scanningprozess analysiert werden.

Dieser läuft mehrstufig ab, wobei im ersten Schritt die zu analysierenden Dateien mittels Regex-Regeln nach auffälligen Mustern durchsucht werden. Um die Performance zu verbessern, hat sich das SAP-Team entschieden, die Hyperscan-Bibliothek von Intel zu verwenden. Hyperscan ist eine High-Performance Regex-Bibliothek, welche das Regex vorkompliiert und an die spezifische Mikroarchitektur der CPU anpasst.

Die Ausführungszeit konnte so um 80% im Vergleich zu einem klassischen Scan gesenkt werden. Der Nachteil von Hyperscan liegt in der stark eingeschränkten Komplexität der einsetzbaren Regex-Regeln. Diese dürfen beispielsweise keine langen „repetition-quantifier“, wie zum Beispiel „{1024}“ verwenden, da sonst die Komplexität für Hyperscan zu stark ansteigt.

Der Auflistung 5.1 lassen sich ausgewählte Suchmuster der Standardkonfiguration entnehmen. Eine Erweiterung um eigene Regeln lässt sich einfach vornehmen. Dabei gilt jedoch, dass jede „category“

6 [44]

7 [44]

8 [24]

nur einmal definiert werden darf. Falls das Muster zu komplex wird, bricht die Ausführung des Scans mit einem Fehler in der Hyperscan-Bibliothek ab.

```

1 rules:
2   - regex: -----BEGIN [A-Z]+ PRIVATE KEY( BLOCK)?-----
3     category: crypto_key
4     description: private key
5
6   - regex: EAACEdEose0cBA[0-9A-Za-z]+
7     category: token
8     description: fb access token

```

Listing 5.1 Auszug aus der Standardkonfiguration von Credential-Digger

Die Ergebnisse des ersten Schrittes werden entweder in einer SQLite- oder eine Postgres-Datenbank abgelegt. Falls die zugehörige Python-Bibliothek verwendet wird, stehen die Ergebnisse auch direkt zur Verfügung und lassen sich in weitere Ausgabeformate überführen.

Der interessante Teil geschieht im nächsten Schritt. Hier nutzt das SAP-Team maschinelles Lernen, um die gefundenen Treffer zu klassifizieren. Hierbei kommen drei unterschiedliche Modelle zum Einsatz, die bei der initialen Ausführung automatisch heruntergeladen werden. Eine zusätzliche Validierung bekannter Token über eine API des API-Providers findet nicht statt.

Das Path-Model analysiert, ob sich das gefundene Secret in einer Datei oder einem Verzeichnis befindet, das für falsch-positive Treffer bekannt ist. Dies betrifft unter Anderem README-Dateien, Dokumentationen oder Verzeichnisse, in denen Tests abgelegt sind. Eine Möglichkeit weitere Muster über entsprechende Konfigurationen hinzuzufügen ist nicht vorhanden.

Das Snippet-Model versucht den gefundenen Treffer im Kontext zu betrachten. Das heißt, es wird analysiert, ob das gefundene Secret im Kontext des umgebenden Programmcodes valide sein könnte.

Beim dritten Modell handelt es sich um das Similarity-Model. Bevor dieses zum Einsatz kommt, ist ein manuelles Review der Ergebnisse nötig. Hierbei werden Treffer manuell klassifiziert. Das Similarity-Model kann dann zukünftig auf Basis dieser Informationen ähnliche Secrets erkennen und entsprechend klassifizieren.⁹

Zusätzlich bietet der Credential-Digger die Möglichkeit eigene Modelle einzuarbeiten. Eine Einbindung über das Pre-Commit-Framework, siehe Kapitel 3.9, wird ebenfalls unterstützt. Auch ein Scan der History eines Repositories ist möglich.

⁹ [32]

5.3.2 TruffleHog

TruffleHog ist ein Produkt der Firma Truffle Security Co. um Dateien nach sensiblen Informationen zu durchsuchen. Es wurde im April 2022 in der Version 3 unter der Affero-General-Public-License (AGPL)-3 auf GitHub veröffentlicht.¹⁰ Das in der Programmiersprache Go geschriebene Programm kann in der Basisvariante kostenlos eingesetzt werden. Die kostenpflichtige Enterprise-Version bietet weiterführende Services wie ein Webportal oder automatische Updates an. Die Basisvariante verfügt über ein Command-Line-Interface (CLI) zur Bedienung.

Das GitHub-Repository¹¹ von TruffleHog verfügte im Oktober 2022 über mehr als 9700 Stars und 1200 Forks. Entsprechend der Analyse durch IsItMaintained sind zum Oktober 2022 8% der Issues offen. Der Median der Bearbeitungszeit eines Issues beträgt dabei 16 Tage und das Projekt zählt 59 Beitragende.¹² Diese Zahlen ,zusammen mit einem Unternehmen und einer kostenpflichtigen Version im Hintergrund, lassen eine längerfristige Lebensdauer des Projekts vermuten. Durch die kostenlose Basisvariante, welche sich großer Beliebtheit erfreut, scheint die Unterstützung des Projekts auch unabhängig von der Firma Truffle Security gesichert zu sein.

TruffleHog ist in der Lage über 700 verschiedene Typen von Keys anhand mitgelieferter Regex-Regeln zu identifizieren. Dies ist möglich, da eine Reihe von APIs über wohldefinierte Token verfügen. Das Projekt hat zusätzlich eine Anleitung veröffentlicht, wie sich zusätzliche Detektoren entwickeln und einbinden lassen.

Nach aktuellem Stand ist es allerdings nicht möglich, eine eigene Liste an Regex-Regeln in Kombination mit den vorgegebenen Regeln zu verwenden. Möglich ist lediglich ein Entweder-Oder-Betrieb.¹³ Dabei erfordert eine neue Secret-Definition die Implementierung eines Go-Packages, welches die Regex-Suche und die anschließende Validierung durchführt.¹⁴

TruffleHog richtet seinen Fokus auf wohldefinierte Ausdrücke, was sich auch darin zeigt, dass keine weiteren Filter, wie beispielsweise über Entropie verfügbar sind. Dafür bietet TruffleHog als einziger evaluerter Scanner die Möglichkeit, gefundene Token gegen die API der Provider auf Validität zu prüfen.

Dies sorgt somit insgesamt für eine geringe falsch-positiv Rate, allerdings auf Kosten der Flexibilität.

Der Scanner kann unterschiedliche Datenquellen wie Git-Verzeichnisse oder das Dateisystem evaluieren, wobei einzelne Verzeichnisse explizit von einem Scan ausgeschlossen werden können. Spezifische Datensätze lassen sich allerdings nicht im Rahmen einer Allowlist von den Suchergebnissen ausschließen.

10[3]

11[13]

12[24]

13[15]

14[16]

Als Ausgabeformat steht JSON zur Verfügung. Eine Einbindung in CI/CD-Pipelines ist möglich. Für das Scannen von Git-Repositories wird auch deren History in Betracht gezogen.

Des Weiteren bietet auch TruffleHog eine Anbindung an das Pre-Commit-Framework, siehe Kapitel 3.9 an.

5.3.3 Gitleaks

Gitleaks ist ein in der Programmiersprache Go entwickelter, CLI-basierter, quelloffener Secrets-Scanner, der unter der Massachusetts-Institute-of-Technology (MIT)-Lizenz über GitHub verfügbar ist.¹⁵

Seit August 2022 gibt es mit der Gitleaks LLC einen Ansprechpartner für kommerziellen Support. Auch soll so die weitere Entwicklung und Verbesserung der Anwendung garantiert werden.¹⁶

Positiv anmerken lässt sich, dass lediglich 2% aller Issues auf GitHub offen sind. Der Median der Bearbeitungszeit beträgt darüber hinaus nur einen Tag.¹⁷ Insgesamt kommt das Projekt mit 229 Beitragenden auf die mit Abstand höchste Anzahl aller evaluierten Scanner. Auf Grundlage der über 11.000 Stars auf GitHub lässt sich Gitleaks als derzeit beliebtester Secrets-Scanner am Markt bezeichnen.

Auch verschiedene kommerzielle Plattformen nutzen Gitleaks in ihren Produkten. So setzt GitLab in seinen unterschiedlichen Code-Scanning-Anwendungen auf Gitleaks, um nach Secrets zu suchen.¹⁸ Für seine Regex-basierte Suche bindet TruffleHog, vergleiche Kapitel 5.3.2, Gitleaks als Abhängigkeit ein.¹⁹

Neben Git-Repositories und deren History lassen sich generell beliebige Textdateien auf Secrets durchsuchen. Der Scanner kann dabei standardmäßig bereits über 140 verschiedene Token identifizieren.

Ein Vorteil von Gitleaks den Mitbewerbern gegenüber, ist die Möglichkeit, eine selbst definierte Liste von Regex-Regeln als Konfigurationsdatei einzupflegen, um nach spezifischen Mustern zu suchen.

Große Flexibilität bei der Anpassung der Suche bietet Gitleaks auch über eine Allowlist. Mit dieser können Verzeichnisse, Dateien und spezifische Datensätze von den Suchergebnissen ausgeschlossen werden. Eine Allowlist kann dabei sowohl auf globaler Ebene als auch spezifisch für eine Regel angelegt werden.

Neben einer rein auf Regex basierenden Suche lässt sich die Ergebnismenge über einen Entropiefilter

15[17]

16[19]

17[23]

18[18]

19[52]

5 Evaluation Secrets-Scanner

weiter einschränken. Weiterführendes zu Entropie und darauf basierenden Filtern kann Kapitel 3.6 entnommen werden.

Eine kurze Beispielkonfiguration einer Regel kann der Auflistung 5.2 entnommen werden.²⁰

Dabei wird eine Regel für die Suche nach fest einprogrammierten Credentials definiert. Das zugehörige Regex in Zeile 4 ist in der Syntax für die Programmiersprache Go geschrieben. Um die Ergebnisse später einordnen zu können, kann für jede Regel eine Reihe von Tags, siehe Zeile 5, definiert werden.

Die Beispielregel verfügt zusätzlich über einen Entropiefilter, der ein Ergebnis nur zurück liefert, falls der Wert der Entropie zwischen 3 und 7 liegt. Die Konfiguration für die Entropie kann den Zeilen 6-9 entnommen werden.

Ein Beispiel für eine lokale und eine globale Allowlist kann diesem Auszug in Zeile 10ff, beziehungsweise Zeile 14ff entnommen werden. Interessant ist hier das Regex in Zeile 17 für das Überspringen von Variablen-Substitution. Diese ist für eine große Anzahl falsch-positiver Treffer verantwortlich, da eine Variable, die beispielsweise \$myClass->_password lautet, auch ein valides Passwort darstellt.

```
1 # Keys
2 [[ rules ]]
3 description = "Hardcoded credentials in source-code files"
4 regex = '''(?i)(?:secret|password|pwd|pass|token)\s{0,10}(?:=|=|=>|:){1}(?:\s?)?
5      [\"''](. )[\"'']'''
6 tags = ["credentials", "hardcoded"]
7 [[ rules.Entropies ]]
8 Group = "1"
9 Max = "7"
10 Min = "3"
11 [rules.allowlist]
12 regexes = ['''(?i)!! enveloped:(?:\S)+''', '''(?:..) ?{{(?:..) ?}}''' ]
13
14 # Globals allowlists
15 [allowlist]
16 description = "Allowlisted files"
17 paths = ['''test ''', '''tests ''']           # Ignoring test directories
18 regexes = ['''$\{|\|(?:..)?(\}|\\)|\\x60(?:..)?\\x60''' ] # skip variable
19      substitution
```

Listing 5.2 Auszug aus einer Beispielkonfiguration von Gitleaks

Der Vorteil großer Flexibilität geht erfahrungsgemäß mit dem Nachteil einer großen Anzahl Treffern einher, von denen viele als falsch-positiv klassifiziert werden müssen. Dieses bekannte Problem konnte 2021 auch in einem internen Testlauf an der IU bestätigt werden. Andererseits konnte damit aber auch die Flexibilität bestätigt werden.

²⁰[14]

Mit Version 8.10.0 wurde ein neues zunächst experimentelles Feature hinzugefügt, welches jedem Treffer einen sogenannten „Fingerprint“ zuweist. Dieser Fingerprint kann in einer *.Gitleaksignore*-Datei eingetragen werden, sodass der Treffer für zukünftige Suchen direkt ignoriert wird. Ein Beispiel kann der Auflistung 5.3 entnommen werden.

```
1 "Fingerprint": "d1925d8548b86acfeb51912909f92756eb687412:weave--scope/scope.yaml:  
generic-api-key:140"
```

Listing 5.3 Beispielhafter Fingerprint zur eindeutigen Identifikation eines Treffers von Gitleaks

Der Fingerprint setzt sich aus der Commit-ID, dem Dateipfad und -namen, der Rule-ID, über welche der Treffer ausgelöst wurde, sowie der Startzeile des Treffers innerhalb der Datei zusammen. Ein Beispiel für diese Werte kann der Auflistung 5.4 entnommen werden, wobei sich der zusammengesetzte Fingerprint in Zeile 20 befindet.

An Ausgabeformaten steht neben JSON und CSV auch das Format SARIF zur Verfügung. SARIF ist ein Format, welches speziell für den Austausch der Ergebnisse statischer (Code-)Analysetools entwickelt wurde.²¹

Der Auflistung 5.4 lässt sich das Ergebnis eines Scans im JSON-Format entnehmen. Dieser wurde mit der Gitleaks-Version 8.15 erzeugt.

Eine Lösung zur Validierung von Token, wie es beispielsweise TruffleHog kann, bietet Gitleaks aktuell nicht an.

Wie TruffleHog ermöglicht auch Gitleaks eine Einbindung in die CI/CD-Pipeline über GitHub-Actions. Eine Verwendung innerhalb von GitLab ist, wie bereits erwähnt, über GitLab selbst möglich. Auch eine Einbindung in das Pre-Commit-Framework wird unterstützt. Für weitere Informationen hierzu sowie einem Beispiel auf Basis von Gitleaks kann Kapitel 3.9 eingesehen werden.

Während der Erstellung dieser Arbeit wurde an der IU diskutiert, inwiefern sich Gitleaks standardmäßig über Basic-Templates in alle CI/CD-Pipelines integrieren lassen könnte. Eine entsprechende Entscheidung steht allerdings bislang noch aus.

