

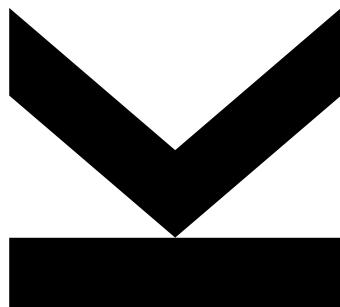
Eingereicht von
Magdalena König

Angefertigt am
**Institut für
Wirtschaftsinformatik –
Software Engineering**

Beurteiler
**Ass.-Prof. Dipl.-Ing.
Dr.techn. Rainer
Weinreich**

Monat Jahr
Dezember 2024

API SECURITY ANALYSIS DASHBOARD auf Basis der OWASP Top 10 API-Sicherheitsri- siken



Masterarbeit
zur Erlangung des akademischen Grades

Master of Science
im Masterstudium

Wirtschaftsinformatik

KURZFASSUNG

Serviceschnittstellen (APIs) sind ein unverzichtbares Konzept in der Softwareentwicklung. Mit der steigenden Anzahl an APIs und der zunehmenden Kommunikation über diese Schnittstellen wächst jedoch auch ihre Angriffsfläche. Cyberkriminelle nutzen APIs vermehrt, um sich Zugang zu Systemen zu verschaffen, Daten zu stehlen oder diese zu manipulieren. API-Security ist daher ein Bereich von aktueller und zukünftiger Relevanz. Die OWASP Foundation (OWASP API Security Project Team, 2023) veröffentlichte 2023 eine aktualisierte Rangliste der 10 größten Risiken für Serviceschnittstellen, um API-Betreiber/-innen dabei zu unterstützen, sichere APIs zu entwickeln und zu betreiben. Im Rahmen dieser Masterarbeit wird ein API Security Analysis Dashboard entwickelt, das die OWASP Top 10 als Grundlage nutzt und den Benutzer/-innen ermöglicht, Schwachstellen einfach und verständlich zu identifizieren. Als Methodik wird der Design Science Ansatz nach Johannesson und Perjons (2014) gewählt. Der Fokus des Dashboards liegt primär auf den Visualisierungen der Schwachstellen, sowie der dazugehörigen OWASP Kategorien, aber auch Benutzerfreundlichkeit und Integrationsmöglichkeiten stehen im Vordergrund, um Schwachstellen anschaulich darzustellen und das Dashboard in bestehende Systeme integrieren zu können. Durch eine Kategorisierung von Sicherheitswerkzeuge im Bereich Webapplikationen und der Analyse sowie dem Vergleich von API-Scannern wird das Tool ZAPProxy (ZAP Dev Team, 2024a) beispielhaft ausgewählt und in das Dashboard eingebunden. Zudem werden im Rahmen des Werkzeugvergleichs Anforderungen für das Dashboard abgeleitet, welche die Implementation bei der abschließenden argumentativen Evaluierung sowie anhand von Fallstudien gut abdecken kann.

ABSTRACT

Service interfaces (APIs) are an indispensable concept in software development. However, with the growing number of APIs and increasing communication via these interfaces, their attack surface is also growing. Cyber criminals are increasingly using APIs to gain access to systems and steal or manipulate data. API security is therefore an area of current and future relevance. The OWASP Foundation (OWASP API Security Project Team, 2023) published an updated ranking of the top 10 risks for service interfaces in 2023 to help API operators develop and operate secure APIs. As part of this master's thesis, an API Security Analysis Dashboard has been developed that uses the OWASP Top 10 as a basis and enables users to identify vulnerabilities in a simple and understandable way. The methodology chosen is the design science approach according to Johannesson and Perjons (2014). The focus of the dashboard is primarily on the visualization of vulnerabilities and the associated OWASP categories, but user-friendliness and integration options are also at the forefront in order to clearly display vulnerabilities and to be able to integrate the dashboard into existing systems. By categorizing security tools in the field of web applications and analyzing as well as comparing API scanners, the tool ZAPProxy (ZAP Dev Team, 2024a) is selected as an example and integrated into the dashboard. In addition, requirements for the dashboard are derived as part of the tool comparison, which the implementation can cover well in the final argumentative evaluation and on the basis of case studies.

INHALTSVERZEICHNIS

1. Einleitung	1
1.1. Zielsetzung	2
1.2. Methodik	3
1.3. Gliederung	5
2. Grundlagen und Begriffe	6
2.1. Cybersecurity	6
2.1.1. Grundlegende Konzepte	6
2.1.1.1. Definitionen	6
2.1.1.2. Gefahren	8
2.1.1.2.a. Malware	8
2.1.1.2.b. Ransomware	8
2.1.1.2.c. (D)DoS Angriff	9
2.1.1.2.d. Brute Force-Angriff	9
2.1.1.2.e. Advanced Persistent Threats	9
2.1.1.2.f. Session Hijacking Angriff	9
2.1.1.2.g. Man-in-the-Middle Angriff	10
2.1.1.2.h. Social Engineering (Phishing)	10
2.1.1.2.i. Faktor Mensch	10
2.1.1.3. OWASP Top Ten	10
2.1.1.3.a. OWASP Top 10 2017 vs. 2021	11
2.1.1.3.b. Broken Access Control	12
2.1.1.3.c. Cryptographic Failures	12
2.1.1.3.d. Injection	13
2.1.1.3.e. Insecure Design	14
2.1.1.3.f. Security Misconfiguration	14
2.1.1.3.g. Vulnerable and Outdated Components	14
2.1.1.3.h. Identification and Authentication Failures	15
2.1.1.3.i. Software and Data Integrity Failures	15
2.1.1.3.j. Security Logging and Monitoring Failures	16
2.1.1.3.k. Server-Side Request Forgery	16
2.1.1.4. Gegenmaßnahmen	17

2.1.1.4.a. Testing.....	17
2.1.1.4.b. Identitäts- und Zugriffsmanagement.....	17
2.1.1.4.c. Zero-Trust Framework	18
2.1.1.4.d. Security by Design.....	18
2.1.1.4.e. Faktor Mensch.....	18
2.2. Kategorisierung der Sicherheitswerkzeuge.....	19
2.2.1. Methodik	19
2.2.2. Ergebnis.....	19
2.3. Service-Schnittstellen (APIs)	22
2.3.1. Grundkonzepte	22
2.3.2. Arten von APIs	23
2.3.2.1. Request-Response APIs	24
2.3.2.1.a. REST	24
2.3.2.1.b. SOAP.....	25
2.3.2.1.c. RPC	25
2.3.2.1.d. GraphQL.....	26
2.3.2.2. Event-Driven APIs	26
2.3.2.2.a. WebHooks	26
2.3.2.2.b. WebSockets.....	27
2.3.2.2.c. HTTP-Streaming	27
2.3.3. Gefahren und Sicherheitstrends.....	27
2.4. API Security.....	28
2.4.1. Konzepte.....	28
2.4.1.1. Authentifizierung und Autorisierung	28
2.4.1.1.a. OAuth	28
2.4.1.1.b. JSON Webtoken (JWT).....	30
2.4.1.1.c. API-Schlüssel.....	31
2.4.1.2 Verschlüsselung	31
2.4.1.3 Input Validierung	32
2.4.1.4 Versionierung	33
2.4.1.5 Security Headers.....	33
2.4.1.6 Rate Limiting	34
2.4.2. OWASP Top 10 API Security	35

2.4.2.1. OWASP Top 10 API Security 2019 vs. 2023	35
2.4.2.2. Broken Object Level Authorization (API1)	37
2.4.2.3. Broken Authentication (API2)	38
2.4.2.4. Broken Object Property Level Authorization (API3)	39
2.4.2.5. Unrestricted Resource Consumption (API4).....	40
2.4.2.6. Broken Function Level Authorization (API5)	42
2.4.2.7. Unrestricted Access to Sensitive Business Flows (API6).....	43
2.4.2.8. Server-Side Request Forgery (API7).....	44
2.4.2.9. Security Misconfiguration (API8)	45
2.4.2.10. Improper Inventory Management (API9).....	46
2.4.2.11. Unsafe Consumption of APIs (API10).....	47
2.4.2.12. Zusammenfassung OWASP Top 10 API Security	48
2.4.3. Testing und Monitoring.....	50
2.4.3.1. API Security Testing.....	50
2.4.3.2. API Security Monitoring	52
3. Vergleich von Werkzeug zur Analyse von Sicherheitsrisiken im Bereich APIs.....	54
3.1. Methodik.....	54
3.1.1. Auswahl der Sicherheitswerkzeuge.....	54
3.1.1.1. Auswahlrunde Eins.....	54
3.1.1.2. Auswahlrunde Zwei	54
3.1.1.3. Auswahlrunde Drei	55
3.1.2. Vergleichskriterien.....	55
3.2. Überblick der Sicherheitswerkzeuge.....	56
3.2.1. ZAPProxy	56
3.2.2. GraphQL Cob.....	56
3.2.3. Wapiti.....	57
3.3. Analyse der Sicherheitswerkzeuge.....	57
3.3.1. ZAPProxy	57
3.3.2. Wapiti.....	60
3.3.3. GraphQL Cop.....	62
3.4. Ergebnis und Vergleich der Sicherheitswerkzeuge	64
4. API Security Dashboard	67
4.1. Zielsetzung und Methodik.....	67

4.2. Anforderungen.....	68
4.3. Architektur und Implementierung	70
4.3.1. Datenschicht	71
4.3.2. Logikschicht	73
4.3.3. Präsentationsschicht	77
4.4. Evaluierung	83
4.4.1. Argumentative Evaluierung	83
4.4.2. Fallstudien.....	87
4.5. Ergebnis des API Security Dashboards	88
5. Zusammenfassung und Diskussion.....	91
5.1. Beantwortung der Forschungsfragen.....	91
5.2. Einschränkungen.....	92
5.2. Implikationen und Ausblick	94
6. Literaturverzeichnis	96
7. Anhang.....	106
Anhang A: Kategorisierung der Werkzeuge.....	106
Anhang B: Auswahl der Werkzeuge	113
Anhang C: Analyse der Sicherheitswerkzeuge.....	118
Anhang D: API Security Analysis-Dashboard Set-Up	133
Anhang E: Visualisierungstypen	134
Anhang F: Beispielhaftes Dashboard	139

ABBILDUNGSVERZEICHNIS

Abbildung 1: Design Science Prozess (Johannesson & Perjons, 2014)	4
Abbildung 2: OWASP Top 10 2017-2021 (OWASP Top 10 Team, 2021).....	12
Abbildung 3: Kategorienverteilung Security-Werkzeuge	20
Abbildung 4: Zugänglichkeit Security-Werkzeuge.....	20
Abbildung 5: APIs Security-Werkzeuge	21
Abbildung 6: Kategorienverteilung APIs Security-Werkzeuge.....	21
Abbildung 7: API Scanner Security-Werkzeuge.....	22
Abbildung 8: OAuth Prozess (angelehnt an Hardt, 2012)	29
Abbildung 9: OWASP Top 10 API Security 2019-2023 (Indusface, 2023)	36
Abbildung 10: Aspekt Ausnutzbarkeit.....	49
Abbildung 11: Aspekt Verbreitung	49
Abbildung 12: Aspekt Erkennbarkeit.....	50
Abbildung 13: Aspekt Auswirkung	50
Abbildung 14: Wichtige Aspekte bei API Security Tests (Sattam & Moulahi, 2023)	51
Abbildung 15: ZAPProxy HTTP-Bericht Alerts	58
Abbildung 16: ZAPProxy HTTP-Bericht Beispielschwachstelle	59
Abbildung 17: Wapiti HTTP-Bericht.....	62
Abbildung 18: GraphQL Cop Bericht	64
Abbildung 19: Architektur	71
Abbildung 20: UML Diagramm Dashboard	72
Abbildung 21: UML Diagramm User Database	73
Abbildung 22: Schnittstellendokumentation	74
Abbildung 23: ZAPProxy Beispielbericht.....	76
Abbildung 24: Dashboard Startseite	78
Abbildung 25: Dashboard Customisation.....	78
Abbildung 26: Dashboard Überblick	79
Abbildung 27: Dashboard Vulnerability-Count Zeitstrahl.....	81
Abbildung 28: Dashboard OWASP Top 10 Distribution	82
Abbildung 29: Dashboard Risk-Scatterplot	83

CODEVERZEICHNIS

Codebeispiel 1: Fehler (Felderer et al., 2016)	7
Codebeispiel 2: GraphQL Abfrage OWASP API1	37
Codebeispiel 3: Beispielabfrage OWASP API2	39
Codebeispiel 4: Beispielabfragen OWASP API3	40
Codebeispiel 5: Beispiel API-Call OWASP API4	41
Codebeispiel 6: Beispielendpunkt OWASP API5.....	42
Codebeispiel 7: Beispiel API-Anfrage OWASP API7	44
Codebeispiel 8: Port-Scanning OWASP API7	44
Codebeispiel 9: Beispiel API-Abfrage OWASP API8	45
Codebeispiel 10: Beispiel http-Anfrage OWASP API10.....	48
Codebeispiel 11: Permanent Redirect OWASP API10	48
Codebeispiel 12: Dockeraufruf	74
Codebeispiel 13: File-Scanning	75
Codebeispiel 14: File-Verarbeitung	76
Codebeispiel 15: Berechnung des Risk-Scores.....	80
Codebeispiel 16: API-Request	82

TABELLENVERZEICHNIS

Tabelle 1: OWASP Sicherheitsrelevante Aspekte	48
Tabelle 2: ZAPProxy Beispielscans	59
Tabelle 3: Wapiti Beispielscans	62
Tabelle 4: GraphQL Cop Beispielscan	64
Tabelle 5: Anforderungen Dashboard.....	68
Tabelle 6: Evaluierung.....	84
Tabelle 7: Fallstudien	87

1. Einleitung

Laut einer Analyse der Wirtschaftskammer Wien haben sich Cyberangriffe in Österreich im Jahr 2023 um beinahe 90% erhöht (WKO, 2023). Gleichzeitig wird die Welt mit einer immer größeren Menge an Daten geflutet, was eine effektive Speicherung dieser verlangt. In diesem Zusammenhang ist besonders die Ablage der Daten in der Cloud auf dem Vormarsch um diese ressourcensparend und kosten-günstig speichern und nutzen zu können. Beinahe 65% aller Unternehmen einer aktuellen Studie geben an, dass Digitalisierung, mitunter speziell Cloud-Migration höchste Priorität haben (Sakovich, 2023). Unter anderem mit dem Trend der Cloud-Migration einhergehend sind unausweichlich Service-Schnittstellen (APIs) welche Daten sicher von einem Service ins andere leiten sollen. Aber auch der Trend der Microservices, sowie die Nutzung von externen Services und Applikationen verlangt den Einsatz von APIs. Kommt es in diesem Bereich zu einer Sicherheitslücke, können nicht nur sensible Daten in die Öffentlichkeit gelangen, auch für potenzielle Cyberkriminelle sind dies ideale Angriffsflächen, um sich Zugang zu Systemen zu verschaffen. Mitunter könnten dann dort wichtige Daten gestohlen oder diese verfälscht werden.

Laut einer Studie von Traceable (2023) stimmen 58% der befragten Expert/-innen zu, dass APIs für einen erweiterte Angriffsfläche im gesamten System verantwortlich sind. Darüber hinaus gehen Studien davon aus, dass Schwachstellen im Bereich APIs für zwischen 4.1% und 7.5% aller Cyberangriffe verantwortlich sind (Leyden, 2022), die Tendenz steigt hier aber ganz klar. Dies bestätigt unter anderem der API-Sicherheitsreport von Wallarm (2024); hier konnte man im letzten Jahr einen 30%igen Anstieg von sicherheitsrelevanten Vorfällen im Bereich APIs verzeichnen. Ein flächendeckender Einsatz von Serviceschnittstellen steigert außerdem die Komplexität von Systemen und die Notwendigkeit von akkurate Dokumentation und Sicherheitsstrategien. Eine Sicherheitsstrategie im Bereich APIs ist allerdings in vielen Fällen unzureichend definiert oder fehlt ganz und auch eine konsequente Dokumentation ist häufig nicht aktuell oder lückenhaft (SALT, 2024). Dabei arbeiten etwa 13% aller Unternehmen laut einer Studie von Traceable (2023) mit über 2500 APIs, was die Komplexität der Serviceslandschaft unterstreicht und die nachhaltige und strukturelle Sicherung dieser unerlässlich macht. Dem gegenüber gaben 95% der befragten Unternehmen in einer Studie von SALT (2024) an, Probleme zu haben, sicherheitsrelevante Vorfälle im Bereich von Serviceschnittstellen eindämmen zu können. 23% der Unternehmen gaben gar Preis, bereits eine Kompromittierung von Daten und System aufgrund eines schwerwiegenden sicherheitsrelevanten Vorfalls im Bereich APIs erfahren zu haben, bei der Studie von Traceable (2023) waren es in den Jahren 2022 und 2023 sogar 60%. Doch nicht nur technische Auswirkungen ziehen unsichere Serviceschnittstellen nach sich, auch organisatorische und strategische Bereiche eines Unternehmens können davon betroffen sein. Über 50% aller Unternehmen gaben beispielsweise an, dass Einführungen von Applikationen aufgrund von Sicherheitslücken in APIs verzögert stattfinden mussten, sowie das Thema API-Sicherheit auch immer öfter in Diskussionen der Management-Ebene einfließt (SALT, 2024).

1.1. Zielsetzung

Die Daten zeigen klar: Es ist unerlässlich eine Service-Schnittstellen entsprechend zu schützen und regelmäßig einer umfassenden Analyse zu unterziehen, um Sicherheitsrisiken aktiv entgegenzuwirken zu können. Eine umfassende Datensammlung, Analyse und Kontextualisierung von Sicherheitsrelevanten Parametern ist ein wichtiger Schritt in Richtung sicherer APIs (Shields, 2024). Die vorliegende Masterarbeit will hier ansetzen und basierend auf den von OWASP definiert zehn Sicherheitsrisiken für APIs (OWASP API Security Project Team, 2023) helfen Schwachstellen im Bereich APIs besser auffindbar und abbildbar zu machen.

Ziel dieser Arbeit ist es, Werkzeuge zu analysieren und zu kategorisieren, die darauf abzielen, sicherheitsrelevante Schwachstellen in Webservices aufzudecken. Auf Basis dieser Kategorisierung werden jene Werkzeuge extrahiert, detailliert analysiert und verglichen, die in der Lage sind, Sicherheitsschwachstellen von Serviceschnittstellen im Hinblick auf die OWASP Top 10 API-Sicherheitsrisiken zu identifizieren. Im Anschluss daran wird ein Prototyp für ein Security Analysis Dashboard entwickelt, sowie ausgewählte Werkzeuge integriert, um die Ergebnisse der Sicherheitsanalysen in einem zentralen Dashboard darzustellen. Dieses Dashboard wird abschließend primär argumentativ aber auch in Form von Case-Studies evaluiert, um seine Effektivität und Benutzerfreundlichkeit zu überprüfen.

Hierbei ergeben sich die folgenden Forschungsfragen:

- FF1: Welche Werkzeuge zur Analyse der Sicherheit von Webservices existieren bereits und wie können diese kategorisiert werden?

Die Beantwortung der Forschungsfrage 1 findet mithilfe der Recherche, Sammlung, sowie Kategorisierung der gefundenen Werkzeuge zur Analyse von Sicherheitsrisiken in Webanwendungen statt. Danach folgt eine visuelle Aufbereitung sowie kategorisierte Liste an Sicherheitswerkzeuge für Webservices, welche als Basis zur Beantwortung von Forschungsfrage zwei dient.

- FF2: Welche Open-Source Werkzeuge zur Analyse von Service-Schnittstellen (APIs) können aus den in FF1 kategorisierten Werkzeuge extrahiert werden und wie können diese zur Aufdeckung der OWASP Top 10 API-Sicherheitsrisiken eingesetzt werden?

Am Beginn der Beantwortung von Forschungsfrage 2 steht die kategorisierte Liste, welche als Artefakt durch die Beantwortung von FF1 entsteht. Diese wird in mehreren Auswahlrunden auf jene Sicherheitswerkzeuge reduziert, welche APIs nach sicherheitsrelevanten Schwachstellen analysieren können, sowie als Open-Source Werkzeug zur Verfügung stehen. Andere Auswahlkriterien umfassen Integrationsfähigkeiten, Dokumentationen oder auch die Möglichkeit der Abbildung der gefundenen Schwachstellen in Kombination der OWASP API-Sicherheitsrisiken. Im Zuge der Analyse der Sicherheitswerkzeuge liegt der Fokus unter anderem auf Aspekten wie Usability, Funktionalität und praktischen Beispielsscans. In dem abschließenden Vergleich der Werkzeuge werden dabei die Stärken, Schwächen und Unterschiede der einzelnen Werkzeuge erläutert.

- FF3: Wie können die aus FF2 aufgedeckten Schwachstellen in Bezug auf Service Schnittstellen (APIs) in einem Security Analysis Dashboard abgebildet und visualisiert werden?

Zur Beantwortung dieser Forschungsfrage werden basierend auf Forschungsfrage FF2 Anforderungen für den zu implementierenden Prototypen extrahiert. Diese Anforderungen werden in einem prototypischen Dashboard umgesetzt, das dann mit Beispiel APIs in verschiedenen Fallstudien (Case Studies) eingesetzt wird. Die Umsetzung des Dashboards soll einen stark explorativen Charakter haben, um verschiedene Arten von Visualisierungen für unterschiedliche Sicherheitsprobleme untersuchen zu können. Die Forschungsfrage wird mit der Evaluierung in Form einer Gegenüberstellung zwischen den Anforderungen und dem realisierten Dashboard im Zuge der Case-Studies abgeschlossen.

1.2. Methodik

In der Masterarbeit wird der Design Science-Ansatz verwendet, um die drei oben genannten Forschungsfragen zu adressieren. Zunächst wird eine umfangreiche Analyse bestehender Sicherheitswerkzeuge für Webservices durchgeführt und hinsichtlich ihrer Charakteristiken kategorisiert (FF1). Anschließend werden aus dieser Kategorisierung Open-Source-Werkzeuge extrahiert und nach ihrer Fähigkeit analysiert, die OWASP Top 10 API-Sicherheitsrisiken zu identifizieren, wobei praktische Anwendungsfälle zur Evaluierung der Werkzeuge dienen (FF2). Schließlich wird basierend auf den gewonnenen Erkenntnissen ein Security Analysis Dashboard entwickelt und implementiert, das Schwachstellen in Service-Schnittstellen (APIs) visualisiert und somit einen praktischen Beitrag zur Verbesserung der API-Sicherheit leistet (FF3). Diese methodische Vorgehensweise ermöglicht es, sowohl theoretische Einsichten zu generieren als auch konkrete Lösungsansätze für die Praxis zu entwickeln, indem sie die Brücke zwischen theoretischer Forschung und praktischer Anwendung schlägt.

Design Science nach Johannesson und Perjons (2014) wird als „scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems [...]“ (S.13) definiert. Der Ansatz wird häufig in wissenschaftlichen Projekten Rund um Informationssysteme verwendet und soll demnach auch die Grundlage für diese Masterarbeit sein. Als Artefakt kann hierbei das API Security Analysis Dashboard gesehen werden, das im Laufe der vorliegenden Arbeit implementiert werden soll.

Dabei definieren Johannesson und Perjons (2014) verschiedene Phasen für den Design Science Ansatz welche in Folge kurz erläutert und in Bezug zu dem methodischen Ansatz in dieser Masterarbeit gesetzt werden. Im Regelfall wird der in Abbildung 1 dargestellte Prozess in mehreren Iterationen wiederholt.

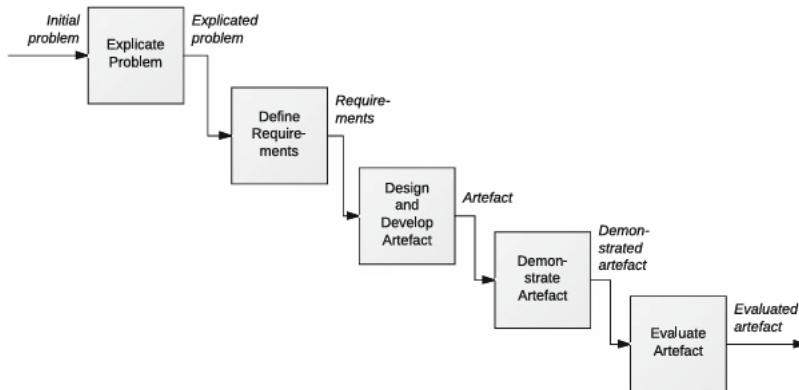


Abbildung 1: Design Science Prozess (Johannesson & Perjons, 2014)

- Phase 1: Problem explizieren: Zum Beginn des Design Science Ansatzes steht die Formulierung sowie Analyse eines Problems, welches eine globale Komponente aufweisen sollte. Im Kontext der vorliegenden Masterarbeit handelt es sich hierbei um einen Anstieg an Cyberkriminalität in Kombination mit der steigenden Wichtigkeit von Serviceschnittstellen, welches eine einfache und schnelle Identifikation der Schwachstellen in APIs benötigt.
- Phase 2: Anforderungen definieren: Die zweite Phase wird von der Schaffung eines Rahmens des vorliegenden Problems mithilfe von der Definition von Anforderungen gekennzeichnet. Abgeleitet von den kategorisierten Sicherheitswerkzeugen (FF1) und der nachfolgenden Werkzeuganalyse (FF2) lassen sich Anforderungen extrahieren, welche dem Artefakt API Security Analysis Dashboard erarbeitet werden.
- Phase 3: Artefakte planen und entwickeln: Die Schaffung des Artefakts zur Lösung des definierten Problems mithilfe der Anforderungen steht hier im Mittelpunkt. Für die vorliegende Masterarbeit bedeutet dies das Design und die Implementierung des prototypischen API Security Analysis Dashboards.
- Phase 4: Artefakte demonstrieren: Nach der Entwicklung des Artefakts kommt es zu dessen Demonstration anhand eines praktischen Beispiels. Das API Security Analysis Dashboard wird in dieser Phase mithilfe von Test-APIs validiert und dessen (Teil-)Lösung des Problems gezeigt.
- Phase 5: Artefakte evaluieren: Die letzte Phase des Design Science Ansatzes ist die Gegenüberstellung der Anforderungen und des tatsächlich implementierten Artefakts. Hier wird evaluiert, inwieweit diese übereinstimmen und welche Verbesserungen oder Überarbeitungen in der nächsten Iteration nötig sind. Im Zuge der vorliegenden Masterarbeit findet die Evaluierung des Artefakts auf argumentativer Ebene statt und das API Security Analysis Dashboard wird so den vorab definierten Anforderungen gegenübergestellt.

Der Design Science-Ansatz in dieser Arbeit verfolgt grundsätzlich einen stark explorativen Charakter und verzichtet auf strenge externe Anforderungen. Stattdessen werden die Anforderungen, insbesondere jene für das Dashboard, direkt aus den Ergebnissen der vorangehenden Forschungsfragen ab-

geleitet. Dieser Ansatz ermöglicht es, bestehende Sicherheitswerkzeuge systematisch zu kategorisieren und in Verbindung mit den OWASP Top 10 API-Sicherheitsrisiken zu analysieren. Darüber hinaus erlaubt er es, die gefundenen Schwachstellen prototypisch visuell im Dashboard darzustellen.

1.3. Gliederung

Die Masterarbeit gliedert sich somit in ein Grundlagenkapitel (Kapitel 2), das fundamentale Ansätze in den Bereichen Cybersecurity, inklusive der Kategorisierung von Sicherheitswerkzeuge, Service-Schnittstellen und API-Security, mit besonderem Fokus auf die OWASP-Top 10 API-Sicherheitsrisiken, näher erläutert. In Kapitel 3 wird die Analyse und der Vergleich der Open-Source API-Sicherheitswerkzeuge durchgeführt. Kapitel 4 umfasst die Implementierung des prototypischen Dashboards zur Visualisierung der Schwachstellen. Abschließend werden die Ergebnisse diskutiert, Implikationen aufgezeigt und ein Ausblick darauf gegeben, wie die in dieser Masterarbeit behandelten Ansätze und Ergebnisse in zukünftigen wissenschaftlichen Arbeiten weiterentwickelt werden könnten.

2. Grundlagen und Begriffe

Im Folgenden werden zentrale Konzepte und Begriffe der Cybersecurity sowie Serviceschnittstellen (APIs) behandelt, bevor im Anschluss die spezifischen Aspekte der API-Security vertieft werden. So soll das Verständnis der für die zur Beantwortung der Forschungsfragen relevanten Themengebiete gewährleistet werden und eine fundierte Basis für die weiteren Ausführungen geschaffen werden.

2.1. Cybersecurity

Cybersecurity, auch oft synonym als IT Security oder IT-Sicherheit bezeichnet, beschreibt den Schutz von Informationen, Anwendungen und vernetzten Systemen vor digitalen Angriffen (IBM, 2023). Eine andere Definition beschreibt Cybersecurity als “measures taken to protect a computer or computer system (as on the Internet) against unauthorized access or attack” (Merriam-Webster, 2023).

Die International Telekommunikationsunion (ITU) erläutert Cybersecurity sehr viel umfassender als eine Sammlung von Instrumenten, Konzepten, Aktionen, Trainings, Technologien, Best Practices, Risikomanagement Ansätze und Guidelines welche den Schutz der IT-Infrastruktur, sowie der Organisations- und Mitarbeiter/-innen-Ressource zur Aufgabe haben. Diese Ressourcen umfassen physische Geräte, wie PCs, Servers oder andere Hardware, als auch Softwareanwendungen, Services oder Telekommunikationssysteme sowie alle Daten welche gespeichert oder versendet werden. Dabei hat Cybersecurity das Ziel, die Verfügbarkeit, Integrität und Vertraulichkeit einer jeder Organisations- und Benutzer-Ressource, sowie der IT-Infrastruktur zu wahren (International Telecommunications Union, 2008 zitiert nach Von Solms & Van Niekerk, 2013).

Oft wird neben IT Security auch die Information Security, als Synonym für Cybersecurity verwendet, jedoch unterscheidet sich hier der Fokus der beiden Konzepte. Während Information Security sich auf den Schutz der Informationen eines Assets oder einer Ressource innerhalb des Cyberraums bezieht, schließt die Cybersecurity auch jene mit ein, die im Cyberraum funktionieren beziehungsweise agieren. Auch jene Assets, die über den Cyberraum hinweg erreicht werden können sind miteingeschlossen. Somit weist Cybersecurity einen weitaus weiteren Blickwinkel auf, als es die Information Security tut (Von Solms & Van Niekerk, 2013).

2.1.1. Grundlegende Konzepte

Im diesem Abschnitt werden grundlegende Konzepte, sowie Definitionen zu einigen ausgewählten Gefahren und Gegenmaßnahmen im Bereich von Cybersecurity näher erläutert.

2.1.1.1. Definitionen

Die folgenden Definitionen bilden das Fundament für das Verständnis der in dieser Arbeit behandelten Themen im Bereich der IT-Sicherheit (Felderer et al., 2016):

- **Vermögenswert (Asset):** Ein Vermögenswert oder Asset ist eine Systemkomponente oder Daten, die vor möglichen Angriffsversuchen geschützt werden müssen.
- **Fehler (Fault):** Die textuelle Repräsentation eines inkorrektens Teils der Verhaltensbeschreibung, welcher ausgebessert werden muss, um eine korrekte Verhaltensbeschreibung zu erhalten. Ein Fehler wird immer von einem anderen Fehler ausgelöst, aber nicht jeder Fehler muss

einen anderen Fehler auslösen, beispielsweise wenn ein Fehler in einem Code, der nie ausgeführt wird auftritt. Ein Fehler könnte beispielsweise folgend in Form eines Python-Codes aussehen (Codebeispiel 1):

```
def divide_numbers(a, b):
    result = a/b
    return result
```

Codebeispiel 1: Fehler (Felderer et al., 2016)

Aufgrund der fehlenden Überprüfung welche Werte a und b annehmen dürfen, kann bei einer potenziellen Division durch 0 ein weiterer Fehler durch diesen Fehler ausgelöst werden.

- **Schwachstelle (Vulnerability):** Ein Fehler wird dann zu einer Schwachstelle, wenn dieser Sicherheitseigenschaften eines Systems betrifft. Eine Schwachstelle ist dabei stets mit einen oder mehreren Assets und Sicherheitseinstellungen verbunden und tritt dann auf, wenn der Sicherheitsmechanismus der die Assets schützen soll fehlerhaft implementiert ist oder ganz fehlt.
- **Ausnutzung (Exploit):** Bei einem Exploit wird durch eine konkrete bösartige Eingabe die Sicherheitseigenschaft des Assets verletzt. Dabei werden die Schwachstellen des ausgewählten Assets ausgenutzt, um auf dessen zugreifen zu können beziehungsweise dessen Eigenschaften zu manipulieren.
- **Bedrohung (Threat) und Angriff (Attack):** Eine potentielle Ursache, welche die Beeinträchtigung des Assets nach sich zieht, wird als Bedrohung bezeichnet. Wird eine die Beeinträchtigung oder Wertveränderung des Assets beabsichtigt oder auch unbeabsichtigt mithilfe des Ausnutzens einer Schwachstelle in die Tat umgesetzt, handelt es sich um einen Angriff. Ein Hacker wird dabei beispielsweise als Bedrohung definiert, während der Hacking-Versuch an sich als Angriff gilt.

Im Folgendem werden jenen Konzepten dargestellt, welche aufgrund von IT-sicherheitstechnischen Maßnahmen gewahrt werden sollen (Committee on National Security, 2010):

- **Vertraulichkeit:** Soll gewährleisten, dass Informationen nicht unbefugt für nicht autorisierte Personen, Prozesse oder Geräte offengelegt werden.
- **Integrität:** Daten sollen unverändert bleiben und nicht unbeabsichtigt oder auch bösartig verändert, manipuliert oder zerstört werden.
- **Verfügbarkeit:** Soll einen zuverlässigen Zugriff auf Daten und Services für autorisierte Entitäten gewährleisten.
- **Authentifizierung:** Ist die Identitätsüberprüfung einer Person, Systeme, Prozesses oder Geräts, sowie die Prüfung der Integrität von Daten und deren Ursprung.
- **Autorisierung:** Zugriffsprivilegien welche Benutzer, Systeme, Prozessen oder Geräten gewährt werden, sowie der Gewährungsprozess selbst.

- **Nicht-Abstreitbarkeit (Non-Reputation):** Hierbei handelt es sich um die Gewährleistung, dass sowohl der Sender einer Nachricht eine Benachrichtigung, über die die Zustellung erhält, sowie der Empfänger der Nachricht Informationen über die Identität des Senders. Somit kann auch zu einem späteren Zeitpunkt keiner der beiden Parteien die Sendung der Nachricht oder deren Informationsverarbeitung abstreiten.

2.1.1.2. Gefahren

Bereits 2011 haben unter anderem die USA und Vereinigte Königreich umfassende Strategien definiert, um nationensintern einheitlich gegen die Gefahren von Cyberkriminalität vorzugehen (Von Solms & Van Niekerk, 2013). Cyberkriminalität kann verschiedene Formen annehmen, einige wichtige Vertreter dieser werden in Folge sowie im nächsten Kapitel 2.1.1.3 OWASP Top 10 Security Threats näher erläutert.

2.1.1.2.a. Malware

Bösartige Software-Varianten wie Trojana oder Viren werden als Malware bezeichnet. Sie sind so konzipiert, dass sie sich unbefugten Zugang auf einen Computer oder Systeme verschaffen beziehungsweise diese beschädigen zu können. Dabei versuchen sie den Schutz von Antiviren-Systemen zu umgehen und sich unbemerkt am Computer einzunisten (IBM, 2023). Wird Malware aktiv von dem/der Nutzer/-in erkannt ist es oft schon zu spät und statt einer Malware-Abwehr kann nur noch Schadensbegrenzung betrieben werden.

2.1.1.2.b. Ransomware

Ransomware ist eine spezielle Art der Malware. Hierbei verschaffen sich Cyberkriminelle unrechtmäßigen Zugang auf (sensiblen) Daten, und haben dessen Veränderung, Veröffentlichung oder Zerstörung als Ziel. Vom Absehen dieses Vorhaben fordern die Cyberkriminellen meist eine bestimmte Geldsumme als Lösegeld und versprechen in Folge die Daten wieder freizugeben beziehungsweise diese nicht zu veröffentlichen (IBM, 2023).

Besonders schwierig die Identität des Kriminellen zu enttarnen, wird es, wenn das Lösungsgeld in einer Kryptowährung, wie Bitcoins, verlangt wird. Eine Zahlung per Kryptowährung ist aufgrund ihres dezentralen Blockchain Charakters um ein Vielfaches anonymer als beispielsweise eine Zahlung per Banküberweisung (Kshetri & Voas, 2017). Die potentielle Bezahlung dieses Lösegelds hängt von vielerlei Faktoren ab, wie beispielsweise die Wichtigkeit und Sensibilität der Daten, den Fakt wie einfach die zerstörten Daten wiederhergestellt werden können, die Reputation des Unternehmens beziehungsweise dessen Schaden sollten Daten veröffentlicht werden (E. Cartwright et al., 2019), sowie nicht zuletzt die demographischen Daten der Entscheidungsträger. Frauen sind demnach statisch häufiger dazu bereits Lösegeld zu bezahlen als Männer (A. Cartwright et al., 2023). Expert/-innen sind sich hier aber klar einig – das Lösegeld zu bezahlen ist der falsche Weg (Mujeye, 2022) und verspricht nur anderen oder auch denselben Täter/-innen in vergleichbaren Szenarien einen ähnlichen Erfolg. Viel sinnvoller ist ein effektives Backup- und Restore-Management, Sicherheitsschulungen der Benutzer/-innen sowie andere Sicherheitsmaßnahmen, welche in Folge in Abschnitt 2.1.1.4 noch näher erläutert werden, damit es erst gar nicht zu dieser Situation kommen kann (Mujeye, 2022).

2.1.1.2.c. (D)DoS Angriff

(Distributed) Denial-of-Service oder kurz (D)DoS Angriffe sind bösartige Angriffe in denen versucht wird einen Server oder Services mithilfe von exzessiven Datenverkehr lahmzulegen, was entweder zu einer extremen Leistungsminderung oder kompletten Absturz dieser führen kann (Bawany et al., 2017). (D)DoS Attacken führen folglich dazu, dass der Server oder die Services keine legitimen Daten mehr empfangen und verarbeiten können (Dahiya & Gupta, 2021). Meist wird diese bösartige Datenflut von mehreren koordinierten Systemen gleichzeitig ausgelöst, was einen solche Angriff zu einer Distributed Denial-of-Service Attacke macht (IBM, 2023). DDoS Angriffe zählen zu den gefährlichsten sowie komplexesten Cyberattacken und sind zumeist mit großen Verlusten für das Unternehmen verbunden (Bawany et al., 2017).

2.1.1.2.d. Brute Force-Angriff

Brute Force Angriffe sind auf Versuch und Irrtum basierende Angriffe, um beispielsweise das Passwort zu einem Benutzerkontos zu knacken. Nachdem der Cyberkriminelle persönliche Informationen des/der Nutzers/Nutzerin gesammelt hat, werden auf dessen Basis zufällig Passwörter generiert und ausprobiert. Je nach Art der Verschlüsselung und Passwortrichtlinien steigt die Zahl der Möglichkeiten exponentiell was den Brute Force-Angriff zu einem sehr langwierigen Prozess machen kann (Dave, 2013). Aufgrund dessen werden Brute Force-Angriffe zumeist automatisiert ausgeführt, dabei sind sie Standart-unabhängig und sehr kostenniedrig (Abdou et al., 2015). Neben der potentiellen Gefahr des Brute-Force Angriffs durch Cyberkriminelle wird er auch oft bei Penetrationstests eingesetzt um das Sicherheitslevel der Applikation zu testen (Tian et al., 2018).

2.1.1.2.e. Advanced Persistent Threats

Anders als DDoS Angriffe die unmittelbar zu einem Zusammenbruch eines Servers oder Services führen, sind Advances Persistent Threats, kurz APTs, schleichende Angriffe deren Ziel es ist so unbemerkt Unternehmensdaten zu erhalten. Dabei wird in der Regel langsam von einem System zum anderen vorgegangen, um das Unternehmen nicht zu alarmieren (Alshamrani et al., 2019). Netzwerke und Systeme bleiben dabei intakt, um Strategien und Daten des Unternehmens ausspionieren zu können (IBM, 2023). ATPs werden in der Regel von Konkurrenzunternehmen oder anderen Sponsoren in Auftrag gegeben beziehungsweise finanziert und spionieren die Systeme so lange aus bis sie entweder entdeckt werden oder alle vom Auftraggeber verlangten Informationen eingeholt haben (Alshamrani et al., 2019).

2.1.1.2.f. Session Hijacking Angriff

Beim Session Hijacking Angriff werden legitime Sessions von Cyberkriminellen übernommen beziehungsweise abgehört, mit dem Ziel Daten zu stehlen. In der Regel bemerken die Opfer die Attacke nicht, da weder Systeme noch Netzwerke damit sichtbar verändert werden. Es wird grundsätzlich zwischen aktive, passive, und hybride Session Hijacking Angriff unterschieden. Beim aktiven Session Hijacking greifen Cyberkriminelle bereits aktive Sessions zwischen Benutzer und Server an, mithilfe eines DDoS Angriffs bringt der Täter sich so in die Rolle des/der Benutzers/Benutzerin und kann so die Datenpakete vom Server hin zum/zur Nutzer/-in abfangen und gegebenenfalls manipulieren. Beim passiven Session Hijacking gibt sich der/die Angreifer/-in als Server aus und fängt so die Datenpakete

vom Nutzer hin zum Server ab. Hybrides Session Hijacking verwendet sowohl die passive als auch aktive Variante und kann so in beide Richtungen Datenpakete abfangen und manipulieren (Baitha & Vinod, 2018).

2.1.1.2.g. Man-in-the-Middle Angriff

Der Man-in-the-Middle, oder kurz MITM, Angriff ist eine der weitverbreitetsten und erfolgversprechendsten Cyberkriminellen Handlungen im Cybersecurity-Bereich (Conti et al., 2016) und zählt zu den Session Hijacking Angriffen. Hier werden Nachrichten zwischen zwei Parteien beziehungsweise Endpunkten abgefangen, manipuliert oder an Dritte weitergeleitet um so Daten zu falsifizieren oder zu stehlen (Conti et al., 2016; IBM, 2023). Eine Form davon ist der Spoofing-Based MITM Angriff; als Spoofing wird das Vortäuschen eines legitimen Netzwerks oder Nutzer/-in bezeichnet woraufhin Angriffe wie DDoS oder MITM resultieren können. Beim Spoofing-Based MITM Angriff wird die Kommunikation zwischen zwei Host abgehört sowie der Datenverkehr kontrolliert, ohne dass die beteiligten Parteien dies bemerken (Conti et al., 2016).