²¹[33]

5 Evaluation Secrets-Scanner

```
1  [
2  {
3      "Description": "Generic API Key",
4      "StartLine": 140,
5      "EndLine": 141,
6      "StartColumn": 8,
7      "EndColumn": 1,
8      "Match": "apiGroup: rbac.authorization.k8s.io",
9      "Secret": "rbac.authorization.k8s.io",
10     "File": "weave-scope/scope.yaml",
11     "SymlinkFile": "",
12     "Commit": "/ removed /",
13     "Entropy": 3.7230742,
14     "Author": "/ removed /",
15     "Email": "/ removed /",
16     "Date": "2021-09-10T15:57:37Z",
17     "Message": "Add weave",
18     "Tags": [],
19     "RuleID": "generic-api-key",
20     "Fingerprint": "<commit-id>:weave-scope/scope.yaml:generic-api-key:140"
21   }
22 ]
```

Listing 5.4 JSON-Ergebnis eines Gitleaks-Scans in Version 8.15

5.4 Praktischer Testlauf

Um eine praktische Einsicht in die Arbeitsweise und die Leistungsfähigkeit der drei untersuchten Scanner zu gewinnen, wurde ein entsprechender Test an einem Quellcode-Repository der IU durchgeführt. Alle drei Scanner wurden dabei ohne weitere Anpassungen an ihren Auslieferungskonfigurationen ausgeführt.

Dieses Repository umfasst mehr als 16.000 Codezeilen in aktuell 40 Zweigen aus über vier Jahren konstanter Weiterentwicklung. Aus Stichprobentests ist bereits bekannt, dass sich in diesem Repository einige potentielle Secrets befinden.

Aus Geheimhaltungsgründen ist es nicht möglich, die Ergebnisse des praktischen Tests zu veröffentlichen. Diese Entscheidung wurde unabhängig davon getroffen, dass die entsprechenden Secrets bereits ausgetauscht wurden. Im Folgenden werden nur die aggregierten Zahlen präsentiert, welche sich Tabelle 5.1 entnehmen lassen.

Die Zeile „Gefundene Secrets (gesamt)“ gibt die Gesamtanzahl aller gefundenen Secrets an. Der Credential-Digger weiß dabei eine extrem hohe Anzahl an Ergebnissen auf. Da der Scanner auf maschinellem Lernen zur Klassifizierung der Ergebnisse setzt, kann vermutet werden, dass aus diesem Grund im ersten Schritt möglichst viele Daten gesammelt werden. Dies beugt auch dem

Tabelle 5.1 Aggregierte Ergebnisse des praktischen Tests der untersuchten Secrets-Scanner

	Credential-Digger	TruffleHog	Gitleaks
Evaluierter Version	4.9.0	3.14.0	8.6.1
Gefundene Secrets (gesamt)	8368	41	95
davon eindeutig	33	11	31
davon falsch-positiv	3	3	0

Problem von wahr-negativen Ergebnissen, vergleiche Kapitel 3.7, vor.

Der Unterschied zur Spalte „davon eindeutig“ ergibt sich daraus, dass ein Secret an beliebig vielen Stellen im Code vorkommen kann. Auch kann es durch Abzweigungen im Repository dazu kommen, dass ein Secret mehrmals identifiziert wird. Der Credential-Digger ist der einzige evaluierte Scanner, der von Haus aus solche Mehrfachtreffer aggregiert. Für Gitleaks und TruffleHog wurde dieser Schritt für die Evaluation manuell durchgeführt.

Die Abweichung von TruffleHog zu den beiden anderen Scannern bei der Anzahl an eindeutigen Treffern ergibt sich daraus, dass TruffleHog standardmäßig nur nach API-Keys und Token sucht. Gitleaks und Credential-Scanner liefern zusätzlich auch Passwörter. Kapitel 3.5 liefert ein anschauliches Beispiel dafür wie sich ein solcher Treffer aufbauen kann.

Da sich die eindeutigen Treffer in einem ähnlichen Rahmen bewegen, ließe sich zunächst annehmen, dass die drei evaluierten Scanner auch ähnliche Ergebnisse zurückliefern. Tatsächlich gibt es aber nur zwei Datensätze, die von allen drei gefunden wurden. Ein API-Key von AWS und ein privater kryptographischer Schlüssel. Ein zweiter AWS-API-Key wurde von Credential-Digger ebenso wenig gefunden wie ein zweiter kryptographischer Schlüssel.

Bei insgesamt 51 gefundenen Passwörtern gab es nur eine einzige Überschneidung von Credential-Digger und Gitleaks. Bei Gitleaks liegt der Grund unter Anderem in der Entropie-Schwelle, die dazu führt, dass eigentlich legitime Zeichenketten ausgefiltert werden.

Bei Credential-Digger wurden ausschließlich Zeichenketten gefunden, die das Wort „PASSWORD“ enthalten.

Dies ist in beiden Fällen ein klares Zeichen dafür, dass die Standardkonfigurationen restriktiver ausgelegt sind, um falsch-positive Treffer einzudämmen.

Hier zeigt sich der Vorteil, mehr als eine SAST-Lösung für ein Problem einzusetzen.

5 Evaluation Secrets-Scanner

Tabelle 5.2 Auswertung der Secrets-Scanner hinsichtlich der Anforderungen aus Kapitel 4.2

Anforderung	Priorisierung	Credential-Digger	TruffleHog	Gitleaks
History-Scanning	Muss	X	X	X
Lokale Ausführung	Muss	X	X	X
Repository-Aktivität und Support	Muss	X	X	X
Maschinellweiterverarbeitbares Ausgabeformat	Muss	X Einbindung als Pythonbibliothek	X	X
Anpassungen von Anti-Pattern	Muss	X eingeschränkt durch Hyperscan	-	X
Umfang der Ausgabedaten	Muss	X	X	X
Einbindung in CI/CD-Pipelines	Soll	X	X	X
Pre-Commit-Hooks	Soll	X	X	X
Anpassungen der Suchpattern	Soll	X eingeschränkt durch Hyperscan	X Default- und Custom-Muster schließen sich aus	X
Einschränkungen des Scanumfangs	Soll	-	-	X
Validierung von Secrets	Kann	-	X	-
Art der Bereitstellung	Kann	X	X	X
Kommerzielle Einschränkungen	Kann	-	-	-

5.5 Entscheidungsgrundlage für Gitleaks

Ziel der Evaluation ist es eine Grundlage für die Entscheidung zu schaffen, welcher Secrets-Scanner für die weitere Entwicklung herangezogen wird. Die Anforderungen, denen die Analyse zu Grunde liegt, können Kapitel 4.2 entnommen werden.

Der Tabelle 5.2 lässt sich eine tabellarische Zusammenfassung der Ergebnisse der Anforderungsanalyse an die drei untersuchten Secrets-Scanner Credential-Digger, TruffleHog und Gitleaks entnehmen.

Die Details können dabei den jeweiligen Kapiteln, in denen die Anwendungen vorgestellt wurden, entnommen werden.

Alle drei evaluierten Scanner bieten verschiedene Vorteile und Nachteile.

So sind alle drei Scanner in der Lage die History eines Git-Repositories zu scannen, können lokal ausgeführt werden und lassen davon ausgehen, dass sie auch mittelfristig noch aktiv weiterentwickelt werden. Der Credential-Digger fällt hier allerdings mit seinem Fokus auf SAP negativ ab.

Beim maschinellweiterverarbeitbaren Ausgabeformat bietet Gitleaks die meisten Optionen,

TruffleHog unterstützt immerhin noch das verbreitete JSON, welches für den Einsatz an der IU ausreichend wäre. Der Credential-Digger bietet keine direkte Möglichkeit, allerdings kann die Ausgabe als interner Datentyp in Python abgegriffen werden, wenn das Programm als Bibliothek verwendet wird. Dies wäre nach Einschätzung durch das ISS-Team ausreichend.

Alle drei Scanner liefern in ihrer Ausgabe ausreichend Informationen mit, um die Daten weiterverarbeiten zu können.

Ebenfalls ermöglichen alle drei Scanner eine Einbindung in die CI/CD-Pipeline sowie einen Einsatz im Rahmen des Pre-Commit-Frameworks.

Ein großer Unterschied zeigt sich erstmalig bei der Anpassbarkeit der Suchpattern. Hier kann lediglich Gitleaks über seine Konfigurationsdatei voll überzeugen. TruffleHog bietet keine Möglichkeit die Default-Konfiguration mit kundenspezifischen Mustern zu ergänzen, sondern erlaubt lediglich einen Entweder-Oder-Betrieb. Darüber hinaus müssen diese Ergänzungen als Go-Klassen programmiert werden. Credential-Digger arbeitet nach einem ähnlichen Prinzip wie Gitleaks und würde theoretisch eine einfache Anpassung über eine Regex-Datei erlauben. Praktisch sind die möglichen Regex-Muster aber durch die Limitierungen der Intel Hyperscan-Bibliothek begrenzt.

Im Bereich der Kann-Anforderungen können alle drei Scanner über die Art der Bereitstellung punkten. Alle Scanner sind über GitHub im Quellcode verfügbar, außerdem stehen vorkompilierte Binaries beziehungsweise DockerFiles zur Verfügung.

Hier hat TruffleHog mit der Möglichkeit, einen Treffer auch automatisch zu validieren, einen Vorteil den beiden anderen Scannern gegenüber.

Relevante kommerzielle Einschränkungen gibt es nicht. Credential-Digger und Gitleaks sind beide vollumfänglich open-source und kostenlos, TruffleHog bietet in der kostenfreien Variante über die CLI alles, was vom ISS-Team zum aktuellen Zeitpunkt benötigte wurde.

Unter Einbeziehung der Ergebnisse aus dem praktischen Teil der Evaluation aus Kapitel 5.4 kann Gitleaks in Summe die meisten Vorteile bieten. Im Hinblick auf die vielfältige Softwarelandschaft der IU bietet Gitleaks mit seiner Konfigurierbarkeit damit die größte Flexibilität.

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

Wie in Kapitel 3.4 dargelegt, kann im Rahmen der statischen Codeanalyse nicht auf eine manuelle Prüfung der Scan-Ergebnisse verzichtet werden. Das Ziel ist es somit, durch verbesserte Regeln und Filterung auf Basis bereits identifizierter falsch-positiver Treffer die Menge an manuell zu prüfenden Ergebnissen möglichst gering zu halten.

Aus diesem Grund wird die prototypische Implementierung eines solchen Programms zur Ergebnisauswertung im folgenden Kapitel aufgezeigt. Nach aktuellem Stand der Recherche gibt es kein verfügbares Programm, das eine solche Verarbeitung von Scan-Ergebnissen ermöglicht. Der in Kapitel 5.3.1 vorgestellte Secrets-Scanner der SAP bietet eine ähnliche Möglichkeit im Rahmen maschinellen Lernens. Allerdings kann er aus anderen Gründen die Anforderungen der IU nicht erfüllen, siehe hierzu Kapitel 5.5.

Als Basis für die prototypische Implementierung dient dabei Gitleaks, das nach Durchführung des Evaluierungsprozesses, der für die Anforderungen der IU am besten geeignete aktuell verfügbare Secrets-Scanner ist. Weitere Informationen zum Evaluierungsprozess können Kapitel 4 entnommen werden.

Sowohl der Prototyp als auch eine geplante Weiterentwicklung sind aktuell nur für den internen Einsatz an der IU gedacht. Dies bietet den Vorteil, dass die Entwicklung zielgerichtet auf die speziellen internen Anforderungen ausgerichtet werden kann.

6.1 Kernanforderungen

Das Programm soll in der Lage sein, die von Gitleaks ausgegebenen Scan-Ergebnisse einzulesen und grafisch ansprechend zur weiteren manuellen Verarbeitung zur Verfügung zu stellen. Dies umfasst insbesondere den Teil des Reviews, ob es sich bei einem Treffer um einen falsch-positiven oder wahr-positiven handelt. Die Ergebnisse sollen hierzu entsprechend markiert werden können. Um eine spätere Nachvollziehbarkeit der Entscheidung zu gewährleisten, muss für jede Klassifizierung eine Begründung sowie ein entsprechender Zeitstempel abgespeichert werden. Da aufgrund der fehlenden Benutzerverwaltung aktuell nur ein Benutzer im Einsatz ist, erübrigt sich für den Prototypen die Abspeicherung des verantwortlichen Reviewers.

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

Auf Basis eines eindeutigen Kennzeichens, wie dem Fingerprint von Gitleaks, soll die Anwendung automatisch alle Treffer aus der Ergebnismenge für ein Review ausschließen, die bereits in der Datenbank vorhanden sind.

Des Weiteren ist die Performance der Anwendung von besonderer Bedeutung, da die zu verarbeitende Menge an Ergebnissen vor allem zu Beginn des Einsatzes von Gitleaks sehr umfangreich ausfallen wird. Weiteres kann hierzu Kapitel 6.7 entnommen werden.

Das eingesetzte Datenmodell soll sowohl flexibel an neue Varianten des Formates der Ergebnisse von Gitleaks angepasst werden können, als auch die Ergebnisse weiterer Secrets-Scanner abbilden können. Aktuell gibt es allerdings noch keine konkreten Pläne für eine Ausweitung auf weitere Secrets-Scanner.

Um die Weiterverarbeitung eines Ergebnisses zu unterstützen, soll das Review-Programm eine Möglichkeit bieten, Ergebnisse zu markieren und somit auf Wiedervorlage legen zu können.

6.2 Prozessablauf zur Auswertung

Eine grafische Darstellung der Einzelschritte des Prozesses zur Auswertung der Scan-Ergebnisse kann Abbildung 6.1 entnommen werden. Im Prozessschritt 1 werden die Ergebnisse eines Gitleaks-Scans durch die Anwendung eingelesen. Hierzu stellt das Backend, vergleiche Kapitel 6.6, ein CLI zur Verfügung. Auch über die Importfunktion des Frontends, vergleiche Kapitel 6.5, ist dies möglich.

Anschließend erfolgt im Prozessschritt 2 ein Abgleich mit den bereits verarbeiteten Daten aus der Datenbank. Dies erfolgt ausschließlich im Backend, da nur dieses mit der Datenbank kommunizieren kann. Technisch erfolgt dieser Abgleich über den Fingerprint, den Gitleaks für jeden Treffer zur eindeutigen Identifikation generiert. Weitere Informationen zum Aufbau des Fingerprints und der dafür benötigten minimalen Gitleaks-Version können Kapitel 5.3.3 entnommen werden.

Bereits identifizierte Treffer werden dabei im Prozessschritt 3 direkt beim Import verworfen, um die Datenmenge für das Review klein zu halten.

Bevor die Daten anschließend in der Datenbank abgelegt werden, werden sie um Metadaten angereichert. Diese sind nötig, um mit den Datensätzen später arbeiten zu können, siehe hierzu auch Kapitel 6.4. Dies entspricht dem Schritt 4 in der Abbildung.

Für Prozessschritt 5, der Benachrichtigung des Reviewers, gibt es im Rahmen des Prototypen aktuell noch keine Umsetzung. Geplant ist hier ein eigener Service im Backend, der eine Integration in Atlassian Jira sowie das Versenden von E-Mails unterstützt. Weiteres kann Kapitel 6.8 entnommen werden.

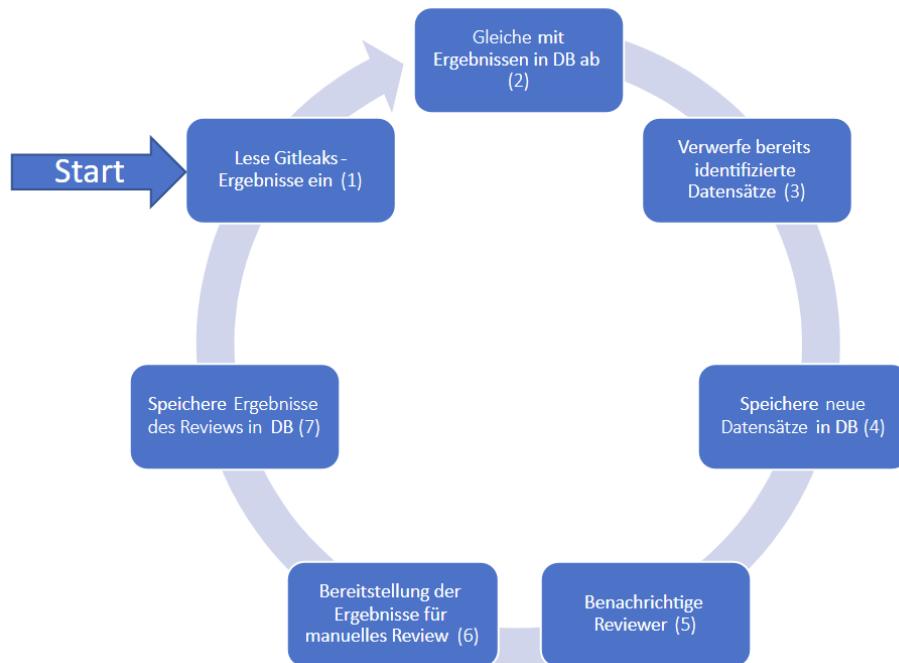


Abbildung 6.1 Grafische Darstellung des Prozessablaufs für das Ergebnis-Review, eigene Darstellung

Über die API, Kapitel 6.6, werden anschließend die Ergebnisse im Schritt 6 für das Review zur Verfügung gestellt. Hier kommt das Frontend ins Spiel, welches diese Daten aufgreift und eine grafische Oberfläche für die Bearbeitung zur Verfügung stellt.

Der finale 7. Prozessschritt erfolgt nach der Bearbeitung im Frontend. Die Daten werden an das Backend zurückgeschickt und nach erfolgreicher syntaktischer Prüfung in der Datenbank abgelegt.

Damit stehen die neuen Daten für einen erneuten Durchlauf des Prozesses zur Verfügung.

6.3 Architektur- und Technikübersicht

Die Systemarchitektur der Anwendung setzt sich aus den drei Komponenten Frontend, Backend und Datenbank zusammen. Frontend und Backend sollen hierbei über eine Restful-API kommunizieren. Die Daten werden in einer NoSQL-Datenbank abgelegt. Um die Integrität und Vertraulichkeit der Daten zu schützen, soll die Datenbank verschlüsselt werden können. Eine Übersicht der Architektur kann der Abbildung 6.2 entnommen werden.

Die API sowie das zugehörige Backend wurden in Python, basierend auf dem FastAPI-Framework¹ geschrieben. Dieses Framework wurde Ende 2018 vorgestellt und zielt auf hohe

¹ [12]

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

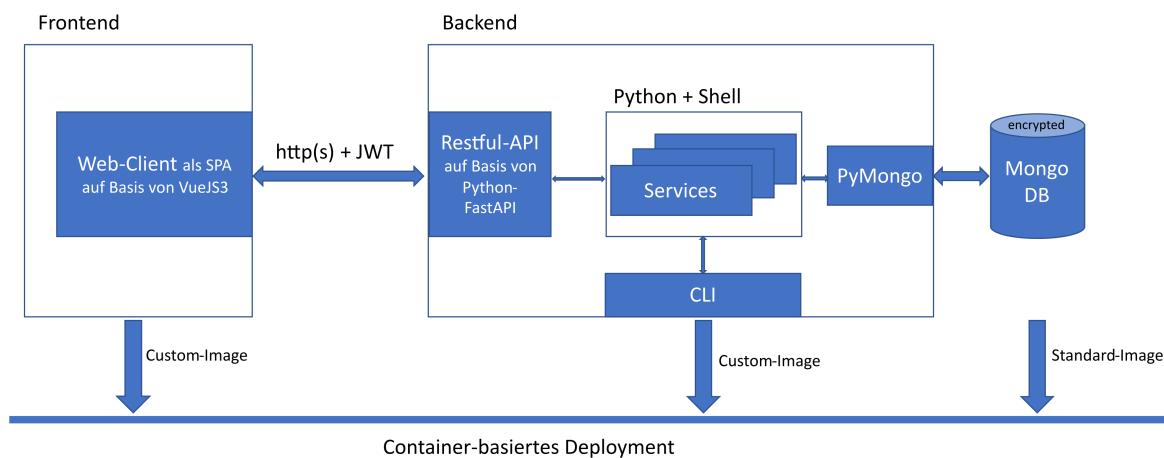


Abbildung 6.2 Architekturübersicht der prototypischen Implementierung, eigene Darstellung

Ausführungsgeschwindigkeit ab. Das Projekt ist quelloffen sowie unter der MIT-Lizenz verfügbar. Neben einer übersichtlichen Dokumentation und einer aktiven Gemeinschaft, bietet es die Möglichkeit ohne zusätzlichen Aufwand eine Swagger-Dokumentation zu generieren.

Das Projekt wird auf GitHub gehostet und zählte zu Beginn der Implementierung circa 54.000 Stars. Die zum Zeitpunkt der Implementierung aktuelle Version war 0.89.1 und wird, wie die Zahlen von IsItMaintained belegen, aktiv weiterentwickelt. Aktuell sind 31 % der Issues offen, die Bearbeitungszeit beträgt dabei 24 Tage.²

Das Frontend wird als Single-Page-Application in VueJS-3³ geschrieben. Dieses Framework wird an der IU für die Entwicklung aller Frontends eingesetzt, was letztendlich der Grund für die Entscheidung war, den Prototypen ebenfalls mit VueJS-3 zu entwickeln.

Da Gitleaks die Ergebnisse unter Anderem im JSON-Format zur Verfügung stellt und sowohl JavaScript als auch Python dieses Format nativ unterstützen, wurde beschlossen, eine NoSQL-Datenbank einzusetzen, womit eine Konvertierung der Ergebnisse entfällt. Hier fiel die Wahl aus den selben Gründen wie bei VueJS auf MongoDB.⁴

Das Deployment der Anwendung erfolgt containerbasiert mittels bereitgestellter Images für Frontend, Backend und die Datenbankumgebung sowie einem entsprechenden Compose-Skript für Docker beziehungsweise Podman.

Da die Anwendung auf einem Ubuntu-basierten Server zum Einsatz kommen wird, müssen keine plattformübergreifenden Anforderungen berücksichtigt werden.

2 [51]

3 [22]

4 [38]

Die gewählten Technologien versprechen eine zeitgemäße und performante Umsetzung, die eine interne Weiterentwicklung in Zukunft ohne großen Einarbeitungsaufwand unterstützen.

Im Rahmen dieser Arbeit wurde eine prototypische Umsetzung der vorgestellten Architektur durchgeführt.

6.4 Datenmodell

Nach dem Einlesen der Datensätze, die von Gitleaks produziert wurden, erfolgt ein Abgleich jedes Ergebnisses mit den Ergebnissen, die bereits in der Datenbank abgelegt wurden. Damit sollen alle Ergebnisse, die bereits in der Datenbank existieren automatisch ausgefiltert werden. Zum manuelle Review werden folglich nur neue Ergebnisse weitergereicht.

Aus diesem Grund liegt ein besonderer Fokus auf dem Entwurf eines passenden und langfristig nutzbaren Datenformats. Die einzelnen Felder des Datenmodells mit einer Beschreibung können der Tabelle 6.1 entnommen werden.

Da Gitleaks nur Daten zu den jeweiligen Treffern an sich abspeichert, werden die originalen Gitleaks-Ergebnisse um eine Reihe von Metadaten angereichert, bevor sie in die Datenbank geschrieben werden.

Diese Metadaten umfassen diverse Informationen zum Scavorgang wie Start- und Endzeit sowie den Namen des gescannten Repositories.

Zu berücksichtigen ist auch, dass in Zukunft Gitleaks durch einen weiteren Scanner ergänzt werden könnte, weswegen grundsätzliche Informationen zum Scanner selbst mit abgespeichert werden müssen. Auch die Version des eingesetzten Scanners wird in den Metadaten abgelegt, sodass auf Änderungen im Datenmodell von Gitleaks reagiert werden kann.

Das Ergebnis der Klassifizierung wird bei jedem Datensatz direkt hinterlegt. Hier ist immer eine Begründung erforderlich, um die Entscheidung später nachvollziehen zu können. Zusätzlich wird das Änderungsdatum mitgeschrieben.

Hiermit werden die entsprechenden Anforderungen aus Kapitel 6.1 umgesetzt.

Auflistung 6.1 kann ein Auszug aus dem JSON des Datenmodells entnommen werden. Zu Beginn stehen die Metadaten zum jeweiligen Scan. Diese umfassen, neben den bereits erwähnten Eigenschaften, auch eine ID zur eindeutigen Identifikation des Treffers in der Datenbank.

Unter „resultRaw“ ist das Gitleaks-Ergebnis im Original abgelegt. Ein vollständiges Ergebnis kann Auflistung 5.4 entnommen werden. Eine solche Kapselung ist nötig, da kein Einfluss auf den Aufbau des Datenmodells von Gitleaks besteht. Falls dieses sich in Zukunft ändert, kann es ohne großen Aufwand und in Abhängigkeit der in den Metadaten hinterlegten Scanner-Version auf das neue Format erweitert werden.

Die Informationen zur Klassifizierung des Treffers sind unter „falsePositive“ gespeichert. Um später nachvollziehen zu können, warum eine bestimmte Klassifizierung vorgenommen wurde, kann ein Datensatz nur aktualisiert werden, wenn auch eine entsprechende Begründung beigelegt wird.

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

Jeder Eintrag in diesem Datenmodell ist dabei ein Pflichteintrag. Neue Ergebnisse werden als „initial“ markiert. Da aufgrund der nicht vertrauenswürdigen Ergebnisse eines Scans ein manuelles Review nötig ist, wurde entschieden, initiale Ergebnisse zunächst pauschal als Falsch-Positiv ein zu sortieren. Hierbei werden eventuelle manuelle Einträge im Datenmodell überschrieben, um einen konsistenten initialen Zustand zu erhalten. Die Markierung als initial dient zusätzlich der Vorbeugung, dass einzelne Treffer beim Review vergessen werden können.

Ein solcher Datenbankeintrag ist etwa zwei Kilobyte groß und lässt sich mit einem beliebigen Texteditor lesen und gegebenenfalls editieren.

```
1  {
2      "_id": "63a03d0b461bc420de4e638c",
3      "scannerType": 0,
4      "scannerVersion": "8.15",
5      "inputType": 0,
6      "repositoryPath": "https://github.com/...",
7      "repositoryName": "applications",
8      "scanStartTime": "2022-12-16T00:00:00",
9      "scanEndTime": "2022-12-16T00:00:00",
10     "save_date": "2022-12-19T11:29:31.810344",
11     "resultRaw": {
12         / ... Gitleaks Scan-Result ... /
13     },
14     "falsePositive": {
15         "isFalsePositive": false,
16         "justification": "4523",
17         "change_date": "2023-01-27T16:37:59.375471"
18     },
19     "isFavourite": true
20 }
```

Listing 6.1 Datenbankeintrag eines Scan-Treffers, angereichert mit Metadaten (Auszug)

6.5 Frontend

Ein Ziel, das mit der Entwicklung dieser Anwendung erreicht werden soll, ist es, dem Reviewer eine Anwendung an die Hand zu geben, die ihn bei seiner Arbeit unterstützt. Die ersten Repository-Scans wurden dabei noch manuell in einem Texteditor durchsucht und potentielle wahr-positive Treffer zur weiteren Überprüfung in eine Excel-Tabelle übertragen. Je nach Größe des Repositories und der Qualität der eingesetzten Regex-Regeln mussten so teilweise tausende Treffer einzeln analysiert werden. Es ist offensichtlich, dass dieser Prozess fehleranfällig, schlecht nachvollziehbar und langsam ist und somit zu Frustration beim Reviewer führt.

Da eine manuelle Nacharbeit im Kontext statischer Codeanalyse, wie in Kapitel 3.4 ausgeführt,

Tabelle 6.1 Übersicht der Felder des Ergebnis-Datenmodells

Objekt	Feld	Beschreibung
metadata	id	automatisch generierte Identifikationsnummer
	scannerType	aktuelle Auswahl Gitleaks
	scannerVersion	Scanner-Version, mit der der Scan durchgeführt wurde
	inputType	neben dem API-Upload können Daten auch direkt über das CLI in die Datenbank geladen werden
	repositoryPath	lokaler Pfad, beziehungsweise URL zum Repository
	repositoryName	Name des Repositories
	scanStartTime	Startzeit des Scansvorgangs
	scanEndtime	Endzeit des Scansvorgangs
	save_date	Datum des Uploads
resultRaw	isFavourite	Kennzeichen, ob der Treffer als Favorit markiert wurde
	eingebettetes Objekt	Ausgabe von Gitleaks
	isFalsePositive	Klassifizierung des Treffers
falsePositive	justification	Begründung der Klassifizierung
	change_date	Änderungsdatum der Klassifizierung

immer nötig ist, soll versucht werden, diese Aufgabe für den Reviewer möglichst angenehm zu gestalten. Hierzu wurde über eine Single-Page-Application ein modernes Webfrontend geschaffen, das die Ergebnisse für jedes Repository in einzelne Objekte zerlegt und diese übersichtlich in Listen anordnet.