2.1.1.2.h. Social Engineering (Phishing)

Social Engineering sind cyberkriminelle Attacken die, anders als DDoS- oder Man-in-the-Middle Angriffe, nicht direkt auf ein digitales System ausgeübt werden, sondern auf jenen Personen, welche mit diesem arbeiten. Dabei benutzen die Täter/-innen psychologische Tricks um über diese Personen Zugang zu den Systemen oder Daten zu erhalten (Peltier, 2006). Phishing stellt dabei eine Art des Social Engineering dar; hier werden scheinbar seriöse Emails oder Textnachrichten an die potentiellen Opfer verschickt die zur Preisgabe von sensiblen Daten auffordern (IBM, 2023).

2.1.1.2.i. Faktor Mensch

Wie schon im oberen Abschnitt beschrieben können nicht nur Systeme, Programme oder Netzwerke ein Sicherheitsschwachstelle für ein Unternehmen bedeuten, sondern auch Mitarbeiter und Mitarbeiterinnen. Hier lassen sich grundsätzlich zwei Arten unterscheiden: Personen, die unsicher oder unwissend im Umgang mit digitalen Systemen sind und so beispielsweise durch eine Phishing-Attacke Opfer von Cyberkriminalität werden und damit eine Sicherheitsrisiko für das Unternehmen bedeuten, sowie Personen die aktuell oder in Vergangenheit Zugang zu unternehmensinternen Systemen oder Netzwerken haben und dies mutwillig für bösartige Zwecke nutzen (IBM, 2023). Weiteres könnten beispielsweise ehemalige Mitarbeiter/-innen oder Geschäftspartner/-innen sein die dem Unternehmen damit schaden wollen. Diverse Aufzeichnungen zeigen, dass Sicherheitsangriffe auf Systeme und Netzwerke oft das fehlende Verständnis rund um Sicherheitsaspekte der Mitarbeiter/-innen, sowie dessen geringe Fähigkeiten und fehlende Kompetenzen in Bezug dessen, ausnutzen (Ani et al., 2019).

2.1.1.3. OWASP Top Ten

Das Open Web Application Security Project, kurz OWASP, agiert als offene Non-Profit Community und hat das Ziel, Unternehmen und Organisationen zu unterstützen vertrauenswürdige Applikationen und Serviceschnittstellen zu entwickeln und zu verwenden. Dabei werden Unternehmen und Organisationen, aber auch Privatpersonen eine Vielzahl an Ressourcen kostenlos zur Verfügung gestellt; unter anderem: Application Security Werkzeuge und Standards, Studienergebnisse, Libraries, Bücher über Security-Testing und sicheres Programmieren, Cheat-Sheets, sowie Gruppen zum Austausch, Events,

Konferenzen und dessen Videoaufzeichnungen (OWASP Foundation, 2023a). Darüber hinaus definiert OWASP Top-Ten-Security-Risks Ranglisten, welche sich auf verschiedene Bereiche der Cybersicherheit spezialisieren, wie zum Beispiel Data Security Top Ten, Mobile Top Ten, Top Ten CI/CD Security Risks, Top Ten Web Application Security Risks oder API Security Top Ten, letztere wird später noch näher beschrieben (OWASP Foundation, 2023b). Die Top Ten Web Application Security Risks wird hierbei oft nur, als OWASP Top Ten bezeichnet. Ihre einzelnen Kategorien werden im nächsten Abschnitt erläutert, um weitere Sicherheitsgrundlagen aufzuzeigen. Grundsätzlich besteht jede Kategorie der OWASP Top Ten aus mehreren sogenannten CWEs, also Common Weakness Enumerations deren Auftretungshäufigkeit gemessen und kaskadiert werden, um so zu einer Priorisierung der einzelnen Kategorien zu kommen. Jedem CEW können ebenfalls in der Regel ein oder mehrere CVEs, also Common Vulnerability and Exposures, zugeordnet werden (OWASP Top 10 Team, 2021k).

2.1.1.3.a. OWASP Top 10 2017 vs. 2021

Die OWASP Foundation definierte erstmalig 2003 die Top Ten Web Application Security Risks, welche unter anderem in den Jahren 2017 und 2021 überarbeitet wurden. In Abbildung 2 werden die Veränderung der beiden Versionen gegenübergestellt. Im Allgemeinen wurden die Prioritäten der einzelnen Security Risks neu gereiht, sowie Kategorien zusammengefügt oder neu hinzugefügt. Die Auswahl der Security Risks sowie deren Priorisierung ist vorrangig abhängig von der vorhanden Datenlage, wobei zwei der zehn Kategorien aus der Top Ten Security Umfrage der Community stammen. Grund hierfür ist, dass OWASP nicht nur auf die Datenlage, also die Vergangenheit blicken will, sondern auch in die Zukunft (OWASP Top 10 Team, 2021k).

Einige wichtige Änderungen sind hier beispielsweise (Haas, 2021):

- Die Zusammenführung von den Kategorien „Injection“ und „Cross Site Scripting“, da letzteres im Kern ebenfalls als „Injeiction“ gewertet werden kann.
- Die Einführung der Kategorie „Insecure Design“, welche Schwachstellen in der Architektur oder Design und nicht in der Implementierung selbst, betrachten. Dies ist besonders wichtig, da Schwachstellen in diesen Bereichen sehr viel schwieriger zu beheben sind, als im Implementierungsstadium.
- Die Verschiebung der Kategorie „XML External Entities“ von 2017 auf die Kategorie „Security Misconfiguration“ 2021, da erstere eine spezielle Form der Fehlkonfiguration darstellt.
- „Software und Data Integrity Failures“ als neue Kategorie, welche die Kategorie „Insecure Deserialization“ von 2017 beinhaltet.
- Das Hinzufügen der Kategorie „Server-Side Request Forgery“, welche besonders bei dem aktuellen Trend zu Microservices zunehmend an Bedeutung gewinnt.
- Aufgrund des Fokus seitens der OWASP Foundation, aber auch der vorhandenen Werkzeuge und Automatisierungsmöglichkeiten in den jeweiligen Bereichen wurde die Reihung der Kategorien und damit auch ihre Priorisierung der Kategorien angepasst. Die beiden führenden Kategorien 2017 wurden nach unten verschoben, während die Kategorien „Broken Access Control“ und „Cryptographic Failures“ 2021 die Liste anführen.

In Folge werden nun die einzelnen Kategorien des OWASP Top Ten 2021 Rankings näher erläutert.

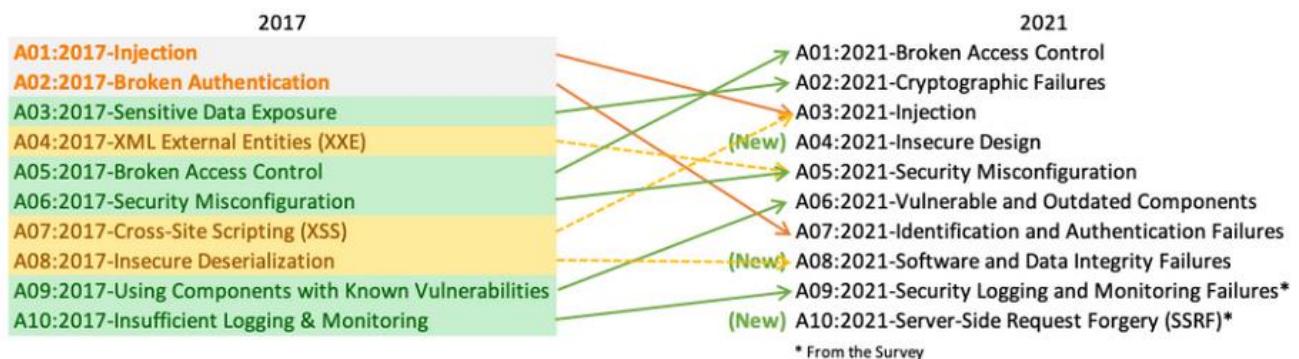


Abbildung 2: OWASP Top 10 2017-2021 (OWASP Top 10 Team, 2021)

2.1.1.3.b. Broken Access Control

Die Kategorie A01, „Broken Access Control“ steht mit 34 zugewiesenen CWEs und 318.000 gemessenen Auftritten dieser, an der Spitze der OWASP Top Ten. 19.013 CVEs können den CWEs zugeordnet werden. Das durchschnittliche Vorkommen von einem oder mehrerer CWEs in dieser Kategorie lag bei 3,81% in den getesteten Applikationen (OWASP Top 10 Team, 2021a).

Zugriffskontrollen sollen gewährleistet, dass Benutzer/-innen innerhalb ihrer zugewiesenen Berechtigungen handeln. Bei Versagen dieses Mechanismus können Informationen preisgegeben beziehungsweise manipuliert werden, sowie Nutzer/-innen ihre Berechtigungsgrenze überschreiten (OWASP Top 10 Team, 2021a). Ein Unternehmen soll nach dem Prinzip des „Principle of Least Privilege“ handeln und so Benutzer rollenbasiert nur jene Ressourcen zur Verfügung stellen die diese/r unbedingt benötigen (Cloudflare, 2023). Es sollte außerdem verhindert werden, dass Nutzer/-innen beispielsweise mithilfe des Manipulierens der URL oder Metadaten die Zugriffskontrolle umgehen und so Zugriff auf Ressourcen erhalten, die unzugänglich für den Benutzeraccount sein sollten. Darüber hinaus sollte kein/e Benutzer/-in, mit Ausnahme eines System-Administrators, den Benutzeraccount manipulieren oder einsehen können. Eine Gewährleistung, dass Privilegien wie definiert funktionieren und keine Benutzer/-in Admin-Privilegien besitzt oder nicht-eingeloggt Aktionen eines Benutzers ausführen kann, ist ebenfalls unabdingbar (OWASP Top 10 Team, 2021a).

Die bedeutensten CWEs sind hierbei: CWE-200 „Exposure of Sensitive Information to an Unauthorized Actor“, CWE-201 „Insertion of Sensitive Information Into Sent Data“ sowie CWE-352 „Cross-Site Request Forgery“ (OWASP Top 10 Team, 2021a).

2.1.1.3.c. Cryptographic Failures

Der Kategorie A02 „Cryptographic Failures“ können 29 CWEs und diesen wiederum 3.075 CVEs zugeordnet werden und weist eine durchschnittliche Inzidenz Rate von 4,49% auf, was eine Auftrittszahl von über 233.000 bedeutet (OWASP Top 10 Team, 2021b).

Für diese Kategorie ist die Frage welche Daten gespeichert und für die Geschäftsprozesse verwendet werden von besonderer Bedeutung und ob diese unter bestimmte Regulationen und Gesetze fallen, wie beispielsweise Gesundheitsdaten. Grundsätzlich sollte jede Art von sensible Daten entsprechend sicher gespeichert werden. Probleme können hier auftreten, wenn Daten ohne jede Art von Verschlüsselung versendet werden, sei es extern und intern, nicht mehr zeitgemäße oder schwache Algorithmen zur Verschlüsselung der Daten verwendet werden oder die Verschlüsselung nicht korrekt ausgeführt wird, beispielsweise durch einen fehlenden HTTP-Header. Auch die unsichere Verwendung von Schlüssel und Zertifikaten ist in dieser Kategorie enthalten, dies schließt ihre Wiederverwendbarkeit, Validierung und Stärke des verwendeten Algorithmus mit ein. Ein gutes Schlüssel- und Zertifikatsmanagement ist für eine sichere Applikation unabdingbar (OWASP Top 10 Team, 2021b).

Zu den bedeutendsten CWEs zählen hierbei: CWE-259 „Use of Hard-coded Password“, CWE-327 „Broken or Risky Crypto Algorithm“ und CWE-331 „Insufficient Entropy“ (OWASP Top 10 Team, 2021b).

2.1.1.3.d. Injection

Zur Kategorie A03 “Injection” können 33 CWEs zugeordnet werden, welche eine durchschnittliche Auftrittsrate von 3,37% in den getesteten Applikationen aufweist und mit 274.000 Vorkommen auf Platz 3 in der OWASP Top 10 nach „Broken Access Control“ und „Cryptographic Failures“ liegt. 94% aller für Tests zur Verfügung stehenden Applikationen wurde auf diese Kategorie getestet. 32.078 CVEs können den CWE's zugeordnet werden, was die Spitzenposition aller Kategorien bedeutet (OWASP Top 10 Team, 2021c).

Injection sind grundsätzlich Angriffe die dann auftreten können, wenn Cyberkriminelle Schwachstellen in Inputfelder beziehungsweise deren Validierungsprozess finden und dies dazu ausnutzen um über eine unrechtmäßige Eingabe beispielsweise Daten aus einer Datenbank preiszugeben (SQL-Injection) oder den Programmcode bösartig zu manipulieren (Code Injection) (Bach-Nutman, 2020). Es existieren eine Vielzahl an Injection-Arten die aber alle im Kerne das identische beschriebene Konzept aufweisen; laut der OWASP Foundation kommen am häufigsten die SQL, NoSQL, OS Command, Object Relation Mapping (ORM), LDAP und Object Graph Navigation Library (OGNL) oder Expression Language (EL) Injection vor. Applikationen die von einer umfassenden Validierung der eingegeben Nutzerdaten absiehen sind hier besonders gefährdet. Die Verwendung von dynamischen Queries, sowie nicht-parametrisierten Aufrufen ohne eine Prüfung der eingegeben Zeichen und deren Effekte erhöht ebenfalls das Risiko eines Injection-Angriffs. Gerade bei SQL-Injections und ORM-Injections muss darauf geachtet werden, dass die Eingabe von bösartigen Daten(-Strukturen) nicht dazu führen kann, dass sensible Daten preiszugeben oder Befehle und Prozeduren manipuliert werden können (OWASP Top 10 Team, 2021c).

Bedeutende CWEs sind in dieser Kategorie: CWE-79 „Cross-site Scripting“, CWE-89 “SQL Injection” sowie CWE-73 “External Control of File Name or Path” (OWASP Top 10 Team, 2021c).

2.1.1.3.e. Insecure Design

Die Kategorie A04 “Insecure Design” umfasst 40 CWEs welche 2.691 CVEs beinhalten und weisen eine durchschnittliche Auftrittsrage von 3%, bei einem Vorkommen von 262.000 in den getesteten Applikationen auf (OWASP Top 10 Team, 2021d).

Die OWASP Foundation unterscheidet klar zwischen einem unsicheren Design und einer unsicheren Implementierung. Ein sicheres Design kann trotzdem Sicherheitslücken aufweisen, wenn die Implementierung nicht korrekt durchgeführt wurde, ein unsicheres Design kann jedoch selbst durch eine perfekte Implementierung nicht sicher gemacht werden. Ein Grund kann beispielsweise das Fehlen einer angemessenen Geschäftsrisikobewertung sein, die zu unsicherem Design führt, da nicht erkannt wird, welches Sicherheitsniveau erforderlich ist. Es ist deswegen wichtig einen weiteren Blickwinkel einzunehmen und auch die Design- und Architekturprozesse vor der eigentlichen Implementierung in den Fokus zu stellen (OWASP Top 10 Team, 2021d).

CWE-209 “Generation of Error Message Containing Sensitive Information”, CWE-256 “Unprotected Storage of Credentials”, CWE-501 “Trust Boundary Violation” und CWE-522 “Insufficiently Protected Credentials” sind hierbei bedeutende CWEs (OWASP Top 10 Team, 2021d).

2.1.1.3.f. Security Misconfiguration

Die Kategorie A05 “Security Misconfiguration” wurde bei 90% aller Applikationen getestet und erreichte hier mit 208.000 Auftritten, eine durchschnittliche Vorkommensrate von 4,5% und kann so eine höhere Priorität als im Jahr 2017 verzeichnen. Sie umfasst 20 CWEs welche wiederum insgesamt 789 CVEs enthalten (OWASP Top 10 Team, 2021e).

Eine Applikation ist in dieser Kategorie besonders gefährdet, wenn nicht verwendete Features, wie Ports, Services oder Accounts, aktiviert oder installiert sind, sowie Standardaccounts und deren Passwörter nicht ordnungsgemäß geändert oder deaktiviert werden. Auch das Fehlen einer korrekten Fehlerbehandlung kann hier zur Sicherheitslücke werden, wenn beispielsweise wichtige Informationen in den Fehlernachrichten enthalten sind, die den Cyberkriminellen bei weiteren Angriffen hilfreich sein können. Auch am Applikationsserver, der verwendeten Datenbank, den Libraries und Frameworks müssen die Sicherheitseinstellungen korrekt konfiguriert sein, sowie bei Updates der bestehende Schutz gewährleistet sein. Darüber hinaus ist der Einsatz von Sicherheitsheadern und korrekt gesicherten Werten beim Applikationsserver von Bedeutung (OWASP Top 10 Team, 2021e).

Bedeutsame CWEs in dieser Kategorie sind beispielsweise CWE-16 „Configuration“ und CWE-611 „Improper Restriction of XML External Entity Reference“ (OWASP Top 10 Team, 2021e).

2.1.1.3.g. Vulnerable and Outdated Components

Kategorie A06 “Vulnerable and Outdated Components” wurde einerseits aufgrund der Community Umfrage Ergebnissen, aber auch den datenanalytischen Gesichtspunkten in die OWASP Top Ten aufgenommen. Aufgrund der Schwierigkeit der Testbarkeit dieser Kategorie können dessen 3 zugeordneten

CWEs, keine CVEs gemappt werden. Da die Schwachstelle nur in etwa 51% aller Applikationen getestet wurde, ist die durchschnittliche Auftrittsrate mit 8,77% sehr hoch, obwohl die Zahl des Vorkommens nur 30.457 ausmacht. Ein bedeutende CWE dieser Kategorie ist CWE-1104 “Use of Unmaintained Third-Party Components” (OWASP Top 10 Team, 2021f).

Diese Kategorie ist besonders relevant, wenn Unwissenheit über die Komponentenversion herrscht, sowie alte oder nicht mehr unterstützte Versionen verwendet werden, auch wenn diese nur über nested Dependencies genutzt werden. Plattformen, Frameworks und Dependencies sollen stets am neusten Stand gehalten, sowie verfügbare Patch installiert werden um die Angriffsfläche für Cyberkriminelle so gering wie möglich zu halten. Auch die Kompatibilität zwischen verschiedenen Versionen oder gepatchten Libraries muss gewährleistet sein (OWASP Top 10 Team, 2021f).

2.1.1.3.h. Identification and Authentication Failures

Der Kategorie A07 “Identification and Authentication Failures” umfasst 22 CWEs welche wiederum 3.897 CVEs beinhaltet. Die Inzidenz liegt mit 132.195 Vorkommnissen der CWEs beziehungsweise deren CVEs bei einer durchschnittlichen Rate von 2,55% aller getesteten Applikationen (OWASP Top 10 Team, 2021g).

Um eine Applikation vor Cyberangriffen im Bereich von Authentifizierung entgegenzuwirken, ist die Bestätigung der korrekten Identität des Nutzers, sowie Management der Sessions wichtig. Schwachstellen in diesem Bereich können beispielsweise den Cyberkriminellen die Möglichkeit zu Brute Force Angriffen oder anderen automatisierten Angriffen geben. Auch das Fehlen von Passwortrichtlinien, unsicherer Prozesse zum Passwort-Zurücksetzen, fehlende Multi-Faktoren-Authentifizierung oder die Speicherung des Passworts ohne zureichenden oder fehlenden Einsatz von Verschlüsselung oder Hashing können Angriffe für die Cyberkriminelle erleichtern. Die Preisgabe von Session IDs, deren Wiederverwendung nach Log-In oder fehlende Invalidierung nach Log-Out können in dieser Kategorie ebenfalls ein Sicherheitsrisiko darstellen (OWASP Top 10 Team, 2021g).

Wichtige CWEs in dieser Kategorie sind CWE-297 “Improper Validation of Certificate with Host Mismatch”, CWE-287 “Improper Authentication” und CWE-384 “Session Fixation” (OWASP Top 10 Team, 2021g).

2.1.1.3.i. Software and Data Integrity Failures

Die Kategorie A08 “Software and Data Integrity Failures” wurde in 75% aller Applikationen getestet, wobei es hier zu 1.152 Vorkommnissen und einer Auftrittsrate von 2,05% gekommen ist. Der Kategorie können 10 CWEs zugeordnet werden, welche 1.152 CVEs beinhalten (OWASP Top 10 Team, 2021h).

“Software and Data Integrity Failures” sind mit dem Code oder Infrastruktur jener Applikationen verbunden, welche nicht ordnungsgemäß gegen Integritätsverletzungen geschützt sind. Dies ist besonders dann der Fall, wenn Applikationen von Plugins, Libraries oder Modulen abhängig sind, dessen

Ursprung nicht vertrauenswürdig beziehungsweise unbekannt ist. Cyberkriminelle können beispielsweise mit dem Einsatz von bösartigen Programmcodes über diese Quellen versuchen auf Daten im Unternehmen zuzugreifen oder andere Cyberattacken durchzuführen (OWASP Top 10 Team, 2021h).

Bedeutsame CWEs in dieser Kategorie beinhalten CWE-829 “Inclusion of Functionality from Untrusted Control Sphere”, CWE-494 “Download of Code Without Integrity Check”, sowie CWE-502 “Deserialization of Untrusted Data” (OWASP Top 10 Team, 2021h).

2.1.1.3.j. Security Logging and Monitoring Failures

Die Kategorie “Security Logging and Monitoring Failures” wurde aufgrund der hohen Reihung in der Community Umfrage in die OWASP Top Ten aufgenommen. Sie weist große Schwierigkeiten in der Testbarkeit auf und kann grundsätzlich nicht gut von CVEs abgedeckt werden, was die Bewertung anhand der vorhanden Datenlage sehr schwierig macht. Demnach konnten nur etwa 53% aller Applikationen auf diese Schwachstelle getestet werden. Dabei kam es zu einer durchschnittlichen Auftrittsrate von 6,51% und einem Vorkommen von insgesamt 53.615 CWEs beziehungsweise CVEs. Der Kategorie können 4 CWEs sowie wiederrum 242 CVEs zugeordnet werden (OWASP Top 10 Team, 2021i).

Korrektes und umfassendes Logging und Monitoring ist essenziell, wenn Sicherheitslücken aufgedeckt werden sollen. Ereignisse wie fehlgeschlagene Log-In-Versuche, wichtige Transaktionen, Warnungen oder Fehler müssen unbedingt aufgezeichnet werden, da sie wichtige Indikatoren für Sicherheitsschwachstellen sind. Probleme treten dann auf, wenn dies nicht ordnungsgemäß, unklar oder nur lokal passiert beziehungsweise keine klarer Benachrichtigungs- oder Eskalationsprozess existiert (OWASP Top 10 Team, 2021i).

CWE-778 “Insufficient Logging to include”, CWE-117 “Improper Output Neutralization for Logs”, CWE-223 “Omission of Security-relevant Information” und CWE-532 “Insertion of Sensitive Information into Log File” werden der Kategorie zugeordnet (OWASP Top 10 Team, 2021i).

2.1.1.3.k. Server-Side Request Forgery

Die Kategorie “Server-Side Request Forgery” wurde seitens der Community als bedeutsamste Web Application Security Risks eingestuft. Auch wenn die Datenlage diesen Trend noch nicht validieren konnte, entschied sich die OWASP Foundation den Community Mitgliedern hier Gehör zu schenken und die Kategorie in die OWASP Top Ten aufzunehmen. Die Kategorie umfasst das CWE-918, mit identem Namen wie die Kategorie selbst, 385 CVEs können dieser zugeordnet werden. 67% aller Applikationen wurden auf diese Sicherheitsschwachstelle getestet und ein durchschnittliche Inzidenzrate von 2,72%, sowie 9.503 Vorkommnisse verzeichnet (OWASP Top 10 Team, 2021j).

Server-Side Request Forgery, oder kurz SSRF, kann dort auftreten, wo Web Applicationen Ressourcen remote aufrufen, ohne zuvor die URL zu prüfen. In diesen Fall haben Cyberkriminelle die Möglichkeit die URL so zu manipulieren, dass Requests der Applikation an ein unerwartetes Ziel gesendet und

so beispielsweise sensitive Daten an Angreifer/-innen preisgegeben werden (OWASP Top 10 Team, 2021j).

2.1.1.4. Gegenmaßnahmen

2.1.1.4.a. Testing

Security Tests prüfen Sicherheitsanforderungen in Bezug auf die bereits im Kapitel 2.1.1.1 erläuterten Prinzipien Vertraulichkeit, Integrität, Verfügbarkeit, Authentifizierung, Autorisierung und Nicht-Abstreitbarkeit. Sie sollen dabei feststellen, ob und inwieweit festgelegte Sicherheitsstandards und definierte Sicherheitseigenschaften wirksam sind und wie beabsichtigt implementiert wurden (Felderer et al., 2016). Nach Tian-Yang et al (2010) kann Security Testing in zwei verschiedene Arten untergliedert werden: Das Sicherheitsfunktionstests und Sicherheitsschwachstellentests. Sicherheitsfunktionstests sollen gewährleisten das die in den Sicherheitsanforderungen festgelegten Anforderungen erfüllt und korrekt implementiert wurden. Sicherheitsschwachstellentests versuchen im Gegensatz dazu Schwachstellen aufzudecken, wie beispielsweise Mängel in der Implementierung eines Systems. Hierbei agieren sie zumeist wie potentielle Angreifer/-innen (Tian-yang et al., 2010). Für weitere Erläuterung im Bereich API-Testing wird an dieser Stelle auf Tian-yang et al. (2010) verwiesen.

Ebenfalls im Kontext von Sicherheitstesting relevant sind die beiden Teststrategien Dynamic Application Security Testing und Static Application Security Testing oder kurz DAST und SAST. DAST prüft dynamisch laufenden Anwendungen auf Schwachstellen, ohne auf den Quellcode der Applikation zugreifen zu müssen was die Strategie als Black-Box-Testing klassifiziert. Dabei führt das DAST-Werkzeug bösartige Angriffe auf das Zielsystem aus und kann so potenzielle Sicherheitsschwachstellen identifizieren. Dem Gegenübergestellt analysiert SAST den Quellcode oder Binärdaten einer Applikation um schon während der Entwicklung und nicht erst zur Laufzeit um frühzeitig Schwachstellen aufdecken zu können. SAST verwendet demnach White-Box-Testing-Techniken, um auf Basis des Quellcodes und die Binärdaten statische Sicherheitsanalysen durchführen zu können. Aus der Kombination dieser beider Strategien kann eine umfassende Sicherheitsabdeckung für Systeme und Applikationen erzielt werden (Sharma, 2021).

2.1.1.4.b. Identitäts- und Zugriffsmanagement

Um einen sicheren Zugang für im Cyberraum agierenden Personen zu schaffen, speziell jenen, die im Unternehmen oder dessen Umfeld tätig sind, ist das sogenannte Identitäts- und Zugriffsmanagement, kurz IAM, wichtig. Der Zugriff auf die Ressourcen sollte so geregelt sein, dass die Mitarbeiter/-innen darauf wenn benötigt zugreifen können, es aber Unbefugten nicht möglich (Microsoft, 2023).

Zum IAM zählen unter anderem die Multifaktor-Authentifizierung oder zweistufige-Authentifizierung, also eine zwei- oder mehrfache Bestätigung der Identität, oft auf unterschiedlichen Geräten, was die einfache Anmeldung mit Benutzername und Kennwort in Bezug auf Sicherheit um ein Vielfaches übersteigt (Microsoft, 2023). Außerdem enthalten sind Konzepte wie Single Sign-On, oder ausgewählte privilegierte Benutzerkonten, welche über Administratorrechte verfügen oder ein klar definiertes Benutzer-Lebenszyklus Management. Während Single Sign-On dafür sorgt, dass Benutzer während einer Sitzung kontinuierlich eingeloggt bleiben, werden im Benutzer-Lebenszyklus Managements Benut-

zertkonten von dessen Erstellung im System bis hin zur Löschung einer konstanten Verwaltung unterzogen (IBM, 2023). Diese Konzepte dienen im Allgemeinen der Identitätsverwaltung, das IAM ist aber darüber hinaus auch für die Zugriffsverwaltung verantwortlich. Nachdem die Identität eines Benutzers validiert werden konnte, muss auch kontrolliert werden welches Benutzerkonto auf welche Ressource zugreifen darf und welche Aktionen mit dieser Ressource ausgeführt werden dürfen. Im Regelfall wird diese Berechtigung unternehmensintern auf verschiedenen Ebenen, abhängig von Sicherheitsstufen und Projektverantwortlichkeiten, vergeben (Microsoft, 2023).

2.1.1.4.c. Zero-Trust Framework

Der Zero-Trust Ansatz, verfolgt die Annahme, dass sich ein potenzieller Angreifer/-innen innerhalb des unternehmenseigenen Cyberraums befindet und somit die zu schützenden Assets vor unternehmensbekannten, wie auch vor unternehmensunbekannten Entitäten gleichermaßen geschützt werden müssen. Das Unternehmen gewährt somit kein impliziertes Vertrauen gegenüber bekannten Entitäten. Jeder Zugriff dieser Entität, sei es Benutzer, Gerät oder Anwendung, wird auf jene beschränkt die unbedingt notwendig sind. Auch die kontinuierliche Authentifizierung und Autorisierung ist Teil des Zero-Trust-Frameworks (Stafford, 2020).

Im Gegensatz zum bereits 2.1.1.3.b erwähnten „Principle of least Privileg“ (POLP) konzentriert sich das Zero-Trust-Framework auf die Autorisierung, während sich POLP auf die Zugriffskontrolle an sich fokussiert. Der Zero-Trust Ansatz ist hierbei sehr viel umfassender und betrachtet neben der Identität des Zugreifers, auch welche Assets betroffen sind und wägt das Risiko bei Zugriff ab. Die beiden Frameworks schließen sich aber keineswegs aus und können sehr gut zusammen arbeiten um eine sichere Systemarchitektur im Unternehmen zu schaffen (Froehlich, 2022).

2.1.1.4.d. Security by Design

Unter Security by Design versteht man in der Soft- aber auch Hardwareentwicklung ein Designkonzept, welches die Sicherheit des Systems, der Anwendung oder des Produkts über den gesamten Lebenszyklus hinweg inklusive dessen Entwicklungsprozess berücksichtigt. Bereits im Designprozess, bevor es zur eigentlichen Implementierung kommt, werden Sicherheitsanforderungen definiert und deren Umsetzung initiiert. Auch das kontinuierliche Testing der definierten Sicherheitsanforderungen sind Teil des Designansatzes. Ziel ist es schon frühzeitig Sicherheitsschwachstellen und somit die Angriffsfläche für potenzielle Cyberkriminelle zu minimieren und damit die Produkte sicherer und robuster für Benutzer/-innen aber auch dem Unternehmen selbst machen. Auch eine Kostensenkung ist eine positive Nebeneffekt dieses Designansatzes, da kostspielige sicherheitsrelevante Veränderungen zu einem späteren, oder gar zu späten, Zeitpunkt am Produkt oder System wegfallen (Luber & Schmitz, 2021).

2.1.1.4.e. Faktor Mensch

Auch die besten technologischen Sicherheitslösung und umfassendste Sicherheitskonzepte haben wenig Erfolg, wenn die Menschen, welche mit den Systemen arbeiten nicht die Wichtigkeit der Sicherheitsanforderungen erkennen und durchsetzen. Das mangelnde Wissen und Verständnis beim Thema Sicherheit der Mitarbeiter/-innen ist ebenso kritisch wie die fehlende Kompetenz bei der Umsetzung dieser Sicherheitsanforderungen. Es ist also essenziell dieses Wissen aktiv zu fördern, personelle

Schwachstellen zu identifizieren und diese durch spezielle Schulungen zu beseitigen. Dieser Prozess soll als kontinuierliche Schleife in das Unternehmen integriert werden, damit kein/e Mitarbeiter/-in den Anschluss verliert – ein Unternehmen ist nur so sicher wie sein schwächstes Glied. Nur durch kontinuierliche Sicherheitsbewertungen inklusive des Faktors Mensch kann ein Unternehmen ein robustes und Resilienz-fähiges Sicherheitskonzept ausarbeiten und durchsetzen (Ani et al., 2019).

2.2. Kategorisierung der Sicherheitswerkzeuge

In diesem Kapitel wird die Kategorisierung von Sicherheitswerkzeugen thematisiert mithilfe dessen sichere Webapplikationen geschaffen und betrieben werden können. Diese Kategorisierung dient dazu, einen strukturierten Überblick über die verschiedenen aktuell verfügbaren Werkzeuge zu schaffen, ihre Funktionsweisen und Einsatzmöglichkeiten zu verstehen und eine Grundlage für weiterführende Vergleiche und Analysen zu legen.

2.2.1. Methodik

Die Werkzeuge werden aus zwei primären Quellen (Kritikos et al., 2019; Tesauro, n.d.), sowie OWASP-Ressourcen (OWASP Foundation, n.d.; Wichers, n.d.) gesammelt und gegenübergestellt, um in Folge eine Kategorisierung unterzogen zu werden. Die Kategorisierung erstreckt sich primär auf die Funktionsweise, wie die verwendeten Teststrategien, der einzelnen Werkzeuge, sowie sekundär auf die Verfügbarkeit dieser. Hier wird zwischen kommerziellen und Open-Source-Werkzeugen unterschieden. Auch die Einsatzfelder oder Spezifikationen einzelner Werkzeuge wird in die Kategorisierung aufgenommen. Eine Visualisierung zur Aufbereitung der Daten wird in Anschluss vorgenommen. Von einer Bewertung der Werkzeuge wird in dieser Kategorisierung aber Abstand genommen, es sollen lediglich gemeinsame Kategorien gefunden und aufgezeigt werden, sowie eine Basis für den später folgenden Vergleich der API-Sicherheits-Werkzeuge und dem Security-Analysis-Dashboard geschaffen werden.

2.2.2. Ergebnis

Es wurden insgesamt 74 Werkzeuge aus den genannten Quellen gesammelt und kategorisiert. Dabei wurden die folgenden Kategorien verwendet, welche ein oder mehrfach an die gesammelten Werkzeuge vergeben werden konnten.

Die Kategorien umfassen:

- SAST: Static Application Security Testing (White-Box-Testing)
- DAST: Dynamic Application Security Testing (Black-Box-Testing)
- API-Scanner: Scanning-Werkzeug, um Schwachstellen von APIs aufzudecken
- Spezifischer Vulnerability Scanner: Spezifischere Schwachstellenscanner für ganze Applikationen oder APIs, beispielsweise zum Erkennen von SQL-Injections, spezifische Schwachstellen in Docker Images, oder ähnliches.
- API-Testing: Werkzeuge zum spezifischen Testen von Serviceschnittstellen
- API-Security: Sicherheitsplattformen zum Überwachen von Serviceschnittstellen

- Fuzzer: Werkzeuge, die zufällige oder unerwartete Daten in das System einspeisen, um Schwachstellen aufzudecken
- Netzwerk-Scanner: Scannen von Netzwerken, um mögliche Angriffspunkte in der Netzwerkinfrastruktur zu erkennen

In Bezug auf die allgemeine Verteilung der Kategorien konnte folgendes Bild erhoben werden (siehe Abbildung 3): API-Scanner konnten bei den gesammelten Werkzeugen am häufigsten mit über 20 Nennungen in dieser Kategorie festgestellt werden, knapp gefolgt von DAST-Werkzeugen. Doch auch Monitoring-Werkzeuge für Serviceschnittstellen, im Zuge der Kategorisierung als „API-Security“ angegeben, konnten oft kategorisiert werden.

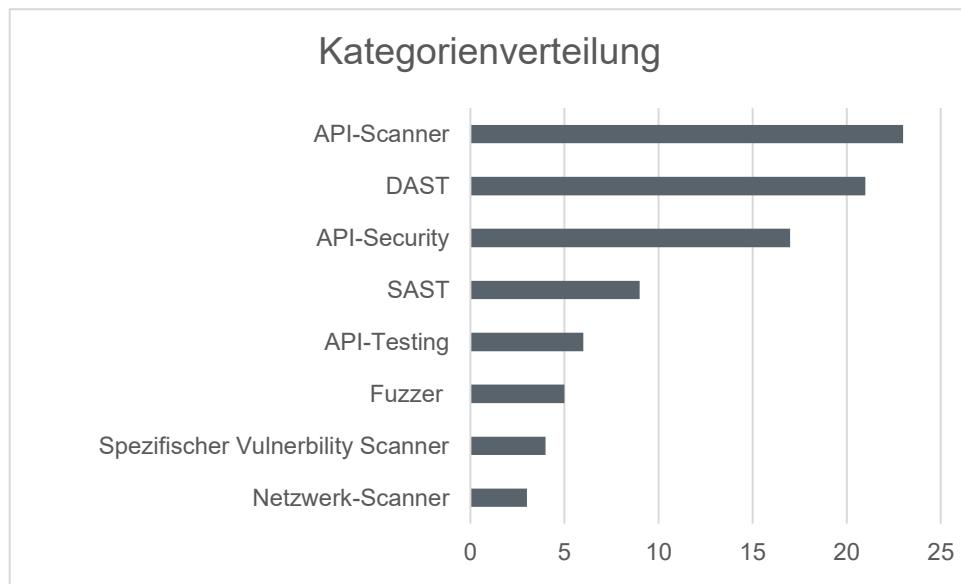


Abbildung 3: Kategorienverteilung Security-Werkzeuge

Im Bereich der Verfügbarkeit der Security-Werkzeuge, war mit 64% der Großteil der Werkzeuge kommerziell und nicht Open-Source verfügbar (siehe Abbildung 4). Einige der kommerziellen Werkzeuge wiesen jedoch kostenfreie zeitlich oder ressourcentechnische begrenzte Versionen auf.

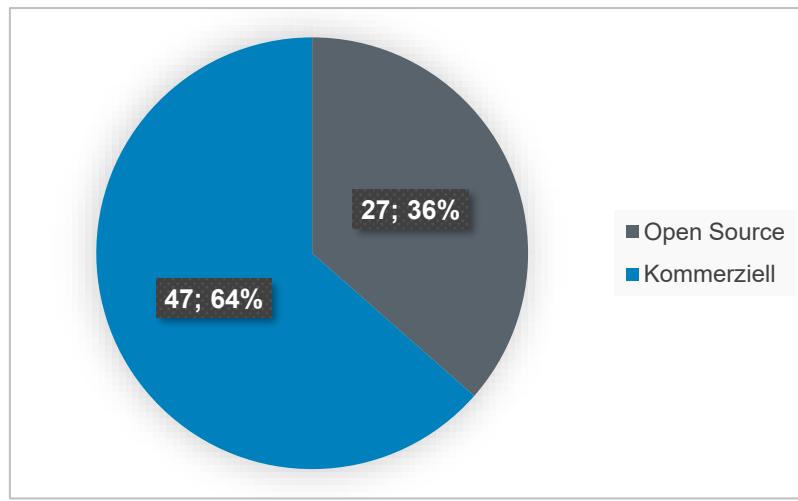


Abbildung 4: Zugänglichkeit Security-Werkzeuge

Wie schon bei der allgemeinen Verteilung der Kategorien sichtbar können eine Vielzahl der gesammelten Werkzeuge ein Fokus im Bereich API zugeordnet werden. Dies unterstreicht die Annahme, das API eine große Priorität im Bereich Cybersecurity aufweist. Wie in Abbildung 5 sichtbar konnten 70% unter anderem einer Kategorie aus den Bereichen API-Security, API-Scanner, Fuzzer, API-Tester oder eine spezifische Schwachstelle im Bereich APIs zugeordnet werden. Die Verteilung der Kategorien innerhalb des Fokus Service-Schnittstellen werden in Abbildung 6 noch detaillierter aufgelistet.

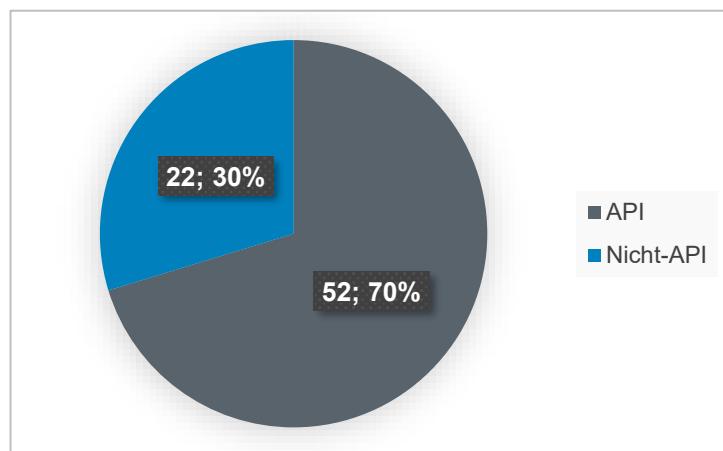


Abbildung 5: APIs Security-Werkzeuge

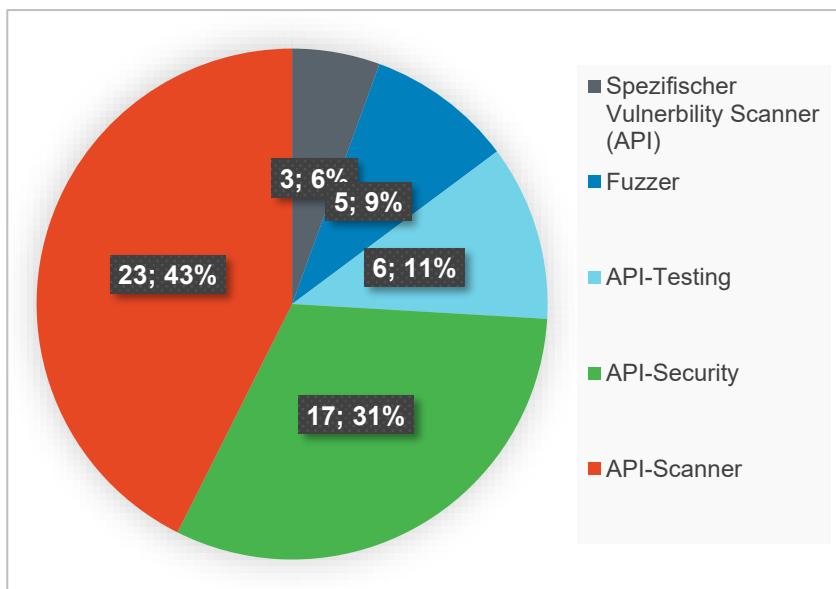


Abbildung 6: Kategorienverteilung APIs Security-Werkzeuge

Für den weiteren Verlauf der Masterarbeit, welche final die Implementierung eines API Security Analysis Dashboards als Ziel hat, sind besonders API-Scanner von Bedeutung, weswegen hier in Abbildung 7 noch einmal eine Aufstellung der Zugänglichkeit der einzelnen Scanner-Werkzeuge für Serviceschnittstellen aufgezeigt wird. Auch hier sind über die Hälfte aller gesammelten API-Scanning-Werkzeuge kommerzieller Natur. Somit bleiben 11 Open-Source API-Scanning-Werkzeuge, welche für den Vergleich sowie tiefgehenden Analysen in Kapitel 3 in Frage kommen.