Zunächst werden alle verfügbaren Repositories in einer Liste dargestellt. Diese Auflistung ist in Abbildung 6.3 unter dem Punkt „Repository List“ als Screenshot aus dem Frontend dargestellt. Die Liste umfasst neben dem Zeitpunkt des letzten Scans auch eine Reihe von Kennzahlen für das Repository. Diese umfassen die Anzahl an wahr-positiven und falsch-positiven Treffern sowie die Anzahl initialer Datensätze, also wie viele noch nicht bearbeitet wurden. Diese sind in der Abbildung 6.3 unter dem Punkt „Overview“ abgebildet.

Zu jedem Repository wird in der Auflistung unter der Spalte „Numbers“, violett eingerahmt, zusätzlich eine Übersicht über die noch zu bearbeitenden Einträge, die wahr-positiven Treffer sowie die falsch-positiven Treffer bereitgestellt. Als zusätzliches Feature wird, sofern die Daten an dieser Stelle korrekt eingetragen wurden, ein Absprung zum Hoster des Repositories ermöglicht. Diese Eigenschaft beruht auf der Erfahrung, dass es oftmals nötig ist, den Kontext in der Quelldatei um einen Treffer herum einzusehen, um eine Klassifizierung vornehmen zu können. Während des manuellen Reviewprozesses musste hierzu aufwendig über die Commit-ID im Repository nach dem

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

The screenshot shows the Gitleaks Review Frontend interface. At the top, a header bar displays "Welcome to Gitleaks Review Frontend". Below it, a purple navigation bar contains the word "Overview". The main content area has five cards: "Documents total 18029", "Repositories total 22", "False Positives total 18026", "True Positives total 3", and "Initial Values total 18021". A second purple navigation bar labeled "Repository List" is shown below. On the right, there is a search bar with the placeholder "Search Repository". The main table lists two repositories: "all-contributors" (Scanner Information: Gitleaks 8.15, Date: January 5th 2023, 00:00:00) and "amplify-js" (Scanner Information: Gitleaks 8.15, Date: January 5th 2024, 00:00:00). Each row includes a "Numbers" section with three colored circles (purple, orange, blue) and their corresponding counts (e.g., initial: 1154, true: 0, false: 1155 for all-contributors). To the right of the table, a "Details" column contains buttons for "Manual Import" (highlighted with a yellow box), "Edit", and "Delete".

Abbildung 6.3 Ansicht der Repositories im Frontend des Review-Programms, eigene Darstellung

Datensatz gesucht werden. In der Abbildung 6.3 ist der Absprung gelb eingerahmt. Der Platzhalter „Manual Import“ stellt den Default-Zustand ohne Verlinkung dar.

Um im nächsten Schritt eine Bearbeitung vornehmen zu können, bietet jeder Eintrag eine detaillierte Auflistung aller Treffer an. Abbildung 6.4 zeigt die Umsetzung im Frontend.

Dieser Bereich stellt den zentralen Arbeitsbereich für das Review dar. Die Auflistung der Treffer erfolgt hierzu zunächst nach Commit-ID sortiert und innerhalb eines Commits alphabetisch. Der Reviewer kann auf einen Blick den Dateinamen in der Spalte „File Name“ sowie den zugehörigen Treffer in der Spalte „Match“ einsehen. Über Tooltips, violett eingerahmt, können weitere Informationen, wie die Rule-ID, über welche der Treffer identifiziert wurde, eingesehen werden. Jeder Eintrag enthält Informationen zu seinem Status in der Spalte „Current Status“.

Um einen Treffer verifizieren oder falsifizieren zu können, kann über ein Pop-Up, abgebildet in Abbildung 6.5, das „resultRaw“, also das Gitleaks-Ergebnis im Detail, eingesehen werden. Um das Ergebnis der Analyse anschließend eintragen zu können, kann entweder ein Pop-Up zur Bearbeitung geöffnet werden oder über die Massenänderungsoption, grün eingerahmt, mehrere Einträge auf einmal bearbeitet werden. Das Pop-Up lässt sich der Abbildung 6.6 entnehmen. Die Massenänderung ist nötig, da bei statischer Codeanalyse oftmals sehr viele Daten anfallen. Für weitere Informationen hierzu siehe Kapitel 3.4 sowie die Tabelle 5.1.

Um einzelne Treffer später wieder aufgreifen zu können, beispielsweise weil erst Rückfragen geklärt werden müssen bevor ein Ergebnis eingetragen werden kann, lässt sich jeder Datensatz als Favorit markieren. Über eine dedizierte Seite können diese direkt eingesehen werden. Diese

The screenshot shows the Gitleaks Review Frontend interface. At the top, a header bar displays "Welcome to Gitleaks Review Frontend" and "REPOSITORY: DOCS". Below this is a purple navigation bar with the text "Overview". The main content area has a white background with several data cards:

- Documents**: total 192
- False-Positives**: total 192
- True-Positives**: total 0
- Initial Values**: total 191
- Date of Last Scan**: Timestamp January 5th 2023, 00:00:00

Below these cards is a section titled "Findings List" with a search bar. A green-bordered button labeled "Bulk Edit Status" is visible. The findings table has columns: File Name, Match, Current Status, Save Date, and Details. One entry is highlighted with a purple box around the "Match" column value "generic-api-key".

File Name	Match	Current Status	Save Date	Details
indexer.md	key": "YXNzZXRFY29lZmZpY2llbnQ=""	False Positive	January 5th 2023, 10:09:50	▼ RAW STATUS
indexer.md	generic-api-key		January 5th 2023, 10:11:56	

Abbildung 6.4 Ansicht der einzelnen Treffer pro Repository im Frontend des Review-Programms, eigene Darstellung

This screenshot shows a detailed view of a finding. A modal window displays a JSON object representing the finding's metadata. The object includes fields such as Description, Startline, Endline, StartColumn, EndColumn, Match, Secret, File, Commit, Entropy, Author, Email, Date, Message, Tags, RuleID, and Fingerprint.

```
{
  "Description": "Generic API Key",
  "Startline": 2413,
  "Endline": 2413,
  "StartColumn": 22,
  "EndColumn": 53,
  "Match": "key": \"YXNzZXRFY29lZmZpY2llbnQ=\"",
  "Secret": "YXNzZXRFY29lZmZpY2llbnQ=",
  "File": "docs/get-details/indexer.md",
  "Commit": "05786364e07031c48dc1d75fda9394231e6de3a6",
  "Entropy": 3.8239348,
  "Author": "Liz Baran",
  "Email": "45209375+ejbaran@users.noreply.github.com",
  "Date": "2021-09-28T00:09:37+00:00",
  "Message": "Formatting fixes to indexer and ATs.",
  "Tags": [],
  "RuleID": "generic-api-key",
  "Fingerprint": "05786364e07031c48dc1d75fda9394231e6de3a6:docs/get-details/indexer.md:generic-api-key:2413"
}
```

A "CLOSE" button is located at the bottom right of the modal.

Abbildung 6.5 Ansicht des Ergebnisses von Gitleaks zu einem Treffer im Frontend des Review-Programms, eigene Darstellung

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

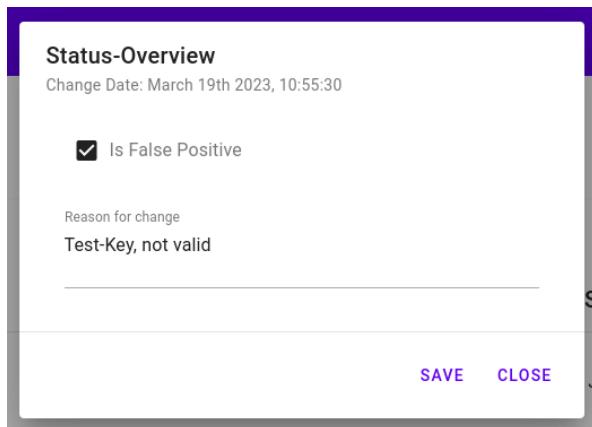


Abbildung 6.6 Pop-Up zur Klassifikation des Treffers im Frontend des Review-Programms, eigene Darstellung

Funktion ist in Abbildung 6.4 links über den Stern, braun eingerahmt, in der Spalte „File Name“ abgebildet.

Über das Pop-Up zur Bearbeitung sowie der Möglichkeit der Markierung als Favorit, wurden die entsprechenden Anforderungen aus Kapitel 6.1 umgesetzt.

Zur Unterstützung der Ergebnisauswertung werden verschiedene Kennzahlen über die entsprechenden Funktionen der Datenbank ermittelt und angezeigt. Aktuelle Kennzahlen sind die Gesamtanzahl aller Dokumente, die Anzahl aller wahr-positiven Treffer, beziehungsweise aller falsch-positiven Treffer, die Anzahl noch nicht bearbeiteter Treffer sowie das Datum des letzten Scan-Laufes.

Die Umsetzung lässt sich der Abbildung 6.3 für alle Repositories sowie der Abbildung 6.4, bezogen auf ein einzelnes Repository, jeweils unter dem Punkt „Overview“ entnehmen.

Zusätzlich bietet das Frontend eine Importfunktion, über welche sich die Ergebnisse neuer Scans hochladen lassen. Die Eingabemaske unterstützt den Reviewer dabei, die Metadaten für den Scan korrekt einzugeben. Diese Maske ist in Abbildung 6.7 dargestellt.

Anschließend wird ein Feedback, abgebildet in Abbildung 6.8, vom Backend-Service übermittelt, wie viele der hochgeladenen Treffer bereits in der Datenbank existieren und wie viele neu aufgenommen wurden.

Alternativ kann ein Upload der Daten über ein CLI im Backend durchgeführt werden. Dies entspricht dem Schritt 1 in der Abbildung 6.1.

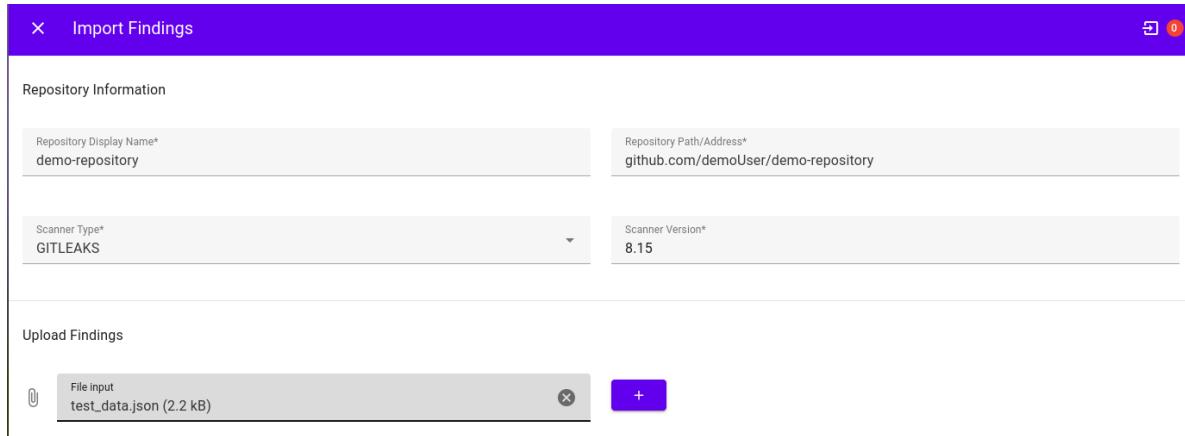


Abbildung 6.7 Assistent zum Import neuer Scan-Ergebnisse im Frontend, eigene Darstellung

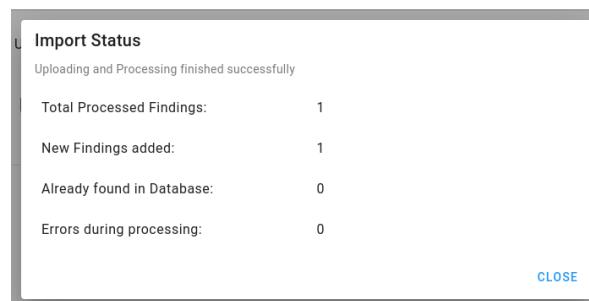


Abbildung 6.8 Rückmeldung zum Status des Imports neuer Scan-Ergebnisse im Frontend, eigene Darstellung

6.6 API und Backend

Das Gegenstück zum Frontend bildet das Backend. Für dieses wurde mit dem Python-Framework FastAPI eine Lösung ausgewählt, welche stark auf Ausführungsgeschwindigkeit optimiert wurde. Auch erfreut sich Python einer großen Beliebtheit, nicht nur im IT-Sicherheitsbereich, sondern auch darüber hinaus.

Das Backend selbst besteht, wie in Abbildung 6.2 dargestellt, aus verschiedenen Services und einem CLI. Auch die Datenbank lässt sich im weiteren Sinne zum Backend zählen.

Die Services umfassen aktuell die Logik zur Verarbeitung der Eingabe- und Ausgabedaten, die Kommunikation mit der Datenbank sowie die Logik, welche hinter der Restful-API steckt. Über direkten Zugriff auf dem Backend-Server steht des Weiteren ein Shell-Skript zur Verfügung, um Gitleaks automatisiert über alle Quellcode-Repositories der IU laufen zu lassen. Dieses Skript ist allerdings kein direkter Bestandteil dieser Arbeit, sondern wurde bereits davor angefertigt.

Eine Dokumentation der API-Endpunkte mittels des Open-Source Werkzeuges Swagger konnte auf Basis der integrierten Funktion von FastAPI unkompliziert erstellt werden.

Für den Bereich der Treffer („Findings“) wurden insgesamt 12 Endpunkte implementiert. Eine Übersicht aus der Swagger-Dokumentation kann der Abbildung 6.9 entnommen werden.