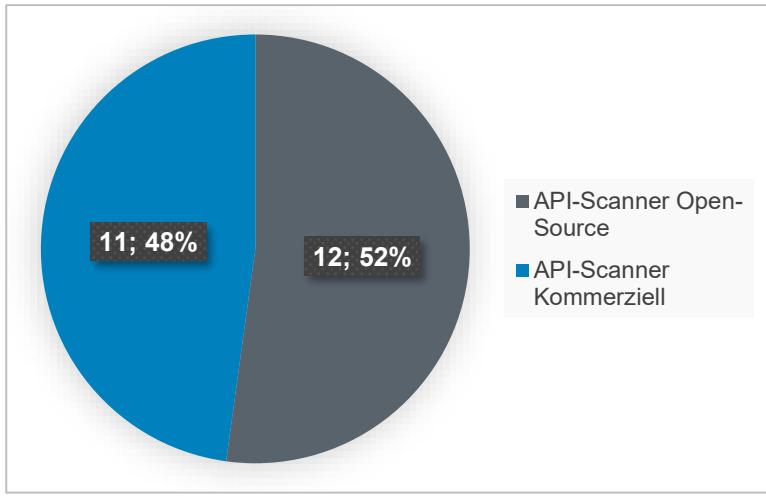


Abbildung 7: API Scanner Security-Werkzeuge

Die gesamten Kategorisierungsdaten sind in Anhang A beigefügt.

2.3. Service-Schnittstellen (APIs)

2.3.1. Grundkonzepte

Application Programming Interfaces, oder kurz APIs, sind Programmierschnittstellen beziehungsweise Serviceschnittstellen, deren Aufgabe darin besteht, Informationen zwischen einzelnen Teilen eines Programms auszutauschen. APIs ermöglichen es so ein Programm verstärkt zu modularisieren und den Programmcode besser zu strukturieren und vereinfachen. Dabei ist eine API nur für die Bereitstellung und Rücksendung der Daten der einzelnen Programmteile verantwortlich, die Logik innerhalb der Module ist für die Programmierschnittstelle nicht von Bedeutung (Luber, 2017).

APIs können sowohl innerhalb einer Anwendung als auch zwischen mehreren Anwendungen eingesetzt werden. Für Softwareentwickler/-innen liegt der Fokus zumeist auf Schnittstellen zu externen Systemen um bestehende Softwarelösungen integrieren zu können (Frank et al., 2021). Dabei lässt sich die Programmierschnittstelle klar von einer Benutzerschnittstelle, dem Userinterface, unterscheiden. Während letzteres die Brücke zwischen der Logik innerhalb des Programms und den Anwendern bildet, kümmert sich die Programmierschnittstelle um die Schnittstelle zwischen der Programmlogik und dem maschinenlesbaren Code (Luber, 2017).

APIs ist nicht nur ein technisch sehr vielseitig einsetzbares Konzept, auch für Unternehmen ergeben sich hier große Vorteile. Durch Programmierschnittstellen können Geschäftseinheiten miteinander verbunden werden, sowie Funktionalitäten so geteilt werden um im Unternehmen effektiver arbeiten zu können (Medjaoui et al., 2021). Ein weiterer Vorteil ergibt sich durch die bereits genannte Modularität, welche der Einsatz von APIs impliziert, aufgrund der Aufteilung des Programmcodes in verschiedene kleinere logische Module wird nicht nur die Lesbarkeit und Nachvollziehbarkeit der Prozesse innerhalb der Anwendung gesteigert, auch die Wartbarkeit wird erhöht, während die Anfälligkeit für Fehler gemindert wird. Des Weiteren eröffnet eine saubere und einheitliche Dokumentation einer jeden API die

Möglichkeit etwaige Programmertätigkeiten auslagern zu können, sowie die Flexibilität und Erweiterbarkeit der APIs so zu gewährleisten (Luber, 2017). Aufgrund dessen ist eine gute Dokumentation unerlässlich und wird auch schon von einigen Werkzeuge, wie beispielsweise Swagger (SmartBear Software, 2024), unterstützt, um den Softwareentwickler/-innen eine schnelle, einfache und einheitliche Art der Dokumentation zu ermöglichen, die auch andere Entwickler/-innen problemlos verstehen können. Auch der Einsatz von standardisierten Spezifikationen wie OpenAPI (2022) erleichtert das Verständnis, sowie die Test- und Wartbarkeit einer Serviceschnittstelle.

Durch den Einsatz von Programmierschnittstellen können außerdem die Stabilität eines System auf lange Sicht gesichert werden, da sich die Schnittstellen trotz Veränderungen in der Logik innerhalb der Module nicht verändern müssen und weiterhin die Programmteile reliabel miteinander verbinden (Luber, 2017). So können Kosten und Aufwand eingeschränkt werden (Luber, 2017) und Ressourcen an anderen Stellen eingesetzt werden um beispielsweise die Applikation weiterzuentwickeln.

In den folgenden Abschnitten wird der Fokus auf Web-APIs gelegt, also APIs die über das Internet angesteuert werden können, da dieser Anwendungsbereich der APIs für die vorliegende Masterarbeit von besonderer Bedeutung ist.

2.3.2. Arten von APIs

APIs lassen sich nach Luber (2017) grundsätzlich in folgende vier Typen einteilen: datenorientierte APIs, funktionsorientierte APIs, protokollorientierte APIs und objektorientierte APIs. Dateiorientierte APIs adressieren einzelne Dateien und dessen Funktionen mithilfe von Dateisystemaufrufe, während funktionsorientierte APIs über Funktionen und deren Parameter kommunizieren. Im Gegensatz zu funktionsorientierten APIs sind protokollorientierte Programmierschnittstellen unabhängig von äußeren Variablen, wie die Hardware oder dem Betriebssystem. Objektorientierte APIs bestechen im Gegensatz zu funktionsorientierten APIs mit ihrer Flexibilität und setzen daher auf Schnittstellenzeiger (Luber, 2017), einem Konzept aus der objektorientierten Programmierung, welches es ermöglicht auf die in der Schnittstelle definierten Methoden zuzugreifen, und so Flexibilität und Interoperabilität zwischen verschiedenen Klassen, die dieselbe Schnittstelle implementieren, zu bieten.

Anders als Luber, schlagen Jacobson et al. (2012) eine Differenzierung nach Zugänglichkeiten einer API vor. Eine API kann demnach entweder öffentlich (public) oder geschlossen (private) zugänglich sein. Dabei machen private APIs die Mehrheit aus und sind nur für jene Personen beziehungsweise Unternehmen zugängliche, welche vertragliche Übereinkünfte mit dem API-Provider getroffen haben. Auch Programmierschnittstellen welche ausschließlich innerhalb einer Applikation oder Unternehmen verwendet werden können, werden als private APIs definiert. Öffentliche Programmierschnittstellen sind für jede Person frei zugänglich, meist ohne oder mit nur sehr geringfügiger vertraglicher Abstimmung (Jacobson et al., 2012). Preibisch et al. (2018) widerspricht der Einteilung der APIs nach ihrer Zugänglichkeit und plädiert auf einer Einteilung von APIs nach Usecase, sowie den beabsichtigten Konsumenten. Ein B2B Unternehmen mit Fokus auf Finanzen benötigt demnach eine völlig andere API als ein B2C Softwareunternehmen. Von einer Kategorisierung von APIs nach festen Kriterien sieht Preibisch et al. (2018) ab.

Anders als von ihrer Zugänglichkeit oder Verwendungszweck, können APIs auch in Implementierung unterschieden werden. Zwei Ansätze werden hierbei in den nächsten beiden Abschnitten näher erläutert, sowie konkrete Beispiele gegeben. Die konkreten Beispiel-APIs basieren auf der im 2023 State of the API Report von Postman (Postman, 2023) veröffentlichten Rangliste der meistverwendeten Serviceschnittstellen Request-Response APIs.

2.3.2.1. Request-Response APIs

Request-Response Programmierschnittstellen stellen im Regelfall eine auf HTTP-basierten Webserver als Schnittstelle bereit und definieren eine Reihe von Endpunkten, welche Seitens des Clients über Anfragen angesprochen werden können. Der Server liefert im Gegenzug Daten, meist in den Formaten JSON oder XML, zurück (Jin et al., 2018). In Folge werden nun die drei gängigsten Architekturen erläutert.

2.3.2.1.a. REST

Der Representational State Transfer, oder kurz REST, ist eine der meistverwendete Architekturen im Bereich der Web-APIs und wurde von Fielding (Fielding, 2000) erstmals vorgestellt. Eine REST-Architektur weist grundsätzlich nach Doglio (2015) unter anderem folgende Merkmale auf:

- Die Netzwerk-Architektur, deren eine RESTful API zu Grunde liegt ist eine Client-Server. Der Server bietet hierbei Services an und steht für die Requests seitens der Clients zur Verfügung, welche einen dieser Services ansprechen. Dieser Netzwerk-Architekturstil erlaubt es nach dem Prinzip „Separation of Concern“ zu arbeiten und so eine Teilung zwischen dem Code, welcher für die Repräsentation der Daten zuständig ist (Client) und dem Code der für die Datenverarbeitung und -speicherung (Server) notwendig ist. Dies führt zu einer erhöhten Flexibilität und Unabhängigkeit dieser beiden Komponenten.
- Die Kommunikation zwischen Client und Server geschieht zustandslos, das bedeutet, dass jede Kommunikation unabhängig von der Vorherigen sein muss. Das setzt voraus, dass in jedem clientseitigen Request alle Informationen beinhaltet sein müssen, welche der Server zur Verarbeitung des Requests benötigt. Dies Zustandslosigkeit weist gerade in den Bereichen Monitoring, Skalierung, Reliabilität, sowie einfacheren Implementierung große Vorteile auf.
- Die einheitliche Schnittstelle oder „uniform interface“ ist ebenfalls ein Merkmal der REST-Architektur. Diese Charakteristika vereinfacht es den Clients auf die Schnittstelle zugreifen zu können beziehungsweise mit dem Server interagieren zu können und gibt den verschiedenen Clients klare Regeln vor, um dessen Implementierung zu erleichtern.
- Der Verwendung des Konzepts von verschiedenen Schichten vereinfacht die Komplexität maßgeblich. Jede Schicht darf in diesem Konzept nur die Schicht darunter verwenden und die Ergebnisse der eigenen Prozesse in die darüberliegende übergeben. Außerdem werden die Skalierbarkeit und Interoperabilität gesteigert.

Der Kern einer jeden RESTful-API sind Ressourcen. Dabei kann jede Form von Information oder Daten eine Ressource sein, sei es ein Bild, ein Text oder eine Webpage. Ressourcen sollen stets eindeutig zuordbar sein und können verschieden strukturiert werden. Sie teilen sich in Repräsentationen, also die repräsentierten Daten zum Beispiel in einem JSON oder XML Format, Identifikatoren, eine URI

(unique resource identifier) der eindeutigen Ressource, Metadaten, welche den Content-Type und Zeitstempel wie das letzte Änderungsdatum beinhalten, sowie Kontrolldaten die beispielsweise Informationen darüber enthalten ob ein cacheing durchgeführt werden soll oder nicht, auf (Doglio, 2015).

2.3.2.1.b. SOAP

SOAP, oder Simple Object Access Protocol, ist ein Nachrichtenprotokoll welches eine Kommunikation via XML Nachrichten über Netzwerke ermöglicht. Dabei macht SOAP die Kernkomponente für die Serviceorientierte Architektur (SOA) aus. Wie auch REST verwendet SOAP mehrheitlich HTTP, um den Austausch von Information über das Internet zu gewährleisten (Soni & Ranga, 2019). SOAP weist dabei im Gegensatz zu einer flexiblen REST-API einen stark strukturierte und klar definierten Standard auf (Bigelow, 2022).

Nach einer Studie von Soni und Ranga (2019) welche REST und SOAP unter verschiedenen Gesichtspunkten verglichen haben, hat SOAP allerdings in den Punkten Zuverlässigkeit (Reliability) aufgrund der eingebauten Fehler-Logik einen großen Vorteil. Auch im Bereich Sicherheit kann SOAP auf die gesamte Bandbreite der WS-Security-Features zurückgreifen, während REST auf HTTPs beschränkt ist. Auf der anderen Seite ist REST jedoch weniger stark an einen Server gekoppelt und eröffnen damit Vorteil, weitaus flexibler und unabhängiger zu sein, beispielsweise wenn es zu Aktualisierungen kommt. In den Bereichen Payload kommt es ebenfalls zu erheblichen Unterschieden zwischen REST und SOAP. Das Verpacken in den streng strukturierten SOAP-Envelop wie eben näher erläutert, erhöht den Payload und die dazu nötige Bandbreite im Gegensatz zu dem sehr leichtgewichtigen und flexiblen REST-Ansatzes um ein Vielfaches (Soni & Ranga, 2019). Im Allgemeinen ist festzuhalten, dass sowohl die REST-Architektur als auch das SOAP-Protokoll Vor- und Nachteile haben und beide je nach Anforderungen und Anwendungsfällen effektiv eingesetzt werden können.

2.3.2.1.c. RPC

Der Remote Procedure Call, oder kurz RPC, ist eine der einfachsten API-Stile, welche wie auch die RESTful-API auf einer Client-Server Netzwerkarchitektur beruht (Jin et al., 2018). RPC verwendet dabei das im vorherigen Abschnitt erklärte SOAP-Protokoll (Soni & Ranga, 2019). Das Grundprinzip von RPC ist die Clientseitige Ausführung eines Codeblocks auf einem Server. Dabei wird sich anders wie die RESTful-API nicht auf Ressourcen, sondern auf die Aktion der Codeausführung an sich fokussiert. Der Client über gibt hierbei den Methodennamen, sowie mögliche Argumente an den dazugehörigen Endpunkt des Servers. Dort wird die Methode ausgeführt und die Antwort meist im JSON- oder XML-Format wieder zurück an den Client gesendet. Dabei wird je nach Aktion das korrekte HTTP-Verb gewählt: get für schreibgeschützte und lesende Requests, POST für schreibende Requests welche Daten modifizieren sollen. Dabei muss eine API, welche auf RPC basiert nicht zwangsläufig mit HTTP kommunizieren, auch andere hoch performante Protokolle wie Apache Thrift können verwendet werden (Jin et al., 2018).

Besonders gRPC, eine moderne Implementierung des RPC-Stils, optimiert die Kommunikation durch Verwendung des HTTP/2-Protokolls, was es effizienter im Vergleich zu älteren RPC-Implementierungen macht, die oft SOAP verwenden. gRPC nutzt Protocol Buffers anstelle von XML, was die Daten-

Übertragung noch schneller und effizienter gestaltet. Diese Aspekte ermöglichen eine verbesserte Leistung bei der Übertragung von komplexen und häufigen Datenpaketen zwischen Client und Server in modernen Anwendungen (gRPC Authors, 2024).

2.3.2.1.d. GraphQL

Die 2012 von Meta entwickelte Abfragesprache für APIs ermöglicht es Clients Daten so beim Server abzufragen, sodass diese in einer vom Client definierten Struktur gesendet werden. Dabei benötigt, GraphQL, anders als REST oder RPC nur einen einzigen Endpoint an den alle Requests gesendet werden können. Alle Informationen die der Server benötigt werden im JSON-Body des Requests angegeben. Dabei ergeben sich im Gegensatz zu den zuvor genannten Ansätzen einige Vorteile. GraphQL kann beispielsweise in einem einzigen Request Daten von verschiedenen Ressourcen fordern, was die Reaktionsgeschwindigkeiten von Applikationen erheblich steigern kann. Außerdem sind die Daten genau auf den Client zugeschnitten, während bei REST oder RPC womöglich Informationen in der Antwort enthalten sind, die der Client gar nicht benötigt und so unnötigen Payload in der Kommunikation erzeugen. Auf der anderen Seite bedeuten aber die exakte Individualisierbarkeit seitens des Clients, einen Mehraufwand seitens des API-Providers, damit der Server auch komplexe Queries verarbeiten und die Antwort in dieser Form liefern kann. Performance-Optimierungen im Bereich GraphQL-Queries kann äußerst schwierig sein, gerade dann, wenn die API hinzu externen Parteien geöffnet wird und die möglichen Usecases (noch) nicht klar sind (Jin et al., 2018).

2.3.2.2. Event-Driven APIs

Request-Response APIs unterstützen häufig Dienste mit sich schnell verändernden Daten nicht optimal, da die Antworten bei diesen Services rasch veralten können. Um bei den sich schnell verändernden Daten immer auf dem Laufenden zu bleiben, greifen viele Entwickler auf ein Monitoring (auch Polling genannt) der API zurück und fragen API-Endpunkte kontinuierlich in bestimmten Intervallen nach neuen Daten ab. Hier das ideale zeitliche Intervall zu finden, gestaltet sich allerdings zumeist schwierig: Wird nicht häufig genug abgefragt fehlen möglicherweise Veränderungen in den Daten seit dem letzten Abfragezeitpunkt. Ist das Intervall aber zu eng gesetzt, werden unnötig Ressourcen verschwendet ohne neue Daten zu erhalten (Jin et al., 2018). Um dieses Problem zu lösen gewannen in den letzten Jahren die sogenannten Event-Driven APIs gegenüber von auf REST basierenden APIs immer mehr an Bedeutung (Medjaoui et al., 2021) und ermöglichte es Daten in Echtzeit über eine API zu erhalten. Drei gängige Mechanismen dieses Typs von APIs werden in Folge nun näher erläutert.

2.3.2.2.a. WebHooks

Ein WebHook stellt eine URL zur Verfügung, welche typischerweise via der HTTP-Methoden POST, angesprochen wird. Ändert sich nun etwas in den Daten wird diese mittels einen Request an den definierten Endpoint geschickt. Auf diese Weise kann der Empfänger der Nachricht (wie etwa eine Applikation) in Echtzeit über die Änderungen informiert werden und darauf reagieren. Unternehmen wie GitHub oder Slack setzen auf WebHooks um Benutzer/-innen in Echtzeit benachrichtigen zu können. (Jin et al., 2018).

Webhooks sind grundsätzlich einfach zu implementieren – für jedes Event muss ein neuer Endpoint angelegt werden, schon können Daten in Echtzeit ausgetauscht werden. Auf der anderen Seite können

sich aus Webhooks auch neue Probleme ergeben – beispielsweise muss eine erneute Sendung der Nachricht gewährleistet sein, sollte es zu Verbindungsproblemen gekommen sein. Auch Sicherheitsaspekte und die Verwendung von Webhooks in Verbindung mit Firewalls muss besondere Beachtung geschenkt werden um eine sichere und funktionierende Kommunikation zwischen den Parteien zu gewährleisten (Jin et al., 2018).

2.3.2.2.b. WebSockets

WebSocket ist ein Protokoll welches einen bidirektionale Kommunikationskanal über eine einzige TCP-Verbindung (Transport Control Protocol) aufbaut. WebSocket APIs werden beispielsweise von Slack oder Blockchain verwendet, um Echtzeitkommunikation zwischen Nutzer/-innen oder Benachrichtigungen an diese zu implementieren. Dabei ermöglichen WebSockets eine simultane Echtzeitkommunikation zwischen Server und Client, ohne einen großen Overhead zu erzeugen. Ein weiterer Vorteil ergibt sich in ihrer Kombinierbarkeit mit Firewalls. Anders als Webhooks, werden WebSockets nämlich nicht von Firewalls geblockt und können somit im Bezug dessen vielseitiger eingesetzt werden. WebSockets sind die ideale Wahl an APIs, wenn eine schnelle Echtzeitübertragung von Daten in Kombination mit einer lang bestehenden Verbindung gewünscht ist. Speziell der letzte Punkt ist essenziell, um WebSockets effektiv nutzen zu können. Bricht die Verbindung ab muss diese Seitens des Clients neu initialisiert werden. Das kann speziell in Gebieten mit nicht-flächendeckender Netzwerkverbindung zu erheblichem Mehraufwand führen. Außerdem weisen WebSocket APIs im Bereich Skalierbarkeit ein Problem auf, da WebSockets einzeln verwaltet und initialisiert werden müssen und es hier zu einem exponentiell steigenden Mehraufwand pro WebSocket kommen kann (Jin et al., 2018).

2.3.2.2.c. HTTP-Streaming

HTTP-Streaming baut auf das Konzept von Request-Response APIs auf, mit dem Unterschied, dass die Länge der Response nicht limitiert ist und die Verbindung nicht nach einmaliger Sendung abbricht. In einer vom Client eröffneten Sitzung kann der Server nun bei Datenänderungen diese in Echtzeit an den Client übertragen, ohne dass ein erneuter Verbindungsauflauf vor jeder Sendung nötig ist. HTTP-Streaming wird beispielsweise von X eingesetzt, um neue Posts über eine einzelne HTTP-Verbindung anstelle von benutzerdefinierten Protokollen an die Applikation senden zu können. Ein Problem des HTTP-Streamings ist Buffering; manche Clients und Proxys weisen eine Buffergrenze auf und beginnen erst ab einem gewissen Grenzwert die Daten an die Applikation zu übertragen, was die Vorteile der Echtzeitübertragung via HTTP-Streaming zunichtemachen würde. Auch ist diese Art von APIs nicht für Clients ideal, die häufig die abgehörten Ereignisse ändern, da dies eine ständige Neuinitialisierung des HTTP-Streams erfordert und so zu einem erheblichen Mehraufwand führen kann (Jin et al., 2018).

2.3.3. Gefahren und Sicherheitstrends

Neben den bereits im Kapitel 2.1.1.2 und 2.1.1.3 beschriebenen Risiken im Kontext von Cybersecurity, die für APIs ebenso relevant sind wie für anderen Systeme, begegnen APIs noch weiteren sicherheitstechnischen Gefahren, welchen entgegengewirkt werden muss, um eine sichere Serviceschnittstelle aufzubauen zu können. Auch können bereits bekannte Risiken im Bereich von Serviceschnittstellen aufgrund von dessen interaktiven Charakter zwischen mehreren Systemen weitaus gravierender sein. So ergibt sich bei APIs beispielsweise ein erhöhtes Risiko im Bereich Authentifizierung und Autorisierung,

wie der unbefugte Zugriff auf Systeme oder Ressourcen, sowie deren potentielle Offenlegung (Badhwar, 2021).

In den frühen 2000ern, mit der Einführung von SOA-orientierten API Paradigmen wie SOAP, wurden Sicherheitsanforderungen vorrangig durch den Einsatz von WS-basierten Authentifizierungs- und Autorisierungsmechanismen, wie beispielsweise WS-Trust, WS-Security oder WS-Policy, umgesetzt (Badhwar, 2021). Auch das Konzept von Basic Authentication wurde seit Einführung des API-Konzepts eingesetzt, dabei werden Zugangsdaten im Header der HTTP-Anfrage mitgesendet. Auch wenn die Daten in der HTTP-Anfrage selbst verschlüsselt sind, müssen diese häufig von der Applikation selbst oder dazwischengeschaltete externe Services im Klartext gespeichert werden, was ein erhebliches Sicherheitsrisiko mit sich bringt (Jin et al., 2018) Darum ging der Trend seit der Einführung von REST und dem Vormarsch von Microservices in die Richtung von Sicherheitstoken wie OAuth-Tokens oder JSON-Webtokens, sowie den Einsatz von API-Schlüsseln und werden seither vorherrschend eingesetzt (Badhwar, 2021). Im folgenden Abschnitt werden diese und weitere API-Konzepte näher erläutert, sowie spezifische API-Sicherheitsrisiken in Form der OWASP Top 10 API Sicherheitsrisiken aufgezeigt.

2.4. API Security

2.4.1. Konzepte

Laut dem OWASP API-Security Team (2024) wird API Security folgend definiert: "API Security focuses on strategies and solutions to understand and mitigate the unique vulnerabilities and security risks of Application Programming Interfaces (APIs)." Um jene Strategien und Lösungen implementieren zu können sind einige Konzepte relevant die in Folge näher erläutert werden.

2.4.1.1. Authentifizierung und Autorisierung

Zwei Konzepte, welche das Fundament einer jeden sicheren API angesehen werden können, sind Authentifikation und Autorisierung, deren Grundlagen bereits in Kapitel 2.1.1.1 näher erläutert werden. Bei jeder sicheren Serviceschnittstelle muss gewährleistet sein, dass nur jene Entitäten mit der API interagieren können, die die jeweiligen Zugangsdaten, Token, Schlüssel, oder ähnlich vorweisen können und nur dann Zugriff auf bestimmte Ressourcen haben, wenn dazu eine Berechtigung vorliegt. 4 der 10 später näher beschriebenen OWASP Top 10 API-Security Kategorien werden direkt von dem Konzept der Authentifizierung und Autorisierung abgeleitet, was die Wichtigkeit dessen unterstreicht. In Folge werden nun einige wichtige Authentifizierungs- und Autorisierungsmechanismen erläutert.

2.4.1.1.a. OAuth

Ein Problem, das bei Authentifizierungs- und Autorisierungsmechanismen auftreten kann, ist, dass verschiedene Applikationen verschiedene Mechanismen implementieren, um die Serviceschnittstelle sicher zu gestalten. Ist dies der Fall, kommt es allerdings zu einer sehr heterogenen und wenig übersichtlichen Landschaft an verschiedenen Mechanismen. Um einen einheitlichen Mechanismus zur Authentifizierung und Autorisierung zu schaffen, wurde deshalb das OAuth, also Open Authorization, Protokoll eingeführt um es API-Entwickler/-innen aber auch API-Konsument/-innen einfacher zu machen einheitliche Schnittstellen zu etablieren beziehungsweise zu verwenden. Das OAuth Protokoll

standardisiert verschiedene heterogene Authentifizierungs- und Autorisierungsmethoden und erlaubt es mehreren Anwendungen so Entitäten über eine einzige Authentifizierung beziehungsweise Autorisierung für alle Anwendungen freizuschalten. Auch können Anwendungen von Drittanbieter mit einer Authentifizierung und Autorisierung auf mehrere Anwendungen zugreifen (Chae et al., 2019). Doch nicht nur in der Vereinheitlichung der Mechanismen liegt der Vorteil des OAuth Protokolls; im Gegensatz zu der bereits 2.3.3 angeführten Basic Authentication müssen beim OAuth Protokoll keine Zugangsdaten mit den jeweiligen Anwendungen geteilt werden (Jin et al., 2018). Beispielsweise kann eine Applikation (Client), welche die Daten eines Benutzers von Google verwenden will, diesen an Google weiterleiten, sodass der/die Benutzer/-in die Autorisierung zur Datenverwendung zustimmen kann. Erfolgte die Freigabe der Daten durch den/die Benutzer/-in (Ressourcen Besitzer) wird bei einem Autorisierungsserver ein Zugangs-Token angefragt. Der Autorisierungsserver autorisiert die Applikation und sendet den Zugangs-Token zurück zum Client. Mithilfe des Tokens kann die Applikation nun bei Google (Ressource Server) die gewünschten Benutzerdaten anfragen, welcher den Token validiert und die Benutzerdaten bei positiver Validierung zur Verfügung stellt (Hardt, 2012; Jin et al., 2018).

Mit Verwendung des OAuth Protokolls kann die Applikation also die Google API aufrufen um Benutzerdaten so erhalten, ohne dass der/die Nutzer/-in sich direkt bei der Applikation authentifizieren oder autorisieren muss. Der gerade erläuterte Prozess der Authentifizierung beziehungsweise Autorisierung über das OAuth Protokoll wird in Abbildung 8 aufgezeigt.



Abbildung 8: OAuth Prozess (angelehnt an Hardt, 2012)

Ein zweiter großer Vorteil bei der Verwendung von OAuth ist die selektive Berechtigungsgabe. Da jede Applikation andere Anforderungen des Datenzugriffs an den API-Betreiber stellt, ist es mit OAuth möglich die Ressourenzugriffe von Applikationen individuell zu verbieten oder gewähren. Sollte wie im eben genannten Beispiel, der Google-Benutzer sich zu einem späteren Zeitpunkt dazu entscheiden, die Applikation nichtmehr nutzen zu wollen, kann der Zugriff auf die Benutzerdaten über das Google-Profil des Benutzers oder der Benutzerin rückgängig gemacht werden. Die Applikation hat nun keinen Zugriff mehr auf die Benutzerdaten, obwohl die Zugangsdaten für Google nicht geändert wurden (Jin et al., 2018).

Mit einer Verwendungsrate von 38% liegt OAuth, beziehungsweise die aktuelle Version OAuth 2.0, 2018 an Platz eins aller bei REST-APIs verwendeten Authentifizierungsmechanismen. Platz zwei machen mit etwa 36% API-Schlüssel aus (Neumann et al., 2018), welche nach dem folgenden Tokenbasierten Authentisierungsmechanismus näher erläutert werden.

2.4.1.1.b. JSON Webtokens (JWT)

JSON Webtokens sind „compact, URL-safe means of representing claims to be transferred between two parties“ (Jones et al., 2015) und lassen sich somit den tokenbasierten Authentifizierungsmechanismen zuordnen. Die zustandslose Authentifizierungstoken können beispielsweise zusammen mit OAuth verwendet und können, während OAuth die Erstauthentifizierung übernimmt, kümmern sich JWT darum den nachfolgenden sicheren Zugriff auf Ressourcen zu gewährleisten (Badhwar, 2021). Im Allgemeinen werden JWT in der Regel nicht für die initiale Authentifizierung verwendet, sondern setzen erst nach der erfolgreichen Authentifizierung durch OAuth oder anderen klassischen Authentifizierungsmethoden, wie Username und Passwort, an. Die Autorisierungsdetails dieser ersten Authentifizierung, wie Benutzerdaten, Rollen oder Berechtigungen, werden zusammen in den JSON Webtokens verpackt und mit einem gemeinsamen Schlüssel verschlüsselt. Danach kann der Client den JWT zusammen mit den Zugriffsanfragen auf Ressourcen an den jeweiligen Server senden. Der Server erhält die Anfrage, entschlüsselt den Token und trifft aufgrund der darin enthaltenen Informationen die Entscheidung ob der Zugriff auf die gewünschte Ressourcen gewährt oder abgelehnt wird (Jánoky et al., 2018).

Dabei ergeben sich beim Einsatz von JWTs Vorteile in der Skalierbarkeit und Performance, da die Informationen zu Authentifizierung und Autorisierung im Zuge der Kommunikation übertragen werden und nicht serverseitig abgerufen werden müssen. Auch bei verteilten Systemen kommt es zu einer verminderten Komplexität beim Einsatz von JWTs. Die Tokens beinhalten bereits alle relevanten Informationen und vereinfachen so den Verbindungsaufbau zwischen den einzelnen Komponenten (Jánoky et al., 2018).

Doch der Einsatz von JSON Webtokens weist auch Nachteile auf: nach Jánoky et al. (2018) ergeben sich sowohl die Vorteile von JWTs als auch dessen Nachteile aus dem dezentralen Charakter der JWT basierten Authentifizierung. Das sogenannte Logout Problem beschreibt die Schwierigkeit den durch den Token gewährten Zugriff wieder zu widerrufen. Anders als bei anderen Token-basierten Methoden, wie das OAuth Protokoll, welches jederzeit vom/von der Benutzer/-in selbst rückgängig gemacht werden kann, ist dies bei JWTs nicht so einfach möglich. Die Gültigkeit eines Tokens wird hier durch den Inhalt und die Fähigkeit diesen zu entpacken seitens des Servers determiniert. Problematisch wird es dann, wenn Benutzersitzungen beispielsweise durch Ausloggen des/der Benutzers/Benutzerin ungültig sind obwohl der JSON Webtoken eigentlich noch gültig wäre (Jánoky et al., 2018). Dies ist sicherheitstechnisch ein nicht zu vernachlässigendes Problem. Badhwar (2021) empfiehlt deshalb die Ablaufzeit des Tokens so kurz wie möglich zu halten, damit so ein Fall gar nicht erst auftreten kann. Dazu müsste jedoch jedes Token bereits zum Erstellungszeitpunkt ein Ablaufdatum zugewiesen bekommen und es würde unweigerlich dazu führen, dass auch Token dessen Benutzersitzungen noch aufrecht sind ihre Gültigkeit verlieren. Ein weiterer Ansatz wäre, dass widerrufene Tokens auf schwarze Listen

gesetzt werden und diese in Folge mit den in Anfragen enthaltenen Token abzugleichen (Jánoky et al., 2018). Dies führt dazu, dass ungültige Token sofort erkannt und abgelehnt werden können, wenn beispielsweise eine Benutzersitzung ausläuft oder die Berechtigung auf eine Ressource widerrufen wird bevor der Token selbst abläuft.

2.4.1.1.c. API-Schlüssel

Im Gegensatz zu den beiden bereits genannten Konzepten OAuth und JWTs, werden API-Schlüssel in der Regel nicht für die Benutzerauthentifizierung oder -autorisierung verwendet (AWS, 2023). Viel mehr beschränkt sich der Einsatzbereich von API-Schlüssel auf die Bereiche Server-Server Kommunikation, der (dauerhaften) Integration von Drittanbieteranwendungen oder die Überwachung von API-Nutzungen beziehungsweise der Fehlerbehebung bei API-Integrationen (AWS, 2023; Helton, 2023). Anders als Token, die Informationen wie Benutzer- oder Sitzungsdaten beinhalten, handelt es sich bei API-Schlüssel um eine Zeichenfolge, die bereits im Programmcode selbst definiert wurde. API-Schlüssel werden demnach in der Regel nur einmal erzeugt und bleiben danach in dieser Form dauerhaft bestehen. Ein API-Schlüssel verwaltet je eine fixierte Anzahl an Berechtigungen für Ressourcen - wer den API-Schlüssel besitzt kann also folglich auf die jeweiligen Ressourcen zugreifen (Helton, 2023). Dies ist der Grund, warum eine Offenlegung eines API-Schlüssels sehr viel verehrender ist, als jene eines kurzlebigen und auf ein Datensegment beschränkten Tokens. Bleibt die Attacke unerkannt hat der/die Angreifer/-in Zugriff auf die jeweils mit dem Schlüssel verwalteten Assets und kann diese manipulieren, wird die Offenlegung rechtzeitig erkannt muss ein neuer API-Schlüssel generiert und der veraltete gelöscht werden.

Der API-Schlüssel ist ein kryptografischer Schlüssel, welcher aus einer zufällig generierten Reihe an Buchstaben und Zahlen besteht (AWS, 2023) und häufig mithilfe eines HMAC, also Hash-basierten Nachrichtenauthentifizierungscodes, verschlüsselt wird (Badhwar, 2021). In der Regel wird eine API-Schlüssel einer Softwarekomponente oder Anwendung zugeordnet, damit diese mit der gewünschten API kommunizieren können (AWS, 2023). Dies hat auch sicherheitstechnische Gründe, da ein/-e Angreifer/-in in diesem Fall nur Daten dieser einen Anwendung und nicht des gesamten Systems abrufen kann (Badhwar, 2021). Die API erhält im Falle einer Datenabfrage den verschlüsselten API-Schlüssel der jeweiligen Anwendungen und validiert diesen. Ist der Schlüssel gültig wird die Anfrage erfüllt und die gewünschten Daten geliefert, bei ungültigem Schlüssel wird der Zugriff verwehrt. API-Schlüssel können, wie bereits erwähnt, auch unterschiedliche Berechtigungen aufweisen, sodass nicht jeder API-Schlüssel denselben Zugriff auf Ressourcen besitzt. Zum Beispiel ist es möglich manchen Anwendungen nur eine Lesefunktion von Daten zu gewähren, wogegen andere Daten auch überschreiben oder verändern dürfen (AWS, 2023).

2.4.1.2 Verschlüsselung

Eine korrekte Authentifizierungsstrategie reicht im Regelfall nicht um eine API sicher gegenüber potentiellem Angreifer/-innen zu machen. Eine Studie von SALT (2024) zeigt, dass es vielen Cyberkriminellen möglich ist, Authentifizierungsmechanismen erfolgreich zu umgehen; so ist es möglich, dass etwa 60% aller Angreifer/-innen nichtauthentifiziert Schaden anrichten können. Demnach sind auch andere Konzepte von Bedeutung, wenn es sich um die Absicherung von Serviceschnittstellen handelt.

Eines dieser Konzepte ist Verschlüsselung, welche in Folge nun näher behandelt wird, bevor andere wichtige Sicherheitskonzepte erläutert werden.

Verschlüsselung im Kontext von API Security ist das Kodieren von Daten zwischen Client und der API, um unbefugten Zugriff oder Manipulation der Daten zu verhindern. Besonders um Übertragung von sensiblen Informationen, also Transaktionen die Finanzinformationen, persönliche oder vertrauliche Daten beinhalten zu schützen ist eine Verschlüsselung ein unverzichtbares Mittel. Die Kommunikation wird durch die Verwendung von kryptografischen Algorithmen geschützt und erschwert so Cyberangriffe wie Man-in-the-Middle-Attack, Datenlecks oder das Abhören von Daten. Secure Sockets Layer/Transport Layer Security (SSL/TSL) ist hierbei ein häufig genutztes kryptografisches Protokoll, welches die Kommunikation zwischen Client und der API mithilfe der Durchführung von sogenannten „Handshakes“ absichern soll (Gutermuth, 2024).

Die Authentifizierung, wie oben näher erläutert, und Verschlüsselung sind zwei verwandte, aber nicht identische Maßnahmen. Während die API-Verschlüsselung dafür sorgt, dass die Datenübertragung zwischen Client und der API vertraulich und unverändert bleibt, dient die Authentifizierung dazu die Identität der Benutzer oder Systeme zu überprüfen. Die Authentifizierung wirkt als Zugangskontrolle und stellt sicher, dass nur autorisierte Benutzer/innen die API nutzen können, während die Verschlüsselung die Daten vor Abhören und Manipulation schützt. Beide Maßnahmen sind somit notwendig für die Sicherheit von APIs (Gutermuth, 2024).

Eine große Herausforderung bei der API-Verschlüsselung ist deren Komplexität in der Implementierung aber auch Verwaltung. Verschlüsselungsprotokolle müssen korrekt konfiguriert, sowie die Schlüssel sicher gespeichert werden. Besonders in verteilten Umgebungen wie Microservices kann die Verwaltung von Verschlüsselung schnell komplex sein. Verschlüsselungen können durch den erhöhten Ressourcenverbrauch, der häufig zur korrekten Verwaltung und Verwendung nötig ist, potentiell die Leistung beeinträchtigen, sowie veraltete Verschlüsselungen gar selbst ein Sicherheitsrisiko darstellen. Eine kontinuierliche Überwachung und Optimierung der Verschlüsselungen, sowie das Monitoring aktuellen Sicherheitsbedrohungen und deren potentielle Gefahren sind daher unumgänglich (Gutermuth, 2024).

2.4.1.3 Input Validierung

Eine umfassende und korrekte Validierung des Inputs in eine Serviceschnittstelle ist nach einer Authentifizierung und Autorisierung der nächste logische Schritt, um APIs abzusichern. Ziel ist es, inkorrekte oder bösartige Eingaben abzuwehren, bevor sie zu dem System potentiell beschädigen können. Injection Angriffe oder Cross-Site-Scripting (XSS) sind beispielsweise Attacken, die mithilfe einer umfassenden Validierung der Eingabedaten vermieden werden können (Belgacem, 2022). Best Practises im Bereich Input Validierung umfassen beispielsweise (Belgacem, 2022):

- Überprüfung des Content-Type Headers und des Datenformats: Validierung des Content-Type Headers und des Datenformats ob die erwarteten Formate vorliegen. Beispielsweise sollte eine API die ausschließlichen JSON-Dateien verarbeitet, überprüfen das im Content Header „application/json“ steht und die Dateien auch wirklich ein gültiges JSON-Format aufweisen.
- Begrenzung der Datengröße: Limitation der Größe hochgeladener Dateien, um den Server vor Überlastungen zu schützen.

- Validierung der Benutzereingaben, um Injection-Agriffe (wie beispielsweise SQL-Injection) zu verhindern

2.4.1.4 Versionierung

„API versioning is the process of managing and tracking changes to an API“ definiert Postman (2024b) Versionierung im Kontext von APIs. Serviceschnittstellen sind keine starren Konstrukte, sondern dynamisch und ändern sich über die Zeit hinweg. Dies ist beispielsweise beim Lösen von Bugs, Beseitigung von Sicherheitslücken oder neuen Funktionen der Fall – hier entstehen oft neue Versionen einer API. Um Probleme mit anderen Teilen des Systems und daraus resultierende Komprimierungen zu vermeiden ist die korrekte Versionierung beim Implementieren dieser Änderungen unabdingbar (Postman, 2024b). Versionierung kann somit als Teil der API-Design angesehen werden und benötigt eine Strategie wie mit verschiedenen Versionen einer API umgegangen wird. Auch bei einer komplexen Servicelandschaft mit vielen APIs sollen deren Versionen korrekt und einheitlich verwaltet werden, um Kompatibilitätsproblemen zu vermeiden (API Lifecycle Management, 2021), dazu gehört beispielsweise auch die Anpassung von Dokumentationen bei Änderungen und daraus resultierenden neuen Versionen (Postman, 2024b). Einige Best Practices hierzu werden in Folge aufgelistet (Postman, 2024b):

- URL-Versionierung: Dieser Ansatz inkludiert die Versionsnummer der API direkt in der URL wie in dieser beispielhaften URL: <https://example.at/v2/customers>. Es wird somit sowohl für Konsument/-innen als auch Entwickler/-innen klar welche Version der API aktuell verwendet wird.
- Query Parameter-Versionierung: Die Versionsnummer wird hier als Parameter in der Anfrage mitgegeben: <https://example.at/customers?version=v2>. Auch bei diesem Ansatz ist für alle Beteiligten klar ersichtlich welche Version verwendet wird.
- Konsumenten-basierte Versionierung: Konsument/-innen der API können bei diesem Ansatz beim ersten Zugriff auf die API die Version wählen, welche in Folge für den jeweiligen Konsumenten gespeicherte und in Zukunft verwendet wird. Eine Änderung der verwendeten Version kann nur durch eine explizite Änderung in der Konfiguration erzielt werden.

2.4.1.5 Security Headers

Bei der Nutzung von APIs mit HTTP-Protokoll sind HTTP Security Header ein Konzept, welches Serviceschnittstellen hilft eine weitere Schicht an Sicherheit hinzuzufügen. Security-Header werden in den HTTP-Response-Headern mitgegeben, die der Server beim Beantworten von Anfragen sendet. APIs sollen so von bekannten Sicherheitsrisiken geschützt werden um Angriffsszenarien wie Cross-Site-Scripting oder Clickjacking minimieren zu können (Kumar, 2024). In Folge werden nun ein paar von OWASP gesammelten Security-Headers näher erläutert (Cheat Sheets Series Team, 2024), dessen Fehlen teilweise auch in Kapitel 3 im Zuge der Analyse der Sicherheitswerkzeuge bei manchen Test-APIs als Schwachstelle aufgezeigt wurden.

- **X-Frame-Options:** Dieser Header wird verwendet, um zu steuern, ob eine Webseite in einem <frame>, <iframe>, <embed> oder <object> dargestellt werden darf. Es kann so verhindert werden, dass die Inhalte in anderen Seiten eingebettet werden können und erschweren so

Clickjacking-Angriffe. Der Security-Header ist damit nur bei http-Antworten mit interaktiven Elementen wie beispielsweise Links relevant.

- **X-Content-Type-Options:** Dieser HTTP-Response-Header wird vom Server verwendet, um den Browsern mitzuteilen, dass die in den Content-Type-Headern angegebenen MIME-Typen (Multipurpose Internet Mail Extensions) befolgt und nicht erraten werden sollten. Es soll so verhindert werden, dass der Browser versucht, den MIME-Typ einer Ressource selbst zu interpretieren und möglicherweise falsche Annahmen trifft, die zu Sicherheitslücken führen könnten.
- **Strict-Transport-Security (HSTS):** Der Security-Header ermöglicht es einer Website, dem Browser mitzuteilen, dass diese ausschließlich über HTTPS anstelle von HTTP aufgerufen werden soll. Dies erhöht die Sicherheit, da die Kommunikation zwischen Website und Browser nun verschlüsselt abläuft und beispielsweise das Risiko von Man-in-the-Middle-Angriffe so gesenkt werden kann.
- **Content-Security-Policy (CSP):** Ziel dieses Security-Headers ist es die Herkunft jener Inhalte zu spezifizieren die auf in eine Webanwendung geladen werden dürfen. So können Sicherheitsrisiken wie Cross-Site-Scripting, oder Injection-Angriffe minimiert werden. Aber auch Szenarien wie Datendiebstahl, das bösartige Verunstaltung von Webseiten oder Verbreitung von Malware kann so entgegengewirkt werden. Der Security Header ist für Webseiten relevant die Skripte und Codes als Input akzeptieren, da hier aktive Inhalte ausgeführt oder dargestellt werden. CSP ist also besonders wichtig für Seiten, die aktiven Code wie beispielsweise JavaScript ausführen.

2.4.1.6 Rate Limiting

Rate Limiting ist ein wichtiges Konzept, um der Überlastung von APIs und deren Servern entgegenzuwirken. Es werden bestimmte Grenzwerte gesetzt, um den Zugriff auf eine API zu limitieren. So können beispielsweise DDoS-Angriffe, die mithilfe einer extrem hohen Anzahl an Anfragen versuchen Zugriff auf Daten oder ins System selbst zu gelangen, abgewehrt werden. Doch auch Gründe über die Datensicherheit hinaus sind entscheidend. Organisatorische und Kosten-minimierende Aspekte machen Rate Limiting zu einem entscheidenden Konzept bei Serviceschnittstellen (Kinsta, 2023).

Dabei können Limit von verschiedenen Ausgangspunkten aus berechnet werden (Kinsta, 2023):

- Benutzer-basierende Limits: Limitierung der API-Aufrufe eines bestimmten Benutzers oder IP-Adresse.
- Standort-basierende Limits: Das Ziel vieler API-Betreiber/-innen ist es die Bandbreite der APIs geographisch gleichmäßig zu verteilen. Kommt es an einem bestimmten Standort zu einer übermäßigen Nutzung und Limits werden überschritten, wird eine Drosselung oder Herunterfahren von Diensten an diesen Standort initiiert.
- Server-basierende Limits: Serverseitig implementierte interne Begrenzung um die gerechte Verteilung von Ressourcen wie Speicher, CPU, und so weiter zu gewährleisten. Dadurch kann sichergestellt werden, dass keine einzelnen Prozesse oder Nutzer eine unverhältnismäßig große Menge an Ressourcen verbraucht.

Auch die Arten welchen Effekt die Limits auf den/der Endbenutzer/-in haben können unterschieden werden. Hier gibt es beispielsweise harte Limits, welche bei Überschreitung die Nutzung der API verwehren. Dem gegenüber stehen weiche Limits, die viel dynamischer sind und auch bei Überschreitung noch einen Zugriff erlauben. Eine weitere Art von Limits, sind jene die bei einer Überschreitung weitere Bezahlung verlangen. Auch Throttles können verwendet werden; dieser Typ von Limits verwehrt nicht sofort den Zugriff auf die API, sondern verlangsamt diesen nach Überschreitung eines Schwellwerts (Kinsta, 2023).

Nachdem einige grundlegende Sicherheitskonzepte von APIs beschrieben wurden, wird im nächsten Abschnitt konkret auf die OWASP Top 10 API-Security Kategorien eingegangen.

2.4.2. OWASP Top 10 API Security

Wie bereits im Kapitel 2.1.1.3 erläutert, erstellen die Open Web Application Security Project, kurz OWASP, Foundation nicht nur Ranglisten für allgemeine, sondern auch spezifische Problemfelder in Bezug auf Cybersecurity. Eine dieser Ranglisten zielt auf den Bereich API-Security ab und wurde erstmals 2019 veröffentlicht. Im letzten Jahr 2023 kam es zu einer Überarbeitung der Liste an potenziellen Sicherheitslücken bei Serviceschnittstellen. Eine aktuelle Studie (SALT, 2024) bestätigt die Relevanz dieser Schwachstellen und zeigt, dass 80 % der Angriffe auf APIs über die in der Rangliste identifizierten Schwachstellen erfolgen, während jedoch nur etwa 58 % der Gegenmaßnahmen auf diese Liste zurückgreifen. In Folge werden die Ranglisten aus beiden Jahren verglichen und danach die neu überarbeitete Liste erläutert und dabei näher auf die Aspekte Angriffsvektor, sicherheitsrelevante Schwachstellen und Auswirkungen bei Ausnutzung dieser eingehen. Am Ende werden diese Aspekte zusammenfassend gegenübergestellt.

2.4.2.1. OWASP Top 10 API Security 2019 vs. 2023

Wie bereits zu Beginn des Kapitels 2.4 eingeleitet, stellen die korrekte Authentifizierung und Autorisierung das Fundament von Serviceschnittstellen dar: “Authorization remains the biggest challenge in API Security. Three out of the top five items are related to authorization” (OWASP API Security Project team, 2023). Der Fokus auf Autorisierung ist, wie in Abbildung 9 zu sehen, sowohl 2019 als auch 2023 noch relevant. Auch Schwachstellen bei der Authentifizierung nehmen in beiden Jahren nach wie vor Platz zwei in der Rangliste ein. Unterschiede der beiden Top Ten Listen lassen sich aber in anderen Aspekten wiederfinden:

- Die Kategorie „Unrestricted Access to Sensitive Business Flows“, wurde hinzugefügt, um das Problem von gefälschten Accounts zu adressieren. Diese Kategorie hebt sich von anderen Kategorien, dessen Fokus stark auf einen sicheren Programmcode liegt, ab und setzt bereits im Planungs- und Designprozess eines Systems an (OWASP API Security Project team, 2023).
- Ebenfalls hinzugefügt wurde die Kategorie „Server Side Request Forgery“, kurz SSRF. Auch wenn es sich hier um kein neues Sicherheitsproblem handelt, ist das Risiko nach wie vor gerade bei APIs vorherrschend. Vor allem der Trend von unter anderem Webhooks, Cloud-basierte REST APIs und Docker macht es Cyberkriminelle einfacher einen SSRF Angriff durchzuführen (OWASP API Security Project team, 2023).

- „Unsafe Consumption of APIs“ wurde der Top 10 API Security Liste hinzugefügt und fokussiert sich auf Drittanbieter-APIs. Softwareentwickler/-innen vertrauen häufig unbekannten oder unsicheren APIs von Drittanbietern was zu einer Schwachstelle im eigenen System führen kann (Akto, 2023).
- Die Kategorien „Logging and Monitoring“, sowie „Injection“ wurden entfernt. Bei letzterer wird die fehlende Exklusivität der Schwachstelle bei Serviceschnittstellen als Grund genannt (Akto, 2023). Injections sind aber nach wie vor ein wesentlicher Teil der Kategorie „Security Misconfiguration“ (Elissen & Namer, 2023).
- Die Kategorie „Broken User Authentication“ wird zur Kategorie „Broken Authentication“. Während sich die 2019 festgelegte Kategorie rein um die Authentifikation von Benutzer/-innen kümmert, sind in der Kategorie 2023 auch Risiken rund um Authentifikationsmechanismen abseits des/der Benutzers/Benutzerin enthalten (Akto, 2023).
- „Excessive Data Exposure“ und „Mass Assignment“ wurden in die Kategorie „Broken Object Level Authorization“ zusammengefasst. Beide Kategorien verfolgen ein einheitliches Ziel, nämlich den Schutz von (sensiblen) Daten gegen unautorisierten Modifikationen und Zugriff (Akto, 2023) und konnten so 2023 vereinheitlicht werden.
- Darüber hinaus kam es zu einer Namensänderung von „Lack of Resources & Rate Limiting“ zu „Unrestricted Ressources Consumption“. Grund hierfür ist der verschobene Fokus von Resourcenmangel und Rate Limiting als Schwachstelle zu den Folgen der Sicherheitslücken wenn es tatsächlich zu einem Angriff in diesem Bereich kommt, nämlich einen uneingeschränkten Ressourcenverbrauch (Akto, 2023).
- Die letzte Änderung betrifft wiederrum eine Namensänderung von „Improper Assets Management“ zu „Improper Inventory Management“ um die Risiken dieser Schwachstelle zu betonen. Eine schlecht organisierte API kann schnell zu einer Offenlegung von sensiblen Daten führen, weshalb ein umfassendes und strukturiertes API-Inventar dabei hilft mögliche Angriffe vorzeitig zu identifizieren und so gegenwirken zu können (Akto, 2023).

Die Auflistung der beiden OWASP Top Ten API Security Kategorien und dessen Veränderungen sind in Abbildung 9 skizziert.

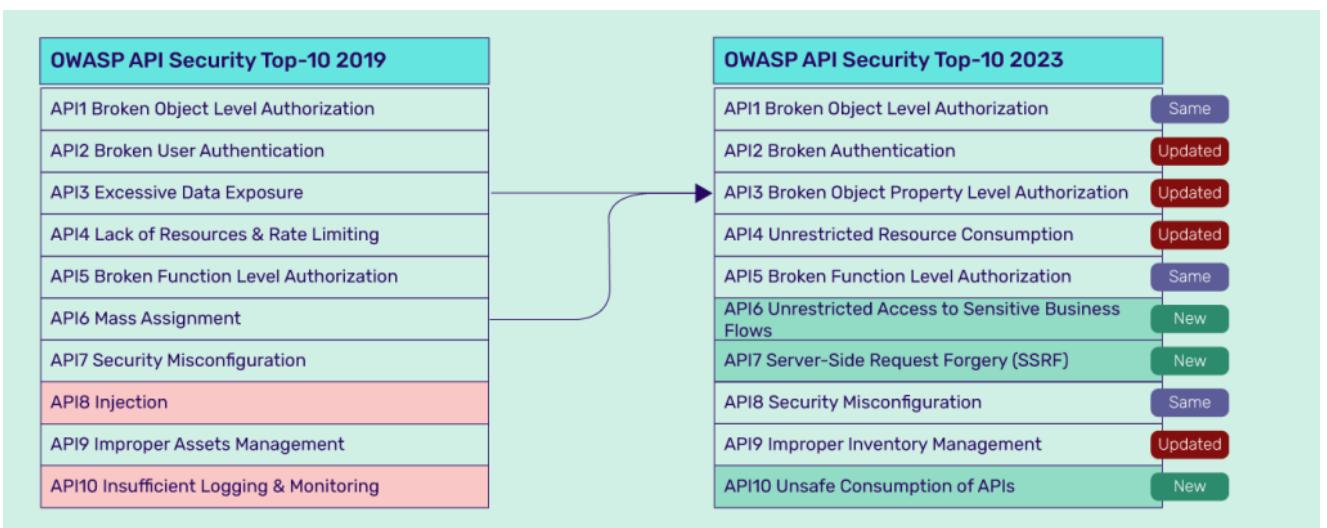


Abbildung 9: OWASP Top 10 API Security 2019-2023 (Indusface, 2023)

In Folge werden nun die einzelnen Kategorien der OWASP API Security Top Ten näher erläutert.

2.4.2.2. Broken Object Level Authorization (API1)

Object Level Authorization, also die Autorisierung auf Objektlevel wird in der Regel bereits im Code selbst implementiert und determiniert welche Entität auf die jeweiligen Objekte zugreifen darf. Nach Empfehlung vom OWASP Top 10 API Security Project Team (2023) soll jede API welche Anfragen zur Anzeige und Manipulation von Daten erhält einen derartigen Mechanismus implementiert haben, um die zugreifenden Entitäten auf ihre Berechtigungen prüfen zu können. Eine Schwachstelle im Autorisierungsmechanismus auf Objektlevel, oder gar ein Fehlen dieses, kann zu einer unerlaubten Offenlegung von Daten, deren Manipulation oder Zerstörung führen. Dabei reicht es in der Regel nicht die ID der aktuellen Benutzersitzung, welche beispielsweise aus einem JSON Webtoken extrahiert wurde mit jenen berechtigten IDs auf Objektebene zu vergleichen. Es können hierbei beispielsweise die IDs manipuliert worden sein, um einen berechtigten Zugriff böswillig zu simulieren. (OWASP Top 10 API Security Project Team, 2023a).

Das konkrete folgende Angriffszenario wird durch eine fehlende Berechtigungsprüfung auf Objektebene bei Aktionen auf Ressourcen ausgelöst. Beispielsweise könnte der in Codebeispiel 2 abgebildeter GraphQL-Aufruf (OWASP Top 10 API Security Project Team, 2023a) zum Löschen eines Dokuments so manipuliert werden ein anderes Dokument zu löschen, auf das der/die Benutzer/-in eigentlich keinen Zugang haben dürfte:

```
POST /graphql
{
  "operationName": "deleteReports",
  "variables": {
    "reportKeys": ["exampleDocumentID"]
  },
  "query": "mutation deleteReports($siteId: ID!, $reportKeys: [String]!) {
    deleteReports(reportKeys: $reportKeys)
  }
}"
```

Codebeispiel 2: GraphQL Abfrage OWASP API1

Das Problem liegt hierbei in der fehlenden Prüfung der Berechtigungen des/der Nutzers/Nutzerin selbst. Es wird lediglich die Dokumenten-ID mitgegeben, nicht aber die konkrete Identifikation des/der Benutzers/Benutzerin um dies Berechtigung der Aktion entsprechend abgleichen zu können. Ein bös-

williger Nutzer könnte hier die Dokumenten-ID ganz einfach manipulieren und ein völlig anderes Dokument löschen, auch wenn er nur zu dieser Aktion auf die originale Ressource berechtigt war (OWASP Top 10 API Security Project Team, 2023a).

Ein/-e Angreifer/-in kann Schwachstellen, die durch eine Verletzung in dieser Kategorie entstehen einfacher ausnutzen, indem wie oben gezeigt die Objekt ID manipuliert wird. Schwachstellen in diesem Bereich sind im Kontext von APIs sehr weit verbreitet, da Server häufig nicht mit den gesamten Benutzerdaten, sondern den Identifikatoren der Objekte arbeiten, um Autorisierungen zu prüfen. Die Erkennbarkeit dieser Schwachstellen ist für Unternehmen aber vergleichsweise einfach und kann deswegen effektiv entgegengewirkt werden. Sollte es zu einem Angriff in diesem Bereich kommen ist mit moderaten technischen Folgen kommen, die ja nach Unternehmen mehr oder weniger gravierend ausfallen können. Wie bereits erwähnt zählen zu möglichen Folgen der unerlaubte Zugriff auf Daten, sowie dessen Manipulation und potentielle Zerstörung (OWASP Top 10 API Security Project Team, 2023a)

In dieser Kategorie stehen Berechtigungen auf Objektlevel im Fokus, die Berechtigung, ob eine Entität Zugriff auf einen konkreten API-Endpunkt beziehungsweise Funktion haben sollte, werden im Abschnitt 2.4.2.6 Broken Function Level Authorization (API5) näher behandelt.

2.4.2.3. Broken Authentication (API2)

Ein korrekt-funktionierendes Authentifizierungsmechanismus ist für APIs essenziell, um unter anderem Benutzeraccounts und Benutzerdaten effektiv schützen zu können. Schwachstellen in diesem Bereich werden mit der Kategorie API2 „Broken Authentication“ abgedeckt. Dabei sollen Mechanismen zur Wiederherstellung von Passworten genauso sicher gehandhabt werden als der Authentifizierungsprozess selbst. Eine API weist eine Schwachstelle in dieser Kategorie auf wenn (OWASP Top 10 API Security Project Team, 2023b):

- Der/die Benutzer/-in schwache Passwörter verwenden kann.
- Authentifizierungstoken oder Passwörter direkt in der URL übertragen werden.
- Token nicht validiert werden.
- die API schwach oder gar nicht signierte JWTs akzeptiert.
- die API das Ablaufdatum der JWTs nicht prüft.
- Passwörter unverschlüsselt oder nur schwach verschlüsselt.
- schwache Schlüssel zur Verschlüsselung verwendet.
- Brute Force Angriffe durch das Ermöglichen von Credential Stuffing begünstigt beziehungsweise keine Mechanismen zur Kontosperrung bei dessen Versuch implementiert.

Ein mögliches Angriffsszenario wäre die Übernahme eines fremden Accounts durch einen zu schwachen Authentifizierungsmechanismus. Um eine Emailadresse eines Benutzeraccounts zu ändern muss folgende Beispieldaten (OWASP Top 10 API Security Project Team, 2023b) an die API übertragen werden:

```
PUT /account
Authorization: Bearer <token>

{ "email": "newemail@test.com" }
```

Codebeispiel 3: Beispielabfrage OWASP API2

In der Anfrage wird keine Identitätsbestätigung mithilfe des Passworts gefordert. Ein Cyberkrimineller kann hier durch den Diebstahl des Authentifizierungstokens das Benutzerkonto übernehmen, indem er diese Anfrage mit der eigenen E-Mailadresse versendet. Mithilfe des Passwort-zurücksetzen-Workflows kann der/die Angreifer/-in in Folge das Passwort beliebig ändern und hat fortan uneingeschränkten Zugang auf das fremde Benutzerprofil (OWASP Top 10 API Security Project Team, 2023b).

Authentifizierungsmechanismen sind ein einfaches Ziel für Angreifer/-innen, da diese in der Regel für jeden sichtbar sind. Auch Werkzeuge, um die Schwachstellen dieser Kategorie ausnutzen zu können typischerweise leicht zu finden. Aufgrund des Missverständnisses der Grenzen von Authentifizierungsmechanismen die Entwickler/-innen häufig aufweisen und den unvermeidbaren Wechselwirkungen zwischen Implementierungselementen sind Schwachstellen in dieser Kategorie durchaus verbreitet. Allerdings ist es auch einfach diese Schwachstellen zu finden. Die Auswirkungen auf das System beziehungsweise das gesamte Unternehmen können sehr schwerwiegend ausfallen, da Angreifer/-innen gesamte Benutzerkonten übernehmen und sensible Informationen auslesen, verändern oder löschen können. Systeme können in der Regel unrechtmäßig übernommene Benutzerkonten nicht von legitimen unterscheiden (OWASP Top 10 API Security Project Team, 2023b). Auch der Vertrauensverlust seitens der Nutzer/-innen, sollte es zu einem solchen Angriff kommen, kann für das Unternehmen negative Folgen haben.

2.4.2.4. Broken Object Property Level Authorization (API3)

Anders als die Kategorie API1, geht die Kategorie „Broken Object Level Authorization“, also der Einsatz von inkorrekte Autorisierungsprozessen auf Ebene der Objekteigenschaften, noch einen Schritt weiter und betrachtet nicht die Ressourcen an sich, sondern deren konkreten Eigenschaften und Ausprägungen. Wenn eine Entität also Zugriff auf einen bestimmten API-Endpunkt hat, muss eine Überprüfung erfolgen, welche spezifischen Eigenschaften der Ressourcen, von Nutzer/-innen eingesehen oder verändert werden dürfen. Eine Schwachstelle in diesem Bereich liegt dann vor, wenn der Entität jene Objekteigenschaften angezeigt werden für die er eigentlich keine Berechtigung besitzt, sowie wenn die Entität die Inhalte dieser Objekteigenschaften verändern, hinzufügen oder löschen kann (OWASP Top 10 API Security Project Team, 2023c).

Ein beispielhaftes Angriffsszenario wäre eine Unterkunftsbuchung über eine Drittplattform, in Zuge dessen der Unterkunftsbetreiber/-in die Objekteigenschaft der Buchung mutwillig verändern, obwohl der/die Betreiber/-in hierzu nicht berechtigt ist. Die folgenden beiden Anfragen (OWASP Top 10 API Security Project Team, 2023c) zeigen eine erlaubte Buchungsbestätigung, sowie danach die böswillige:

```
{
  "approved": true,
  "comment": "Check-out is before 10am"
}
```

```
{
  "approved": true,
  "comment": " Check-out is before 10am "
  "total_price": "$100,000"
}
```

Codebeispiel 4: Beispielabfragen OWASP API3

In diesem Fall fehlte die Berechtigungsprüfung auf Objekteigenschaftsebene gänzlich - der/die Betreiber/-in hätte hier auf die Objekteigenschaft „total_price“ nicht zugreifen dürfen, sowie deren Inhalt nicht verändern. Folglich hätte der/die Bucher/-in einen viel höheren Preis bezahlt als auf der Plattform gekennzeichnet (OWASP Top 10 API Security Project Team, 2023c).

Serviceschnittstelle, speziell REST-APIs, neigen in der Regel dazu Endpunkte, welche alle Objekteigenschaften zurückgeben preiszugeben was eine gravierende Schwachstelle in dieser Kategorie bedeutet. GraphQL basierte APIs sind zwar besser geschützt, doch auch hier existieren bereits Werkzeuge, die zur mutwilligen Aufdeckung der Objekteigenschaften Unterstützung bieten. Somit lassen sich diese Schwachstellen in der Regel leicht von potenziellen Angreifer/-innen ausnutzen. Aufgrund des einfachen Zugangs zu Ressourceninformationen reicht im Regelfall eine Inspektion von APIs aus diesen Informationen zu erhalten. Auch Fuzzing wird oft dazu verwendet um Objekteigenschaften, speziell versteckte, zu erhalten. Schwachstellen in diesem Bereich sind also durchaus üblich und können auch leicht identifiziert werden. Die Auswirkungen bei Ausnutzung dieser Schwachstelle ist auch wie schon bei API1, moderat und kann zu unrechtmäßigem Auslesen, Modifizieren oder Löschen von Daten führen (OWASP Top 10 API Security Project Team, 2023c).

2.4.2.5. Unrestricted Resource Consumption (API4)

Eine Serviceschnittstellen kann nur dann effektiv funktionieren, wenn genug Ressourcen wie Netzwerkbandbreite, CPU und Speicher verfügbar sind. Manche APIs beziehen einzelne Ressourcen von externen Dienstleistern, die in der Regel pro Anfrage bezahlt werden – wie beispielsweise das Senden von Emails, SMS, Anrufe, oder ähnlichem. Die Kategorie „Unrestricted Resource Consumption“ oder uneingeschränkter Ressourcenverbrauch, legt den Fokus auf die Schwachstellen dieser APIs in Verbindung mit den Ressourcen die sie von Drittanbietern beziehen. Eine API die Schwachstellen in dieser Kategorie aufweist setzt kein oder ein unpassendes Limit für folgende Ressourcen (OWASP Top 10 API Security Project Team, 2023d):

- Maximale Ausführungszeit
- Maximale zuweisbare Speichermenge

- Maximale Datengröße bei Upload
- Maximale Anzahl von Prozessen
- Maximale Ausgaben für die Dienste von Drittanbieter
- Maximale Anzahl von Datensätzen pro Seite in einer Anfrage oder Antwort
- Maximale Anzahl von Operationen die in einer einzelnen API-Anfrage durchgeführt werden sollen

Folgendes Angriffsszenario (OWASP Top 10 API Security Project Team, 2023d) ist in dieser Kategorie möglich: Eine Plattform hat einen „Passwort Vergessen“ Button der von Nutzer/-innen benutzt werden kann um das Passwort zurücksetzen zu können. Der Plattform-Betreiber nutzt den SMS-Dienst eines Drittanbieters und sendet im Falle eines Klicks auf „Passwort vergessen“ eine SMS an die hinterlegte Telefonnummer. Im Hintergrund wird nun die Telefonnummer des/der Benutzers/Benutzerin an das Back-End des Plattformbetreibers gesendet, sowie darauffolgend im Hintergrund des Back-End der API an den Dienstleister mit dem Hostnamen smsservice.net.

```
POST /initiate_forgot_password

{
  "step": 1,
  "user_number": "06645861254"
}
```

```
POST /sms/send_reset_pass

Host: smsservice.net

{
  "phone_number": "06645861254"
}
```

Codebeispiel 5: Beispiel API-Call OWASP API4

Nach Erhalt der Anfrage kümmert sich der Dienstleister um die Sendung der SMS und verrechnet der Plattform pro SMS 5 Cent. Ein Cyberkrimineller schreibt nun ein Skript, um den API Request hin zum Back-End der API abertausende Male durchführen zu lassen. Für jeden POST-Request ins Back-End wird auch ein Request zum Dienstleister ausgelöst und eine SMS verschickt, sowie die Kosten an die Plattform verrechnet. Innerhalb von kürzester Zeit kann es hierbei zu einem erheblichen finanziellen Schaden für die Plattformbetreiber kommen (OWASP Top 10 API Security Project Team, 2023d). Da der Plattformbetreiber keine ausreichende Limitierung für Datenübertragungen oder Ausgaben an den Dienstleister gesetzt hat merkt dieser den Angriff erst viel zu spät oder gar nicht und verliert tausende von Euro.

Um Schwachstellen in dieser Kategorie ausnutzen zu können, benötigt der/die Angreifer/-in lediglich einfache API-Anfragen und wird vom OWASP Top 10 API Security Project Team (2023d) als durchschnittlich im Aspekt Ausnutzbarkeit eingestuft. Es existieren auch Werkzeuge welche Cyberkriminellen hier Unterstützung bieten, diese sind aber vermehrt dafür ausgelegt DOS-Attacken (2.1.1.2.c) durchzuführen und so die Beeinträchtigung der Servicerate von APIs als Ziel haben. APIs, welche ihre Ressourcen und Interaktionen nicht einschränken sind weitverbreitet, außerdem können diese Schwachstellen sehr leicht ausgemacht werden. Die Auswirkungen auf das Unternehmen können äußerst schwer ausfallen, sei es finanziell wie im Angriffsszenario beschrieben oder aufgrund von einem erheblichen Anstieg an Cloud, CPU oder Speicherressourcen, als auch technisch wenn es beispielsweise zu einem DoS Angriff kommt (OWASP Top 10 API Security Project Team, 2023d).

2.4.2.6. Broken Function Level Authorization (API5)

Im Gegensatz zu API1 welche sich auf den Zugriff von Objekten und Ressourcen fokussiert, sind Schwachstellen in der Kategorie „Broken Function Level Authorization“ auf Ebene der Funktionen und Endpunkte. Um Schwachstellen in dieser Kategorie zu finden ist eine genaue Analyse des Autorisierungsmechanismus nötig inklusive den Aspekten Benutzerrollen und deren Hierarchien. Können Benutzer/-innen beispielsweise Endpunkte aufrufen die eigentlich nur Administratoren vorbehalten sind? Ist es Benutzer/-innen möglich durch das Ändern eines HTTP-Verbs in der URL, Aktionen aufzuführen, für die sie nicht berechtigt sind? Wie sieht es mit unberechtigten Funktionsaufrufen aus, wenn die dazugehörigen URLs und deren Parameter erraten werden können? Kann eine oder mehrere dieser Fragen mit Ja beantwortet werden weist ein Unternehmen Schwachstellen in dieser Kategorie auf (OWASP Top 10 API Security Project Team, 2023e).

Ein mögliches Angriffsszenario könnte durch die Schwachstelle eines beispielhaften ungeschützten administrativen Endpunkts (OWASP Top 10 API Security Project Team, 2023e) ausgelöst werden:

```
GET /api/test/v1/users/all
```

Codebeispiel 6: Beispielendpunkt OWASP API5

Beim Aufrufen dieses Endpunkts werden den Administratoren alle Benutzerdaten angezeigt, weshalb normale Nutzer/-innen hierauf keinen Zugriff haben sollten. Der Autorisierungsmechanismus auf Funktionsebene wurde für diesen Endpunkt aber nicht korrekt implementiert. Ein böswillige/-r Nutzer/-in ohne Administratorrechte konnte die URL durch Ausprobieren ausfindig machen und hat nun Zugriff auf die gesamten Benutzerdaten (OWASP Top 10 API Security Project Team, 2023e).

Damit potenzielle Angreifer/-innen diese Schwachstelle ausnutzen können müssen diese in der Rolle von nichtberechtigtem/nichtberechtigter Benutzer/-in legitime API-Aufrufe tätigen. Das gestaltet sich in der Regel für Cyberkriminelle verhältnismäßig einfach. Öffentlich gemachte Endpunkte laufen besonders Gefahr von Angreifer/-innen ausgenutzt zu werden. Normalerweise werden Autorisierungsmechanismen bereits im Code oder auf Konfigurationsebene festgelegt, da sich hier aber eine Suche nach Schwachstellen bei vielen modernen Anwendungen aufgrund der vielen Gruppen, Rollen und

Hierarchien sehr komplex gestaltet, ist es einfacher die Schwachstellen auf API-Ebene auffindbar zu machen. Die Verbreitung der Schwachstellen wird als üblich, aber nicht weitverbreitet kategorisiert. Die Auswirkungen auf das Unternehmen bei erfolgreichen Angriffen können sehr schwerwiegend ausfallen, da Funktionen und Endpunkte welche nur Administratoren vorbehalten sind, als attraktives Ziel für Cyberkriminelle gelten. In der Regel ist es so möglich auf eine umfassende Anzahl an Daten zuzugreifen und diese zu manipuliert. Unter Umständen können auch Unterbrechungen des von der API zur Verfügung gestellten Services herbeigeführt werden (OWASP Top 10 API Security Project Team, 2023e).

2.4.2.7. Unrestricted Access to Sensitive Business Flows (API6)

Die Kategorie “Unrestricted Access to Sensitive Business Flows”, also den uneingeschränkten Zugang zu sensiblen Geschäftsabläufen, rückt den überwiegend technischen Fokus der letzten Kategorien hin zu einer unternehmenszentrierten Sichtweise. Die Erstellung einer API setzt nicht nur eine gute und sichere technische Implementierung, sondern auch das Wissen welche Geschäftsabläufe damit der Öffentlichkeit zugängig gemacht werden. Gerade durch APIs offengelegte sensible Geschäftsabläufe können eine sicherheitsrelevante Schwachstelle darstellen. Dazu zählen unter anderem folgende Geschäftsabläufe (OWASP Top 10 API Security Project Team, 2023f):

- Kauf eines Produktes – hier besteht das Risiko, dass ein/-e Angreifer/-in unmittelbar den gesamten Bestand eines Artikels mit hoher Nachfrage aufkauft und mit hoher Gewinnspanne weiterverkauft.
- Kommentar- oder Beitragserstellung – das System könnte mit Spamkommentaren oder Beiträgen überflutet werden.
- Terminreservierungen – Störung des Normalbetrieb durch die Reservierung aller verfügbaren Zeitfenster

Ein beispielhaftes Angriffsszenario könnte folgend aussehen (OWASP Top 10 API Security Project Team, 2023f): Eine Applikation die Fahrgemeinschaften vermittelt, verspricht jeden/jeder Benutzer/-in der einen neuen Nutzer/-innen an Boot holt Credits welche in Folge für vergünstigte oder freie Fahrten genutzt werden können. Ein/-e Angreifer/-in nutzt diesen sensiblen Geschäftsablauf aus und spielt ein Skript ein, welches die Credits eines/einer jeden Benutzers/Benutzerin bei Neueinladung auf ihn überträgt. Der/die böswillige Benutzer/in kann nun uneingeschränkte Freifahrten genießen oder die Credits teurer weiterverkaufen.

Die Ausnutzbarkeit von Schwachstellen in dieser Kategorie ist für Angreifer/-innen in der Regel einfach, sobald sie das dahinterliegende Geschäftsmodell verstanden haben. Sensible Abläufe können so schnell gefunden und die Zugriffe auf diese automatisiert werden, um dem Unternehmen zu schaden. Aufgrund der oft fehlenden ganzheitlichen Sichtweise und einem zu starken Fokus auf reine Implementierungsschwachstellen sind Mängel in dieser Kategorie weitverbreitet und für Unternehmen auch nicht immer sofort identifizierbar. Die Auswirkungen bei einer Ausnutzung der Schwachstellen wird auf moderat eingestuft und sind stark Unternehmensabhängig (OWASP Top 10 API Security Project Team, 2023f). In der Regel handelt es sich dabei auch nicht um technische Auswirkungen, sondern sie sind eher finanzieller Natur oder senken Kundenvertrauen oder -zufriedenheit.

2.4.2.8. Server-Side Request Forgery (API7)

Die Kategorie „Server-Side Request Forgery“ findet sich auch in den allgemeinen OWASP Top 10 wieder und wird im Kapitel 2.1.1.3.k bereits beschrieben. Deshalb fokussiert sich dieser Abschnitt auf API-spezifische Aspekte, die noch nicht zuvor abgedeckt wurden.

Server-Side-Request-Forgery treten dann auf, wenn eine API eine Ressource aufruft, ohne die vom Client zur Verfügung gestellten URL zuvor zu validieren. Das Risiko von SSRFs steigt aufgrund von dem Einsatz von modernen Konzepten wie Webhooks, URL-Vorschauen und dem Datenabruf via URLs stetig. Außerdem erleichtern aktuell häufig verwendete Technologien wie Docker, Kubernetes oder Cloud Services aufgrund der Offenlegung von bekannten HTTP-Kanälen zur Verwaltung und Kontrolle, es den Angreifer/-innen eine SSRF-Angriff durchzuführen (OWASP Top 10 API Security Project Team, 2023g).

Schwachstellen in dieser Kategorie können beispielsweise durch folgendes Angriffszenario ausgelöst werden. Benutzer/-innen laden auf einer Social Media Plattform Fotos als Profilbilder hoch. Dabei können die Bilder mit der folgenden API-Anfrage (OWASP Top 10 API Security Project Team, 2023g) als URL zur Verfügung gestellt werden.

```
POST /api/test/profile_picture_upload

{
  "profile_picture_url": "http://test.com/profile_pic.jpg"
}
```

Codebeispiel 7: Beispiel API-Anfrage OWASP API7

Ein/-e Angreifer/-in könnte nun durch die zur Verfügungstellung einer böswilligen URL ein Port-Scanning initiieren:

```
POST /api/test/profile_picture_upload

{
  "profile_picture_url": "localhost:8080"
}
```

Codebeispiel 8: Port-Scanning OWASP API7

Anhand der Antwortzeit kann der/die Angreifer/-in in Folge herausfinden, ob ein Port offen oder geschlossen ist und kann mithilfe der Information über eine neue potentielle Schwachstelle andere Angriffe entsprechend planen und durchführen (OWASP Top 10 API Security Project Team, 2023g).

Die Schwachstellen in dieser Kategorie lassen sich in der Regel einfach ausnutzen. Ein/e Angreifer/-in muss hierfür einen API-Endpunkt finden, welcher eine von Nutzer/innen bereitgestellte URL ausführt, um einen SSRF-Angriff zu starten. Eine Schwachstelle, die eine SSRF-Attacke begünstigt ist aufgrund der vielfältigen Möglichkeiten vom Client bereitgestellte URL zum Vorteil der Anwendung zu nutzen nicht selten. Viele dieser bereitgestellten URL werden aber ohne umfassende Validierung ausgeführt was SSRF-Angriffe ermöglicht. Die Sicherheitsschwachstelle dieser Kategorie zu finden ist aber für das Unternehmen durch regelmäßige Analysen von Anfrage und Antworten relativ einfach. Auswirkungen von SSRF-Attacken werden als moderat klassifiziert und können von Port-Scanning, Offenlegung von Informationen, Umgehen von Firewalls oder anderen Sicherheitsmechanismen bis hin zu einem DoS Angriff ausmachen (OWASP Top 10 API Security Project Team, 2023g).

2.4.2.9. Security Misconfiguration (API8)

Wie auch die vorherige Kategorie ist auch „Security Misconfiguration“ bereits in der allgemeinen OWASP Top 10 Rangliste zu finden und mit Kapitel 2.1.1.3.f abgedeckt. Auch hier wird nun der Fokus auf jene Aspekte, welche spezifisch für APIs sind, gerückt. APIs welche anfällig für diese Kategorie an Schwachstellen sind weisen unter andrem eine oder mehrere der folgenden Aspekte auf (OWASP Top 10 API Security Project Team, 2023h):

- Fehlen von Sicherheitsmaßnahmen im API-Stack
- fehlerhafte Einstellungen von Berechtigungen von Cloud-Diensten
- aktive unnötigen Funktionen
- inkorrekte oder Inkonsistente Verarbeitung von Anfragen von Servern in der HTTP-Serverkette
- der Transport Layer Security (TSL) ist nicht vorhanden
- Fehlermeldungen enthalten sensible Informationen und Fehlerbeschreibungen
- Sicherheits- oder Cache-Control-Einstellungen werden nicht korrekt an die Clients gesendet
- inkorrekte Richtlinien für Cross-Origin Ressource Sharing (CORS)
- fehlende Sicherheitsheader

Schwachstellen in diesem Bereich können folgendes beispielhaftes Angriffsszenario auslösen. Eine Social Media Plattform erlaubt es Benutzern und Benutzerinnen über private Nachricht zu kommunizieren. Um neue Nachrichten abfragen zu können wird die folgende API Anfrage (OWASP Top 10 API Security Project Team, 2023h) seitens der Plattform übermittelt:

```
GET /chat/user_updates.json?conversation_id=22598&cursor=JFENF55AD96IL
```

Codebeispiel 9: Beispiel API-Abfrage OWASP API8

Da die Antwort der API nicht die Cash-Eigenschaft „Cash-Control“ im HTTP-Kopf beinhaltet werden die privaten Nachrichten im Webbrowser des/der Nutzers/Nutzerin im Cash gespeichert und können in Folge von Angreifer/-innen ausgelesen werden (OWASP Top 10 API Security Project Team, 2023h).

Angreifer/-innen versuchen stets über nicht-gepatchte Schwachstellen, unsichere (Standard)-Konfigurationen oder ungeschützte Dateien Zugriff zu einem System zu verschaffen oder Informationen darüber zu erhalten. Darüber hinaus sind Schwachstellen, besonders jene die weit verbreitet sind, häufig öffentlich bekannt was es potenziellen Angreifen noch einfacher macht Schwachstellen zu finden und auszunutzen. Die Schwachstelle ist sehr weit verbreitet und kann auf jeder Ebene des API-Stacks vorkommen. Doch auch Unternehmen können automatisierte Werkzeuge verwenden, um Schwachstellen aufzudecken. Die Auswirkung bei erfolgreicher Ausnutzung der Schwachstellen in dieser Kategorie sind für die Unternehmen in der Regel schwer, da sie nicht nur Benutzerdaten preisgeben können sondern auch wichtige Details zum System selbst und es so zu einer gesamten Übernahme des Servers durch Angreifer/-innen kommen kann (OWASP Top 10 API Security Project Team, 2023h).

2.4.2.10. Improper Inventory Management (API9)

Die hohe Rate an Konnektivität, die mit dem Charakter von APIs einhergeht, birgt auch neue Gefahren, die adressiert werden müssen. API-Betreiber/-innen müssen damit nicht nur die eigene API und deren Endpunkte kennen und verstehen, sondern auch Wissen über jene APIs haben, mit denen sie kommunizieren, Daten teilen oder speichern. Auch das Betreiben von APIs in mehr als einer Version bedarf einen erhöhten Managementaufwand und erhöht das Risiko eines Cyberangriffs (OWASP Top 10 API Security Project Team, 2023i). Die Kategorie „Improper Inventory Management“, zu Deutsch unzureichendes Inventur- beziehungsweise Inventarmanagement, befasst sich mit den Gefahren, wenn Unternehmen keine vollständige und aktuelle Übersicht über ihre API-Ressourcen haben. Es ist deshalb unabdingbar das jede API eine aktuelle und umfassende Dokumentation vorweisen können. Folgende Aspekte weisen darauf hin, dass eine APIs Schwachstellen in dieser Kategorie aufweisen (OWASP Top 10 API Security Project Team, 2023i):

- Unklarheit über den Zweck des API-Hosts
- fehlende Spezifikation in welcher Umgebung die API läuft
- Unklarheit darüber wer Zugriff auf die API haben soll
- keine Angaben um welche API-Version verwendet wird
- keine oder eine veraltete Dokumentation der API
- keine Information über den Prozess bei API-Versionswechsel
- fehlendes oder veraltetes Inventar des Hosts

Außerdem spielen in dieser Kategorie auch die Sichtbarkeit und das Inventar von sensiblen Datenflüssen eine entscheidende Rolle, besonders dann, wenn es auf Seiten des Drittanbieters zu einem sicherheitsrelevanten Vorfall kommt. Schwachstellen in diesem speziellen Bereich kommen dann vor, wenn eine API sensible Daten mit einer Drittanbieteranwendung teilt. Ein erhöhtes Risiko ergibt sich, wenn keine Genehmigung beziehungsweise kein Zweck für die die Übertragung der sensiblen Daten vorliegt oder es keine Kenntnisse darüber gibt welche Daten überhaupt geteilt werden (OWASP Top 10 API Security Project Team, 2023i).

Ein mögliches Angriffsszenario in dieser Kategorie könnte folgend aussehen (OWASP Top 10 API Security Project Team, 2023i): Eine Social Media Plattform erlaubt die Integration von anderen Applikationen. Mithilfe einer Bestätigung des/der Benutzers/Benutzerin können die Applikationen auf die Daten des/der jeweiligen Benutzers/Benutzerin zugreifen. Der Datenfluss zwischen der Plattform und

den Applikationen ist allerdings nicht genug gesichert, sodass die Applikationen nicht nur auf die Daten des/der Benutzers/Benutzerin, sondern auch auf die privaten Daten von auf der Plattform registrierten Freunden des/der Benutzers/Benutzerin zugreifen können. Ein/e potenzielle/r Angreifer/-in baut eine Applikation und nutzt die API der Social Media Plattform, um die Erlaubnis des Datenzugriffs von hunderttausenden Nutzer/-innen zu erhalten. Aufgrund des zu wenig gesicherten Zugriffs auf Informationen erhalten die Betreiber der neuen Applikation nun Millionen von Nutzerdaten, welche sie für viel Geld weiterverkaufen können.