Für die Umsetzung weiterer Funktionalitäten wie dem Login und dem Datei-Upload des Prototypen mussten insgesamt neun weitere Endpunkte umgesetzt werden. Ein Auszug aus der entsprechenden Swagger-Dokumentation kann der Abbildung 6.10 entnommen werden.

Zur Verwaltung externer Zugriffe ist die API über einen JSON-Web-Token (JWT) abgesichert. Um einen JWT einsetzen zu können, musste eine einfache Benutzerverwaltung entworfen werden. Diese wurde für den Prototypen zunächst auf das Nötigste reduziert und bietet somit keine Verwaltungsmöglichkeiten wie Ändern des Passworts oder Neuerstellung des Benutzers. Dies ist aktuell nur über direkten Zugriff auf die Datenbank möglich. Der Swagger-Dokumentation aus Abbildung 6.10 lassen sich die entsprechenden Endpunkte entnehmen. Der Endpunkt zum „User-Signup“ ist zum aktuellen Zeitpunkt nicht funktional.

Ein erwähnenswerter Vorteil von FastAPI ist die Möglichkeit, Request-Bodies aus Anfragen an die API automatisch über die Modelle, welche das Schema der Datenbank bilden, prüfen zu lassen. Dabei generiert FastAPI selbstständig Fehlermeldungen und sendet eine passende Antwort an den Aufrufer zurück.

Ein entsprechender Auszug aus der Schema-Definition kann der Abbildung 6.11 entnommen werden.

Damit konnte die Umsetzung des Prototypen deutlich beschleunigt und mehr Zeit in die Umsetzung des Frontends investiert werden.

Finding

<code>GET</code>	<code>/finding/count</code>	Get Finding Overview Count	
<code>GET</code>	<code>/finding/repository/{repository_id}/count</code>	Get Finding Overview Count	
<code>GET</code>	<code>/finding/overview</code>	Get Finding Overview	
<code>GET</code>	<code>/finding/favourites</code>	Get All Favourite Findings	
<code>GET</code>	<code>/finding/true_positives</code>	Get All Favourite Findings	
<code>GET</code>	<code>/finding/</code>	Get All Findings	
<code>GET</code>	<code>/finding/{finding_id}</code>	Get Single Finding	
<code>GET</code>	<code>/finding/repository/{repository_id}</code>	Get Repository Findings	
<code>PUT</code>	<code>/finding/{finding_id}/fp</code>	Put False Positive	
<code>PUT</code>	<code>/finding/{finding_id}/fav</code>	Put Favourite	
<code>POST</code>	<code>/finding/raw_upload</code>	Post Findings Raw	
<code>POST</code>	<code>/finding/file_upload</code>	Post Finding File	

Abbildung 6.9 Auszug aus der Swagger-Dokumentation zu den Treffer-Endpunkten der API, eigene Darstellung

User

<code>GET</code>	<code>/user/{user_id}</code>	Get User	
<code>GET</code>	<code>/user/</code>	Get All User	
<code>POST</code>	<code>/user/signup</code>	Create User	

Login

<code>GET</code>	<code>/token/</code>	Get Current User	
<code>POST</code>	<code>/token/</code>	Get Access Token	

Miscellaneous

<code>GET</code>	<code>/misc/scanner</code>	Get Scanner Types	
------------------	----------------------------	-------------------	--

ScanManager

<code>POST</code>	<code>/scan_manager/start_import/{file}</code>	Get Import Start	
-------------------	--	------------------	--

Abbildung 6.10 Auszug aus der Swagger-Dokumentation zu weiteren relevanten Endpunkten der API, eigene Darstellung

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

Schemas
AvailableScanner > [...]
Body_get_access_token_token_post >
Body_get_import_start_scan_manager_start_import_file_post >
Body_post_finding_file_finding_file_upload_post >
GitleaksRawResultModel >
HTTPValidationError >
InputType >
UpdateFalsePositive >
UpdateFindingModelFalsePositive >
UpdateFindingModelFavourite >
UploadNewFindingModelRaw >
UserModel >
ValidationError >

Abbildung 6.11 Auszug aus der Definition der Schemas aus der Swagger-Dokumentation, eigene Darstellung

Das Backend stellt zusätzlich, neben dem Import über das Frontend, eine CLI zur Verfügung, um neue Treffer einlesen zu können. Der Vorteil dieser Option ist, auf einen Schlag große Datenmengen, beispielsweise für einen initialen Datei-Upload, zu verarbeiten.

6.7 Tests

Eine dedizierte Testumgebung sowie Unit-Tests für Frontend und Backend wurden für die Umsetzung des Prototypen im Rahmen dieser Arbeit nicht bereitgestellt. Für eine zukünftige Weiterentwicklung ist dies allerdings zu empfehlen. Hierzu kann auf die bereitgestellte Swagger-Dokumentation zurückgegriffen werden.

Sowohl das Backend als auch das Frontend wurden allerdings gegen einen umfangreichen Satz an Testdaten entwickelt. Dies war nötig, da, wie in Kapitel 7.1 und 7.2 ersichtlich sowie durch den praktischen Test der evaluierten Scanner in Kapitel 5.4 zusätzlich belegt, die Menge der auszuwertenden Datensätze in einem fünfstelligen Bereich liegen kann.

Aus diesem Grund wurden insgesamt etwa 50 öffentlich einsehbare Github-Repositories mit Gitleaks gescannt. Von diesen Repositories wurden anschließend über das CLI etwa 20.000 Datensätze in die Datenbank eingespielt.

So konnte von Beginn an im Auge behalten werden, wie sich der Prototyp im Umgang mit größeren Datensätzen verhält. Aus diesem Grund wurde beispielsweise entschieden, die Anzeige der einzelnen Treffer pro Repository über Paginierung anstelle einer nachladenden „ewigen“ Liste umzusetzen.

6.8 Offene Punkte

Bei der Entwicklung des Review-Programms handelt es sich um „Work-In-Progress“. Im Rahmen dieser Arbeit sind folglich einige Punkte zunächst offen geblieben.

Für eine zeitige Umsetzung empfiehlt sich vor allem ein State-Management im Frontend. Aktuell werden die Datensätze mit jedem Neuladen der Seite vom Backend geladen, auch wenn sich diese Daten nicht geändert haben.

Auch die Verschlüsselung der Datenbank konnte im Rahmen dieser Arbeit nicht umgesetzt werden, obwohl dies eine kritische Anforderung ist, bevor ein produktiver Einsatz angedacht werden kann. Dies gilt auch für den Einsatz von HTTPS zur Kommunikation zwischen Frontend und Backend. Dies war aktuell nicht möglich, da sich beide Komponenten für die Entwicklung auf dem selben Host befinden und somit kein gültiges Zertifikat generiert werden kann, welches auch von modernen Browsern akzeptiert wird.

Relevant für einen produktiven Einsatz wird auch die Umsetzung einer Benachrichtigungsfunktion wie sie in Abbildung 6.1 im Punkt 5 dargestellt ist. Dies ließe sich zunächst beispielsweise über das Versenden einer E-Mail umsetzen. Später wäre auch eine Anbindung an Atlassian Jira für die

6 Konzeptionierung Review Programm für Gitleaks-Ergebnisse

automatische Erstellung eines Tickets denkbar.

Weniger kritisch für den produktiven Einsatz wäre eine Export Funktion der gespeicherten Datensätze, beispielsweise im SARIF für den Austausch mit weiteren Anwendungen aus dem SAST-Bereich.

Auch die automatische Generierung einer *.Gitleakignore*-Datei, wie sie in Kapitel 5.3.3 vorgestellt wurde, um Gitleaks Fingerprints zur Verfügung zu stellen, welche automatisch ignoriert werden, soll in Zukunft implementiert werden.

7 Praktischer Einsatz von Gitleaks

Als Ergebnis der Evaluation in Kapitel 6 wurde Gitleaks als Scanner zum Aufspüren von sensiblen Informationen im Quellcode der IU ausgewählt.

In diesem Kapitel wird der Einsatz von Gitleaks im Rahmen der Erstellung dieser Arbeit dokumentiert.

Hierzu wird zunächst in Kapitel 7.1 beschrieben, wie die Konfigurationsdatei der Regex-Muster für Gitleaks über die Zeit angepasst und verbessert werden konnte. Das Ziel dieser Überarbeitungen war es, eine eigene, an die Bedürfnisse der IU angepasste, Konfiguration zu entwickeln.

In Kapitel 7.2 wird der praktische Einsatz von Gitleaks an den Quellcode-Repositories beschrieben. In insgesamt sechs Durchläufen wurden dabei alle der circa 500 Repositories gescannt und anschließend ausgewertet.

Das Kapitel schließt mit einer detaillierten Betrachtung der aktuell eingesetzten Variante der Konfigurationsdatei in Kapitel 7.3.

7.1 Evolution der Konfiguration

Wie in Kapitel 3.5 vorgestellt, stellen Regex-basierte Suchmuster eine Möglichkeit dar, nach Secrets im Quellcode zu suchen. Generische Suchmuster haben allerdings den Nachteil, eine hohe Anzahl falsch-positiver Treffer zu verursachen. Aus diesem Grund gilt es eine Balance bei der Erstellung generischer Suchmuster zu finden.

Gitleaks bietet auf Basis einer Konfigurationsdatei die Möglichkeit, diese Suchmuster individuell zu erstellen. Beispiele hierzu können Kapitel 5.3.3 sowie der Auflistung 5.2 entnommen werden.

Die Gitleaks-Konfiguration stellt hierbei die Umsetzung der entsprechenden Anforderung „Anpassungen der Suchpattern“ und „Anpassungen von Ausschlusspattern“ aus Kapitel 4.2 dar.

Um ausgehend von der Defaultkonfiguration eine IU-spezifische Konfiguration zu entwickeln, wurden die Quellcode-Repositories mit insgesamt fünf angepassten Versionen gescannt.

Als Repository wurde jenes aus Kapitel 5.4 wiederverwendet. Dieses umfasst mehr als 16.000 Codezeilen in aktuell 40 Zweigen aus über vier Jahren konstanter Weiterentwicklung. Aus

7 Praktischer Einsatz von Gitleaks

Geheimhaltungsgründen ist es nicht möglich, die Ergebnisse des praktischen Tests zu veröffentlichen. Im Folgenden werden nur die aggregierten Zahlen herangezogen.

Die Übersicht der Ergebnisse kann Tabelle 7.1 entnommen werden. Die Tabelle gibt dabei unter Anderem an, wie viele Regex-Regeln in der jeweiligen Konfiguration eingesetzt wurden und wie viele davon selbst erstellt wurden. Die Spalte „Treffer gesamt“ gibt die Gesamtanzahl der Treffer für das Repository an, davon abgezogen wurden in „Treffer eindeutig“ alle Treffer, die mehr als einmal gefunden wurden. „Anzahl valide Treffer“ gibt alle Treffer an, die gültig sind. Hier wurde aber noch nicht berücksichtigt, ob ein Treffer verifiziert werden konnte, also ob er noch aktiv genutzt wird.

Config_v1 - Config_v5

Die „Config_v1“ entspricht dabei der Defaultkonfiguration, wie sie von Gitleaks in Version 8.6.1 verwendet wurde. Diese wurde in der „Config_v2“ um generische Muster zum Finden von Passwörtern, API-Token, Benutzernamen und Anderen erweitert. Hierbei handelt es sich um die Variante, die bereits vorab für den ersten manuellen Test, wie er in Kapitel 1.2 erwähnt wurde, zum Einsatz gekommen ist.

Die Varianten Config_v3, Config_v4 und Config_v5 sind prinzipiell Erweiterungen der Config_v2. Hierbei wurden weitere Variationen für Zuweisungen hinzugefügt, welche aus der Analyse der Treffer abgeleitet werden konnten. Über die Option „Allowlist“ wurde zudem erstmals die Treffermenge eingeschränkt und gezielt bereits bekannte falsch-positive Treffer über entsprechende Muster von den Scanergebnissen ausgenommen.

Diese Anpassungen führten allerdings, erwartungsgemäß, zu deutlich mehr Treffern, da die Formulierung der Regeln nun generisch erfolgte. Eine Erklärung dieses Verhaltens kann Kapitel 3.5 entnommen werden.

Zunächst wurde deswegen versucht, durch weitere Einschränkungen über die Allowlist bessere Ergebnisse zu erzielen. Diese Versuche erwiesen sich allerdings als nicht zielführend, da die Regex-Regeln immer komplexer wurden und immer mehr Einträge in die Allowlist eingepflegt werden mussten.

Config_v6

In Config_v6 wurde schließlich ein anderer Ansatz gewählt, der die Definition eines gültigen Secrets wieder enger fasst. So wurde sich zunutze gemacht, dass in den meisten Fällen Passwörter als Zeichenketten gespeichert werden, was bedeutet, dass sie, in den an der IU eingesetzten Programmiersprachen, in einfache oder doppelte Anführungszeichen gesetzt werden müssen. Dies wurde auch in Kapitel 3.5 dargestellt. Eine prominente Ausnahme stellen Umgebungsvariablen, beziehungsweise die Datei `.env` dar. Da diese allerdings ohnehin kein Bestandteil eines Repositories

Tabelle 7.1 Versionsvergleich der Regex-Konfigurationen von Gitleaks

Scan-Konfiguration	Regeln (gesamt)	Regeln (custom)	Treffer (gesamt)	Treffer (eindeutig)	Treffer (valide)	Verhältnis (Eindeutig/Valide)
Config_v1	92	0	95	31	31	100%
Config_v2	103	10	637	118	58	49%
Config_v3	105	10	1000	194	80	41%
Config_v4	103	10	862	208	118	56%
Config_v5	102	9	1061	264	154	58%
Config_v6	97	5	347	74	67	90%

sein sollte, wurde eine Regel definiert, damit sie immer in die Suchergebnisse übernommen wird. Die Erfahrung des manuellen Reviews hat hier gezeigt, dass solche .env-Dateien interessante, sensible Informationen beinhalten, auch wenn es sich dabei nicht immer um Secrets handelte.

7.2 Analyse der IU-Repositories

Begleitend zu der in Kapitel 7.1 beschriebenen Erarbeitung einer an die Bedürfnisse der IU angepassten Version der Gitleaks Konfiguration, wurden drei vollständige Durchläufe über alle Repositories der IU durchgeführt und anschließend ausgewertet. Mit diesen Scans wurden zwei Ziele verfolgt.