Schwachstellen in dieser Kategorie auszunutzen ist besonders aufgrund eines erleichterten Zugriffs auf veralteten und demnach oft weniger sicherere API-Versionen oder Endpunkten, leicht möglich. Auch die Offenlegung von Daten durch Drittanbieter ist ein Weg, den viele potentielle Angreifer/-innen oft wählen. Schwachstellen sind in der Regel sehr weitverbreitet, außerdem fällt es dem Unternehmen häufig schwer aufgrund von fehlender oder veralteter Dokumentation, sowie fehlende Strategien bei Versionswechsel, den Überblick über alle Ressourcen und Endpunkte der API zu behalten und Schwachstellen zu identifizieren. Die Auswirkungen bei Angriffen wird für das Unternehmen als moderat eingestuft. Angreifer/-innen können sich Zugang zu sensiblen Daten verschaffen oder Server ganz übernehmen. Auch der unrechtmäßige Zugriff auf administrative Funktionen oder auf aktuelle Datenbanken über veraltete API Versionen können vorkommen (OWASP Top 10 API Security Project Team, 2023i).

2.4.2.11. Unsafe Consumption of APIs (API10)

Die letzte Kategorie “Unsafe Consumption of APIs“, also der unsichere Verbrauch von APIs ist eng verwandt mit der Kategorie API09, während API09 sich aber um die Verwaltung von APIs im eigenen System fokussiert, rückt API10 die konkrete Verwendung von fremden APIs in den Mittelpunkt. Ein großes Problem in diesem Bereich sind Entwickler/-innen die Drittanbieteranwendungen, besonders von bekannten Unternehmen, zu sehr vertrauen und demnach weniger starke Sicherheitsmechanismen in den eigenen Systemen beispielsweise in Bezug auf Input-Überprüfung implementieren. Besonders anfällig für Schwachstellen in dieser Kategorie sind APIs die (OWASP Top 10 API Security Project Team, 2023j):

- mit anderen APIs über unverschlüsselte Kanäle kommunizieren.
- keine umfassende Validation und Säuberung von Daten aus anderen APIs vornehmen, bevor sie an andere Komponenten des Systems weitergeleitet werden.
- ohne Prüfung Weiterleitungen folgen.
- unbegrenzte Ressourcen zur Verarbeitung von Antworten der Drittanbieteranwendungen aufbringen.
- keine zeitlichen Limits zu den Interaktionen mit fremden APIs definieren.

Ein beispielhaftes Angriffsszenario, welches aus einer Schwachstelle dieser Kategorie resultieren könnte, wird in Folge aufgezeigt: Eine API hat zur Speicherung von sensiblen medizinischen Daten einen Drittanbieterservice integriert. Um Daten dort sicher zu speichern wird folgende HTTP-Anfrage (OWASP Top 10 API Security Project Team, 2023j) gesendet:

```

POST /user/store_phr_record
{
  "patient": "LSMFMD12512"
  "genome": "ACTAGTAG__TTGADDAAIICCTT..."
}

```

Codebeispiel 10: Beispiel http-Anfrage OWASP API10

Cyberkriminelle haben eine Schwachstelle in der API des Drittanbieters gefunden und konnten sich einen Zugang zu der API verschaffen. Auf jede Speicheranfrage wird nun mit der folgenden „308 Permanent Redirect“ Nachricht geantwortet:

```

HTTP/1.1 308 Permanent Redirect
Location: https://attacker.com/

```

Codebeispiel 11: Permanent Redirect OWASP API10

Da die API Weiterleitungen von Drittanbieter blind folgt und diese nicht überprüft, wird die identische Anfrage inklusive sensiblen Patientendaten wiederholt, aber nun an den Server der Cyberkriminellen (OWASP Top 10 API Security Project Team, 2023j).

Damit Angreifer/-innen die Schwachstellen aus dieser Kategorie ausnutzen können müssen unsichere APIs gefunden werden, die in der Ziel-API integriert wurden. Das OWASP Top 10 API Security Project Team (2023j) klassifiziert die Ausnutzbarkeit von Schwachstellen in dieser Kategorie als einfach. Aufgrund des bereits erwähnten häufig vorkommenden Grundvertrauen von Entwickler/-innen an Drittanbieter-APIs sind Schwachstellen in dieser Kategorie häufig vertreten. Die Unternehmensseitige Erkennbarkeit der Schwachstellen wird als durchschnittlich eingestuft, was darauf schließen lässt das Schwachstellen in diesen Bereich oft übersehen werden oder gar nicht erst in Betracht gezogen, da das Problem ja nicht in der „eigenen“ API liegt. Die Folgen von ausgenutzten Schwachstellen können je nach Unternehmen mitunter sehr schwer ausfallen. Es kann zu einer Offenlegung von sensiblen Daten kommen, sowie Injection oder DoS-Angriffen (OWASP Top 10 API Security Project Team, 2023j).

2.4.2.12. Zusammenfassung OWASP Top 10 API Security

Nachdem nun alle Kategorien der OWASP Top 10 API Security Rangliste im Detail erläutert wurden, folgt in Tabelle 1 eine zusammenfassende Auflistung der Kategorien sowie ihre Ausprägungen in den von OWASP (2023) definierten Sicherheitsrelevanten Aspekten.

Tabelle 1: OWASP Sicherheitsrelevante Aspekte

Kategorie	Angriffsvektor	Sicherheitsschwachstelle	Auswirkung
-----------	----------------	--------------------------	------------

	Ausnutzbarkeit	Verbreitung	Erkennbarkeit	Technisch / Unternehmen
API1	Einfach	Weitverbreitet	Einfach	Moderat
API2	Einfach	Üblich	Einfach	Schwer
API3	Einfach	Üblich	Einfach	Moderat
API4	Durchschnittlich	Weitverbreitet	Einfach	Schwer
API5	Durchschnittlich	Weitverbreitet	Einfach	Schwer
API6	Einfach	Weitverbreitet	Durchschnittlich	Moderat
API7	Einfach	Üblich	Einfach	Moderat
API8	Einfach	Weitverbreitet	Einfach	Schwer
API9	Einfach	Weitverbreitet	Durchschnittlich	Moderat
API10	Einfach	Üblich	Durchschnittlich	Schwer

Wie in klar in Abbildung 10 und Abbildung 11 sichtbar wird, ist die Ausnutzbarkeit von Schwachstellen der API Security Rangliste in den meisten Kategorien sehr einfach, sowie die Mehrzahl der Schwachstellen weitverbreitet. Für einen potenziellen Angriff auf eine der Schwachstellen benötigt ein/e Angreifer/-in weder herausragende Fähigkeiten noch viele zeitliche Ressourcen, um eine API mit entsprechenden Mängeln zu finden.



Abbildung 10: Aspekt Ausnutzbarkeit



Abbildung 11: Aspekt Verbreitung

Abbildung 12 zeigt jedoch, dass Sicherheitsmängel, auch wenn diese weit verbreitet sind, in der Regel einfach vom Unternehmen selbst identifiziert werden können. Prozesse und Werkzeuge, die hierbei hilfreich sein könnten, werden im nächsten Kapitel erläutert, sowie ausgewählte konkrete API Scanning-Werkzeuge in Folge analysiert und evaluiert. Zuletzt zeigen sich in Abbildung 13 die Auswirkungen auf das Unternehmen im Falle eines Cyberangriffs, seien sie nun technischer Natur oder auf finanzielle beziehungsweise kundenorientierte Aspekte fokussiert. Hier zeigt sich das jene potentiellen Auswirkungen der Schwachstellen in den genannten Kategorien zumindest moderater Natur ist, die Hälfte davon sind gar schwerwiegend. Dies unterstreicht nochmals die Notwendigkeit genügend Wissen und Verständnis über Schnittstellensicherheit für Techniker/-innen im gesamten Lebenszyklus einer API aufzubauen, sowie geeignete Sicherheitsmechanismen zu finden, die Angreifer/-innen effektiv abwehren können, um die Assets und damit das gesamte Unternehmen vor Cyberkriminalität zu schützen.

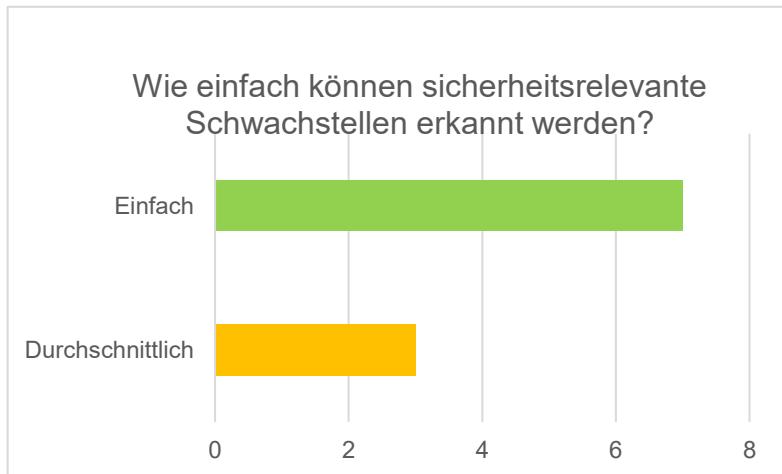


Abbildung 12: Aspekt Erkennbarkeit

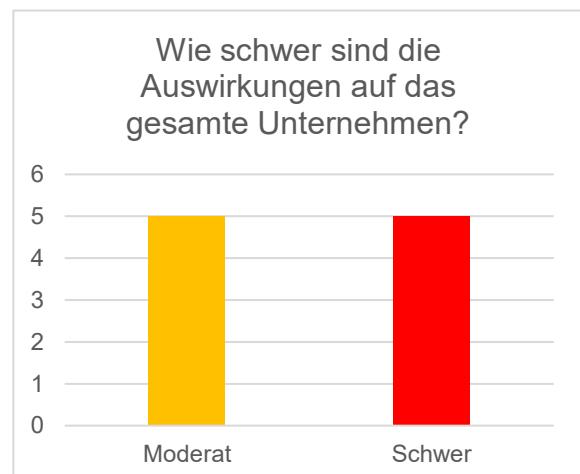


Abbildung 13: Aspekt Auswirkung

2.4.3. Testing und Monitoring

In einer dynamischen IT-Landschaft, in der APIs eine zentrale Rolle spielen, reicht es nicht aus, lediglich präventive Sicherheitsmaßnahmen zu implementieren. Regelmäßige Tests und kontinuierliches Monitoring sind essenziell, um potenzielle Schwachstellen frühzeitig zu erkennen und auf Angriffe rechtzeitig reagieren zu können.

2.4.3.1. API Security Testing

API-Tests sollen die Funktionalität, Performance und Sicherheit der APIs überprüfen und sind häufig Teil von Integrationstests. Die Tests finden bei Applikationen, welche häufig in einer 3-Schichtenarchitektur implementiert werden und so aus Präsentations-, Logik und Datenschicht bestehen, auf der mittleren der drei Ebenen, der Logikschicht, statt. Diese Schicht wird als die kritischste angesehen, da die gesamte Systemlogik und Verarbeitung der Daten hier stattfinden. Dabei sind API-Tests für automatisierte Tests und CI/CD Prozesse unerlässlich und benötigen im Gegensatz zu UI-Tests weniger Wartungsaufwand. Grundsätzlich weisen API-Tests im Vergleich zu UI-Tests und Unit-Tests einen sehr guten ROI (Return on Investment) auf und erzielen schon bei wenig Aufwand einen großen Nutzen (Katalon, 2024).

Sattam & Moulahi (2023) diskutieren wichtige Aspekte, die sich stark mit Konzepten aus der OWASP Top 10 API Security Rangliste decken. Authentifizierung, Autorisierung, die korrekte Validation von Input und Output, das Sichern von Kommunikationskanälen, Fehlerbehandlung, Sitzungsmanagement, Integration von Drittanbieteranwendungen, Ressourcenlimitierung, sowie Logging und Monitoring werden von Sattam & Moulahi (2023) als jene Aspekte genannt welche besonders im Zuge von API-Tests relevant sind. In Abbildung 14 werden diese nochmals zusammenfassend wiedergegeben.

Aspect of Security	Testing Approach
Authentication	Test for weak authentication mechanisms, such as weak passwords or lack of multi-factor authentication.
Authorisation	Test for improper access controls, such as privilege escalation attacks or inadequate role-based access controls.
Input Validation	Test for proper validation of user inputs to prevent injection attacks, such as SQL injection or cross-site scripting (XSS).
Error Handling	Test for proper error handling, such as ensuring error messages do not reveal sensitive information or cause application crashes.
Session Management	Tests for proper session management, such as preventing session fixation attacks or session hijacking attacks.
API Rate Limiting	Test for proper API rate limiting to prevent denial of service (DoS) attacks or brute force attacks.
Integration Testing	Test for security vulnerabilities in third-party APIs or services that the API interacts with.

Abbildung 14: Wichtige Aspekte bei API Security Tests (Sattam & Moulahi, 2023)

Tests, um die Sicherheit einer API zu überprüfen sind demnach essenziell, um Serviceschnittstellen effektiv nutzen zu können, ohne andere Entitäten, die auf die API zugreifen oder dessen Daten zu gefährden. Im Folgenden werden nun einige wichtige Testtypen des API-Security Testings näher erläutert:

- Penetration Testing:
Bei diesem Testtypen versuchen Nutzer/-innen, welche nur beschränktes Wissen über die API besitzen, die Serviceschnittstelle anzugreifen und so Schwachstellen im System zu identifizieren (Katalon, 2024; Sattam & Moulahi, 2023). Dabei können Penetration Tests manuell von Menschen oder auch automatisiert durchgeführt werden (Patel, 2019).
- Input Validation Testing:
Die Validation des Inputs in Anwendungen ist wichtig, um die Eingabe von bösartigen Scripts oder Queries zu unterbinden und so Injection- oder Cross-Site-Scripting-Angriffe abwehren zu können (Rodríguez et al., 2020). Auch Validierungsmechanismen müssen Tests durchlaufen, damit dessen korrekte Funktionalität garantiert werden kann. Dies ist beispielsweise mithilfe von Boundary oder Fuzz Tests möglich.
- Boundary Testing:
Um das Risiko von Integer Overflow, aber auch Injection-Angriffe zu minimieren werden APIs mit verschiedenen Inputs außerhalb des akzeptierten Rahmens getestet (Zhiwei & Zhongliang, 2020 zitiert nach Sattam & Moulahi, 2023). Daraufhin wird evaluiert, wie die API reagiert und ob sich in spezifischen Input-Szenarien etwaige Sicherheitslücken aufmachen.
- Fuzz Testing:

Diese Art von API Security Tests stellt der API eine große Anzahl an zufälligen Input-Daten zu Verfügung und versucht damit Fehlfunktionen hervorzurufen oder andere negative Effekte auszulösen, um die durch die Inputdaten erkannten Schwachstellen nachträglich beheben zu können (Katalon, 2024).

- Vulnerability Scanning: Um Mängel und Schwachstellen einer API gegenwirken zu können und so Daten, Anwendungen und andere Systeme zu schützen, müssen API Betreiber diese identifizieren bevor es zu einem Angriff kommt (Bhuiyan et al., 2018). Um dies zu erreichen, existieren einige Werkzeuge die Anwendungen auf Schwachstellen überprüfen und Rückmeldungen über gefundene Mängel geben. Werkzeuge, die Anwendungen und APIs auf Sicherheitschwachstellen, scannen erstellen in der Regel automatisierte Berichte die dem jeweiligen Betreiber zur Verfügung gestellt werden um Schwachstellen auffindbar und behebbar zu machen (Sattam & Moulahi, 2023).

Im Kontext von Vulnerability Scanning kann zwischen aktiven und passiven Scanvorgängen unterschieden werden. Dabei senden aktive Scans Testdatenverkehr an die jeweilige Applikation oder API und simulieren somit Angriffe auf das System. In Folge wird die Reaktion des Systems ausgewertet und dessen Ergebnis später im Bericht wiedergegeben. Passive Scans hingegen senden selbst keine Daten, sondern überwachen lediglich den Datenverkehr und interagieren damit nicht direkt mit dem System und lösen keine Angriffsreaktion aus. Dementsprechend sind aktive Scans in der Regel viel tiefgreifender und umfassender als passive Scans, können aber mitunter Schäden oder Leistungsminderungen am gescannten Systems auslösen (RiskOptics, 2022).

2.4.3.2. API Security Monitoring

Sowohl API Testing, als auch API Monitoring haben als Ziel, eine sichere, zuverlässige und effiziente Serviceschnittstelle zu gestalten. Dabei wird API Testing, im Gegensatz zu API-Monitoring in der Regel bereits während der Entwicklung vorgenommen, während API Monitoring zumeist erst im Echtbetrieb stattfindet, um Benutzer/-innen, dessen Daten und andere kommunizierende Systeme zu schützen. API Monitoring nutzt hierbei ähnliche logische Ansätze wie API Testing und sammelt, sowie visualisiert häufig Daten um diese in Folge den Techniker/-innen zur Verfügung stellen zu können (Postman, 2024a).

API-Monitoring ist für API-Betreiber und deren Entwicklungs-Team unabdingbar, um APIs zu beobachten und Schwachstellen und Mängel zu beheben, bevor diese zu einem sicherheitsrelevanten Problem werden und so Nutzer/-innen und Partner(-systeme) gefährden. Das Hauptziel von API Monitoring ist es die Mean Time to Resolution (MTTR), also jene Zeit die durchschnittlich von Erkennung bis hin zur Behebung des Problems vergeht, zu minimieren (Postman, 2024a). Dabei müssen API Monitoring Strategien sehr individuell auf das jeweilige Unternehmen zugeschnitten sein es gibt aber einige Monitoring-Szenarien die in den meisten Strategien enthalten sind. Diese umfassen die Validierung ob ein Request korrekt an die API ankommen, verarbeitet wird sowie (ja nach Anfrageart) eine Antwort rückübermittelt wird. Das ist besonders in jenen Unternehmen wichtig die sehr agil arbeiten und es zu häufigen Anpassungen im Code kommt. Wie bereits in der OWASP API Security Top 10 Rangliste mit

API10 abgedeckt, sind auch die Gefahren von Drittanbieteranwendungen und deren APIs nicht zu unterschätzen. Auch hier sollte eine konstante Überprüfung des Kommunikationskanals erfolgen. Ein dritter wichtiger Anwendungsfall für API Monitoring umfasst die bereits erwähnte Validierung von Benutzererfahrungen die in der Regel nicht nur eine API, sondern eine Kette an zusammenhängenden APIs verwenden (Postman, 2024a). Auch diese zusammenhängenden API-Ketten sollten nicht aus einer Monitoring-Strategie ausgeschlossen werden, um auch die Wechselwirkungen zwischen verschiedenen APIs zu überprüfen beziehungsweise den Datenfluss analysieren und bei Schwachstellen adaptieren zu können.

API Monitoring Systeme werden außerdem häufig dazu verwendet APIs, wie bereits im vorherigen Abschnitt eingeleitet, auf Mängel zu überprüfen und sogenannte Vulnerability Scans durchzuführen. Diese Überprüfungen finden meist sowohl im Entwicklungs- als auch im produktiven Stadium der API statt und sind häufig Teil der CI/CD-Pipeline (Postman, 2024a).

In Folge werden nun Werkzeuge zum Scannen von Sicherheitsschwachstellen im Bereich APIs auf Basis der in Kapitel 2.2 kategorisierten Liste an Sicherheitswerkzeuge gesammelt und ausgewählte Werkzeuge in Bezug auf die OWASP API Security Top 10 Rangliste analysiert beziehungsweise miteinander verglichen. Im Anschluss wird in Kapitel 4 ein Dashboard für die Analyse von API-Sicherheitsaspekten auf der Basis der identifizierten Werkzeuge vorgestellt.

3. Vergleich von Werkzeug zur Analyse von Sicherheitsrisiken im Bereich APIs

3.1. Methodik

Die vorliegende Masterarbeit verfolgt einen Design-Science Ansatz mit dem Ziel eine Lösung für ein Analyse-Dashboard für API-Sicherheit zu gestalten. Um dieses Ziel einen Schritt näher zu kommen, ist die Auswahl eines geeigneten Werkzeugs, um die Schwachstellen einer Serviceschnittstelle aufzeigen zu können entscheidend, um dieses in den Analyse-Prozess einbinden zu können. Nachdem eine im Folgenden näher beschriebene Auswahl der Sicherheitswerkzeuge stattgefunden hat, werden die Werkzeuge anhand von in Kapitel 3.1.2. beschriebenen Vergleichskriterien analysiert. Dabei werden auch tatsächliche Scans von Beispiel-APIs miteinbezogen. Da besonders aktive Scans von APIs, wie es eine Vielzahl der Werkzeuge bereitstellen, rechtlich als unerlaubter Angriff der API gilt, werden nur jene APIs für die beispielhaften Scans ausgewählt, die explizit für Tests zur Verfügung stehen. Jene APIs werden somit nicht in einer produktiven Umgebung verwendet und beinhalten keine Echtdaten, die womöglich durch die Konsequenzen von aktiven Scans preisgegeben werden könnten.

3.1.1. Auswahl der Sicherheitswerkzeuge

Die Basis der Auswahl bildete die in Kapitel 2.2 kategorisierte Liste der Security Werkzeuge, woraus die relevanten Werkzeuge extrahiert werden und in mehreren Runden gegenübergestellt werden. Die Auswahlkriterien umfassen die Zugänglichkeit der Werkzeuge, die Spezialisierungen in bestimmte Felder, die Systemvoraussetzungen, sowie die Abdeckungen der OWASP API Security Top 10 Kriterien. Aus insgesamt 74 Werkzeuge wurden final drei ausgewählt, analysiert und miteinander in Bezug auf Konzepte wie Usability, Funktionalität und Abdeckung der OWASP Top 10 API Security Schwachstellen verglichen.

3.1.1.1. Auswahlrunde Eins

Ziel der ersten Auswahlrunde war es, jene Werkzeuge zu finden, welche als Open Source-Projekte frei zugänglich sind, sowie im Zuge der kategorisierten Liste in die Kategorie API-Scanner eingeordnet werden können. Außerdem wurden jene Werkzeuge aussortiert, deren Funktionalität nicht auf das Scannen von Schwachstellen ausgelegt war (wie beispielsweise Fuzzer oder Werkzeuge die explizit nur für Injection oder Cross-Site-Scripting ausgelegt sind). Das minimierte die Zahl der Werkzeuge von 74 auf 12, die für die Analyse der Sicherheitswerkzeuge in Frage kommen würden.

3.1.1.2. Auswahlrunde Zwei

Die zweite Auswahlrunde minimierte die Werkzeuge von 12 auf 5. Fokus hierbei ist eine mögliche Integration in bestehende Systeme (beispielsweise ob Ergebnisse via API-Call abgerufen werden können oder das Werkzeug anderswertig in den Workflow eingebunden werden kann). Weiters werden Systemanforderungen, wie spezielle Voraussetzungen gegenübergestellt. Werkzeuge die hohe Systemanforderungen aufweisen oder veralteter Standards aufweisen, werden ausgeschlossen. Außerdem werde in dieser Auswahlrunde die Verknüpfung zwischen den Werkzeug-Funktionalitäten und den OWASP-Kategorien geprüft. Können Werkzeuge hier keine genauen Angaben über ihre Funktionalität machen oder konnten diese nicht mit den OWASP-Kategorien verknüpft werden, werden diese ebenfalls ausgeschlossen.

3.1.1.3. Auswahlrunde Drei

Die Auswahlrunde drei startet mit 5 Werkzeuge und machte damit die letzte Auswahlrunde aus, bevor es zur eigentlichen Analyse der Sicherheitswerkzeuge und Vergleich kommt. In dieser Runde wurden die einzelnen Werkzeuge bereits anhand ihrer Abdeckung der OWASP API Security Top 10 Kategorien analysiert. Die aus der jeweiligen Website extrahierten Informationen über die Funktionalität der Werkzeuge werden in die verschiedenen Kategorien eingeordnet – dabei ergibt sich folgendes Bild:

- ZAPProxy: API1, API2, API3, API4, API5, API7, API8, API9, API10
- Wapiti: API1, API2, API3, API7, API9, API10
- w3af: API1, API2, API5, API8, API10
- APIClarity: API1, API3, API5, API9
- GraphQL Cop: API2, API3, API4, API9

Die genauen Verknüpfungen zwischen Werkzeug-Funktionalität und OWASP Kategorien werden im Anhang B wiedergegeben.

Auch die Integrationsmöglichkeit sowie eine funktionierende Installation in Windows Umgebung werden in dieser Auswahlrunde überprüft. Obwohl das Werkzeug w3af eine sehr umfassende Abdeckung von Sicherheitsschwachstellen im Bereich APIs aufweist, wird es aufgrund eines nicht-funktionierenden Installation in Windows-Umgebung zum Zeitpunkt der Analyse der Sicherheitswerkzeuge nicht in die Analyse mitaufgenommen. Während auch das Werkzeug API-Clarity grundsätzlich für die vorliegende Masterarbeit relevante Bereiche abdeckt, kann aufgrund der Integrationsmöglichkeiten mit einer sehr beschränkten Palette an Technologien nicht gewährleistet werden eine umfassende Analyse des Werkzeugs zu erreichen. Daher werden diese beiden Werkzeuge auch in Folge nicht für die Integration in das Dashboard in Frage kommen und in der Vorauswahl ausgeschlossen.

Somit ergeben sich für die anschließende Analyse der Sicherheitswerkzeuge die drei Werkzeuge, ZAPProxy, GraphQL Cop und Wapiti.

3.1.2. Vergleichskriterien

Die Vergleichskriterien der im Folgenden näher beschriebenen Werkzeuge sollen neben einigen Basisinformationen und Angaben zur Usability den Fokus auf den tatsächlichen potenziellen Einsatz, in dem im späteren Verlauf der Masterarbeit entwickelten, API Security Analysis Dashboard legen. Dabei ist die Funktionalität, Integrationsmöglichkeit inklusive Bereitstellung der Daten, Abdeckung der OWASP Top 10 API Security-Kategorien sowie auch die Visualisierbarkeit der Daten von besonderer Bedeutung und soll demnach im Mittelpunkt der Analyse der Sicherheitswerkzeuge beziehungsweise des darauffolgenden Vergleichs stehen. Zum Abschluss der Analyse der Sicherheitswerkzeuge werden die Werkzeuge zusätzlich anhand einer konkreten Beispiel-API erhoben und so die Funktionalität der einzelnen Werkzeuge besser veranschaulicht. Der Analysebogen, welchen jedes Werkzeug durchlaufen wird, setzt sich aus den folgenden Bestandteilen zusammen:

- Basisinformationen: Hier werden grundlegende Informationen über das Werkzeug, wie der Entwickler oder die aktuelle Version erhoben.

- Usability: In diesem Abschnitt geht es primär um die Benutzerfreundlichkeit des Werkzeugs, etwa wie klar die Dokumentation gestaltet ist oder es den Anforderungen des Nutzers angepasst werden kann.
- Funktionalität und Integration: Im Fokus steht hier wie einfach das Werkzeug in einen bestehenden Prozess eingebunden kann, sowie welche APIs auf welche Art und Weise analysiert werden können.
- Abdeckung der OWASP Top 10 API Security-Kategorien: Weiterführend zum Punkt Funktionalität, steht hier die Fähigkeit der Werkzeuge Schwachstellen aus der OWASP Top 10 API Security Rangliste erheben zu können.
- Einbindung von Authentifizierungsmechanismen: Da nicht nur Scans von öffentlichen APIs ohne Authentifizierung möglich sein sollen, sondern auch jene die eine Authentifizierung beinhalten wird auch dieser Punkt in die Analyse der Sicherheitswerkzeuge miteingezogen.
- Bericht: Um die Ergebnisse der Scans auch effektiv nutzen zu können, ist die Art der Berichterstattung der gefundenen Schwachstellen von großer Bedeutung, gerade in Hinsicht auf die mögliche Einbindung in ein Security Analysis Dashboard.
- Praktische Anwendung: Im letzten Schritt wird anhand von mehreren Beispiel-APIs die Funktionalität der Werkzeuge praktisch aufgezeigt. Verwendete Test-APIs sind hierbei VAmPI (erev0s, 2020), crAPI (OWASP Foundation, 2021), Rest API Goat (Hartz, 2020) und Damn Vulnerable GraphQL Application (Exposed Atoms, 2020). Es werden hier je nach Möglichkeiten der Test-API Versuche mit und ohne Authentifizierungsparametern durchgeführt.
- Fazit: In diesem Abschnitt wird der Gesamteindruck sowie die mögliche Einbindung in das Security Analysis Dashboard diskutiert.

3.2. Überblick der Sicherheitswerkzeuge

3.2.1. ZAPProxy

ZAPProxy ist eine Open Source Scanning-Software deren Ziel es ist, Sicherheitsschwachstellen in Web-Anwendungen aufzudecken. Zed Attack Proxy ist damit das weltweit meistgenutzte Scanning-Werkzeug mit einer großen Community um die stetigen Weiterentwicklung voranzutreiben (ZAP Dev Team, 2024c). Es weist dabei eine hoher Verschränkung zu den OWASP-Ranglisten auf und bietet etwaige Best Practices und Features an, um gegen die Top 10 Schwachstellen von OWASP effektive, vorgehen zu können. ZAPProxy wird aktuell von der Crash Override Open Source Fellowship zur Verfügung gestellt und ist dabei sowohl als On-Premises Variante zum Download und mitsamt grafischer Benutzeroberfläche als auch via CL (Command Line) Kommandos oder Docker Image verfügbar. ZAPProxy beinhaltet eine Vielzahl an Funktionalitäten und deckt damit eine breite Palette an Sicherheitsrisiken, auch im Bereich APIs, ab. In der nachfolgenden Analyse der Sicherheitswerkzeuge wird das Werkzeug in der Docker-Variante eingesetzt und vor allem der aktive und passive API-Scan näher analysiert.

3.2.2. GraphQL Cob

GraphQL Cob ist im Gegensatz zu dem sehr großen und umfassenden ZAPProxy Werkzeug ein sehr leichtgewichtiges auf Python basierendes Open-Source Werkzeug, um Sicherheitslücken in GraphQL

APIs aufzudecken. Aufgrund der leichtgewichtigen Scans ist besonders ein Einsatz in CI/CD Prozessen vorteilhaft (GraphQL-Cop, 2022). Um das Werkzeug einzusetzen, muss das GitHub Repository heruntergeladen und das Python Programm über die CL ausgeführt werden. Danach werden die Ergebnisse des Scans in der Konsole ausgegeben. Wie bereits sprechend im Namen enthalten ist dieses Werkzeug ausschließlich auf die Analyse von Sicherheitslücken im Bereich von GraphQL APIs beschränkt.

3.2.3. Wapiti

Wapiti ist wie schon ZAPProxy ein Open-Source Scanning-Werkzeug, um Webapplikationen auf sicherheitsrelevante Schwachstellen zu untersuchen. Dabei führt Wapiti Blackbox-Scans durch und versucht wie ein Fuzzer mögliche Schwachstellen aufzudecken (Nicolas, 2022). Das Werkzeug steht frei zum Download zur Verfügung und läuft am besten in einer Linux-Umgebung, weswegen es für die Analyse der Sicherheitswerkzeuge in einem Docker ausgeführt wird, um die benötigte Systemumgebung gewährleisten zu können. Nachdem der Scan ausgelöst wurde, wird eine HTML-Seite zur besseren Veranschaulichung der Ergebnisse in den entsprechenden Ordner gespeichert und kann für weitere Analysezwecke verwendet werden. Auch wenn das Werkzeug, wie ZAPProxy nicht reine APIs in den Fokus rückt so umfasst der Scan auch sehr wohl für APIs sicherheitsrelevante Schwachstellen.

3.3. Analyse der Sicherheitswerkzeuge

Die folgenden Analysen der Sicherheitswerkzeuge basieren auf den Dokumentationen der jeweiligen Werkzeuge (GraphQL-Cop, 2022; Nicolas, 2022; ZAP Dev Team, 2024a), sowie eigenen Erfahrungen, Erkenntnissen und Messungen während der Nutzung der Werkzeuge.

3.3.1. ZAPProxy

ZAPProxy wird zum Zeitpunkt der Analyse der Sicherheitswerkzeuge in Version 2.14.0 betrieben, wobei die letzte Aktualisierung am 12.10.2023 stattfand. ZAPProxy weist eine On-Premises Version auf, die eine Desktopanwendung beinhaltet und eine Benutzeroberfläche bietet. Die Installation hierfür ist sehr einfach und auch von Laien durchführbar. Für die Analyse der Sicherheitswerkzeuge wird aber vorrangig die Ausführung der Scans über das Dockerimage bevorzugt. ZAPProxy weist eine ausführliche Dokumentation auf, sowohl für die Desktopanwendung, als auch für die vorhandenen Docker-Images und mögliche programmatische Erweiterungen. ZAPProxy kann mittels anderem Plug-ins und Skripte auf die individuellen Anforderungen des Nutzers angepasst werden, sowie in bestehende Systeme beispielsweise über die bereitgestellte Serviceschnittstelle eingebunden werden.

ZAPProxy unterstützt den aktiven, sowie passiven Scan von APIs bei allen gängigen Typen von Serviceschnittstellen (REST, SOAP, GraphQL,...). Aktive Scans können hierbei aufgrund der Scantiefe von der gescannten Applikation beziehungsweise Serviceschnittstelle als Angriff gewertet werden (ZAP Dev Team, 2024b), weshalb nur die in Abschnitt 3.1.2 genannten Beispiel-APIs VAmPI, crAPI, Rest API Goat und Damn Vulnerable GraphQL Application für Scans im Zuge der Analyse verwendet werden. Die Scans basieren auf einer Reihe von vordefinierten Regeln welche, im Zuge der Scans überprüft werden, und von den Nutzern selbst angepasst werden können. Um aktive Scans zu starten, muss eine Schnittstellen-Dokumentation in einem gängigen und standardisierten Format, wie beispielsweise OpenAPI, zur Verfügung gestellt werden, für passive Scans reicht die URL um eine API

scannen zu können. Da eine standardisierte Schnittstellenbeschreibung nicht bei allen gescannten Test-APIs auffindbar war, konnten in manchen Beispielszenarien kein aktiver Scan durchgeführt werden. Speziell für GraphQL-Schnittstellen sind Plug-Ins mit auf diesen Typ abgestimmte Funktionalitäten verfügbar, welche für den Scan der GraphQL-API über die Desktopanwendung verwendet wird.

Der Output der Scans kann in verschiedenen Formaten dargestellt werden, darunter XML, JSON, Text oder HTML, was eine Weiterverarbeitung der Ergebnisse und somit Einbindung des Werkzeugs in bestehende Workflows oder Systeme leicht möglich macht. Außerdem werden die Ergebnisse je nach Ausführung in der Konsole oder Desktopanwendung ausgegeben. Inkludiert sind hier jeweils mitunter die Nennung der Schwachstelle, dessen Priorität (High – Medium – Low - Information), eine Beschreibung der Schwachstelle und konkrete Angriffsszenarien, sowie Gegenmaßnahmen. Die Auflistung der Ergebnisse ist im jeweiligen Format übersichtlich. Gerade die Ergebnisdarstellung via HTML weist eine gute Übersicht auf, welche allerdings nur textueller und tabellarischer Natur ist und von Visualisierungen absieht (Abbildung 15). Eine Interpretierbarkeit ist hier nur mit explizitem Vorwissen möglich. Die einzelnen Schwachstellen sind mit Beschreibungen, Lösungsansätzen, sowie Links zu OWASP-Informationsseiten versehen, um mehr Informationen zu erhalten (Abbildung 16). Auch eine Zuordnung zu wie in Kapitel 2.1.1.3 erläuterten CVE's können dem Bericht entnommen werden, welche jedoch nur der allgemeinen OWASP Rangliste zugeordnet werden können. Das Mapping der gefundenen Schwachstellen und der Kategorien der OWASP API Rangliste passiert demnach manuell. Die Automatisierbarkeit der Berichterstellung ist beispielsweise über CI/CD-Werkzeuge wie Jenkins oder GitHub Actions gut möglich, allerdings können gerade aktive Scans bei Wapiti mitunter sehr zeitintensiv ausfallen.

Alerts

Name	Risk Level	Number of Instances
SQL Injection	High	1
.env Information Leak	Medium	1
Bypassing 403	Medium	1
CORS Misconfiguration	Medium	45
Content Security Policy (CSP) Header Not Set	Medium	7
Hidden File Found	Medium	4
A Server Error response code was returned by the server	Low	742
Application Error Disclosure	Low	4
Permissions Policy Header Not Set	Low	7
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	12
Unexpected Content-Type was returned	Low	874
A Client Error response code was returned by the server	Informational	1714
Authentication Request Identified	Informational	4
Non-Storable Content	Informational	12
Session Management Response Identified	Informational	1

Abbildung 15: ZAPProxy HTTP-Bericht Alerts

Medium	.env Information Leak
Description	One or more .env files seems to have been located on the server. These files often expose infrastructure or administrative account credentials, API or APP keys, or other sensitive configuration information.
URL	http://host.docker.internal:8888/.env
Method	GET
Parameter	
Attack	
Evidence	HTTP/1.1 200 OK
Other Info	
Instances	1
Solution	Ensure the .env file is not accessible.
Reference	https://www.google.com/search?q=db_password+filetype%3Aenv https://mobile.twitter.com/svblxyz/status/1045013939904532482
CWE Id	215
WASC Id	13
Plugin Id	40034

Abbildung 16: ZAPProxy HTTP-Bericht Beispieldaten

ZAPProxy deckt die Kategorien der OWASP API-Top 10 Rangliste gut ab – lediglich jene Kategorien, die einen starken organisatorischen oder strategischen und weniger technischen Fokus aufweisen konnten, nicht abgedeckt werden (API6). Speziell die Kategorie API8 - Security Misconfiguration kann häufig in Beispieldaten gefunden werden.

Dabei unterstützt ZAPProxy Scans mit und ohne Authentifizierung, wobei hier im Zuge der Analyse mit Hilfe der Beispiel-APIs keine großen Unterschiede beim Auffinden der Schwachstellen festgestellt werden konnte. Grundsätzlich ist es möglich alle gängigen Authentifizierungsmechanismen zu werden (OAuth, Basic Authentication, ...), die auch einfach in Docker-Commands eingebunden oder im Zuge eines Scans über die Desktopanwendung dort hinterlegt werden können.

Die Ergebnisse der gescannten Beispiel-APIs können in Tabelle 2 eingesehen werden. Hier zeigt sich, dass besonders der aktive Scan eine gute Abdeckung der Kriterien aufweist. Hier kann das Werkzeug beispielsweise im Falle einer Test-API beim Scan Schwachstellen in 9 der 10 OWASP Kategorien abdecken. Bei passiven Scans derselben API können nur Schwachstellen aus 3 Kategorien erkannt werden. Ein Scan mit und ohne Authentifizierung macht hierbei keinen Unterschied. Mit steigender Anzahl der gescannten URLs steigt jedoch auch die Scan Zeit exponentiell an. Während ein passiver Scan bei 4 gescannten URLs etwa eine Minute brauchte, sind es bei einem aktiven Scan von crAPI mit 434 URLs über 16 Minuten. Die Durchschnittszeit aller Scans beträgt 528 Sekunden bei aktiven und 60 Sekunden bei passiven Scans. Eine genaue Auflistung der gefundenen Schwachstellen, sowie der Analysebogen der Analyse der Sicherheitswerkzeuge werden im Anhang C beigelegt.

Tabelle 2: ZAPProxy Beispieldaten

API	Aktiv/Passiv	Authentifizierung	Prioritäten	OWASP-Kategorien
vAmPI	Passiv	Ja	Medium: 2x Low: 5x Information: 2x	API1: 1x API3: 1x API8: 7x
vAmPI	Passiv	Nein	Medium: 2x Low: 5x Information: 2x	API1: 1x API3: 1x API8: 7x
vAmPI	Aktiv	Ja	Medium: 5x	API1: 2x

			Low: 5x Information: 4x	API2: 3x API3: 2x API5: 1x API8: 9x API9: 1x API10: 2x
vAmPI	Aktiv	Nein	Medium: 5x Low: 5x Information: 4x	API1: 2x API2: 3x API3: 2x API5: 1x API8: 9x API9: 1x API10: 2x
crAPI	Passive	Ja	Medium: 2x Low: 5x Information: 2x	API1: 1x API3: 1x API8: 7x
crAPI	Passive	Nein	Medium: 2x Low: 5x Information: 2x	API1: 1x API3: 1x API8: 7x
crAPI	Aktiv	Ja	Medium: 5x Low: 5x Information: 4x	API1: 1x API2: 3x API3: 1x API5: 1x API8: 9x API9: 1x API10: 2x
crAPI	Aktiv	Nein	High: 1x Medium: 5x Low: 5x Information: 4x	API1: 2x API2: 3x API3: 2x API5: 1x API8: 9x API9: 1x API10: 2x
REST API GOAT	Passive	Ja	Medium: 2x Low: 3x Information: 1x	API3: 1x API8: 5x
REST API GOAT	Passive	Nein	Medium: 2x Low: 3x Information: 1x	API3: 1x API8: 5x
Damn Vul- nerable GraphQL	-	Nein	Medium: 2 Low: 4 Information: 7	API2: 2x API9: 9x

3.3.2. Wapiti

Wapiti ist zum Zeitpunkt der Analyse in der Version 3.1.8 verfügbar, die letzte Aktualisierung fand am 16.05.2024 statt. Das Werkzeug besitzt keine Benutzeroberfläche und wird über CLI-Commands gesteuert. Das Set-Up des auf Python-basierenden Werkzeuge kann in einer Linux-Umgebung als ein-

fach betrachtet werden, in einer Windows-Umgebung, wie es bei dieser Analyse der Sicherheitswerkzeuge der Fall war, ist das Set-Up anspruchsvoller. Um die Ausführung zu vereinfachen beziehungsweise mögliche Probleme und Wechselwirkungen mit einer Ausführung mit Windows zu vermeiden, werden die Scans in einem selbst erstellten Dockercontainer gestartet.

Die Qualität, sowie der Umfang der Dokumentation und hinter dem Open-Source-stehende Community lassen sich als eher spärlich und wenig umfangreich beschreiben. Auch die Anpassung an individuelle Anforderungen der Nutzer ist nur beschränkt über Scanoptionen möglich. Wapiti unterstützt die gängigen Service-Schnittstellen-Typen wie REST, SOAP und GraphQL, wobei letzteres im Zuge der Analyse der Sicherheitswerkzeuge nicht getestet wird. Das Werkzeug arbeitet mit aktiven Scans, welche, wie schon bei ZAPProxy erwähnt, als potentielle Angriffe für die Serviceschnittstelle erkannt werden könnte. Demnach ist es auch bei diesem Werkzeug wichtig, Tests nur mit eigenen oder dafür vorgesehene APIs durchzuführen. Die Datenbereitstellung ist mittels Scanoption festlegbar und in vielen gängigen Formaten, wie HTML, XML, TXT, JSON oder CSV möglich. Eine Integration zu anderen Werkzeuge oder Systemen ist nur begrenzt möglich. Wapiti ist speziell für den Einsatz von CI/CD-Pipelines konzipiert, was eine Einbindung hier einfach macht. Andere Einbindungen oder Kommunikationswege sind allerdings stärker beschränkt oder mit größerem programmatischem Aufwand verbunden.

Die Abdeckung der OWASP Top 10 API-Security Kategorien kann als ausreichend angesehen werden. Alle Kategorien bis auf API4 – API6 können auf Basis der Dokumentation ausgemacht werden. In den Beispielsscans sind allerdings nur Schwachstellen der Kategorie API8 (Security Misconfiguration) auffindbar. Scans können mit und ohne Authentifizierung durchgeführt werden, dabei werden allerdings primär einfache Authentifizierungsmechanismen wie Basic Auth unterstützt. Die Einbindung dieser Authentifizierungstypen ist einfach, bei komplexeren Typen wie OAuth, kann eine Einbindung aber mitunter schnell komplex werden. Eine Berichterstattung nach dem Scan ist standartmäßig via HTML verfügbar. Hier beschränken sich die Informationen auf textuelle Beschreibungen der Schwachstellen, ohne Visualisierung oder Priorisierung dieser (Abbildung 17). Einige allgemeine Links zu potentiellen Lösungen werden außerdem im Bericht angegeben. Die Automatisierung der Berichterstattung ist aufgrund des idealtypischen Einsatzes in der CI/CD-Pipeline sehr einfach und schnell durchführbar.

Wapiti vulnerability report

Target: <http://host.docker.internal:8888/>

Date of the scan: Sun, 28 Apr 2024 13:21:06 +0000. Scope of the scan: folder. Crawled pages: 21

Summary

Category	Number of vulnerabilities found
Backup file	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	1
Cross Site Request Forgery	0

Abbildung 17: Wapiti HTTP-Bericht

Das Ende der Analyse markiert die Beispielscans der Test-APIs welche in Tabelle 3 eingesehen werden können. Wie bereits erwähnt, kann in den Scans lediglich Schwachstellen der Kategorie API8 ausgemacht werden. Die Scans sind allerdings mit einer Durchschnittszeit von 12.7 Sekunden pro Scan vergleichsweise sehr schnell, was für einen effektiven Einsatz in CI/CD Pipelines spricht. Eine genaue Auflistung der gefundenen Schwachstellen, sowie der Analysebogen der Analyse der Sicherheitswerkzeuge werden im Anhang C beigelegt.

Tabelle 3: Wapiti Beispielscans

API	Authentifizierung	OWASP-Kategorien
vAmPI	Nein	API8: 2x
crAPI	Ja	API8: 2x
crAPI	Nein	API8: 2x
REST API GOAT	Ja	API8: 2x
REST API GOAT	Nein	API8: 2x

3.3.3. GraphQL Cop

Das Werkzeug GraphQL Cop ist zum Zeitpunkt der Analyse der Sicherheitswerkzeuge in der Version 1.12 verfügbar und wurde am 03.11.2022 das letzte Mal aktualisiert. GraphQL Cop ist Python basieret und kann über ein via Git-Hub zur Verfügung gestelltes Repository gestartet werden. Mithilfe von CLI-Commands können die Scans in Folge initiiert werden. Das Set-Up ist in Linux-Umgebung vergleichsweise einfach, in Windows-Umgebung etwas komplexer allerdings nach dessen Konfiguration einwandfrei ausführbar. Für diesen Analyse der Sicherheitswerkzeuge wird ein Set-Up in Windows Umgebung durchgeführt. Die Qualität und Umfang der Dokumentation ist eher spärlich und es werden nur

Basisinformationen über das Werkzeug und dessen Set-Up preisgegeben. Die Anpassung des Werkzeugs an die Bedürfnisse der Nutzer ist deshalb auch nur wenig bis gar nicht beschrieben – es existieren Scanoptionen, allerdings keine Plug-Ins oder ähnliches.

GraphQL Cop ist wie der Name schon verspricht, ausschließlich für die Sicherheitsanalyse von GraphQL Serviceschnittstellen geeignet, weshalb hier auch nur eine Test-API gefunden und analysiert werden konnte. Das Werkzeug verwendet, wie auch Wapiti und ZAPProxy aktive Scans zum Auffinden der Schwachstellen, was beim Scan einer produktiven API zu Angriffsreaktionen führen könnte. Die Daten können nach erfolgreichem Scan in Text oder JSON-Format dargestellt werden. Weshalb eine Integrierung mit anderen Werkzeugen oder Systemen mithilfe von CLI-Commands möglich ist, wie beispielsweise die Einbindung in einer CD/CI Pipeline. Eine Integration über eine bestehende API oder ähnliches ist allerdings nur mit größerem programmatischem Aufwand möglich. Auch beim Thema Authentifizierung ist dies der Fall. Einfache Authentifizierungstypen wie Basic Authentication werden unterstützt und können als Scanoptionen mit im CLI-Command angegeben werden. Komplexere Mechanismen benötigen allerdings einen programmatischen Mehraufwand.

Das Werkzeug deckt basierend auf der in der Dokumentation erläuterten möglichen auffindbaren Schwachstellen 4 der 10 OWASP Top 10 API Security Kategorien ab. Im Beispieldaten kann man auch Schwachstellen zu allen 4 Kategorien gefunden werden. Aufgrund der Spezialisierung auf GraphQL-Serviceschnittstellen lässt sich vermuten, dass diese vier Kategorien bei diesen Typen von API von besonderer Bedeutung sind und deswegen verstärkt überprüft werden.

Die standardmäßige Berichterstattung findet über die Konsole statt. Hier werden die jeweiligen Schwachstellen und deren Prioritäten in absteigender Reihenfolge angegeben. Die einzelnen Schweregrade der Schwachstellen sind farblich unterlegt, jedoch ansonsten rein textuell (Abbildung 18). Es werden keine Lösungsansätze genannt oder auf Ressourcen wie OWASP verwiesen, auch Visualisierungen fehlen. Ohne technisches Vorwissen im Bereich API Security, sowie GraphQL Serviceschnittstellen ist eine Interpretation der Ergebnisse ein schwieriges Unterfangen. Die Automatisierung der Berichterstattung ist aber aufgrund der Leichtgewichtigkeit des Werkzeugs sehr einfach. Beispielsweise könnte ein Scan in einer CI/CD Pipeline eingebaut werden und bei jedem Commit ausgelöst werden, um die aktuelle Sicherheit der Serviceschnittstelle zu evaluieren. Mit einer Scanzeit von 5 Sekunden wäre dies ohne große zeitliche Verluste möglich. Die Effizienz des Werkzeugs wirkt daher sehr solide, was bei größeren APIs und demnach einer steigenden Anzahl an zu scannenden URLs passiert, konnte bei dieser Analyse der Sicherheitswerkzeuge allerdings nicht überprüft werden.

```

http://localhost:5013/playground does not seem to be running GraphQL. (Consider using -f to force the scan if GraphQL does exist on the endpoint)
http://localhost:5013/console does not seem to be running GraphQL. (Consider using -f to force the scan if GraphQL does exist on the endpoint)
[HIGH] Alias Overloading - Alias Overloading with 100+ aliases is allowed (Denial of Service - /graphql)
[HIGH] Array-based Query Batching - Batch queries allowed with 10+ simultaneous queries (Denial of Service - /graphql)
[HIGH] Directive Overloading - Multiple duplicated directives allowed in a query (Denial of Service - /graphql)
[HIGH] Directive Overloading - Multiple duplicated directives allowed in a query (Denial of Service - /graphql)
[HIGH] Field Duplication - Queries are allowed with 500 of the same repeated field (Denial of Service - /graphql)
[LOW] Field Suggestions - Field Suggestions are Enabled (Information Leakage - /graphql)
[LOW] Field Suggestions - Field Suggestions are Enabled (Information Leakage - /graphql)
[MEDIUM] GET Method Query Support - GraphQL queries allowed using the GET method (Possible Cross Site Request Forgery (CSRF) - /graphql)
[LOW] GraphQL IDE - GraphQL Explorer/Playground Enabled (Information Leakage - /graphql)
[HIGH] Introspection - Introspection Query Enabled (Information Leakage - /graphql)
[HIGH] Introspection-based Circular Query - Circular-query using Introspection (Denial of Service - /graphql)
[MEDIUM] POST based url-encoded query (possible CSRF) - GraphQL accepts non-JSON queries over POST (Possible Cross Site Request Forgery - /graphql)
[INFO] Unhandled Errors Detection - Exception errors are not handled (Information Leakage - /graphql)

```

Abbildung 18: GraphQL Cop Bericht

Wie schon ZAPProxy und Wapiti wird auch GraphQL Cop mithilfe von Test-APIs beispielhaft eingesetzt, um Sicherheitsscans durchzuführen. Die Ergebnisse der Sicherheitsanalyse lassen sich in Tabelle 4 finden. Im Gegensatz zu den vorherigen beiden Scans zeigt sich hier eine überdurchschnittliche Auffindung von hohen Sicherheitsrisiken, was womöglich auf die konkrete Spezialisierung des Werkzeugs und die darauf abgestimmte Scanweise zurückzuführen ist. Eine genaue Auflistung der gefundenen Schwachstellen, sowie der Analysebogen der Analyse der Sicherheitswerkzeuge werden im Anhang C beigefügt.

Tabelle 4: GraphQL Cop Beispielscan

API	Prioritäten	OWASP-Kategorien
Damn Vulnerable GraphQL	High: 6x Medium: 2x Low: 3x Information: 1x	API2: 2x API3: 1x API4: 5x API9: 3x

3.4. Ergebnis und Vergleich der Sicherheitswerkzeuge

Bei der Analyse und dem Vergleich der Werkzeuge ZAPProxy, Wapiti und GraphQL Cop in Bezug auf ihre Leistungsfähigkeit und Eignung für Sicherheitsscans verschiedener API-Schnittstellen, ergeben sich verschiedene Aspekte, die beachtet werden müssen:

ZAPProxy zeigt dabei eine umfassende Fähigkeit, eine breite Palette von Sicherheitsrisiken zu erkennen, wie in den Beispielscans dargestellt. Mit Unterstützung für sowohl aktive als auch passive Scans bietet es eine tiefe und flexible Analysemöglichkeit. Die einfache Einbindung in CI/CD-Pipelines und die Unterstützung verschiedener Authentifizierungsmechanismen machen es zu einem vielfältig einsetzbaren Werkzeug. Die Möglichkeit, Ergebnisse in verschiedenen Formaten auszugeben, erleichtert die Integration in bestehende Workflows.

Wapiti, obwohl es über keine grafische Benutzeroberfläche verfügt und hauptsächlich über CLI gesteuert wird, bietet effektive aktive Scans. Die Dokumentation und Community-Unterstützung scheint etwas eingeschränkter als bei ZAPProxy. Die Scans sind schnell, was sie ideal für die Integration in

CI/CD-Pipelines macht, obwohl die Flexibilität in Bezug auf die Anpassung und die Authentifizierungsoptionen begrenzter ist. Im Bereich Auffindbarkeit der Schwachstellen gibt es jedoch Abstriche.

GraphQL Cop ist spezialisiert auf GraphQL-APIs und bietet durch diese Spezialisierung eine tiefere Analyse spezifischer Sicherheitsrisiken dieser Technologie. Die Scan-Fähigkeiten sind spezifisch auf den Einsatz mit GraphQL-Schnittstellen ausgerichtet. Die Ergebnisse zeigen dabei eine hohe Erkennungsrate für spezifische Sicherheitsprobleme, ein Einsatz mit anderen API-Typen ist allerdings somit nicht möglich.

Im direkten Vergleich lassen sich die folgenden drei Hauptvergleichspunkte ausmachen:

- **Abdeckung von Sicherheitskategorien:** ZAPProxy zeigt eine breite Abdeckung der OWASP Top 10 Kategorien, auch in komplexen Szenarien mit und ohne Authentifizierungsmechanismen. Wapiti scheint hier in der Abdeckung begrenzter zu sein und kann auch bei komplexeren Authentifizierungsszenarien nicht mithalten. Bei den Beispielscans wird deutlich, dass ZAPProxy und auch GraphQL Cop hier umfangreichere Ergebnisse erzeugen als Wapiti.
- **Usability:** ZAPProxy besticht hier durch das ein einfaches Set-Up, die umfassende Dokumentation und bei Bedarf eine Desktopanwendung mit integrierten Userinterface, während die anderen Werkzeuge leichtgewichtiger sind und auf CLI-Kommandos setzen. Dokumentation und die dahinterstehende Community sind bei diesen Werkzeugen weniger umfassender und ein Set-Up primär für Linux-Umgebung ausgerichtet.
- **Integration und Automatisierung:** Alle drei Werkzeuge unterstützen die Integration in CI/CD-Pipelines, wobei ZAPProxy und Wapiti durch ihre Fähigkeit, verschiedene Ausgabeformate zu unterstützen, besonders geeignet scheinen. GraphQL Cop ist ebenfalls gut integrierbar, bietet jedoch möglicherweise weniger Ausgabeoptionen. Aufgrund der Scangeschwindigkeit bieten sich hier besonderes die leichtgewichtigen Werkzeug GraphQL Cop und Wapiti an. ZAPProxy ist hierbei vielleicht langsamer, allerdings bietet es dafür ein umfassendes Angebot an Integrationsmöglichkeiten.
- **Spezialisierung:** GraphQL Cop ist besonders für GraphQL-APIs geeignet und könnte in Umgebungen, in denen GraphQL eine zentrale Rolle spielt, die bevorzugte Wahl sein. Sollte das Werkzeug aber möglichst viele Typen von Service-Schnittstellen abdecken empfiehlt sich eher der Einsatz von ZAPProxy bei Bedarf in Kombination mit etwaigen Plugins.

Abschließend lässt sich hier sagen, dass, ZAPProxy sich gut für breit gefächerte Anwendungsfälle und komplexe Umgebungen eignet, während Wapiti schnelle und effiziente Scans in weniger komplexen Setups bietet. GraphQL Cop ist ideal für spezialisierte GraphQL-Umgebungen, in denen tiefere Kenntnisse der GraphQL-Sicherheit erforderlich sind.

Die Wahl eines geeigneten Werkzeugs hängt grundsätzlich stark von den spezifischen Sicherheitsanforderungen, der Systemumgebung und den bestehenden Infrastrukturen ab. Für die Implementierung des Dashboards wäre es ideal, alle drei Werkzeuge parallel einzubinden, um eine möglichst breite Abdeckung von Schwachstellen zu erreichen und deren Schweregrad sowie Priorität optimal

zu bewerten. Der Fokus liegt dabei besonders auf einer klaren Zuordnung der identifizierten Schwachstellen zu den OWASP-Kategorien, um eine transparente Sicherheitsbewertung sicherzustellen. Aufgrund begrenzter Ressourcen fiel die Entscheidung, im Prototyp vorerst nur ZAPProxy als Werkzeug einzusetzen. ZAPProxy überzeugt durch seine umfassende Dokumentation, hohe Integrationsfähigkeit, die Bereitstellung der Daten in verschiedenen Formaten und die Möglichkeit, Scans unabhängig von der Umgebung durchzuführen. Ein weiterer Vorteil ist die Unterstützung aller gängigen API-Typen, was ZAPProxy zu einer idealen Wahl für die prototypische Implementierung macht. Zeitliche Aspekte der Scans sind hier im ersten Schritt weniger relevant.

4. API Security Dashboard

Die Visualisierung von Sicherheitsrisiken ist von zentraler Bedeutung, um API-Betreiber/-innen eine effektive und intuitive Möglichkeit zur Analyse und Überwachung der Sicherheitslage ihrer APIs zu bieten. In den folgenden Kapiteln soll aufgezeigt werden, wie ein Dashboard unter Berücksichtigung der OWASP Top 10 API-Sicherheitsrisiken konzipiert und implementiert werden kann, um einen besseren Überblick über potenzielle Schwachstellen zu ermöglichen und die Sicherheit APIs fortlaufend zu gewährleisten.

4.1. Zielsetzung und Methodik

Der Ansatz zur Umsetzung des API Security Analysis Dashboards weist einen starken explorativen Charakter auf, dessen Ziel es ist, API-Schwachstellen visuell bestmöglich abbildbar zu machen. In der aktuellen Phase des Design Science Ansatzes nach Johannesson und Perjons (2014) sollen Anforderungen für das Artefakt definiert werden, bevor es zur Implementierung, Demonstration und Evaluierung des API Security Analysis Dashboards kommt.

Hauptziel des Dashboards ist die prototypische Visualisierung von API-Schwachstellen mit einem starken Fokus auf die OWASP Top 10 API-Sicherheitsrisiken. Hier soll eine Zuordnung zwischen den einzelnen Schwachstellen und den OWASP Kriterien stattfinden, um eine transparente und universell verständliche Einordnung der gefundenen Schwachstellen gewährleisten zu können. OWASP gilt in vielen Quellen wie in Veröffentlichungen vom National Institute for Standards and Technology oder Security Standards Council (NIST, 2022; Penetration Test Guidance Special Interest Group & PCI Security Standards Council, 2017) als einflussreiche Ressource für Best Practices und Schwachstellen-Einordnung im Bereich Cybersecurity, welche Schnittstellenbetreiber/-innen als wichtige Informationsquelle dienen sollen.

Dabei stehen die Visualisierungen im Fokus, da diese bei den in Kapitel 3 analysierten API-Sicherheitswerkzeuge nicht vertreten sind. Komplexe quantitative Informationen können von Individuen besser visuell als in reiner textueller Form verarbeitet werden (Tufte & Graves-Morris, 1983). Statt einer textuellen Darstellung der gefundenen Schwachstellen stehen daher deren grafische Abbildung im Mittelpunkt, was so aus informationsverarbeitungstechnischen Sicht (Tufte & Graves-Morris, 1983) den Nutzer/-innen dabei helfen soll, ein tiefergehendes Verständnis des Sicherheitsstatus der API zu erlangen. Ein beispielhafter JSON-Output des Sicherheitswerkzeuge ZAPProxy kann in Abbildung 23 eingesehen werden. Die darauf folgende Kategorisierung und Strukturierung der Visualisierungen soll ebenso die Übersichtlichkeit des Dashboards wahren und das Verständnis der bereitgestellten Informationen gewährleisten, selbst wenn diese unterschiedliche Fokusse aufweisen (Koffka, 2013; Tufte & Graves-Morris, 1983).

Ein weiteres Ziel des API Security Analysis Dashboard leitet sich aus der dynamischen Komponente von APIs und somit auch deren Sicherheitsschwachstellen ab. APIs sind sehr schnelllebig und genauso sind es deren Schwachstellen (OWASP Top 10 API Security Project Team, 2023a, 2023i) demnach ist es unerlässlich eine Serviceschnittstelle nicht nur einmal zu analysieren, sondern dies laufend zu initiieren. Auch weisen Informationen, welche auf einem Zeitstrahl angezeigt werden einen informa-

tionsverarbeitungstechnischen Mehrwert auf und helfen so Nutzer/-innen Informationen besser verarbeiten zu können (Tufte & Graves-Morris, 1983). Weitere Visualisierungstypen, welche beispielsweise das derzeitige Risiko der API beleuchten oder auf die Verteilung der OWASP-Kategorie fokussieren, sind ebenfalls entscheidend, um den Schnittstellenbetreiber/-innen dabei zu helfen eine API absichern zu können (OWASP Foundation, n.d.-a).

4.2. Anforderungen

Abgeleitet von der geraden beschriebenen Zielsetzung, sowie von der in Kapitel 3 durchgeföhrten Analyse der Sicherheitswerkzeuge können einige Anforderungen für das API Security Analysis-Dashboard definiert werden. Besonders wichtig sind hier jene Aspekte, welche über einen reinen API-Scan der analysierten Werkzeuge nicht zureichend zufriedengestellt werden können. Ziel ist es, ein Dashboard zu entwickeln, das über die grundlegenden Funktionen der vorhandenen Werkzeuge hinausgeht und eine tiefere und benutzerfreundlichere Sicherheitsanalyse ermöglicht. Hauptaugenmerk soll hier klar wie im Zuge der Forschungsfrage fixiert, auf der Visualisierung der Schwachstellen liegen. Die abgeleiteten Anforderungen werden nun in Folge näher erläutert.

Tabelle 5: Anforderungen Dashboard

Bereich	Anforderung	Beschreibung/Begründung
Visualisierung	OWASP Top 10 Mapping	<p>Beschreibung: Das Dashboard soll alle identifizierten Sicherheitsrisiken den OWASP Top 10 API-Sicherheitsrisiken zuordnen und visualisieren.</p> <p>Begründung: Während die Werkzeuge wie ZAPProxy und GraphQL Cop Schwachstellen identifizieren und textuell wiedergeben, fehlt eine visuelle Darstellung der gefundenen Schwachstellen, welche diese quantitative Information besser verständlich machen. Auch eine direkte Abbildung zwischen Schwachstelle und der OWASP Top 10 API-Sicherheitsrisiken findet bei den Werkzeugen nicht statt.</p>
	Zeitliche Darstellung von Schwachstellen	<p>Beschreibung: Visualisierung der Schwachstellen über einen definierten Zeitraum hinweg.</p> <p>Begründung: Die Werkzeuge bieten keine Möglichkeit, Schwachstellen und deren Trends über die Zeit hinweg zu analysieren. Eine solche Funktion würde helfen, die Effektivität von Sicherheitsmaßnahmen zu bewerten und die Entwicklung von Risiken besser zu verstehen.</p>
	Risikoübersicht und Priorisierung	<p>Beschreibung: Darstellung der Risiken im Kontext der gesamten gescannten API, inklusive individuelle Gewichtungen</p> <p>Begründung: Die Werkzeuge liefern zwar Risikobewertungen, aber sie bieten keine kontextbasierten und individuellen Gewichtungen, welche API-Betreiber/-innen dabei helfen, die wichtigsten Risiken gezielt zu adressieren.</p>
	Verteilung der OWASP-Kategorien	<p>Beschreibung: Eine grafische Darstellung, wie sich Schwachstellen über die verschiedenen OWASP-Kategorien hinweg in einer API verteilen.</p>

		<p>Begründung: Die Werkzeuge können Schwachstellen identifizieren und textuell in einen Bericht wiedergeben, aber sie bieten keine zusammengefasste Übersicht über die Verteilung dieser Schwachstellen im Kontext der OWASP Top 10 API-Sicherheitsrisiken, was für die Priorisierung von Sicherheitsmaßnahmen nützlich wäre.</p>
	Komplexitätsgerechte Visualisierungen	<p>Beschreibung: Visualisierungen, die sowohl für technische Experten als auch für weniger technikaffine Menschen zur Veranschaulichung verwendet werden können.</p> <p>Begründung: Die Ergebnisse der Werkzeuge sind oft komplex und textbasiert, was die Kommunikation und das Verständnis der Risiken für Nicht-Experten erschwert. Ein Dashboard, das die Komplexität reduziert und visualisiert, würde die Verständlichkeit und Entscheidungsfindung verbessern.</p>
Visualisierung/ Integration	Möglichkeit von Aktiven und Passiven Scans	<p>Beschreibung: Möglichkeit sowohl aktive als auch passive Scans über das Dashboard zu initiieren sowie den Unterschied visuell darzustellen.</p> <p>Begründung: Während Werkzeuge wie ZAProxy sowohl aktive als auch passive Scans unterstützen, fehlt die Möglichkeit, die Ergebnisse beider Scans nebeneinander zu analysieren. Ein Dashboard, das diese Funktion bietet, würde Nutzer/-innen helfen, ein vollständigeres Bild der Sicherheitslage zu erhalten. Es könnte beispielsweise visuell darstellen, welche Schwachstellen nur durch aktive Scans entdeckt wurden und welche auch bei passiven Scans sichtbar sind, wodurch die Effektivität der eingesetzten Sicherheitsmaßnahmen besser bewertet werden kann.</p>
Integration	Umfassende API-Unterstützung	<p>Beschreibung: Unterstützung einer breiteren Palette von API-Typen und Protokollen.</p> <p>Begründung: Während ZAProxy und Wapiti mehrere API-Typen unterstützen, fehlt oft die nahtlose Integration und zentrale Verwaltung dieser APIs in einem Dashboard. Ein zentrales Dashboard könnte diese Lücke schließen und die Analyse verschiedener API-Typen effizienter gestalten. So mit soll eine Visualisierung der Schwachstellen universell und typunabhängig stattfinden können.</p>
Erweiterung	Programmatische Erweiterbarkeit	<p>Beschreibung: Das Dashboard soll leicht programmatisch erweiterbar sein und APIs bieten, um mit anderen Sicherheitswerkzeugen oder -systemen zu kommunizieren</p> <p>Begründung: Während alle der analysierten Werkzeuge Integrationen ermöglichen, ist dies oft mit hohem Aufwand verbunden. Ein Dashboard mit gut dokumentierten und einfach zu nutzenden APIs würde die Integration und Erweiterung erleichtern.</p>

4.3. Architektur und Implementierung

Zur Implementierung des prototypischen API Security Analysis Dashboards wird eine 3-Schichten-Architektur (Fowler, 2012) gewählt (Abbildung 19). Die Datenschicht wird hier mithilfe einer Postgres SQL-Datenbank abgebildet, welche in einem Docker Container mit den erstellten Datenbänken läuft. Die Datenbanken kommunizieren mit einem Python Flask Server, eine auf REST basierende API, welche die Logikschicht des Dashboards ausmacht. Die finale Schicht ist die Präsentationsschicht mit dem Userinterface welche mit React und Vite umgesetzt wird.

Vorteile dieser 3-Schichten-Architektur sind die getrennte und vergleichsmäßig unabhängige Wartbarkeit und Flexibilität der einzelnen Schichten (Microsoft, 2010). Die Architektur setzt außerdem auf eine strikte Schichtentrennung, sodass die jeweilige Schicht nur mit jener direkt unter ihr kommunizieren kann. Die Datenschicht kann demnach nur mit einer expliziten Abfrage der Logikschicht Daten übertragen, genauso ist es zwischen Logik- und Präsentationsschicht der Fall, was zu einer erhöhten Wartbarkeit, Flexibilität und Wiederverwendbarkeit führt (Microsoft, 2010).

Die Architektur des Dashboards ist in Abbildung 19 dargestellt. Um den Sicherheitsstatus einer API zu überprüfen, wird die URL oder die Schnittstellenspezifikation der API im Dashboard eingegeben. Die bereitgestellten Informationen werden an ein Python-Backend übermittelt, das daraufhin einen Docker-Befehl ausführt, um einen ZAPProxy Scan der API zu starten. Nach Abschluss des Scans wird der Sicherheitsbericht im JSON-Format in einem manuell definierten Verzeichnis gespeichert. Das Backend überwacht dieses Verzeichnis und erkennt neue Reports, die dann gescannt und relevante Informationen in eine Datenbank geladen werden. Sobald die Datenbank aktualisiert wurde, wird der neue Sicherheits-Scan im Dashboard angezeigt. Benutzer/-innen können diesen Scan auswählen und die entsprechenden Visualisierungen einsehen. Die Visualisierungen werden über API-Aufrufe abgerufen, bei denen das Backend die Daten aus der Datenbank abfragt und im JSON-Format an das Benutzerinterface zurückgibt.

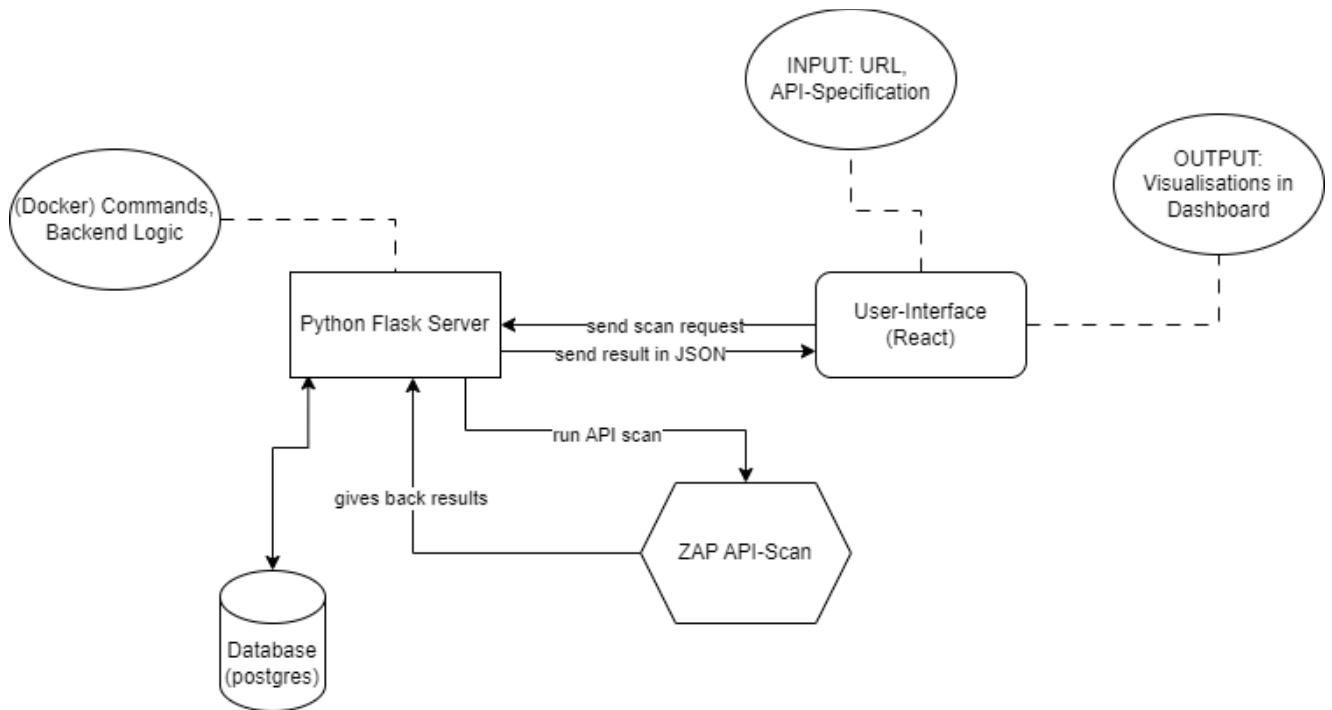


Abbildung 19: Architektur

Das prototypische API Analysis Security Dashboard kann aktuell mithilfe eines Docker-Containers sowie durch das Ausführen einer bat-Datei gestartet werden. Voraussetzung hierfür ist der Download und Initialisierung des Back- und Frontends in einer geeigneten IDE. Eine detaillierte Beschreibung zur Installation wird im Anhang D angefügt.

4.3.1. Datenschicht

Die Datenschicht hat die Aufgabe Daten konsistent zu speichern, um sie dem API Security Analysis Dashboard laufend zur Verfügung zu stellen. Sie besteht dabei aus zwei voneinander unabhängigen Datenbanken, welche auf einer Postgres Instanz liegen. Es handelt sich somit um relationale SQL-Datenbanken um eine systematische und konsistente Speicherung der Daten zu erzielen.

Die Datenbank „api_dashboard“, dessen Modell in Abbildung 20 gezeigt wird, verwaltet hierbei alle Sicherheitsscans, die darin gefundenen Schwachstellen, sowie die Abbildung zwischen den OWASP-Kategorien und Prioritäten auf die jeweiligen Schwachstellen. Auch die Werkzeuge werden darin verwaltet, sodass bei Bedarf mehrere Scanning-Werkzeuge eingebunden werden könnten. Jeder Scan weist eine ID, einen Scanzeitpunkt, ein Werkzeug und URL auf, sowie die Information, ob es sich um einen aktiven oder passiven Scan handelt. In einem Scan werden null bis n Schwachstellen gefunden, welche eine ID, einen Namen, eine Beschreibung, eine Priorität, sowie Anzahl an Vorkommen der Schwachstellen und einen Boolean-Wert, ob die entsprechende Schwachstelle bei Scans derselben URL im letzten Monat vorgekommen ist, beinhalten. Die ID der Schwachstelle ist dabei in einer separaten Tabelle zusammen mit den OWASP Kategorien abgebildet, da es vorkommen kann, dass einen

Schwachstelle mehrere OWASP-Kategorien aufweist. Sowohl die OWASP Top 10 API-Sicherheitskategorien als auch die Prioritäten werden am Beginn durch ein Initialisierungsskript konfiguriert und während der Verwendung des API Security Analysis-Dashboards nicht verändert.

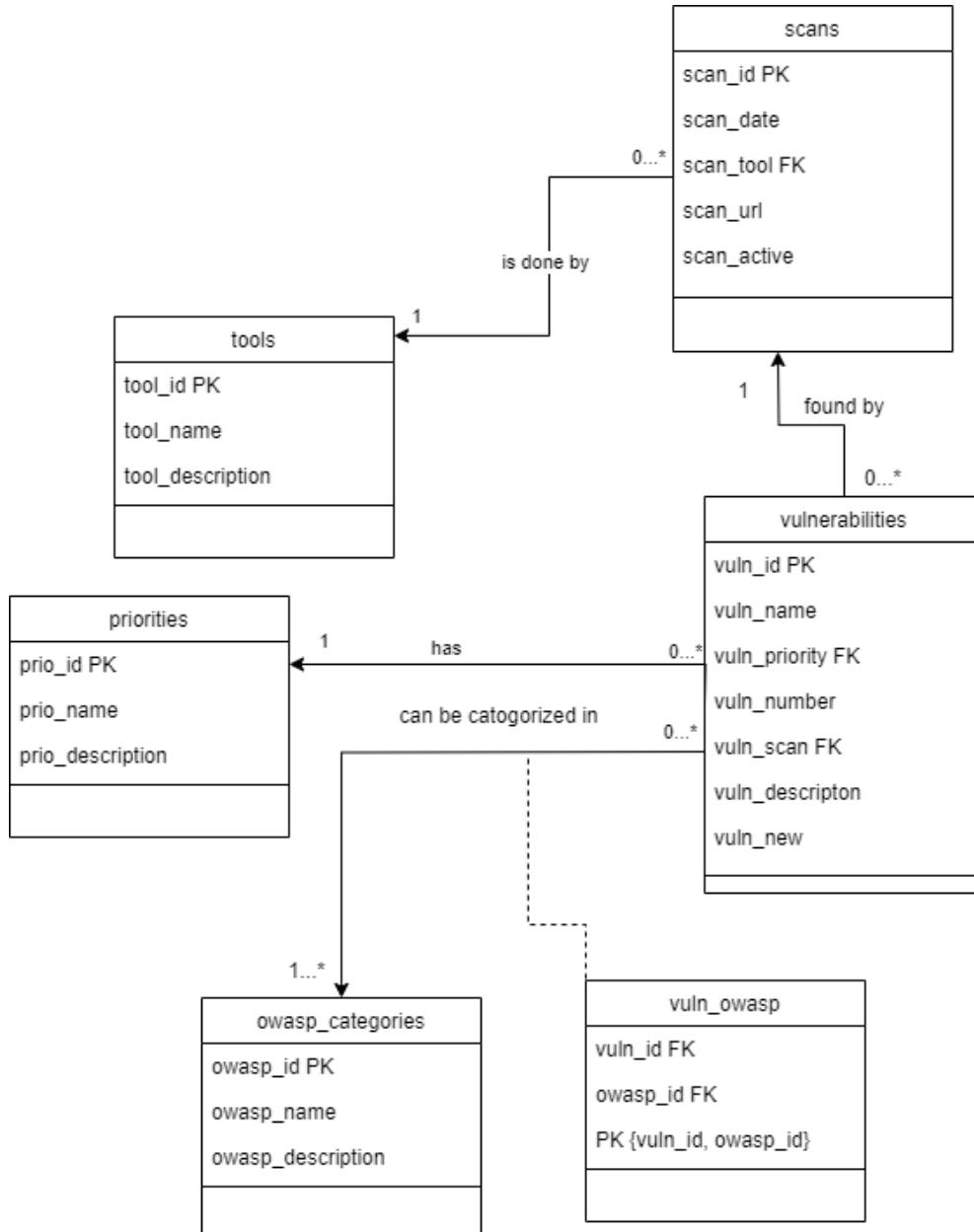


Abbildung 20: UML Diagramm Dashboard

Das Modell der zweiten Datenbank „user_database“ ist in Abbildung 21 zu sehen. Diese Datenbank ist elementarer und wird derzeit nur für die Anpassung des Dashboards verwendet, konkret für die Gewichtung der einzelnen OWASP-Kategorien zur Berechnung eines Risiko-Scores. Sie ist mit dem Gedanken implementiert, bei Bedarf verschiedene Benutzer mit unterschiedlichen Costumizing-Einstellungen oder Einschränkungen bei der Anzeige von Scans, anzulegen. Aufgrund der vorhanden Spalten Name, E-Mail und Passwort wäre das Dashboard auch über einen Log-In Prozess erweiterbar.

Dies ist aber in diesem Prototyp noch nicht realisiert. Stattdessen wird während der Initialisierung der Datenbanken ein Standardbenutzer angelegt, welcher die jeweiligen Gewichtungen hinterlegt hat.

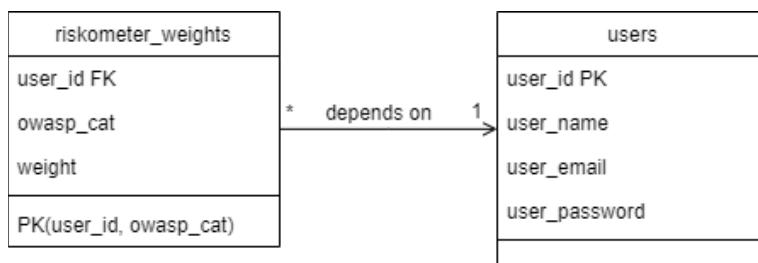


Abbildung 21: UML Diagramm User Database

4.3.2. Logikschicht

Die Logikschicht speichert und ruft die in der Datenbank vorhandenen Informationen ab und stellt sie im Falle einer API-Abfrage der Präsentationsschicht zur Verfügung. Das Backend hat daher zwei Hauptaufgaben; erstens die in der Datenbank gespeicherten Informationen dem Dashboard zur Verfügung zu stellen und zweitens die vom ZAPProxy Scan abgelegten Reports zu scannen und deren Informationen in der Datenbank zu speichern. Zur Umsetzung dieser Aufgaben werden zwei Python Anwendungen implementiert.

Die erste Python Anwendung ist für Bereitstellung der Informationen aus den gescannten Berichten über eine Restful API verantwortlich. „API.py“ wird hierbei als ein Python Flask Server implementiert und verarbeitet API-Anfragen des Frontends. Die jeweiligen Daten werden im Anschluss über eine Datenbankabfrage im JSON-Format an das Frontend zurückgegeben. Neben Endpunkten welche die Daten von Scans, Vulnerabilities und Benutzer bereitstellen sind auch Endpunkte vorhanden, welche die jeweiligen API-Scans starten. Die verschiedenen Endpunkte können als Übersicht in Abbildung 22, sowie detaillierter bei lokal laufender API unter <http://localhost:5000/apidocs> eingesehen werden. Wird ein Request an den /run-active-scan oder /run-passive-scan Endpoint gesendet, wird ein Docker-Container gestartet, welcher den jeweiligen ZAPProxy Scan initiiert. Hierbei ist sowohl ein aktiver als auch-passiver Scan möglich, für den aktiven Scan wird die Schnittstellenspezifikation im OpenAPI_Standard und im JSON- oder yaml-Format benötigt, für den passiven Scan lediglich die zu scannende URL. Im Codebeispiel 12 wird die Konstruktion des Docker-Commands sowie dessen Aufruf für passive Scans gezeigt.

GET	/customisation	Retrieves customisation data based on user ID and/or OWASP category.	get_customisation
POST	/customisation	Updates customisation data for a user based on OWASP category.	post_customisation
POST	/run-active-scan	Runs an active API security scan on the provided OpenAPI file using OWASP ZAP.	post_run_active_scan
POST	/run-passive-scan	Runs a passive API security scan on the provided URL using OWASP ZAP.	post_run_passive_scan
GET	/scans	Retrieves a list of scans.	get_scans
GET	/vulnerabilities	Retrieves a list of vulnerabilities.	get_vulnerabilities
GET	/vulnerability_trend	Retrieves the trend of vulnerabilities over time for a specific scan URL.	get_vulnerability_trend

Abbildung 22: Schnittstellendokumentation

```
@app.route('/run-passive-scan', methods=['POST'])
def run_docker_passive():
    data = request.json
    if 'url' not in data:
        return jsonify({"error": "URL is missing"}), 400

    url = data['url']
    timestamp = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')
    json_report_file = f'api-passive-scan-report_{timestamp}.json'
    print({output_directory})

    docker_command = (
        f'docker run -v {output_directory}:/zap/wrk -t owasp/zap2docker-stable '
        f'zap-baseline.py -g api-passive-scan.conf -t {url} -J {json_report_file}'
    )
    try:
        print("Starting scan")
        result = subprocess.run(docker_command, shell=True, stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE)
```

Codebeispiel 12: Dockeraufruf

Nachdem der jeweilige Scan ausgelöst wurde, beginnt ZAPProxy mit dem aktiven oder passiven Scan der jeweiligen API und extrahiert Schwachstellen aus diesen, welche in Folge in Form eines JSON-Reports in das Zielverzeichnis gespeichert werden. Mithilfe der gesetzten Umgebungsvariable „DOWNLOAD“ in der .env Datei kann das Zielverzeichnis zur Speicherung des Berichts angepasst werden.