Das primäre Ziel war es, hoch kritische sensible Daten aus den Repositories zu entfernen. Der in Kapitel 1.2 beschriebene erste Lauf hatte bereits gezeigt, dass sich sensible Informationen in den Repositories befinden.

Das sekundäre Ziel der Scan-Läufe war es, ausgewählte Konfigurationen praktisch zu testen.

Hierzu wurde Gitleaks in Version 8.6.1 mit den Config-Versionen v4, v5 und v6 über alle verfügbaren Repositories der IU ausgeführt. Die Ergebnisse können Tabelle 7.2 entnommen werden. Für eine Visualisierung kann Abbildung 7.1 herangezogen werden.

Diese Ansicht zeigt anhand des grauen Graphen, wie deutlich die Anzahl an Treffern mit der Config_v6 gegenüber den beiden Vorgängern gesenkt werden konnte. Und dies, obwohl die Gesamtanzahl an Repositories in diesem Scan-Laufen um ca 10% zugenommen hatte.

Die Anzahl an Repositories schwankt dabei zwischen den einzelnen Scans, da diese mit einem zeitlichen Versatz ausgeführt wurden. Der zweite Scan umfasste 454 Repositories und wurde etwa eine Woche nach dem ersten Scan mit 462 ausgeführt. Der dritte Scan umfasste bereits 501 Repositories und wurde mit einem Versatz von circa acht Wochen nach dem zweiten ausgeführt. Der Grund für den Versatz liegt in dem hohen zeitlichen Aufwand, bei dem jeder Treffer einer manuellen Überprüfung unterzogen wurde.

7 Praktischer Einsatz von Gitleaks

Tabelle 7.2 Auswertung der Ergebnisse dreier Scans mit Gitleaks über alle Repositories der IU

Scan-Konfiguration	Repositories (gesamt)	Repositories (ohne Treffer)	Treffer (gesamt)	Treffer (eindeutig)	Median Treffer pro Repository
Config_v4	462	358	49326	6627	17
Config_v5	454	368	153005	17074	27
Config_v6	501	291	14105	2952	7

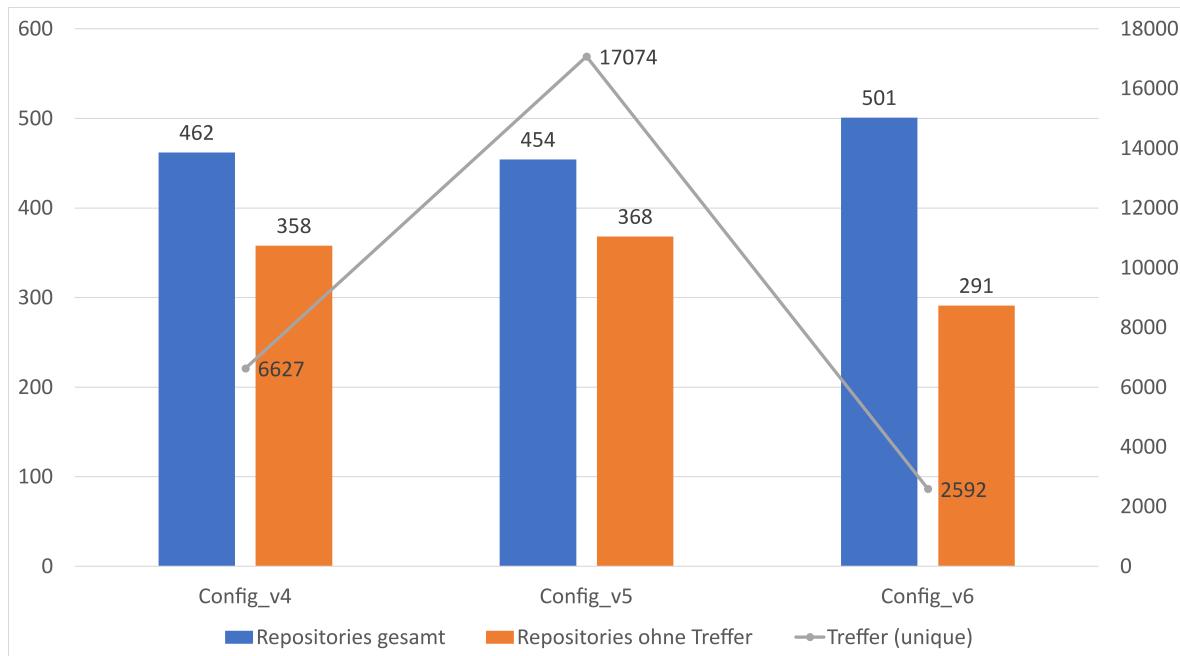


Abbildung 7.1 Grafische Übersicht der Auswertung der Gitleaks-Scans über alle Repositories der IU, eigene Darstellung

Aus Gründen der Geheimhaltung können keine detaillierten Ergebnisse in dieser Arbeit aufgeführt werden. Allerdings wurden von den 291 gültigen Treffern des dritten Scans 45 zur weiteren Bearbeitung übergeben.

7.3 Vorstellung der aktuellen Konfiguration

In diesem Kapitel wird die aktuell verwendete Version der Konfiguration von Gitleaks vorgestellt. Diese Config_v6 ist das Ergebnis eines Entwicklungsprozesses, welcher dem Kapitel 7.1 entnommen werden kann.

Weitere grundsätzliche Informationen zur Konfigurationsdatei können auch dem Kapitel 5.3.3 entnommen werden.

Ein Auszug aus dieser Konfiguration, der sich auf die „interessanteren“ Aspekte wie Passwörter, Benutzernamen und Allowlist fokussiert, kann Auflistung 7.1 entnommen werden.

Die Konfigurationsdatei ist dabei dreigeteilt aufgebaut. Zunächst wurden die Regex-Muster für verschiedene API-Keys, wie sie von der Gitleaks-Default-Konfiguration vorgegeben wurden, übernommen. Da einige API-Provider ihre API-Keys nach einem vorgegebenen Muster aufbauen, lässt sich gezielt danach suchen.

Für die IU potentiell interessante Muster umfassen dabei unter Anderem die APIs von Google, AWS, Atlassian, Slack, Facebook und GitLab.

Aber auch die Muster für private SSH-Keys sind in diesem Abschnitt standardmäßig eingepflegt. Die Datei umfasst dabei insgesamt 92 solcher Muster.

Der zweite Teil der Datei, welcher auch auszugsweise in Auflistung 7.1 einsehbar ist, besteht aus den speziell für die IU erstellten Mustern. Diese sind deutlich allgemeiner formuliert und tragen somit stark zum großen Aufkommen von falsch-positiven Treffern stark bei.

Die Muster sind dabei in drei Gruppen unterteilt. Gruppe eins umfasst „Hardcoded credentials“ und zielt beispielsweise auf folgende Stichworte:

- secret
- password
- pwd
- pass
- token

Die zweite Gruppe konzentriert sich auf „Hardcoded usernames“ und umfasst beispielsweise folgende Stichpunkte:

- user
- username
- userid

Da Konfigurationsdateien in YAML mehrzeilige Zeichenketten erlauben, wurden zusätzliche Suchmuster hinzugefügt, um diese Fälle abdecken zu können. Die Auswertung der Ergebnisse des Scans mit Config_v6 zeigen allerdings, dass mehrzeilige Zeichenketten in YAML aktuell nicht an der IU verwendet werden. Aus diesem Grund sind diese Regeln nicht Teil der Auflistung 7.1.

Wie in Kapitel 4.2 mit der Anforderung „Anpassungen der Suchmuster“ explizit gefordert, wird mit einem weiteren Suchmuster nach .env-Dateien gesucht. Diese enthalten oft sensible Informationen und sollten nicht in Repositories geladen werden.

7 Praktischer Einsatz von Gitleaks

Der dritte Abschnitt der Konfigurationsdatei, welcher ebenfalls auszugsweise Auflistung 7.1 entnommen werden kann, umfasst die globale Allowlist, mit welcher die Anforderung „Anpassungen von Ausschlusspatten (Anti-Pattern)“ umgesetzt wurde. Vom Scanning ausgeschlossene Verzeichnisse umfassen hierbei beispielsweise node_modules, Testverzeichnisse, Lock-Dateien und generell alle Cascading-Style-Sheets (CSS)-Dateien.

Neben der globalen Allowlist ermöglicht Gitleaks auch eine lokale Allowlist im Kontext jeder eigene Regel zu definieren. Über ein entsprechendes Regex wurde versucht Variablen-Substitution zu erlauben.

```
1  [[ rules ]]
2  description = "Hardcoded credentials in source-code files"
3  regex = '''(?i)(?:secret|password|pwd|pass|token)\s{0,10}(?:|=|=|=>|:) {1}(?:\s ?
4    [\"''](. )[\"'']'''
5  tags = ["credentials", "hardcoded"]
6  [[ rules.Entropies]]
7  Group = "1"
8  Max = "7"
9  Min = "3"
10 [rules.allowlist]
11 regexes = [
12   '''${(?:.)+}''',
13   '''(?i)\(\((?:\s)?(?:grab|concat)(?:.)?(?:\s)?\)\)''',
14   '''(?i)!!enveloped:(?:\$)+''',
15   '''(?:.)?{{(?:.)?}}''',
16   '''\$({|\(\)(?:.)?}|)|\x60(?:.)?\x60''',
17   '''(?:\s?)[\"'']\$(. )[\"'']''',
18 ]
19
20 [[ rules]]
21 description = "Hardcoded usernames in source-code files"
22 regex = '''(?i)(?:user|username|userid)\s{0,10}(?:|=|=|=>){1}(?:\s?)[\"'\x60](. )
23   [\"'\x60]'''
24 tags = ["usernames", "hardcoded"]
25 [[ rules.Entropies]]
26 Group = "1"
27 Max = "7"
28 Min = "3"
29 [rules.allowlist]
30 regexes = [
31   '''${(?:.)+}''',
32   '''(?i)\(\((?:\s)?(?:grab|concat)(?:.)?(?:\s)?\)\)''',
33   '''(?i)!!enveloped:(?:\$)+''',
34   '''(?:.)?{{(?:.)?}}''',
35   '''\$({|\(\)(?:.)?}|)|\x60(?:.)?\x60''',
36   '''(?:\s?)[\"'']\$(. )[\"'']'''
```

```
37 [ allowlist ]
38   description = "Allowlisted files"
39   paths = [
40     '''node_modules''',
41     '''vendor''',
42     '''test''',
43     '''tests''',
44     '''^ \.? gitleaks.toml$ ''',
45     '''^(.?) (jpg|gif|doc|pdf|bin)$ ''',
46     '''^ (.?) _test \.go$ ''',
47     '''^ (.?) \.(spec|test) \.(j|t)s$ ''',
48     '''(go.mod|go.sum)$ ''',
49     '''vendor\json''',
50     '''Gopkg\.(lock|toml)''',
51     '''package-lock\json''',
52     '''package\json''',
53     '''composer\json''',
54     '''composer\lock''',
55     '''yarn\lock''',
56     '''\css''',
57     '''UserFactory.php''',
58     '''\xib''',
59   ]
60   regexes = [
61     '''test''',
62     '''\$({|\\() (?::) ?({|\\))|\\x60(?::) ?\\x60'''',
63   ]
```

Listing 7.1 Auszug aus der aktuellen Version 6 der IU-Gitleaks-Konfiguration

8 Fazit und Zukunftsausblick

Im folgenden Kapitel wird die Arbeit mit einem Fazit über die erreichten Ziele, einer Zusammenfassung des Standes der in den einzelnen Kapiteln erreichten Ziele sowie einem kurzen Ausblick in die Zukunft des Secrets-Scanning und der statischen Codeanalyse abgeschlossen.

8.1 Fazit

Insgesamt lässt sich feststellen, dass die gesteckten Ziele erreicht werden konnten.

Zunächst wurde in Kapitel 1 die Notwendigkeit begründet, die circa 500 Quellcode-Repositories der IU nach sensiblen Informationen wie beispielsweise Passwörtern, API-Keys und API-Token zu durchsuchen und somit der Hintergrund für die Erstellung dieser Arbeit definiert.

Um die zu ergreifenden Maßnahmen aus den Blickwinkeln von Entwicklern, dem ISMS und dem ISS-Team zu betrachten, wurde der Themenbereich in Kapitel 2 erfolgreich eingeordnet.

Nach der Schaffung eines gemeinsamen Verständnisses und der Definition wichtiger Grundlagenbegriffe in Kapitel 3 wurde in Kapitel 4 der Evaluation eines Secrets-Scanners als Teil der statischen Codeanalyse an der IU nachgegangen.

Hierzu wurden zunächst im Rahmen von Expertengesprächen Anforderungen an die Anwendung definiert und diese anschließend nach dem Prinzip der Requirements-Triage priorisiert.

Die drei im Rahmen einer Marktanalyse ermittelten, vielversprechendsten Anwendungen wurden genauer untersucht und im Rahmen dieser Arbeit vorgestellt. Ein praktischer Test an einem größeren Repository der IU mit 16.000 Zeilen Code wurde genutzt, um die Scanner im Praxiseinsatz zu testen.

Die Ergebnisse des praktischen Tests und die Übereinstimmung mit den Anforderungen wurden abschließend für eine Entscheidungsfindung in einer Tabelle zusammengefasst. Entschieden wurde sich schlussendlich für das Open-Source-Programm Gitleaks.

Statische Codeanalyse liefert vergleichsweise viele falsch-positive Ergebnisse, insbesondere wenn sie auf Basis von Regex arbeitet. Aus diesem Grund wurde in Kapitel 6 eine Lösung für das Problem der manuellen Nachbearbeitung der Scan-Ergebnisse erarbeitet.

Diese Lösung besteht aus einer webbasierten Anwendung, welche es ermöglicht, die Scan-Ergebnisse nach falsch-positiv oder wahr-positiv zu klassifizieren. Sollte ein bereits erfasstes Ergebnis im

8 Fazit und Zukunftsausblick

Rahmen eines nachfolgenden Scans erneut aufgeworfen werden, wird es automatisch gefiltert. Dies ist möglich, da Gitleaks zu jedem Treffer einen eindeutigen Fingerprint liefert, auf dessen Basis der Vergleich der Ergebnisse ausgeführt werden kann.