Für die Überwachung der ins Zielverzeichnis gespeicherten ZAPProxy Berichte ist die zweite Python Anwendung verantwortlich. Die Anwendung „Insert_Real_Data.py“ verwendet hierbei die Imports Observer und FileSystemEventHandler aus den Libraries Watchdog.Observers beziehungsweise Watchdog.Events um ein lokales Zielverzeichnis dauerhaft überwachen zu können. Genauso wie bei der API

ist auch der Pfad zum Zielverzeichnis in der .env Datei gespeichert, damit der Speicherort des Sicherheitsreports und das überwachte Verzeichnis ident bleiben. Wie im Codebeispiel 13 sichtbar wird im Falle eines neuen JSON-Reports ein Scan des Files gestartet, die relevanten Informationen wie die URL, den Scanzeitpunkt, und die dazugehörigen Schwachstellen sowie deren Prioritäten, herausgelesen, mit den dazugehörigen OWASP-Kategorien gemappt und im Anschluss in die Datenbank gespeichert. Die Abbildung der Schwachstellen auf die jeweiligen OWASP-Kategorie wird manuell, mithilfe einer Excelliste durchgeführt, da eine konkrete Zuordnung nicht Teil der API-Scan Reports ist, sowie ein LLM (Large Language Model) für diesen Einsatz zum derzeitigen Stand noch keine zuverlässigen Ergebnisse liefert. In Abbildung 23 wird ein JSON-Report des API-Scans einer Beispielschnittstelle gezeigt, dass darauffolgende Codebeispiel 14 zeigt wie die Berichte verarbeitet werden. Hier wird zunächst die Struktur des JSON-Dokuments geparsst, um die relevanten Daten, wie die URL der betroffenen Schnittstelle und die erkannten Schwachstellen, zu extrahieren. Anschließend werden diese Daten mit den entsprechenden OWASP-Kategorien verknüpft und in die relevanten Tabellen scans, vulnerabilities, vuln_owasp sowie tools, sollte es sich um ein noch unbekanntes Werkzeug zur Schwachstellenerkennung handeln, in die Datenbank „api_dashboard“ eingespeist, um eine konsistente Speicherung der Daten zu gewährleisten.

```
def on_created(self, event):
    if event.is_directory or not event.src_path.endswith('.json'):
        return
    print(f"Processing new file: {event.src_path}")
    self.process_json(event.src_path)

def process_json(self, file_path):
    with open(file_path, 'r') as file:
        data = json.load(file)
    self.insert_into_db(data, file_path)
```

Codebeispiel 13: File-Scanning

```
{
    "@programName": "ZAP",
    "@version": "2.14.0",
    "@generated": "Mon, 19 Aug 2024 16:22:49",
    "site": [
        {
            "@name": "http://host.docker.internal:8000",
            "@host": "host.docker.internal",
            "@port": "8000",
            "@ssl": "false",
            "alerts": [
                {
                    "pluginid": "10021",
                    "alertRef": "10021",
                    "alert": "X-Content-Type-Options Header Missing",
                    "name": "X-Content-Type-Options Header Missing",
                    "riskcode": "1",
                    "confidence": "2",
                    "riskdesc": "Low (Medium)",
                    "desc": "<p>The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.</p>",
                }
            ]
        }
    ]
}
```

Abbildung 23: ZAPProxy Beispielbericht

```
scan_tool_name = data.get('@programName')
scan_date = datetime.strptime(data.get('@generated'), '%a, %d %b %Y %H:%M:%S')

for site_info in data.get('site', []):
    base_url = site_info.get('@name')
    port = site_info.get('@port')

    if ':' in base_url:
        scan_url = base_url
    else:
        if base_url.endswith('/'):
            scan_url = f"{base_url[:-1]}:{port}"
        else:
            scan_url = f"{base_url}:{port}"

    [...]

    for alert in site_info.get('alerts', []):
        vuln_name = alert['alert']
        vuln_prio = alert['riskcode']
        vuln_description = alert['desc']
        vuln_number = alert['count']
```

Codebeispiel 14: File-Verarbeitung

Nachdem die Scans ausgelöst, deren Ergebnisse in die Datenbank gespeichert und vom Backend gelesen, sowie an das Frontend weitergegeben werden können, wird nun im letzten Abschnitt die Präsentationsschicht nähere erläutert.

4.3.3. Präsentationsschicht

Die Präsentationsschicht, auch als Frontend bezeichnet, bildet das zentrale Element der Applikation. Sie stellt den sichtbaren Teil dar, den die Schnittstellenbetreiber/-innen als Dashboard nutzen, um Sicherheitsüberprüfungen ihrer API durchzuführen.

Das Frontend wird mit dem JavaScript-Framework React (Meta Platforms, 2023) sowie dem Build-Tool Vite (VoidZero Inc. & Vite Contributors, 2019) umgesetzt und verwendet primär die d3.js (Bostock & Observable, 2024) Bibliothek um die Visualisierungen zu implementieren. React folgt einem modularen, komponentenbasierten Ansatz, der es ermöglicht, wiederverwendbare und isolierte Komponenten zu entwickeln. Jede Komponente kapselt spezifische Funktionalitäten und Darstellungen, wodurch der Code besser strukturiert, wartbar und erweiterbar ist. Somit teilt sich das Dashboard (Abbildung 24) in die Komponente zur Initiierung von neuen Scans, die Abbildung der vergangenen Scans und einen Button, um zur Customizing-Komponente des Dashboards zu gelangen.

In Zuge der Anpassungsfähigkeit des Dashboards ist aktuell die individuelle Gewichtung der OWASP-Kategorien möglich (Abbildung 25). Hier können Benutzer/-innen insgesamt 100 Punkte vergeben und diese beliebig auf die OWASP-Kategorien aufteilen. Je mehr Punkte eine Kategorie aufweist desto mehr Gewicht hat sie in der Risikobewertung. Übersteigt die Gesamtpunktzahl 100 kommt es zu einer Fehlermeldung und die Gewichtung kann nicht gespeichert werden.

Start New API Scan

Active Scan

Keine Datei ausgewählt

Passive Scan

Previous scans

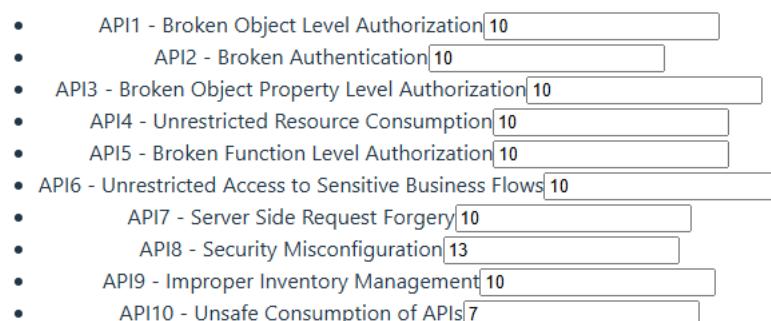
- Sun, 25 Aug 2024 17:22:44 GMT (<http://host.docker.internal:8888>)
- Sun, 18 Aug 2024 20:19:02 GMT (<https://www.example.com>)
- Sat, 17 Aug 2024 20:01:51 GMT (<https://www.example.com>)
- Wed, 07 Aug 2024 14:16:02 GMT (<http://host.docker.internal:5000>)
- Tue, 23 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 16 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 09 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 11 Jun 2024 14:13:24 GMT (<http://host.docker.internal:5000>)
- Thu, 23 May 2024 19:58:53 GMT (<http://host.docker.internal:5000>)
- Sat, 27 Jul 2024 19:20:53 GMT (<https://www.example.com>)
- Wed, 17 Jul 2024 19:19:53 GMT (<https://www.example.com>)

Abbildung 24: Dashboard Startseite

[Go Back to Scan Page](#)

Customisation

Riskometer



Total weight used: 100 / 100

Abbildung 25: Dashboard Customisation

Das Dashboard des durchgeführten API-Sicherheitsscans erscheint bei Auswahl einer der bereits durchgeführten Scans (Abbildung 26). Damit gelangt man auf die Komponente, welche die Scan-Details wiedergibt und in jener alle anderen Visualisierungskomponenten eingebunden sind. Mithilfe einer API-Anfrage an das Backend werden die Daten des aktuellen Scans in das Dashboard geladen und wiedergegeben. Wie in Abbildung 26 dargestellt ist der Startpunkt des Dashboards ein Überblick über die gescannte API, wie dessen Scanzeitpunkt, die URL, sowie ob der Scan aktiver oder passiver Natur war. Auch die Anzahl an gefundenen Schwachstellen, die OWASP Kategorie, die am häufigsten vertreten ist und die nach Priorität und Gewichtung gemessenen schwerwiegendsten Schwachstellen sind ein Teil des Überblicks. Darunter folgt eine Aufzählung der einzelnen Schwachstellen inklusive dessen ausklappbare Beschreibung und ein Link zu der jeweiligen OWASP-Seite um mehr Informationen, Lösungsansätze und Best Practices zu der einzelnen OWASP-Kategorie zu finden. Schwachstellen, die noch nie oder seit dem letzten Monat nicht in einem API-Sicherheitsscan einer URL aufgetaucht sind, werden hervorgehoben und als „New“ deklariert. Dies soll helfen, speziell neue oder bereits in Vergangenheit gelöste Sicherheitsschwachstellen visuell besser erkennbar zu machen.

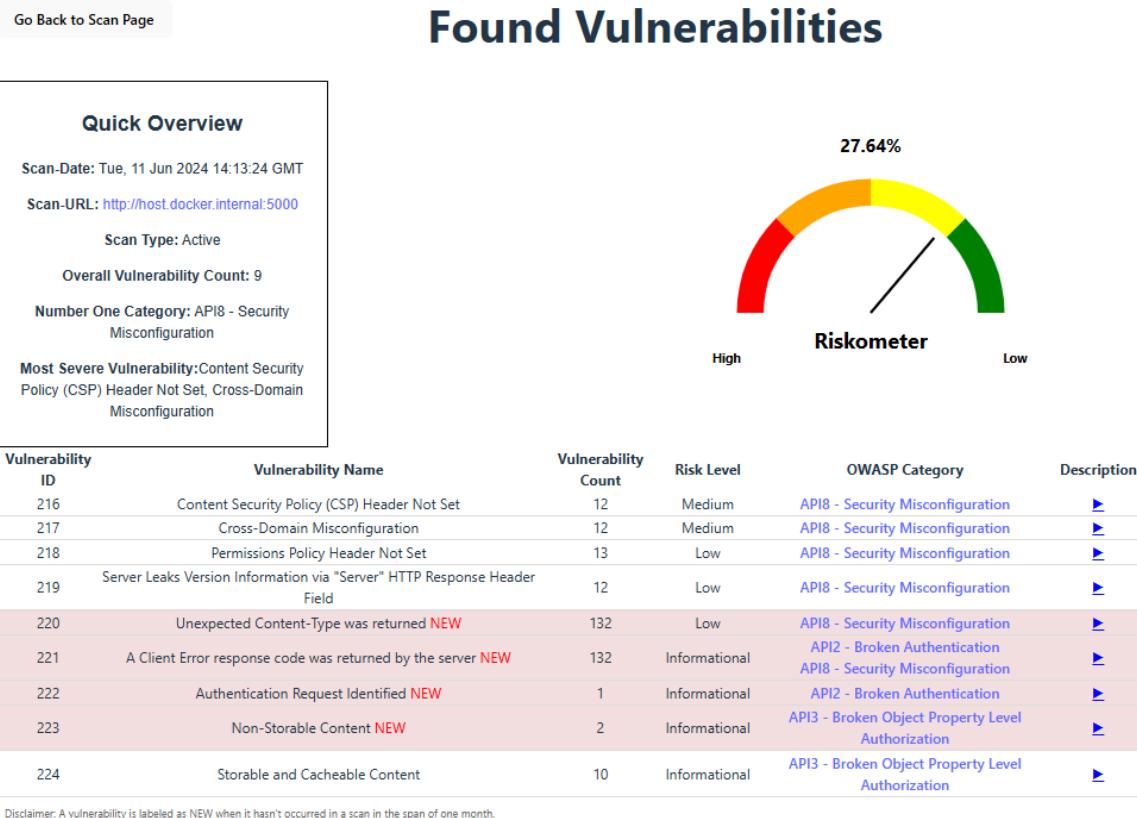


Abbildung 26: Dashboard Überblick

Auch auf den ersten Blick erkennbar ist das sogenannte „Riskometer“, ein Tachometer, welches das Risiko der gescannten API wiedergibt. Die Risikoberechnung stützt sich hier auf drei Variablen: die Priorität der Schwachstelle, welche direkt seitens des API-Scanning-Werkzeuge übermittelt wird, die Gewichtung der OWASP-Kategorien (siehe Abbildung 25), die individuell von Benutzer/-innen selbst vergeben werden kann, sowie die Anzahl der gefundenen Schwachstellen. Im Codebeispiel 15 lässt sich die Berechnung des Risk-Scores finden. Wichtig ist hierbei anzumerken, dass eine Schwachstelle

mehrere OWASP-Kategorien aufweisen kann; ist dies der Fall wird jene OWASP-Kategorie für die Risikobewertung herangezogen, welche die höchste Gewichtung aufweist. Die Konstante *Scaling-Factor* wird dazu benötigt, um ein wachsendes Risiko bei einer wachsenden Anzahl von Schwachstellen abbilden zu können.

```

const calculateRiskScore = (vulnerability) => {
    const priorityOrder = {'high': 4, 'medium': 3, 'low': 2, 'informational': 1 };
    const priority = priorityOrder[vulnerability.prio_name] ?
        vulnerability.prio_name.toLowerCase() : 'no Priority' || 0;

    const weight = vulnerability.highestWeight || 0;
    return priority * weight;
};

[...]

const uniqueVulnerabilities = filterUniqueVulnerabilities(vulnerabilities);

const totalBaseRiskScore = uniqueVulnerabilities.reduce((total, vuln) => {
    return total + calculateRiskScore(vuln);
}, 0);

const scalingFactor = 1 + (uniqueVulnerabilities.length - 1) * 0.5;

const adjustedRiskScore = totalBaseRiskScore * scalingFactor;

const maxPriority = 4;
const maxWeight = 100;
const maxPossibleBaseScore = uniqueVulnerabilities.length * maxPriority * max-
Weight;

const normalizedRiskScore = (adjustedRiskScore / maxPossibleBaseScore);

```

Codebeispiel 15: Berechnung des Risk-Scores

Nach dem allgemeinen Überblick, sowie der Risikobewertung der gescannten API werden die Schwachstellen basierend auf ihren OWASP-Kategorien in verschiedenen konzeptionellen Themen eingeteilt. Alle Visualisierungen können im Anhang E, sowie Anhang F anhand eines Beispielsscans eingesehen werden. In Folge werden einige ausgewählte Visualisierungen aufgezeigt.

Visualisierungen, die Zeitreihen abbilden sind zeitbasierte Darstellungen die speziell auf die dynamische Komponente von APIs, sowie deren Schwachstellen abzielen. In Abbildung 27 wird beispielsweise die zeitliche Veränderung der Anzahl an gefundenen Schwachstellen einer API, also einer URL, dargestellt. Eine andere Visualisierung stellt die temporäre Veränderung des Risk-Scores in den Fokus. Auch der Vergleich zwischen neuen und bereits bekannten Schwachstellen pro Zeitpunkt oder die zeitliche Veränderung der Anzahl der OWASP-Kategorien an sich sind Teil dieses Konzepts. Auch

wird bei allen diesen Visualisierungen stets unter aktiven und passiven Scans unterschieden, um missverständliche Interpretationen der Visualisierungen zu vermeiden, da aktive Scans nach Erfahrung in der Analyse der Sicherheitswerkzeuge und auch der Natur von aktiven Scans, durchschnittlich mehr Schwachstellen finden als passive. Der Scan, auf dessen Dashboard sich der/die Benutzer/-in aktuell befindet, ist mit einer roten Markierung gekennzeichnet, was es den Nutzer/-innenerleichtern soll, sich in den Grafiken zurechtzufinden. Bei einem Klick auf einen der anderen Datenpunkte in den Visualisierungen wird der/die Benutzer/-in automatisch auf das entsprechende Dashboard des jeweiligen Scans weitergeleitet.

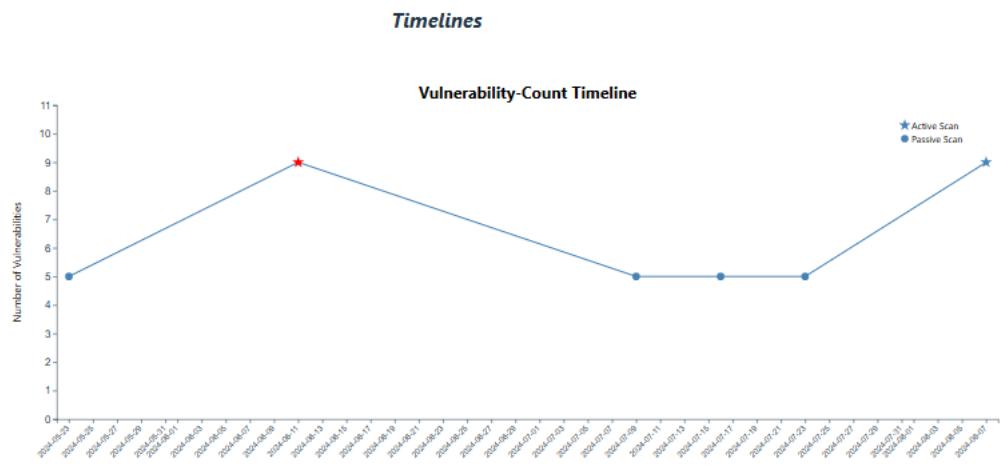


Abbildung 27: Dashboard Vulnerability-Count Zeitstrahl

Das Codebeispiel 16 zeigt einen beispielhaften repräsentativen API-Aufruf des Backends. In diesem Fall ist dieser Codeausschnitt aus der Komponente der in Abbildung 27 gezeigten Vulnerability-Count Zeitstrahl-Visualisierung entnommen. Die API_URL ist hierbei wie bereits auch im Backend beschrieben eine Umgebungsvariable welche in der .env Datei gespeichert und bei Bedarf angepasst werden kann. Innerhalb des useEffect-Hooks wird eine asynchrone Funktion *fetchData* definiert und unmittelbar ausgeführt, um Daten zu den Sicherheitslücken von der API über das Backend abzurufen. Die Daten werden verarbeitet und der Zustand der Komponente mit der Gesamtanzahl der gefunden Schwachstellen pro Scanzeitpunkt aktualisiert. So stellt die Komponente sicher, dass sie stets die aktuellen Daten für die Darstellung der Sicherheitslücken verwendet.

```
useEffect(() => {
  const fetchData = async () => {
    try {
      const response = await axios.get(` ${API_URL}/vulnerabilities?scan_url=${scanUrl}`);
      const aggregatedData = aggregateData(response.data);
      setData(aggregatedData);
    } catch (error) {
      console.error('Failed to fetch scan data:', error);
    }
  };
};
```

```
    fetchData();
}, [scanUrl]);
```

Codebeispiel 16: API-Request

Ein weiteres wichtiges Konzept der Visualisierungen ist die Verteilung von Schwachstellen in Bezug auf ihre OWASP-Kategorien. Abbildung 28 zeigt hier die allgemeine Verteilung in einem Tortendiagramm, die eine einfache und schnelle Interpretation ermöglicht. In diesem Beispiel ist die OWASP-Kategorie 8 klar die am häufigsten vertretene Kategorie, was dem/der Benutzer/-in einen erhöhten Fokus auf diese Kategorie empfehlen könnte. Ebenfalls vertreten in diesem Konzept sind die Anzahl an Vorkommnissen einer Schwachstelle in einem Scan, dessen Information ebenfalls direkt aus dem API-Sicherheitsscan kommt. Diese Anzahl kann stark ja nach API und Schwachstelle schwanken, während des Scans konnten hier beispielsweise von 1 bis über 1000 Vorkommen einer Schwachstelle ausgemacht werden

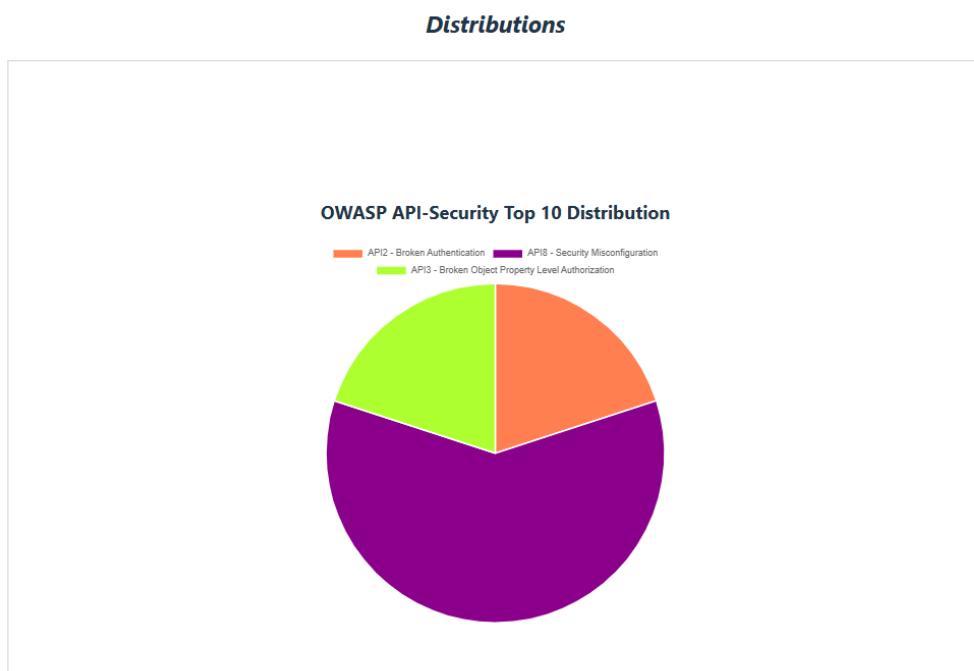


Abbildung 28: Dashboard OWASP Top 10 Distribution

Der letzte Kontext des Dashboards ist jener der Prioritäten. Eine Schwachstelle mit hohem Risiko soll demnach schnell erkannt und gelöst werden, aber auch die Risikoverteilung über die gesamte API soll helfen, einen besseren Gesamteindruck über den Sicherheitsstatus der API zu erlangen. In Abbildung 29 ist hier ein Scatter-Plot Diagramm abgebildet, mit den Risikolevels auf einer, und den Schwachstellen auf der anderen Achse. Als dritte Dimension werden die Farben, welche eine OWASP-Kategorie, beinhaltet hinzugefügt. Hier wird aufgezeigt, welche Schwachstelle besondere Aufmerksamkeit benötigen und welche Prioritäten die einzelnen gefundenen Schwachstellen eines Scans aufweisen. Im konkreten Beispiel von Abbildung 29 kann die Schwachstellen „Content Security Policy (CSP) Header not Set“ mit einem moderaten Risikolevel ausgemacht werden. Die Farbe des Datenpunkts verweist

außerdem auf die OWASP-Kategorie, in diesem Fall Kategorie 8 „Security Misconfiguration“. Sollte ein Datenpunkt zwei Kategorien zugeordnet werden können, ist es in dieser Visualisierung mithilfe eines zweifarbigem Datenpunkts sichtbar, wie es beispielsweise bei der Schwachstelle „A Client Error response code was returned by the server“ der Fall ist.

Eine weitere Visualisierung der Prioritäten findet in einem Balkendiagramm statt, um zu veranschaulichen, wie die Verteilung der einzelnen Prioritäten aussieht.

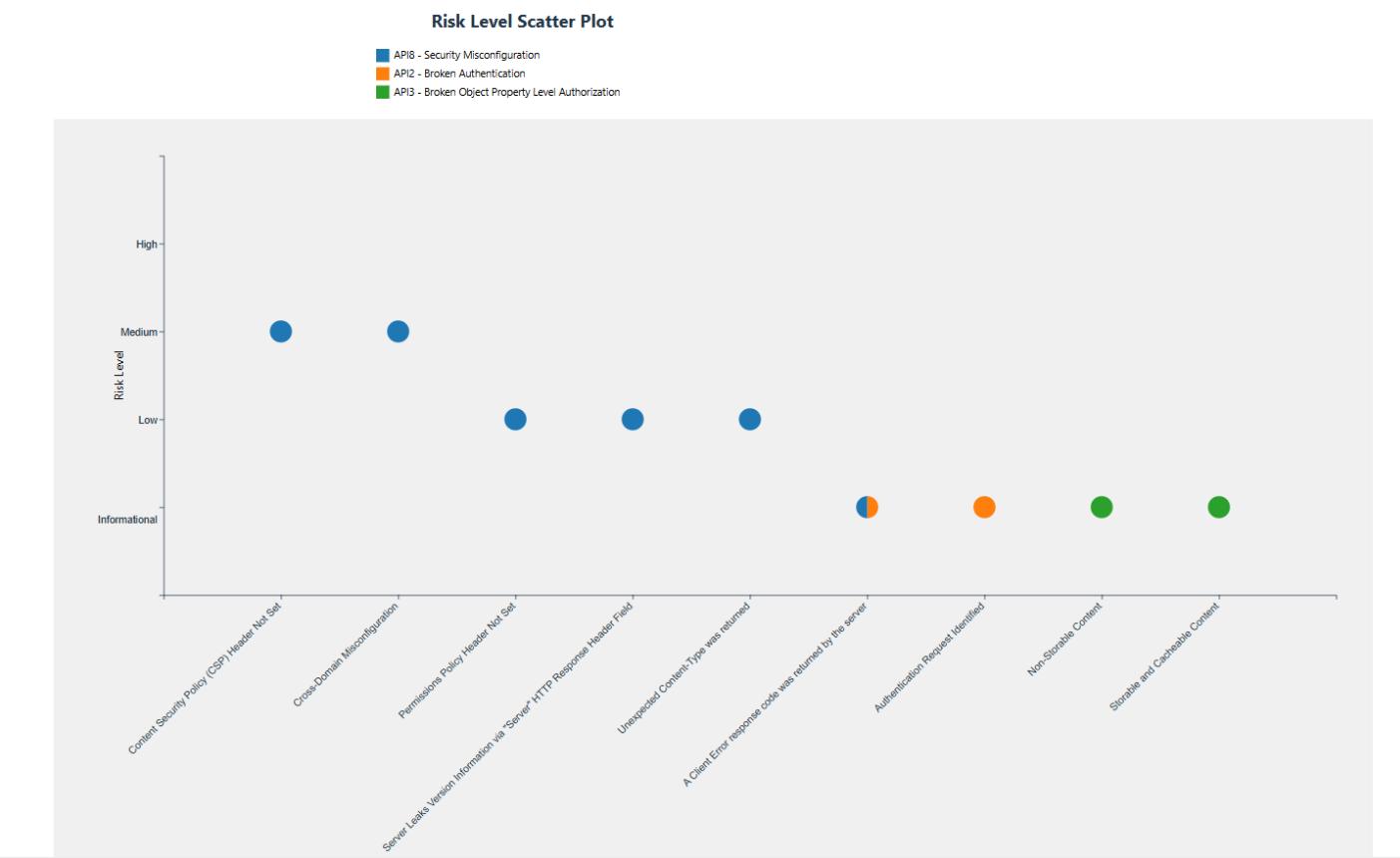


Abbildung 29: Dashboard Risk-Scatterplot

4.4. Evaluierung

Der abschließende Schritt im Design Science Prozessmodell nach Johannesson und Perjons (2014) ist die Evaluierung des erstellten Artefakts. Aus ressourcentechnischen Gründen wird die Evaluierung des vorliegenden Dashboards auf argumentativer Ebene durchgeführt, sowie mithilfe von beispielhaften Fallstudien. Eine Validierung seitens Expert/-innen oder potenziellen Benutzer/-innen wäre hier aber sicher eine gute Wahl bei weiterführenden Arbeiten an dem Dashboard.

4.4.1. Argumentative Evaluierung

In Folge werden die zuvor definierten Anforderungen mit der Implementierung gegenübergestellt, sowie in Tabelle 6 zusammengefasst aufgezeigt.

Tabelle 6: Evaluierung

Bereich	Anforderung	Umsetzung/Argumentation
Visualisierung	OWASP Top 10 Mapping	Das Dashboard ordnet Schwachstellen den OWASP Top 10 zu und stellt sie visuell dar. Dies bietet eine klare Zuordnung und erleichtert die Priorisierung von Schwachstellen, was gegenüber textbasierten Werkzeugen einen Mehrwert schafft.
	Zeitliche Darstellung von Schwachstellen	Ermöglicht Nutzern, Sicherheitsentwicklungen und die Wirkung von Maßnahmen über die Zeit zu verfolgen. Verschiedene Diagrammtypen (zum Beispiel Linien- und Balkendiagramme) zeigen die zeitliche Entwicklung und helfen, den Erfolg von Sicherheitsmaßnahmen zu analysieren.
	Risikoübersicht und Priorisierung	Visualisiert API-Risiken im Kontext, unter anderem mit einem Riskometer und Balkendiagramm zur Verteilung nach Risikokategorien. Scatterplots und zeitliche Diagramme ermöglichen eine umfassende Analyse.
	Verteilung der OWASP-Kategorien	Kreis- und Blasendiagramme zeigen die Schwachstellenhäufigkeit pro OWASP-Kategorie, sodass besonders kritische Bereiche gezielt adressiert werden können. Zeitliche Diagramme verdeutlichen Trends und unterstützen die Erkennung von Mustern in der Schwachstellenerkennung einer spezifischen API.
	Komplexitätsgerechte Visualisierungen	Ampelsysteme und intuitiv gestaltete Diagramme (zum Beispiel Riskometer oder Scatterplots) erleichtern es verschiedenen Gruppen von Nutzer/-innen eine schnelle Risikoeinschätzung. OWASP-Ressourcen bieten zusätzlichen Kontext zu Risiken und Lösungsansätzen.
Visualisierung/Integration	Möglichkeit von Aktiven und Passiven Scans	Das Dashboard unterstützt aktive und passive Scans parallel, um Schwachstellen mit beiden Methoden zu erfassen. Die Ergebnisse werden nebeneinander visualisiert, was eine umfassende Sicherheitsbewertung ermöglicht.
Integration	Umfassende API-Unterstützung	Flexibilität für verschiedene API-Typen; initialer Fokus auf ZAPProxy, wobei weitere Werkzeuge integriert werden können.
Erweiterung	Programmatische Erweiterbarkeit	Drei-Schichten-Architektur und dokumentierte Endpunkte ermöglichen die einfache Integration zusätzlicher Werkzeuge und Funktionen. Die API ist für mehrere Nutzer/-innen mit individuellen Konfigurationen anpassbar und flexibel erweiterbar, sodass langfristige Anpassungsfähigkeit gewährleistet ist.

Das prototypische Dashboard bietet eine benutzerfreundliche Visualisierung von Sicherheitsrisiken in APIs. Eine zentrale Funktion ist die Abbildung der gefundenen Schwachstellen auf die OWASP Top

10 API-Sicherheitsrisiken, was eine klare und intuitive Zuordnung ermöglicht, und den Nutzer/-innen hilft, Risiken schnell zu identifizieren und zu priorisieren. Durch die visuelle Darstellung wird ein Vorteil gegenüber textbasierten Werkzeugen wie ZAPProxy, Wapiti und GraphQL Cop geschaffen, da quantitative Informationen so greifbarer und leichter verständlich vermittelt werden. Die Abbildung der Schwachstellen auf die OWASP-Kategorien gibt den Anwender/-innen die Möglichkeit, direkt auf O-WASP-Ressourcen wie Lösungsansätze und Best Practices zuzugreifen und so die Sicherheitslage der API greifbarer zu machen.

Eine weitere wichtige Komponente des Dashboards ist die zeitliche Darstellung der Schwachstellen (Anhang E – zeitstrahlbasierte Visualisierungen), die den Nutzer/-innen erlaubt, Trends und Entwicklungen über einen definierten Zeitraum hinweg zu beobachten. Verschiedene Visualisierungstypen, welche in Anhang E eingesehen werden können, verdeutlichen die zeitliche Entwicklung, sowie die Auswirkungen der ergriffenen Sicherheitsmaßnahmen. Zum Beispiel zeigt ein Liniendiagramm den Verlauf der Risikobewertungen über die Zeit hinweg und hilft so, den Erfolg implementierter Sicherheitsmaßnahmen zu analysieren. Zusätzlich zeigt ein Balkendiagramm die Anzahl der entdeckten Schwachstellen in bestimmten Zeiträumen, was es den Nutzer/-innen erleichtert, die Wirksamkeit von Maßnahmen einzuschätzen. Ebenfalls hilft es zu beurteilen, ob neue Schwachstellen aufgetreten sind oder ob bestehende Schwachstellen reduziert werden konnten. Ein weiteres Liniendiagramm analysiert zeitliche Aspekte im Hinblick auf die OWASP-Kategorien und gibt darüber Aufschluss, ob die API vermehrt Schwachstellen in bestimmten Kategorien aufweist. Diese zeitbasierten Visualisierungen bieten eine Grundlage, um aktuelle Schwachstellen zu bewerten und so zukünftige Maßnahmen gezielt planen zu können.

Die Risikodarstellung und Priorisierung ist ein weiterer zentraler Aspekt des Dashboards. Die Visualisierungen, die in Anhang E unter risikobasierten Visualisierungen gezeigt werden, umfassen unter anderem ein Riskometer, das den Risikostatus in Form eines Tachometers mit Farbkodierung darstellt und so eine schnelle Einschätzung des Sicherheitsstatus ermöglicht. Der Risk-Score wird auf Basis der gefundenen Schwachstellen und der benutzerdefinierten Gewichtungen der OWASP-Kategorien berechnet, was eine umfassende Risikoeinschätzung erlaubt. Zusätzlich veranschaulicht ein Balkendiagramm die Verteilung der Schwachstellen auf verschiedene Risikokategorien, wodurch die Priorisierung von Ressourcen unterstützt wird. Ein Scatterplot-Diagramm ergänzt die Darstellung, indem es mehrere Dimensionen des Sicherheitsstatus zeigt: Schwachstellen, Risikolevels und OWASP-Kategorien. So können die Benutzer/-innen auf einen Blick die kritischsten Schwachstellen und die am häufigsten betroffenen Kategorien erkennen. Ein Risk Percentage Zeitstrahl visualisiert die zeitliche Entwicklung des Riskscores, um Trends und die Wirksamkeit von Maßnahmen zu analysieren und langfristige Sicherheitstrends zu erkennen.

Das Dashboard visualisiert zudem die Verteilung der OWASP-Kategorien und zeigt so die Verteilung der Schwachstellen in der API. In Anhang E sind unter anderem jene Visualisierungen dargestellt, welche die prozentuelle Verteilung der Schwachstellen über die OWASP-Kategorien hinweg verdeutlichen. Das Kreisdiagramm bietet eine schnelle Übersicht über die am stärksten vertretenen Schwachstellenkategorien, sodass die API-Betreiber/-innen gezielt Ressourcen auf die kritischsten Bereiche konzentrieren können. Ein Blasendiagramm zeigt die Häufigkeit der Schwachstellen pro OWASP-Kategorie, wobei größere Blasen Kategorien eine höherer Anzahl an Schwachstellen repräsentieren und

so eine gezielte Priorisierung ermöglichen. Ein weiteres Liniendiagramm stellt den Trend der Schwachstellenanzahl pro Kategorie über die Zeit hinweg dar und unterscheidet zwischen aktiven und passiven Scans. Diese Visualisierungen bieten eine verständliche und übersichtliche Darstellung der Sicherheitslage der API und unterstützen eine schnelle und gezielte Maßnahmenplanung.

Für technische Experten und weniger technikaffine Benutzer/-innen enthält das Dashboard Visualisierungen, in unterschiedlichen Komplexitätsstufen. Die intuitive Gestaltung der Darstellungen reduziert die Komplexität der gezeigten Informationen und erleichtert eine schnelle Entscheidungsfindung. Beispielsweise visualisiert das Riskometer die Schwachstellen mithilfe eines Ampelsystems (grün, gelb, rot), was eine schnelle Risikoeinschätzung ermöglicht, insbesondere für weniger technikaffine Nutzer. Für tiefere Analysen stehen Scatterplot-Diagramme und Liniendiagramme zur Verfügung, die Schwachstellen, Risikostatus und OWASP-Kategorien über die Zeit aufschlüsseln. Balken- und Kreisdiagramme bieten eine schnelle Übersicht über die Verteilung der Schwachstellen auf die OWASP-Kategorien und unterstützen die Analyse der kritischsten Sicherheitsbereiche. Darüber hinaus enthält das Dashboard (Abbildung 26) Links zu OWASP-Ressourcen, die den Nutzer/-innen detaillierte Informationen zu spezifischen Sicherheitsrisiken und deren Lösung bieten.

Die Möglichkeit, sowohl aktive als auch passive Scans durchzuführen und die Ergebnisse direkt nebeneinander darzustellen, erhöht die Transparenz und ermöglicht den Benutzer/-innen eine präzise Bewertung der Wirksamkeit ihrer Sicherheitsmaßnahmen. In vielen Visualisierungen (Anhang E – zeitstrahlbasierte Visualisierungen), insbesondere in den Liniendiagrammen, sind passive Scans durch einfache Punkte und aktive Scans durch Sterne gekennzeichnet, während im Balkendiagramm ein Muster zur Unterscheidung verwendet wird. Diese Funktion bietet den Nutzer/-innen eine umfassendere Analyse ihrer Serviceschnittstellen.

Dank der umfassenden API-Unterstützung ist das Dashboard nicht auf einen speziellen API-Typ begrenzt und unterstützt eine breite Palette, was eine universelle Visualisierung der Schwachstellen ermöglicht. Die Einbindung von ZAPProxy stellt dies sicher. Wie im Zuge der Werkzeug-Analyse (Kapitel 3) identifiziert bietet die Integration von mehreren Sicherheitswerkzeugen einen Mehrwert, um noch mehr oder auch spezifischere Schwachstellen auffindbar zu machen. Die Infrastruktur ist bereits dahingehend vorbereitet, sodass weitere Werkzeuge in die Datenschicht eingebunden werden können. Der Parser muss in diesem Fall der jeweiligen Berichtstrukturen angepasst werden, um so die die Sicherheitsanalyse für verschiedene Scanning-Werkzeuge möglich zu machen, was die Flexibilität und den Nutzen des Dashboards für eine breite Benutzerbasis erhöht. Auch speziellere Scanning-Werkzeuge wie GraphQL Cop könnten so eingebunden werden, um neben API-Typ-unabhängigen Schwachstellen auch für einen API-Typen spezifische Schwachstellen auffindbar zu machen.

Schließlich ist das Dashboard anpassungsfähig gestaltet. Die Drei-Schichten-Architektur und der komponentenbasierte Ansatz erleichtern das Hinzufügen weiterer Funktionen. Die API-Endpunkte sind dokumentiert und können bei laufender Applikation über <http://localhost:5000/apidocs> abgerufen werden, was die Integration mit anderen Systemen, sowie das Hinzufügen neuer Funktionen unterstützt. Auch die Ergänzung von mehreren Nutzern mit individuellen Logins und Gewichtungen für die OWASP-Kategorien ist somit mit geringem Aufwand möglich. Diese Erweiterbarkeit stellt sicher, dass das Dashboard in bestehende Sicherheitsinfrastrukturen integriert und flexibel an neue Anforderungen angepasst werden kann.

4.4.2. Fallstudien

Neben der rein argumentativen Evaluierung wurde das Dashboard auch mithilfe von einigen Fallstudien getestet. Hier wurden wie auch schon im Zuge der Analyse der Sicherheitswerkzeuge und des Vergleichs aus rechtlichen Gründen Test-APIs gewählt. Die Ergebnisse zweier wichtigen Fallstudien, welche ebenfalls während der Implementierung als primäre Test-APIs zur Validierung verwendet wurden, werden in Folge nun genauer erläutert (Tabelle 7). Alle getesteten APIs sind im Anhang C am Ende der jeweiligen Werkzeug-Analysen zu finden.

Tabelle 7: Fallstudien

Beispiel-API	Scan-Typ	Risiko der gefundenen Schwachstellen	Auflistung der gefundenen Schwachstellen (Hoch – Moderat – Niedrig – Information)
VAmPI	Passiv	<ul style="list-style-type: none"> • Niedrig: 2 • Information: 1 • <i>Summe:</i> 3 	<ul style="list-style-type: none"> • Server Leaks Version Information via "Server" HTTP Response Header Field (API8) • X-Content-Type-Options Header Missing (API8) • Storable and Cacheable Content (API3)
VAmPI	aktiv	<ul style="list-style-type: none"> • Niedrig: 4 • Information: 4 • <i>Summe:</i> 8 	<ul style="list-style-type: none"> • A Server Error response code was returned by the server (API8/API10) • Server Leaks Version Information via "Server" HTTP Response Header Field (API8) • Unexpected Content-Type was returned (API8) • X-Content-Type-Options Header Missing (API8) • Storable and Cacheable Content (API3) • Non-Storable Content (API3) • Authentication Request Identified (API2) • A Client Error Response Case was returned by surver (API8/API2)
crAPI	passiv	<ul style="list-style-type: none"> • Moderat: 2 • Niedrig: 5 • Information: 2 • <i>Summe:</i> 9 	<ul style="list-style-type: none"> • Content Security Policy (CSP) Header Not Set (API8) • Missing Anti-clickjacking Header (API8) • Dangerous JS Functions (API1) • Permissions Policy Header Not Set (API8) • Server Leaks Version Information via "Server" HTTP Response Header Field (API8) • Timestamp Disclosure – Unix (API8)

			<ul style="list-style-type: none"> • X-Content-Type-Options Header Missing (API8) • Information Disclosure - Suspicious Comments (API8) • Modern Web Application (No threat) • Non-Storable Content (API3)
crAPI	aktiv	<ul style="list-style-type: none"> • Hoch: 1 • Moderat: 5 • Niedrig: 5 • Information: 4 • <i>Summe: 15</i> 	<ul style="list-style-type: none"> • SQL Injection (API1/API3) • .env Information Leak (API8) • Bypassing 403 (API1/API5) • CORS Misconfiguration (API8) • Content Security Policy (CSP) Header Not Set (API8) • Hidden File Found (API9) • A Server Error response code was returned by the server (AP10/API8) • Application Error Disclosure (API10/API8) • Permission Policy Header not Set (API8) • Server Leaks Version Information via "Server" HTTP Response Header Field (API8) • Unexpected Content-Type was returned (API8) • A Client Error response code was returned by the server (API2/API8) • Authentication Request Identified (API2) • Non-Storable Content (API3) • Session Management Response Identified (API2)

Die Fallstudien konnten hierbei mit einem positiven Ergebnis durchlaufen und der Sicherheitsstatus der gescannten APIs im Dashboard visualisiert werden. Ein konkretes Beispiel eines Dashboards, welches in Zuge einer Fallstudie mit der Test-API VAmPI erstellt werden konnte, ist im Anhang F zu finden. Teilweise wurden zu Testzwecken die Ergebnisse der getesteten APIs manipuliert um speziell Visualisierungen, die den Verlauf der Schwachstellen über einen bestimmten Zeitraum aufzeigen, besser abbildbar zu machen.