Im Rahmen dieses Kapitels wurde dabei die entworfene Architektur des Prototypen vorgestellt, sowie die Techniken, mit denen die Implementierung ausgeführt wurde. Auch eine Übersicht des zugrundeliegenden Prozesses wurde vorgestellt.

Insgesamt wurden die Repositories der IU im Rahmen dieser Arbeit sechs mal mit Gitleaks gescannt. Dabei wurde in jedem Durchlauf die Konfiguration der Suchmuster angepasst. Drei dieser Scans wurden vollständig ausgewertet. Die Ergebnisse der Auswertung wurden anschließend grafisch aufbereitet, um die Entwicklung über die Zeit aufzuzeigen.

8.2 Zusammenfassung

Im folgenden Kapitel werden die Ergebnisse der Kapitel 2, 4, 5, 6 und 7 zusammengefasst.

8.2.1 Einordnung des Secrets-Scannings

Als Vorreiter im Bereich der Online-Lehre arbeitet die IU an verschiedenen Maßnahmen zur Verbesserung der IT-Sicherheit. Aus diesem Grund war es zunächst nötig, das Secrets-Scanning als Teil statischer Codeanalyse passend einzurichten.

Zur Dokumentation wurde dabei in Kapitel 2.1 zunächst erarbeitet, wie das Thema von MITRE und OWASP beschrieben wird.

MITRE beschreibt das Problem von „Hard-coded Credentials“ dabei in der CWE-798. In der CWE-Top-25 Liste von 2022 liegt die CWE-798 dabei, als einzige statische Schwachstelle im Ranking, an Position 15.

Auch ein entsprechender Eintrag in der ATT&CK-Matrix existiert unter der ID 1552.001 als Bestandteil der Technik „Credentials Access“.

OWASP wiederum liefert eine Klassifizierung von „high“, falls sensible Informationen über den Quellcode veröffentlicht werden. Für einen anschließenden Exploit liegt die Eintrittswahrscheinlichkeit sogar bei „very high“. Dies konnte mit einem aktuellen Beispiel anhand von Atlassian Confluence in Kapitel 1.5.3 unterstrichen werden.

Für die Einordnung in die Maßnahmen von Seiten der IU ergaben sich konkret drei Bereiche. Aus Entwicklersicht wurde das Secrets-Scanning als Teil von statischer Codeanalyse in Kapitel 2.4 in den agilen Software-Entwicklungszyklus eingeordnet. Dabei wurde festgestellt, dass Sicherheit

nicht mehr am Ende des Entwicklungszyklus abgedeckt werden kann, sondern, im Sinne von DevSecOps, ein essentieller Bestandteil jedes Prozessschrittes des SDLC sein muss.

Aus Sicht des ISMS handelt es sich bei sensiblen Informationen im Quellcode um „Fehlende Sicherheit in der Entwicklung (Fehlende Sicherheit im Entwicklungs- und Unterstützungsprozess)“. Diese Beurteilung ist allerdings nur ein Beispiel, da eine vollständige Risikoanalyse den Rahmen dieser Arbeit überschritten hätte.

Eine Vorstellung der Maßnahmen des CIS, wie sie vom ISS-Team umgesetzt werden, wurde in Kapitel 2.3 vorgenommen. Dabei wurde in Kapitel 2.3.1 gezeigt, wie sich Secrets-Scanning als Teil des Controls 5, Verwaltung von Accounts, einordnen lässt. In Kapitel 2.3.2 wurde eine Einordnung in das Control 16, Sicherheit von Anwendungssoftware, vorgenommen.

8.2.2 Anforderungsdefinition und Auswahlprozess

In Kapitel 4.2 wurden zunächst 13 Anforderungen definiert, die für die Auswahl eines Secrets-Scanners herangezogen werden mussten. Diese Anforderungen wurden im Rahmen von Expertengesprächen ermittelt und anschließend in Kapitel 4.3 nach der Requirements-Triage in Muss-, Kann- und Soll-Anforderungen unterteilt.

Anschließend wurde im Rahmen einer Marktanalyse in Kapitel 5.2 evaluiert, welche Scanner bereits am Markt existieren. Dabei wurden Kandidaten ausgeschlossen, welche die Muss-Anforderungen nicht erfüllen können. Dies betraf insbesondere Anwendungen, welche auf künstlicher Intelligenz, beziehungsweise maschinellem Lernen basieren, da diese, bis auf eine Ausnahme, keine lokale Analyse ermöglichen. Die Richtlinien der IU verbieten es, potentiell sensible Informationen an Drittanbieter zu senden, um sie dort analysieren zu lassen.

Die drei vielversprechendsten Kandidaten Credential-Digger, TruffleHog und Gitleaks wurden anschließend in den Kapiteln 5.3.1, 5.3.2 und 5.3.3 vorgestellt und einem praktischen Test unterzogen. Dieser Test wurde an einem größeren Repository der IU durchgeführt. Anschließend konnte in Kapitel 5.5 eine Entscheidung getroffen werden, welche Gitleaks als neuen Secrets-Scanner für die IU empfiehlt.

8.2.3 Prototyp für manuelles Review

Um das Problem der großen Anzahl an Ergebnissen und dem hohen Anteil falsch-positiver Treffer innerhalb der Ergebnismenge bei statischer Codeanalyse zu begegnen, ist es nötig, die Ergebnisse einem manuellen Review zu unterziehen.

Hierzu wurde eine webbasierte Anwendung entworfen und prototypisch implementiert, um das Review zu erleichtern. Die Anwendung basiert dabei auf einem in VueJS geschriebenen Frontend, welches in Kapitel 6.5 vorgestellt wurde, einer Restful-API auf Basis von FastAPI im Backend und

8 Fazit und Zukunftsausblick

einer MongoDB um die Ergebnisse zu persistieren. API, Backend und Datenbank wurden in Kapitel 6.6 vorgestellt.

Auch eine Übersicht offener Punkte und potentieller Verbesserungsmöglichkeiten wurde in Kapitel 6.8 erarbeitet.

Besonderes Augenmerk wurde in Kapitel 6.4 auf das Datenmodell für die Datenbank gelegt, da die abgespeicherten und klassifizierten Datensätze die Basis für die Filterung neuer Datensätze bilden. Hier wurde das entworfene Model im Detail vorgestellt.

Die Architektur, der eingesetzte Technik-Stack sowie der Ablaufprozess des Reviews wurden in den Kapiteln 6.3 beziehungsweise 6.2 vorgestellt.

8.2.4 Gitleaks Konfiguration

In Kapitel 7.1 wurde vorgestellt, wie sich die Konfiguration der Regex-basierten Suche von Gitleaks im Zeitraum der Erarbeitung dieser Arbeit entwickelt hat.

Insgesamt wurden sechs Varianten der Konfigurationsdatei für die Scans der Quellcode-Repositories verwendet. Wie diese sich voneinander unterscheiden und entwickelt haben, wurde im Detail in Kapitel 7.1 vorgestellt.

In Kapitel 7.2 wurden die Ergebnisse dreier vollständiger Scans über alle circa 500 Repositories der IU ausgewertet. Dabei kamen drei unterschiedliche Konfigurationen zum Einsatz, darunter auch die aktuell verwendete Version 6.

Hierbei wurde gezeigt, wie die Anzahl an Ergebnissen und die Qualität der ermittelten Treffer über die Zeit hinweg verbessert werden konnte.

Zum Abschluss erfolgte in Kapitel 7.3 ein detaillierter Einblick in den Aufbau der aktuellen Version 6 der Konfiguration.

8.3 Zukunftsausblick

In Kapitel 5.2 wurden Scanner angesprochen, welche auf Basis von künstlicher Intelligenz dabei helfen sollen, sensible Informationen in Dateien zu finden. Allerdings erfolgte die Analyse der Dateien immer auf den Servern der Drittanbieter, sodass eine genaue Abwägung im Sinne der Vertraulichkeit erforderlich ist, ob potentiell sensible Informationen an Drittanbieter übergeben werden dürfen.

Der Credential-Digger der SAP aus Kapitel 5.3.1 ist die einzige recherchierte Ausnahme hiervon, da die Ausführung lokal und offline erfolgt. Dies erfordert allerdings zunächst dessen Datenmodell zu trainieren.

Es ist stark erwartbar, dass Anwendungen auf Basis künstlicher Intelligenz zukünftig für die Suche nach sensiblen Informationen auf breiter Basis zum Einsatz kommen werden.

CH can you find sensitive information in this text:

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "metabase/metabase",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": [3000, 3777, 3778]
    }
  ],
  "Command": ["-e", "MB_DB_TYPE=postgres", "-e",
    "MB_DB_DBNAME=aws_postgres_ebdb", "-e", "MB_DB_PORT=5432", "-e",
    "MB_DB_USER='[REDACTED]', -e", "MB_DB_PASS=[REDACTED]", "-e",
    "MB_DB_HOST=[REDACTED], eu-central-1.rds.amazonaws.com"]
  }
}
```

Abbildung 8.1 Anfrage an ChatGPT zur Identifikation sensibler Informationen im Quellcode, eigene Darstellung

Eine dieser Anwendungen, die sich im Bearbeitungszeitraum dieser Arbeit in den Fokus gestellt hat, ist ChatGPT. Bei diesem handelt es sich um einen Chatbot, der vom US-Forschungslabor OpenAI im November 2022 als Prototyp veröffentlicht wurde.¹

Da ChatGPT, im Gegensatz zu den anderen recherchierten Anwendungen auf Basis von künstlicher Intelligenz, kostenlos und unkompliziert ausprobiert werden kann, wurde das Programm für einen kurzen Zukunftsansicht gewählt.

Zunächst wurde das Programm gefragt, was denn sensible Informationen im Kontext von Quellcode sind. Da es sich hierbei nur um die Abfrage einer Definition handelt, wurde diese Frage passend beantwortet.

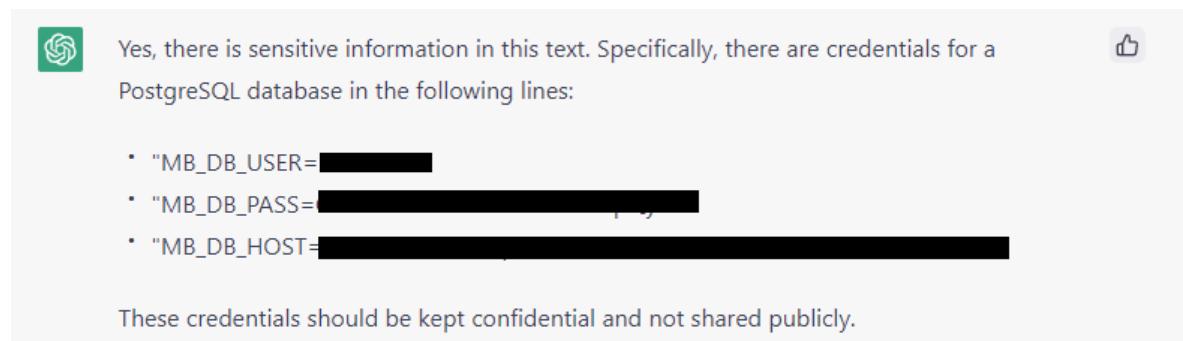
In der nächsten Frage, dargestellt in Abbildung 8.1, wurde ChatGPT anschließend ein Quellcodeauszug mit sensiblen Informationen übergeben. Um den Richtlinien der IU zu entsprechen, wurde dieser Datensatz vorab geprüft und invalidiert.

Innerhalb weniger Sekunden konnte das Programm alle drei Stellen identifizieren und zusätzlich noch eine kurze Erklärung beifügen, warum diese Informationen als sensibel eingestuft wurden. Die Antwort von ChatGPT kann Abbildung 8.2 entnommen werden.

Dieses Ergebnis lässt die Vermutung zu, dass zukünftig vor allem im Bereich statischer Codeanalyse mithilfe künstlicher Intelligenz ein deutlich höheres Sicherheitsniveau erreicht werden kann als

¹ [40]

8 Fazit und Zukunftsansblick

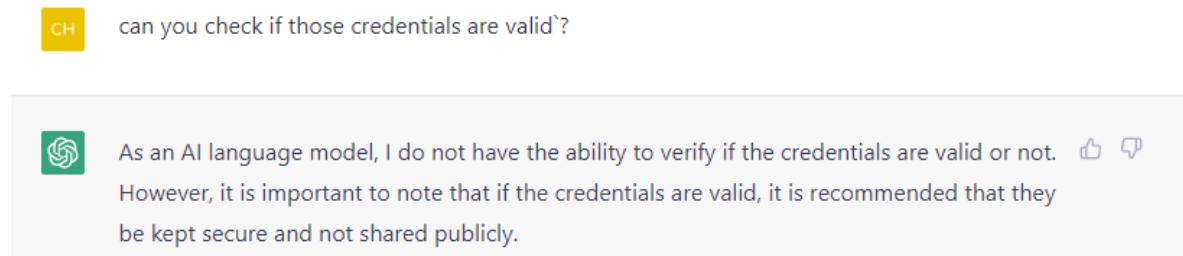


Yes, there is sensitive information in this text. Specifically, there are credentials for a PostgreSQL database in the following lines:

- "MB_DB_USER= [REDACTED]
- "MB_DB_PASS= [REDACTED]"
- "MB_DB_HOST=[REDACTED]"

These credentials should be kept confidential and not shared publicly.

Abbildung 8.2 Antwort von ChatGPT zur Identifikation sensibler Informationen in Quellcode, eigene Darstellung



can you check if those credentials are valid`?

As an AI language model, I do not have the ability to verify if the credentials are valid or not. However, it is important to note that if the credentials are valid, it is recommended that they be kept secure and not shared publicly.

Abbildung 8.3 Frage an ChatGPT die identifizierten Treffer zu validieren, eigene Darstellung

das heute bereits der Fall ist.

Hier gilt allerdings, wie bereits in Kapitel 4.2 angesprochen, abzuwegen, ob das Risiko vertretbar ist, Quellcode mit potentiell sensiblen Informationen an Dritte zur Analyse zu geben.

Eine solche Entscheidung müsste an das ISMS zurückgegeben werden.

Ein Punkt, der allerdings unabhängig von künstlicher Intelligenz weiter verfolgt werden sollte, ist die automatische Validierung gefundener Ergebnisse. Hier gibt es bereits interessante Ansätze in Anwendungen wie TruffleHog, während ChatGPT an dieser Stelle noch nicht zu einer Antwort in der Lage war, siehe Abbildung 8.3.

Denkbar für die Zukunft wäre somit eine Kombination aus Gitleaks und TruffleHog an der IU einzusetzen. So könnten gefundene API-Keys direkt validiert werden ohne die Flexibilität von Gitleaks bei der Suche nach sonstigen sensiblen Informationen zu verlieren.

Dies würde auch der Empfehlung aus Kapitel 4 entsprechen bei SAST-Scans mehr als ein Programm für die selbe Aufgabe einzusetzen.

Solange künstliche Intelligenz nicht dabei helfen kann, die Datenmenge zu reduzieren, müssen die Ergebnisse weiter manuell geprüft werden. Hier kann der vorgestellte Prototyp als Grundlage dienen. Der geplante produktive Einsatz an der IU wird dabei helfen, diese Anwendung weiter zu verbessern.

8.3 Zukunftsansicht

Auch die Frage, inwiefern sich sensible Informationen in Repositories bereits beim Commit verhindern lassen, muss in zukünftiger Arbeit weiter analysiert werden. Hier wird vor allem eine Durchsetzung des verpflichtenden Einsatzes von Secrets-Scannern in den lokalen Entwicklungsumgebungen der Entwickler an der IU eine Herausforderung darstellen.

Literatur

- [1] Ars Technica. *Twitch source code, creator earnings exposed in 125GB leak*. 8. Feb. 2023. URL: <https://arstechnica.com/information-technology/2021/10/twitch-admits-to-major-leak-exposing-source-code-creator-earnings/> (besucht am 08.02.2023).
- [2] Atlassian. *Questions For Confluence Security Advisory 2022-07-20 | Confluence Data Center and Server 8.0 | Atlassian Documentation*. 8. Feb. 2023. URL: <https://confluence.atlassian.com/doc/questions-for-confluence-security-advisory-2022-07-20-1142446709.html> (besucht am 08.02.2023).
- [3] Dylan Ayrey. *Introducing TruffleHog v3 - Truffle Security*. 2022. URL: <https://trufflesecurity.com/blog/introducing-trufflehog-v3/index.html> (besucht am 14.11.2022).
- [4] Bundesamt für Sicherheit in der Informationstechnik. “Die Lage der IT-Sicherheit in Deutschland 2021”. In: (). URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2022.pdf?__blob=publicationFile&v=6 (besucht am 29.10.2022).
- [5] C. E. Shannon. “A Mathematical Theory of Communication”. In: DOI: 10.4135/9781412959384.n229.
- [6] Center for Internet Security. “CIS-Controls-v8”. In: 8 (2021). Hrsg. von SANS. URL: <https://learn.cisecurity.org/cis-controls-download> (besucht am 27.01.2023).
- [7] CWE Content Team. *CWE - CWE-798: Use of Hard-coded Credentials (4.9)*. 13. Nov. 2022. URL: <https://cwe.mitre.org/data/definitions/798.html> (besucht am 13.11.2022).
- [8] Dave Wichers, itamarlavender, will-obrien, Eitan Worcel, Prabhu Subramanian, kingthorin, coadaflorin, hblankenship, GovorovViva64, pfhorman, GouveaHeitor, Clint Gibler, DSotnikov, Ajin Abraham, Noam Rathaus, Mike Jang. *Source Code Analysis Tools*. Hrsg. von OWASP Foundation. 2022. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (besucht am 02.11.2022).
- [9] Deutsches Institut für Normung e. V., Hrsg. *ISO 27001-2017 DE*.
- [10] Gabriel Bassett, C. David Hylender, Philippe Langlois, Alexandre Pinto, Suzanne Widup. “Verizon-2021-data-breach-investigations-report”. In: (2021). URL: <https://www.verizon.com/business/resources/reports/dbir/2022/introduction/> (besucht am 10.10.2022).

Literatur

- [11] GitGuardian. "THE STATE OF SECRETS SPRAWL 2022". In: 2022 (). URL: https://res.cloudinary.com/da8kiytlc/image/upload/v1646148528/GitGuardian_StateOfSecretsSprawl2022.pdf (besucht am 26. 10. 2022).
- [12] GitHub. *GitHub - tiangolo/fastapi: FastAPI framework, high performance, easy to learn, fast to code, ready for production.* 5. Feb. 2023. URL: <https://github.com/tiangolo/fastapi> (besucht am 05. 02. 2023).
- [13] GitHub. *GitHub - trufflesecurity/trufflehog: Find credentials all over the place.* 5. Feb. 2023. URL: <https://github.com/trufflesecurity/trufflehog> (besucht am 05. 02. 2023).
- [14] GitHub. *gitleaks/gitleaks.toml at master · zricethezav/gitleaks.* 5. Feb. 2023. URL: <https://github.com/zricethezav/Gitleaks/blob/master/config/gitleaks.toml> (besucht am 05. 02. 2023).
- [15] GitHub. *Option to use custom Regex with and without the already available detectors · Issue #596 · trufflesecurity/trufflehog.* 5. Feb. 2023. URL: <https://github.com/trufflesecurity/trufflehog/issues/596> (besucht am 05. 02. 2023).
- [16] GitHub. *trufflesecurity/trufflehog: Find credentials all over the place.* 14. Nov. 2022. URL: <https://github.com/trufflesecurity/trufflehog> (besucht am 14. 11. 2022).
- [17] GitHub. *zricethezav/gitleaks: Protect and discover secrets using Gitleaks.* 14. Nov. 2022. URL: <https://github.com/zricethezav/gitleaks> (besucht am 14. 11. 2022).
- [18] GitLab B.V. *Secure your application | GitLab.* 12. Nov. 2022. URL: https://docs.gitlab.com/ee/user/application_security/ (besucht am 12. 11. 2022).
- [19] gitleaks.io. *Gitleaks.* 13. Okt. 2022. URL: <https://gitleaks.io/> (besucht am 14. 11. 2022).
- [20] Michael Günsch. "Twitch-Hack: Streamer-Auszahlungen und Steam-Konkurrent offenbart". In: *ComputerBase* (10. Juni 2021). URL: <https://www.computerbase.de/2021-10/twitch-hack-streamer-auszahlungen-und-steam-konkurrent-offenbart/> (besucht am 08. 02. 2023).
- [21] *Is it maintained-Webseite.* 2022. URL: <https://isitmaintained.com/> (besucht am 18. 11. 2022).
- [22] *Is it maintained? fastapi.* 5. Feb. 2023. URL: <https://isitmaintained.com/project/tiangolo/fastapi> (besucht am 05. 02. 2023).
- [23] *Is it maintained? gitleaks.* 14. Nov. 2022. URL: <https://isitmaintained.com/project/zricethezav/gitleaks> (besucht am 14. 11. 2022).
- [24] *Is it maintained? hog.* 14. Nov. 2022. URL: <https://isitmaintained.com/project/trufflesecurity/trufflehog> (besucht am 14. 11. 2022).

- [25] IU – Internationale Hochschule. *IU REACHES NEW MILESTONE OF 100,000 STUDENTS* | *IU News*. 13. Nov. 2022. URL: <https://www.iu.de/en/news/iu-international-university-of-applied-sciences-reaches-new-milestone-of-100000-students/> (besucht am 13.11.2022).
- [26] IU Internationale Hochschule GmbH. “company presentation_IU_international university_EN_010922”. In: (2021). unveröffentlicht, auf Anfrage über Pressestelle der IU, presse@iu.org.
- [27] Mackenzie Jackson. “Uber Breach 2022 – Everything You Need to Know”. In: *GitGuardian Blog - Automated Secrets Detection* (16. Okt. 2022). URL: <https://blog.gitguardian.com/uber-breach-2022/> (besucht am 08.02.2023).
- [28] Tanya Janca. *Alice and Bob Learn Application Security*. 1st edition. Erscheinungsort nicht ermittelbar und Boston, MA: Wiley and Safari, 2020. ISBN: 9781119687351. URL: <https://learning.oreilly.com/library/view/-/9781119687351/?ar>.
- [29] Jay Chen, Palo Alto Networks u. a. *Unsecured Credentials: Credentials In Files: Sub-technique T1552.001*. 8. Nov. 2022. URL: <https://attack.mitre.org/techniques/T1552/001/> (besucht am 13.11.2022).
- [30] Sebastian Klipper. *Information Security Risk Management: Risikomanagement mit ISO/IEC 27001, 27005 und 31010 ; mit 10 Tabellen und 14 Fallbeispielen*. 1. Aufl. Edition <kes>. Wiesbaden: Vieweg + Teubner, 2011. ISBN: 9783834813602. URL: <https://link.springer.com/content/pdf/10.1007/978-3-8348-9870-8.pdf> (besucht am 04.10.2022).
- [31] Alexander Königstein. “Twitch-Leak: Einnahmen aller Streamer und Quellcodes veröffentlicht”. In: *heise online* (10. Juni 2021). URL: <https://www.heise.de/news/Twitch-Leak-Einnahmen-aller-Streamer-und-Quellcodes-veroeffentlicht-6210552.html> (besucht am 08.02.2023).
- [32] Sofiane Lounici u. a. “Optimizing Leak Detection in Open-source Platforms with Machine Learning Techniques”. In: (), S. 145–159. DOI: 10.5220/0010238101450159. URL: <https://www.scitepress.org/Papers/2021/102381/102381.pdf> (besucht am 03.12.2022).
- [33] Luke Cartey, David Keaton, Stefan Hagen. *OASIS Static Analysis Results Interchange Format (SARIF) TC | OASIS: Defining a standard output format for static analysis tools*. 13. Nov. 2022. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sarif (besucht am 13.11.2022).
- [34] Mayank Deshmukh. *Atlassian Confluence: Questions for Confluence App Hardcoded Credentials Vulnerability (CVE-2022-26138)* | *Qualys Security Blog*. 2022. URL: <https://blog.qualys.com/qualys-insights/2022/08/17/atlassian-confluence-questions-for-confluence-app-hardcoded-credentials-cve-2022-26138> (besucht am 08.02.2023).

Literatur

- [35] Michael Meli, Matthew R. McNiece und Bradley Reaves. "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories". In: DOI: 10.14722/ndss.2019.23418. URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf (besucht am 30.10.2022).
- [36] Microsoft Corporation. *Microsoft Security DevOps: Making security principles and practices an integral part of DevOps while maintaining improved efficiency and productivity*. 2022. URL: <https://www.microsoft.com/en-us/securityengineering/devsecops> (besucht am 12.11.2022).
- [37] Mirco Lang. "Git-Hook-Management mit Pre-Commit. Einstieg ins Pre-Commit-Framework". In: 48925098 (2023). URL: <https://www.dev-insider.de/git-hook-management-mit-pre-commit-a-84e48ab4399984ccf6158c0955e10a18/?r=ext> (besucht am 21.02.2023).
- [38] MongoDB. *MongoDB: The Developer Data Platform*. 5. Feb. 2023. URL: <https://www.mongodb.com/> (besucht am 05.02.2023).
- [39] nist.gov. *NVD - CVE-2022-26138 Detail*. 8. Feb. 2023. URL: <https://nvd.nist.gov/vuln/detail/CVE-2022-26138> (besucht am 08.02.2023).
- [40] OpenAI. "ChatGPT: Optimizing Language Models for Dialogue". In: *OpenAI* (30. Nov. 2022). URL: <https://openai.com/blog/chatgpt/> (besucht am 15.02.2023).
- [41] OWASP Foundation. *Source Code Analysis Tools | OWASP Foundation*. 2022. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (besucht am 05.12.2022).
- [42] OWASP Foundation, Inc. *Use of hard-coded password*. 10. Nov. 2022. URL: https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password (besucht am 13.11.2022).
- [43] Scott Chacon und Ben Straub. "Pro Git: Everything you need to know about Git". In: (2022). URL: <https://github.com/progit/progit2-de> (besucht am 08.01.2023).
- [44] Slim Trabelsi. *Credential Digger: Using Machine Learning to Identify Hardcoded Credentials in Github*. 2022. URL: <https://blogs.sap.com/2020/06/23/credential-digger-using-machine-learning-to-identify-hardcoded-credentials-in-github/> (besucht am 14.11.2022).
- [45] Sonarsource S.A. *As a service | SonarCloud | Sonar*. 12. Nov. 2022. URL: <https://www.sonarsource.com/products/sonarcloud/> (besucht am 12.11.2022).
- [46] Sonarsource S.A. *PHP static code analysis: Hard-coded credentials are security-sensitive*. 11. Nov. 2022. URL: <https://rules.sonarsource.com/php/RSPEC-2068> (besucht am 13.11.2022).

- [47] Stan Bühne, Dr. Andrea Herrman. “Handbuch Requirements Management nach IREB Standard: Aus- und Weiterbildung zum IREB Certified Professional for Requirements Engineering Advanced Level “Requirements Management””. In: (). URL: https://www.gasq.org/files/content/gasq/downloads/certification/IREB/IREB%20AL/ireb_cpre_handbuch_requirements_management_advanced_level_v1.pdf (besucht am 24.10.2022).
- [48] Tony Stubblebine. *Reguläre Ausdrücke - kurz & gut: Kurz & gut*. 2. Aufl. O'Reillys Taschenbibliothek. Köln: O'Reilly, 2008. ISBN: 9783897215351. URL: <https://learning.oreilly.com/library/view/-/9783897215351/?ar>.
- [49] The 2022 CWE Top 25 Team. *CWE - 2022 CWE Top 25 Most Dangerous Software Weaknesses*. 13. Nov. 2022. URL: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (besucht am 13.11.2022).
- [50] The MITRE Corporation. *MITRE ATT&CK®*. 8. Nov. 2022. URL: <https://attack.mitre.org/> (besucht am 13.11.2022).
- [51] *Vue.js - The Progressive JavaScript Framework | Vue.js*. 5. Feb. 2023. URL: <https://vuejs.org/> (besucht am 05.02.2023).
- [52] Zach. “Gitleaks LLC — Announcement - Gitleaks”. In: *Gitleaks* (20. Juni 2022). URL: <https://blog.gitleaks.io/gitleaks-llc-announcement-d7d06a52e801> (besucht am 14.11.2022).