4.5. Ergebnis des API Security Dashboards

Die Entwicklung des Artefakts API Security Analysis Dashboard hat gezeigt, dass ein Dashboard basierendes auf den OWASP-Top 10 API Sicherheitsrisiken mithilfe des Design-Sciene-Ansatzes implementiert werden kann um ein universelles und gut verständliches Werkzeug vorzustellen, was sich von

den oft rein-textbasierten API-Scannern abhebt. Dies soll es so Benutzer/-innen erleichtern Sicherheitslücken in der API auffindbar und verständlich zu machen um frühzeitig reagieren zu können. Die wesentlichen Ergebnisse dieser Implementierung sind wie folgt:

- Fokus auf Visualisierungen statt textuellen Darstellungen:
 - OWASP Top 10 Mapping: Das Dashboard ermöglicht es, alle identifizierten Schwachstellen den relevanten OWASP Top 10 API-Sicherheitsrisiken zuzuordnen und diese visuell darzustellen. Diese Visualisierung hat die Verständlichkeit der Sicherheitsrisiken verbessert, da Benutzer/-innen nun auf einen Blick erkennen können, welche Schwachstellen besonders kritisch sind, sowie welcher OWASP Kategorie sie angehören.
 - Zeitbasierte Darstellungen: Die Integration einer zeitlichen Visualisierung von Schwachstellen kann es den Nutzer/-innen ermöglichen, Trends und Entwicklungen über die Zeit hinweg zu beobachten. Dies hilft Benutzer/-innen den Sicherheitsstatus der API besser zu verstehen, sowie Schwachstellen welche häufig vorkommen besondere Aufmerksamkeit zu schenken.
- Erhöhte Benutzerfreundlichkeit und Customizing:
 - Risikobasierte Priorisierung: Die Möglichkeit, Risiken im Kontext der gesamten API zu priorisieren, inklusive individueller Gewichtungen, hilft Nutzer/-innen, die wichtigsten Schwachstellen gezielt zu bearbeiten.
 - Komplexitätsgerechte Visualisierungen: Durch die Bereitstellung von Visualisierungen, die sowohl für technische Experten als auch für weniger technikaffine Benutzer/-innen verständlich sind, konnte die Kommunikation von Sicherheitsrisiken verbessert werden. Somit kann der Sicherheitsstatus der API Professionsübergreifend verstanden und die Priorität einer sicheren API kommuniziert werden.
- Erweiterte Funktionalitäten:
 - Unterstützung von aktiven und passiven Scans: Das Dashboard bietet die Möglichkeit, sowohl aktive als auch passive Scans durchzuführen und die Ergebnisse visuell neben-einander darzustellen. Dies bieten den Nutzer/-innen ein umfassenderes Bild der Sicherheitslage und kann die Bewertung der Effektivität der Sicherheitsmaßnahmen erleichtert.
 - Umfassende API-Unterstützung: Die Integration einer breiten Palette von API-Typen und Protokollen macht das Dashboard flexibel und universell einsetzbar. Dadurch können Sicherheitsanalysen effizient über verschiedene API-Typen hinweg durchgeführt werden.
- Integration und Erweiterbarkeit:
 - Erweiterbarkeit: Aufgrund der gewählten Architektur und Frameworks, sowie einer umfassenden Schnittstellendokumentation ist das Dashboard leicht erweiterbar und kann in bestehende Sicherheitssysteme integriert werden. Dies stellt sicher, dass das Dashboard zukunftssicher ist und an die spezifischen Bedürfnisse der Benutzer/-innen angepasst werden kann.

Das prototypische API Security Analysis Dashboard konnte die festgelegten Anforderungen und Zielsetzungen gut erfüllen und bietet somit einen auf den OWASP Top 10 API Sicherheitsrisiken ba-

sierenden Ansatz für die Sicherheitsanalyse von APIs mit Fokus auf Visualisierungen und Benutzerfreundlichkeit. Das Dashboard ermöglicht nicht nur eine Sicherheitsanalyse, sondern verbessert auch die Kommunikation und das Verständnis von Sicherheitsrisiken kann. Dennoch gibt es Bereiche, in denen das Dashboard weiter optimiert werden kann, insbesondere in Bezug auf die Automatisierung des OWASP-Kategorie-Mappings, der Integration von weiteren Sicherheitswerkzeugen oder die Erweiterung um weiteren Visualisierungen um das Dashboard noch umfassender zu gestalten. Diese Aspekte bieten Potenzial für zukünftige Verbesserungen, um das Dashboard noch leistungsfähiger und benutzerfreundlicher zu gestalten. Die Evaluierung wurde aktuell nur auf Ebene der Machbarkeit mittels Fallstudien durchgeführt. Eine ausführlichere Analyse mittels Nutzstudien wäre hier sinnvoll, konnte aber aus ressourcengründen nichtmehr durchgeführt werden. Im Zuge des Prototyps wurde die wesentliche Idee für die Gestaltung eines Dashboards umgesetzt und argumentiert, dass es einen Mehrwert gegenüber den verfügbaren Werkzeugen für Vulnerability-Scans liefern. Die empirische Validierung das auch Nutzer/-innen dies so empfinden fehlt aktuell noch.

5. Zusammenfassung und Diskussion

Die vorliegende Masterarbeit beschäftigte sich mit der Sicherheit von Serviceschnittstellen. Das Fundament stellten hierbei die 2023 aktualisierte OWASP Top 10 API-Sicherheitsrisiken Rangliste dar. Neben umfassenden Erläuterungen zu Grundlagen im Bereich Cybersecurity, APIs und API-Security, sowie der tiefergehenden Erklärung der OWASP Rangliste wurden die drei Forschungsfragen, welche am Beginn der Masterarbeit definiert wurden, beantwortet.

Dafür wurde im ersten Teil der vorliegenden Arbeit der Fokus auf allgemeine Sicherheitswerkzeuge für Webapplikationen gelegt. Mithilfe einer Recherche der existierenden Sicherheitswerkzeuge wurden diese erfasst, kategorisiert und deren Verteilung grafisch aufbereitet. Die zweite Forschungsfrage setzte an dessen Kategorisierung an und definierte den Fokus auf Open-Source API-Scanner welche Schwachstellen einer Serviceschnittstelle extrahieren können. Nachdem diese nach verschiedenen Gesichtspunkten gefiltert wurden, durchliefen die finalen drei Werkzeuge eine umfassende Analyse und einen abschließenden Vergleich. Ziel dieser Analyse war es die Stärken und Schwächen der einzelnen Werkzeuge zu erheben. Nicht zuletzt dienten die Analyse der Sicherheitswerkzeuge und der Vergleich auch als Ausgangspunkt für die Anforderungsdefinition von Forschungsfrage FF3. Dessen Ziel war es ein API Security Analysis Dashboard zu entwerfen und prototypisch umzusetzen. Basierend auf die OWASP Top 10 API-Sicherheitsrisiken und einem primären Fokus auf Visualisierungen, aber auch Benutzerfreundlichkeit und Integration, wurde das Artefakt umgesetzt. Dabei wurde der Design Science Ansatz verfolgt, wobei die Validierung des Dashboards hier rein auf argumentativer Ebene stattfand und die Machbarkeit in Fallstudien überprüft wurde.

5.1. Beantwortung der Forschungsfragen

- FF1: Welche Werkzeuge zur Analyse der Sicherheit von Webservices existieren bereits und wie können diese kategorisiert werden?

Die Beantwortung der ersten Forschungsfrage erfolgte durch eine umfassende Recherche existierender Sicherheitswerkzeuge, die sich auf Webservices und Webanwendungen fokussieren. Dabei wurden die identifizierten Werkzeuge systematisch erfasst und in verschiedene Kategorien eingeteilt, beispielsweise nach ihrem Einsatzzweck und ihrer Verfügbarkeit. Die Kategorisierung half einen Überblick über das breite Spektrum an Sicherheitswerkzeugen zu gewinnen, und diente als Grundlage für die nachfolgende Untersuchung. Die grafische Aufbereitung der Verteilung der Werkzeuge ermöglichte zudem eine anschauliche Darstellung der verschiedenen Kategorien und deren Anwendungsbereiche. Diese Erkenntnisse bilden die Basis für die Beantwortung der zweiten Forschungsfrage, indem sie die spezifische Auswahl von API-Scannern ermöglichten.

- FF2: Welche Open-Source-Tools zur Analyse von Service-Schnittstellen (APIs) können aus den in FF1 kategorisierten Werkzeugen extrahiert werden und wie können diese zur Aufdeckung der OWASP Top 10 API-Sicherheitsrisiken eingesetzt werden?

Aufbauend auf den in FF1 erarbeiteten Kategorien wurden in FF2 spezifisch jene Werkzeuge identifiziert, die API-Schnittstellen auf sicherheitsrelevante Schwachstellen hin analysieren können. Durch

einen mehrstufigen Auswahlprozess, der sowohl die Open-Source-Verfügbarkeit als auch die Fähigkeit zur Identifizierung von Schwachstellen gemäß den OWASP Top 10 API-Sicherheitsrisiken berücksichtigt, konnten drei geeignete Werkzeuge ausgewählt werden. Diese wurden anschließend in einem Vergleich hinsichtlich ihrer Usability, Integrationsfähigkeit, Dokumentation und Funktionalität analysiert. Der Fokus lag dabei auf der praktischen Anwendung der Werkzeuge zur Identifizierung von Schwachstellen in API-Schnittstellen. Am Ende dieser Analyse wurde außerdem ein Tool ausgewählt, welches basierend auf seinen analysierten Stärken in das API Security Analysis Dashboard (FF3) integriert werden sollte, da aus ressourcengründen nicht alle Werkzeuge eingebunden werden konnten.

- FF3: Wie können die aus FF2 aufgedeckten Schwachstellen in Bezug auf Service-Schnittstellen (APIs) in einem Security Analysis Dashboard abgebildet und visualisiert werden?

Zur Beantwortung der dritten Forschungsfrage wurden die Anforderungen für das API Security Analysis Dashboard aus den Ergebnissen von FF2 abgeleitet. Anschließend wurde ein Prototyp entwickelt, der es ermöglicht, die durch die API-Scanner identifizierten Schwachstellen visuell darzustellen. Das Dashboard konzentriert sich auf die übersichtliche Darstellung der Schwachstellen in Übereinstimmung mit den OWASP Top 10 API-Sicherheitsrisiken und zielt darauf ab, den Nutzer/-innen eine intuitive und benutzerfreundliche Schnittstelle zu bieten. Verschiedene Visualisierungstechniken wurden explorativ untersucht, um die Sicherheitslage einer API möglichst klar und verständlich darzustellen. Die Implementierung des Dashboards folgte dem Design Science Ansatz und wurde abschließend in Form von Case-Studies getestet, bei denen Beispiel-APIs und die in FF2 ausgewählten Werkzeuge zum Einsatz kamen. Die Evaluierung des Dashboards erfolgte durch den Vergleich der definierten Anforderungen mit den realisierten Funktionen, wobei der Fokus auf der Benutzerfreundlichkeit, der Integration und der visuellen Aufbereitung der Sicherheitsinformationen lag.

5.2. Einschränkungen

Nach der Beantwortung der Forschungsfragen ergeben sich jedoch einige Einschränkungen in Bezug auf die Validität der Ergebnisse. Diese Einschränkungen werden im Folgenden von der ersten bis zur letzten Forschungsfrage im Detail erläutert, um mögliche Limitationen der Studie aufzuzeigen und potenzielle Verbesserungsvorschläge für zukünftige Arbeiten zu formulieren.

Bei Forschungsfrage 1 können als Limitation die eingeschränkte Verwendung von Quellen zur Sammlung von Sicherheitswerkzeuge für Webapplikationen angegeben werden. Hier hätte eine tiefergehende Recherche stattfinden können, damit noch mehr Werkzeuge einer Kategorisierung unterzogen werden können. Aufgrund des Fokus auf Sicherheitswerkzeuge im Bereich Serviceschnittstellen und einer begrenzten Recherchezeit wurde sich auf die in Kapitel 2.2 genannten Primärquellen (Kritikos et al., 2019; Tesauro, n.d., OWASP Foundation, n.d.; Wichers, n.d.) beschränkt.

Da die Analyse und Vergleich der Sicherheitswerkzeuge, welche im Zuge von FF2 durchgeführt wurden, direkt von FF1 abhängig waren, ist hier ebenfalls die identische Limitation anzuwenden. Hinzuzufügen ist hier, dass reine API-Scanner sehr schwer zu finden sind, speziell im Open-Source Bereich und deswegen vermehrt auch Security-Scanner für Webanwendungen, welche ebenfalls für APIs angewendet werden können, verwendet wurden. Manche Sicherheitswerkzeuge wurden aufgrund ihrer technischen Voraussetzungen ausgeschlossen, oder weil aus der Dokumentation nicht erkennbar war,

welche Art von Schwachstellen gefunden werden konnten und so vorab keine Abbildung auf die OWASP Top 10 API-Sicherheitsrisiken stattfinden konnte. Aus Zeitgründen konnten diese Werkzeuge nicht in die Analyse aufgenommen werden. Eine Fallstudie von dem Entwickler der Test-API vAmPI, welche auch im Zuge der Analyse der Sicherheitswerkzeuge für Tests verwendet wurde, zeigt außerdem, dass viele API-Scanner nicht die erwartete Leistung erzielen. Eine Kombination von verschiedenen Scanning-Werkzeugen, Einsatz von Plug-Ins oder Anpassungen in den Konfigurationen könnte hier hilfreich sein, um die Ergebnisse zu verbessern (erev0s, 2022).

Im Zuge der portotypen Implementierung des Dashboards ließen sich ebenfalls einige Limitationen finden. Aufgrund des prototypischen Charakters und der derzeit noch unzureichenden Unterstützung von LLMs (Large Language Models) werden im aktuellen Dashboard die gefundenen Schwachstellen noch nicht automatisch auf die OWASP-Kategorien abgebildet, sondern diese muss manuell konfiguriert werden (durch einen Eintrag in eine Excel Liste). Scans sind aktuell aufgrund der Art der Visualisierungen und derzeitigen Implementierung, sowie des Datenbankschemas auf einen Scan pro Tag und URL beschränkt. Wird versucht die identische URL an einem Tag nochmals zu scannen wird ein Fehler im Backend ausgelöst und der Scan nicht gestartet. Auch ist das Dashboard aktuell auf einen Standard-Benutzer limitiert. Hier müssten bei Bedarf API-Endpunkte sowie das Userinterface und Backend angepasst werden um das Dashboard mit verschiedenen Benutzer/-innen ausführbar zu machen. Wie bereits erwähnt, ist die Evaluierung des Dashboards nicht ideal gewählt und eine Befragung von Expert/-innen oder potenzielle Nutzer/-innen wäre hier weitaus aussagekräftiger gewesen. Aus Ressourcengründen musste auf diese Art der Evaluierung verzichtet werden und ein argumentativer und Fallstudien-gestützter Ansatz gewählt werden.

Das aktuell integrierte Scanning-Werkzeug ZAPProxy weist ebenfalls Limitationen auf, welche erwähnt werden sollten. Da das Werkzeug mit Standardkonfigurationen ausgeführt wird, kann es hier, wie bereits im vorherigen Abschnitt erwähnt, vorkommen, dass Schwachstellen nicht auffindbar sind (erev0s, 2022). Außerdem dauern Scans bei ZAPProxy mitunter sehr lange, gerade bei aktiven Scans und größeren APIs die viele URLs beinhalten, was die User Experience und Effizienz negativ beeinflussen könnte. Dies wurde aber im Falle des Prototyps hingenommen, da der Fokus auf die Visualisierungen selbst lag. An dieser Stelle wird erneut darauf hingewiesen, dass eine Integration von mehreren Werkzeugen zur Schwachstellenerkennung einen positiven Mehrwert bringen würde. Dies wurde im Zuge der Implementierung des Dashboards bereits teilweise vorbereitet, sowie auf die korrekte Dokumentation von Schnittstellen geachtet, um eine programmatische Erweiterung zu vereinfachen (siehe Punkt Erweiterbarkeit in Kapitel 4.4).

Die Test-APIs welche zur Toolanalyse, Tests und Verifikation verwendet wurden, können ebenfalls, als ein limitierender Faktor betrachtet werden. Aufgrund ihres intentional fehlerhaften Charakters können dessen lokale Ausführung mitunter temporär nicht korrekt funktionieren oder das Aufsetzen dieser zu Problemen führen. Aufgrund von rechtlichen Aspekten dürfen nur eigene oder für Testzwecken entwickelte APIs gescannt werden, weshalb hier keine alternativen Service-Schnittstellen verwendet werden konnten.

5.2. Implikationen und Ausblick

Zur Weiterarbeit am API Security Analysis Dashboard in einem wissenschaftlichen Umfeld können einige Implikationen erwähnt werden, welche sich mitunter von den Limitationen im vorherigen Kapitel ableiten lassen. Ein Einsatz von Künstliche Intelligenz, besonders von Large-Language-Models (Xu et al., 2024) könnte speziell beim bei der Abbildung der von den Analysewerkzeugen gelieferten Schwachstellen auf die OWASP Kategorien hilfreich sein. Abseits von Large-Language-Models könnte auch der Einsatz von Machine Learning zur Priorisierung und Risikobewertung von Schwachstellen erforscht werden. Ein ML-Modell, das auf historischen Sicherheitsdaten trainiert wurde, könnte dabei helfen, Schwachstellen besser zu priorisieren und spezifische Handlungsempfehlungen abzuleiten.

Auch der Einsatz von weiteren API-Scanning Werkzeuge, Plug-Ins oder spezielle Anpassungen in den Konfigurationen können dabei helfen, mehr Schwachstellen auffindbar zu machen und so einen Mehrwert bieten. Im Bereich der Risikoberechnung einer API könnten neben der individuellen Gewichtung der Kategorien, der Anzahl an gefundenen Schwachstellen, sowie der Prioritäten auch andere Faktoren der OWASP-Kategorien miteinbezogen wird; beispielsweise die in Tabelle 1 beschriebenen Ausprägungen Ausnutzbarkeit, Verbreitung, Erkennbarkeit sowie die Auswirkungen bei Nutzung der Schwachstelle für Cyberangriffe.

In zukünftigen Arbeiten könnte außerdem ein Modul entwickelt werden, das basierend auf den gefundenen Schwachstellen automatisierte Handlungsempfehlungen gibt, wie zum Beispiel spezifische Patches oder Konfigurationsänderungen. Dies könnte den Prozess der Schwachstellenbehebung deutlich effizienter gestalten und mithilfe von Künstlicher Intelligenz entwickelt werden. Auch die Einbindung in einen CD/CI Prozess könnte eine Möglichkeit sein, automatisierte Scans durchzuführen und dessen Ergebnisse im Dashboard abbildbar zu machen.

Neben der bereits erwähnten Maßnahme von erweiterten Evaluierungen durch Expert/-innen und potentiellen Benutzer/-innen wäre auch eine Langzeitstudie denkbar, um die Wirksamkeit des Dashboards und der vorgeschlagenen Verbesserungen zu evaluieren. Dabei könnten Unternehmen über einen längeren Zeitraum begleitet werden, um zu messen, wie sich die Sicherheit ihrer APIs durch den Einsatz des Dashboards verändert.

Abschließend soll die vorliegende Masterarbeit ein kurzer Ausblick in aktuelle Sicherheitstrends im Bereich API-Security geben, welche ebenfalls potentielle Themenfelder für die Erweiterung der API Security Analysis Dashboard sein könnten. Das Thema API-Security bleibt nach wie vor ein sehr präsentes mit großer Wichtigkeit (Nicosia, 2024; Roy, 2024). Trends wie der Einsatzes von Künstlicher Intelligenz im Zuge der Implementierung (Roy, 2024) aber auch Sicherung von APIs (Nicosia, 2024), sowie der Einsatz von Blockchains um APIs robuster zu machen (Kong, 2024) sind auf dem Vormarsch. Aber auch explizite Risiken rücken in den Vordergrund; aufgrund des starken Anstiegs an APIs stehen unter anderem veraltete und nicht mehr verwendete Shadow-APIs in den Fokus, welche ein erhebliches Sicherheitsrisiko mit sich bringen (Allison, 2024). Auch ein Anstieg an automatisierte Angriffe mitunter mit Fokus auf die Unternehmenslogik seitens der Cyberkriminellen wird in Zukunft zu verzeichnen sein (Allison, 2024). Nicht zuletzt ist auch zu erwarten, dass sich das Ausnutzen von Sicherheitslücken von APIs nicht nur auf Computer, Smartphones und Servers beschränken, sondern auch auf Internet of Things Instanzen vorkommen und es hier vermehrt zu cyberkriminellen Angriffen

kommen kann (Kong, 2024). Gerade bei kritischen Systemen wie selbstfahrende Fahrzeuge oder medizinischen Geräten wäre eine solche Entwicklung verehrend und unbedingt abzuwenden.

6. Literaturverzeichnis

- Abdou, A., Barrera, D., & van Oorschot, P. C. (2015). What lies beneath? Analyzing automated SSH bruteforce attacks. In *International conference on PASSWORDS* (pp. 72–91). Cham, Schweiz: Springer International Publishing.
- Akto. (2023). *What's changed in OWASP API Security Top 10 2023 Release Candidate from 2019?* <https://www.akto.io/blog/whats-changed-in-owasp-top-10-2023>
- Allison, P. R. (2024). *The top API risks of 2024 and how to mitigate them.* <https://www.it-pro.com/security/the-top-api-risks-and-how-to-mitigate-them>
- Alshamrani, A., Myneni, S., Chowdhary, A., & Huang, D. (2019). A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, 21(2), 1851–1877. IEEE.
- Ani, U. D., He, H., & Tiwari, A. (2019). Human factor security: Evaluating the cybersecurity capacity of the industrial workforce. *Journal of Systems and Information Technology*, 21(1), 2–35. Emerald Publishing Limited.
- API Lifecycle Management. (2021). *Best Practices For Your API Versioning Strategy.* <https://www.akana.com/blog/api-versioning>
- AWS. (2023). *Was ist ein API-Schlüssel?* <https://aws.amazon.com/de/what-is/api-key/>
- Bach-Nutman, M. (2020). Understanding the top 10 owasp vulnerabilities. *arXiv Preprint arXiv:2012.09960.*
- Badhwar, R. (2021). Intro to API Security-Issues and Some Solutions! In *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms* (pp. 239–244). Cham, Schweiz: Springer International Publishing.
- Baitha, A. K., & Vinod, S. (2018). Session hijacking and prevention technique. *International Journal of Engineering & Technology*, 7(2.6), 193–198. Science Publishing Corporation.
- Bawany, N. Z., Shamsi, J. A., & Salah, K. (2017). DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42, 425–441. Springer.

- Belgacem, H. (2022). *API Security Best Practices: Part 4/6—Input validation*. <https://medium.com/@hassene/how-to-secure-your-api-part-4-6-input-validation-best-practices-db2c28d7a991>
- Bhuiyan, T., Begum, A., Rahman, S., & Hadid, I. (2018). API vulnerabilities: Current status and dependencies. *International Journal of Engineering & Technology*, 7(2.3), 9–13. Science Publishing Corporation.
- Bigelow, S. (2022). *Die unterschiedlichen API-Typen und ihre Merkmale*. <https://www.computer-weekly.com/de/tipp/Die-unterschiedlichen-API-Typen-und-ihre-Merkmale>
- Bostock, M., & Observable. (2024). *D3.js* [Computer software]. <https://d3js.org/>
- Cartwright, A., Cartwright, E., Xue, L., & Hernandez-Castro, J. (2023). An investigation of individual willingness to pay ransomware. *Journal of Financial Crime*, 30(3), 728–741. Emerald Publishing Limited.
- Cartwright, E., Hernandez Castro, J., & Cartwright, A. (2019). To pay or not: Game theoretic models of ransomware. *Journal of Cybersecurity*, 5(1), tyz009. Oxford University Press.
- Chae, C.-J., Kim, K.-B., & Cho, H.-J. (2019). A study on secure user authentication and authorization in OAuth protocol. *Cluster Computing*, 22, 1991–1999. Springer.
- Cheat Sheets Series Team. (2024). *HTTP Security Response Headers Cheat Sheet*. https://cheatsheets.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
- Cloudflare. (2023). *Was ist das Prinzip des minimalen Zugangs („Least Privilege“)?* <https://www.cloudflare.com/de-de/learning/access-management/principle-of-least-privilege/>
- Committee on National Security. (2010). *National Information Assurance Glossary* (Technical 4009). Fort Meade, USA: National Security Agency.
- Conti, M., Dragoni, N., & Lesyk, V. (2016). A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 18(3), 2027–2051. IEEE.
- Dahiya, A., & Gupta, B. (2021). How iot is making ddos attacks more dangerous. *Cyber Security Insights Magazine*. Insights2Techinfo.
- Dave, K. T. (2013). Brute-force attack ‘seeking but distressing.’ *Int. J. Innov. Eng. Technol. Brute-Force*, 2(3), 75–78. Innovative Research Publications.

- Doglio, F. (2015). *Pro REST API Development with Node.js*. New York: Apress.
- Elissen, M., & Namer, N. (2023). *OWASP Top 10 API-Sicherheitsrisiken: Die Ausgabe 2023 ist endlich da*. <https://www.akamai.com/de/blog/security/owasp-top-10-api-security-risks-2023-edition>
- erev0s. (2020). *vAmPI* [Computer software]. <https://github.com/erev0s/VAmPI>
- erev0s. (2022). *Automated security scanning against VAmPI*. <https://erev0s.com/blog/vampi-against-automated-api-scanning/>
- Exposed Atoms. (2020). *Damn Vulnerable GraphQL Application* [Computer software]. <https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application>
- Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., & Pretschner, A. (2016). Security testing: A survey. In *Advances in Computers* (Vol. 101, pp. 1–51). Elsevier.
- Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.
- Fowler, M. (2012). *Patterns of enterprise application architecture*. Upper Saddle River, USA: Pearson Education.
- Frank, R., Strugholtz, S., & Meise, F. (2021). *Bausteine der digitalen Transformation*. Heidelberg: Springer Books.
- Froehlich, A. (2022). *Compare zero trust vs. The principle of least privilege*. <https://www.techtarget.com/searchsecurity/answer/Compare-zero-trust-vs-the-principle-of-least-privilege>
- GraphQL-Cop. (2022). *GraphQL-Cop*. <https://github.com/dolevf/graphql-cop>
- gRPC Authors. (2024). *Documentation*. <https://grpc.io/docs/>
- Gutermuth, M. (2024). *What is API encryption?* <https://blog.postman.com/what-is-api-encryption/>
- Haas, H.-W. (2021, December 5). Changes in OWASP Top 10: 2017 vs 2021. *Changes in OWASP Top 10: 2017 vs 2021*. <https://medium.com/digitalfrontiers/changes-in-owasp-top-10-2017-vs-2021-7cea4183288b>
- Hardt, D. (2012). *The oauth 2.0 authorization framework: Bearer token usage (No. rfc6750)*. Internet Engineering Task Force (IETF).
- Hartz, S. (2020). *Rest API Goat* [Computer software]. <https://github.com/optiv/rest-api-goat>

- Helton, A. (2023). *API keys vs tokens—What's the difference?* <https://www.gomomento.com/blog/api-keys-vs-tokens-whats-the-difference>
- IBM. (2023). *Was ist Cybersicherheit?* <https://www.ibm.com/de-de/topics/cybersecurity>
- Indusface. (2023). *What's New in OWASP API Top 10 2023: The Latest Changes and Enhancements.* <https://www.indusface.com/blog/whats-new-in-owasp-api-top-10-2023/>
- Jacobson, D., Brail, G., & Woods, D. (2012). *APIs: A strategy guide*. Sebastopol, USA: O'Reilly.
- Jánoky, L. V., Levendovszky, J., & Ekler, P. (2018). An analysis on the revoking mechanisms for JSON Web Tokens. *International Journal of Distributed Sensor Networks*, 14(9), <https://doi.org/10.1177/1550147718801535>. SAGE Publications.
- Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs: Building APIs That Developers Love*. Sebastopol, USA: O'Reilly.
- Johannesson, P., & Perjons, E. (2014). *An introduction to design science* (Vol. 10). Heidelberg: Springer.
- Jones, M., Bradley, J., & Sakimura, N. (2015). *Json web token (jwt)*. Internet Engineering Task Force (IETF).
- Katalon. (2024). *Introduction to API testing in Katalon Studio*. <https://docs.katalon.com/docs/katalon-studio/get-started/supported-application-under-tests/introduction-to-api-testing-in-katalon-studio#where-is-api-testing-performed>
- Kinsta. (2023). *API Rate Limiting: The Ultimate Guide*. <https://kinsta.com/knowledgebase/api-rate-limit/>
- Koffka, K. (2013). *Principles of Gestalt psychology* (Original erschienen 1935). New York: Routledge.
- Kong. (2024). *The Critical Role of API Security in the Internet of Things (IoT)*. <https://konghq.com/blog/enterprise/iot-api-security-guide>
- Kritikos, K., Magoutis, K., Papoutsakis, M., & Ioannidis, S. (2019). A survey on vulnerability assessment tools and databases for cloud-based web applications. *Array*, 3. <https://doi.org/10.1016/j.array.2019.100011>. Elsevier.
- Kshetri, N., & Voas, J. (2017). Do crypto-currencies fuel ransomware? *IT Professional*, 19(5), 11–15.
- Kumar, C. (2024). *Wie implementiert man Sicherheits-HTTP-Header, um Schwachstellen zu verhindern?* <https://geekflare.com/de/http-header-implementation/>

- Leyden, J. (2022). *One in every 13 incidents blamed on API insecurity – report*. <https://portswigger.net/daily-swig/one-in-every-13-incidents-blamed-on-api-insecurity-report>
- Luber, S. (2017). *Was ist eine API?* <https://www.dev-insider.de/was-ist-eine-api-a-583923/>
- Luber, S., & Schmitz, P. (2021). *Was ist Security by Design?* <https://www.security-insider.de/was-ist-security-by-design-a-1071181/>
- Medjaoui, M., Wilde, E., Mitra, R., & Amundsen, M. (2021). *Continuous API management* (2. Edition). Sebastopol, USA: O'Reilly.
- Merriam-Webster. (2023). *Cybersecurity*. <https://www.merriam-webster.com/dictionary/cybersecurity>
- Meta Platforms. (2023). *React* (Version 18) [Computer software]. <https://react.dev/>
- Microsoft. (2010). *Chapter 5: Layered Application Guidelines*. [https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658109\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v=pandp.10))
- Microsoft. (2023). *Was ist Identity & Access Management (IAM)?* <https://www.microsoft.com/de-de/security/business/security-101/what-is-identity-access-management-iam>
- Mujeye, S. (2022). Ransomware: To Pay or Not to Pay? The results of what IT professionals recommend. *2022 The 5th International Conference on Software Engineering and Information Management (ICSIM)*, 76–81. IEEE.
- Neumann, A., Laranjeiro, N., & Bernardino, J. (2018). An analysis of public REST web service APIs. *IEEE Transactions on Services Computing*, 14(4), 957–970. IEEE.
- Nicolas, S. (2022). *Wapiti—The web-application vulnerability scanner*. <https://wapiti-scanner.github.io/>
- Nicosia, M. (2024). *API Predictions for 2024*. <https://salt.security/api-security-trends>
- NIST. (2022). *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. national institute of standards of technology. <https://csrc.nist.gov/pubs/sp/800/218/final>
- OpenAPI. (2022). *What is OpenAPI?* <https://www.openapis.org/>
- OWASP API Security Project team. (2023). *OWASP API Security Top 10 2023 has been released*. <https://owasp.org/blog/2023/07/03/owasp-api-top10-2023>
- OWASP API Security Project Team. (2023). *OWASP Top 10 API Security Risks – 2023*. <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>

- OWASP Foundation. (n.d.-a). *OWASP Risk Rating Methodology*. https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
- OWASP Foundation. (n.d.-b). *Vulnerability Scanning Tools*. https://owasp.org/www-community/Vulnerability_Scanning_Tools
- OWASP Foundation. (2021). *crAPI* [Computer software]. <https://github.com/OWASP/crAPI>
- OWASP Foundation. (2023a). *About OWASP*. About OWASP. <https://owasp.org/Top10/A00-about-owasp/>
- OWASP Foundation. (2023b). *OWASP Top Ten*. OWASP Top Ten. https://owasp.org/www-project-developer-guide/draft/foundations/owasp_top_ten/
- OWASP Top 10 API Security Project Team. (2023a). *API1:2023 Broken Object Level Authorization*. <https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/>
- OWASP Top 10 API Security Project Team. (2023b). *API2:2023 Broken Authentication*. <https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/>
- OWASP Top 10 API Security Project Team. (2023c). *API3:2023 Broken Object Property Level Authorization*. <https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/>
- OWASP Top 10 API Security Project Team. (2023d). *API4:2023 Unrestricted Resource Consumption*. <https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/>
- OWASP Top 10 API Security Project Team. (2023e). *API5:2023 Broken Function Level Authorization*. <https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization/>
- OWASP Top 10 API Security Project Team. (2023f). *API6:2023 Unrestricted Access to Sensitive Business Flows*. <https://owasp.org/API-Security/editions/2023/en/0xa6-unrestricted-access-to-sensitive-business-flows/>
- OWASP Top 10 API Security Project Team. (2023g). *API7:2023 Server Side Request Forgery*. <https://owasp.org/API-Security/editions/2023/en/0xa7-server-side-request-forgery/>
- OWASP Top 10 API Security Project Team. (2023h). *API8:2023 Security Misconfiguration*. <https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration/>

OWASP Top 10 API Security Project Team. (2023i). *API9:2023 Improper Inventory Management*.

<https://owasp.org/API-Security/editions/2023/en/0xa9-improper-inventory-management/>

OWASP Top 10 API Security Project Team. (2023j). *API10:2023 Unsafe Consumption of APIs*.

<https://owasp.org/API-Security/editions/2023/en/0xaa-unsafe-consumption-of-apis/>

OWASP Top 10 Team. (2021a). A01: 2021 – *Broken Access Control*. https://owasp.org/Top10/A01_2021-Broken_Access_Control/

OWASP Top 10 Team. (2021b). A02: 2021 – *Cryptographic Failures*. https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

OWASP Top 10 Team. (2021c). A03: 2021 – *Injection*. https://owasp.org/Top10/A03_2021-Injection/

OWASP Top 10 Team. (2021d). A04: 2021 – *Insecure Design*. https://owasp.org/Top10/A04_2021-Insecure_Design/

OWASP Top 10 Team. (2021e). A05: 2021 – *Security Misconfiguration*. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

OWASP Top 10 Team. (2021f). A06: 2021 – *Vulnerable and Outdated Components*. https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

OWASP Top 10 Team. (2021g). A07: 2021 – *Identification and Authentication Failures*. https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

OWASP Top 10 Team. (2021h). A08: 2021 – *Software and Data Integrity Failures*. https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

OWASP Top 10 Team. (2021i). A09: 2021 – *Security Logging and Monitoring Failures*. https://owasp.org/Top10/A09_2021-SecurityLogging_and_Monitoring_Failures/

OWASP Top 10 Team. (2021j). A10: 2021 – *Server-Side Request Forgery (SSRF)*. https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

OWASP Top 10 Team. (2021k). OWASP Top 10: 2021. OWASP Top 10: 2021. <https://owasp.org/Top10/>

Patel, K. (2019). A survey on vulnerability assessment & penetration testing for secure communication. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 320–325). IEEE.

- Peltier, T. R. (2006). Social engineering: Concepts and solutions. *Information Security Journal*, 15(5), 13. Taylor & Francis.
- Penetration Test Guidance Special Interest Group & PCI Security Standards Council. (2017). *PCI Data Security Standard (PCI DSS)*. Security Standards Council. https://listings.pcisecuritystandards.org/documents/Penetration-Testing-Guidance-v1_1.pdf
- Postman. (2023). *2023 State of the API Report*. <https://www.postman.com/state-of-api/api-technologies/>
- Postman. (2024a). *API monitoring*. <https://www.postman.com/api-platform/api-monitoring/>
- Postman. (2024b). *API versioning*. <https://www.postman.com/api-platform/api-versioning/>
- Preibisch, S. (2018). *API Development*. New York: CA Technologies.
- RiskOptics. (2022). *Vulnerability Scanners: Passive Scanning vs. Active Scanning*. <https://reciprocity.com/blog/vulnerability-scanners-passive-scanning-vs-active-scanning/>
- Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, 166. <https://doi.org/10.1016/j.comnet.2019.106960>. Elsevier.
- Roy, A. (2024). *The Trends and Innovations in the Application Programme Interface World for 2024*. <https://www.testhouse.net/blogs/key-trends-and-innovations-in-the-api-world-for-2024/#:~:text=API%20security%20remains%20a%20critical,APIs%20from%20vulnerabilities%20and%20breaches>
- Sakovich, N. (2023, July 13). *16 Latest Software Development Trends in 2023*. <https://www.sam-solutions.com/blog/software-development-trends/>
- SALT. (2024). API Security Trends 2024. <https://salt.security/api-security-trends>
- Sattam, J. A., & Moulahi, T. (2023). *API Security Testing: The Challenges of Security Testing for Restful APIs*. <https://doi.org/10.5281/ZENODO.7988410>
- Sharma, M. (2021). Review of the benefits of DAST (dynamic application security testing) versus SAST. *International Journal of Management and Engineering Research*, 1(1), 05–08. Innovative Research Publications.

- Shields, T. (2024). *API Security in 2024: Navigating New Threats and Trends*. <https://www.cyber-security-insiders.com/api-security-in-2024-navigating-new-threats-and-trends/>
- SmartBear Software. (2024). *Swagger.io*. <https://swagger.io/>
- Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, 8(9), 664–671. Blue Eyes Intelligence Engineering & Sciences Publication.
- Stafford, V. (2020). Zero trust architecture. *NIST Special Publication*, 800, 207. National Institute of Standards and Technology.
- Tesauro, M. (n.d.). *API Security Tools*. https://owasp.org/www-community/api_security_tools#
- Tian, Z., Qiao, H., Tian, J., Zhu, H., & Li, X. (2018). An automated brute force method based on web-page static analysis. In *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)* (pp. 100–103). IEEE.
- Tian-yang, G., Yin-Sheng, S., & You-yuan, F. (2010). Research on software security testing. *International Journal of Computer and Information Engineering*, 4(9), 1446–1450. World Academy of Science, Engineering and Technology.
- Traceable. (2023). 2023 API Security Trends Revealed: Insights, Risks, and Strategies. <https://www.traceable.ai/blog-post/2023-state-of-api-security-trends>
- Tufte, E. R., & Graves-Morris, P. R. (1983). *The visual display of quantitative information* (Vol. 2). Cheshire, USA: Graphics press.
- VoidZero Inc., & Vite Contributors. (2019). *Vite* [Computer software]. <https://vite.dev/>
- Von Solms, R., & Van Niekerk, J. (2013). From information security to cyber security. *Computers & Security*, 38, 97–102. Elsevier.
- Wallarm. (2024). *Introducing the Wallarm 2024 API ThreatStats™ Report*. <https://lab.wallarm.com/introducing-the-wallarm-2024-api-threatstatstm-report/>
- Wichers, D. (n.d.). *Free for Open Source Application Security Tools*. https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools
- WKO. (2023, October 18). Zahl der Cyberangriffe stieg um 89 Prozent. <https://www.wko.at/wien/news/zahl-der-cyberangriffe-stieg-um-89-prozent>

- Xu, H., Wang, S., Li, N., Wang, K., Zhao, Y., Chen, K., Yu, T., Liu, Y., & Wang, H. (2024). Large language models for cyber security: A systematic literature review. *arXiv Preprint arXiv:2405.04760*.
- ZAP Dev Team. (2024a). *Documentation*. <https://www.zaproxy.org/docs/>
- ZAP Dev Team. (2024b). *Scan Policy*. <https://www.zaproxy.org/docs/desktop/start/features/scanpolicy/>
- ZAP Dev Team. (2024c). ZAP. <https://www.zaproxy.org/>
- Zhiwei, L., & Zhongliang, P. (2020). Realization of Integrity Test of Boundary-Scan Structure. In *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)* (pp. 722–724). IEEE.

7. Anhang

Anhang A: Kategorisierung der Werkzeuge

Name	Link	Kategorie	Verwendung	Verfügbarkeit
OWASP Attack Surface Detector	https://owasp.org/www-project-attack-surface-detector/	SAST	Analysiert Code, um Angriffsflächen zu identifizieren	Open Source
Zed Attack Proxy (ZAP-Proxy)	https://www.zaproxy.org/	DAST / API-Scanner	Dynamisches Scannen von Webanwendungen und APIs	Open Source
GitHub code scanning	https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning	SAST	Automatisches Scannen von Code auf Schwachstellen und Fehler in Github-Projekten	Open Source
Contrast Scan	https://www.contrastsecurity.com/contrast-scan	SAST / API-Scanner	Web App and API code scanners	Kommerziell
Coverity Scan Static Analysis	https://scan.coverity.com/	SAST	Statische Analyse von Open Source Projekten	Kommerziell

HCL AppScan CodeSweep	https://www.hcl-software.com/appscan/products/appscan-codesweep	SAST	Scannen von Code auf Schwachstellen in GitHub-Projekten	Kommerziell
Grabber	http://rgaucher.info/beta/grabber/	DAST	Webanwendung Sicherheits-Scanner	Kommerziell
Nikto2	https://cirt.net/nikto2	DAST	Scanner für Webserver-Sicherheitslücken	Open Source
Golismero	https://github.com/golismero/golismero	DAST/ SAST/ API-Scanner	Sicherheitsanalyse von Webanwendungen und APIs	Open Source
Vega	https://subgraph.com/vega/	DAST / API-Scanner	Webapplikations-Sicherheits-Scanner	Open Source
Wapiti	https://wapiti-scanner.github.io/	DAST / API-Scanner	Dynamisches Scannen von Webanwendungen und APIs	Open Source
OWASP-Xenotix-XSS-Exploit-Framework	https://github.com/ajinabraham/OWASP-Xenotix-XSS-Exploit-Framework	Spezifischer Vulnerability Scanner (API), API-Testing	Cross-Site Scripting Angriffe erkennen und testen	Open Source
OpenVAS	https://openvas.org/	DAST	Netzwerk-Sicherheitsscanner	Kommerziell
Arachni	https://ecsypno.com/pages/arachni-web-application-security-scanner-framework	DAST	Framework für Sicherheits-scans von Webanwendungen	Kommerziell
IronWASP	https://github.com/swatv3nub/IronWASP	DAST	Webanwendung Sicherheits-Scanner, nicht mehr aktuell (veraltet)	Open Source
w3af	http://w3af.org/	DAST / API-Scanner	Webanwendung Angriffs- und Audit-Framework	Open Source

OpenSCAP	https://www.open-scap.org/getting-started/	SAST	Werkzeuge und Richtlinien zur Bewertung der Sicherheit von Systemkonfigurationen	Open Source
Clair	https://www.redhat.com/en/topics/containers/what-is-clair	Spezifischer Vulnerability Scanner	Schwachstellen in Container/Docker-Images erkennen	Open Source
Vulcan	https://vulcan.io/lp/demo/?nab=1	API-Scanner	Scanning und Erkennen von Sicherheitslücken in APIs	Kommerziell
HPI-Vuln	https://hpi-vdb.de/	Datenbanken	Bereitstellung einer Datenbank mit Informationen zu Software-Schwachstellen	Kommerziell
SQL-Map	https://sqlmap.org/	Spezifischer Vulnerability Scanner (API)	Automatisiertes Erkennen und Ausnutzen von SQL-Injection-Schwachstellen	Open Source
Havij	https://github.com/ZiaurRezaJoy/Havij	Spezifischer Vulnerability Scanner (API)	SQL-Injection-Tool	Open Source
Skipfish	https://www.kali.org/tools/skipfish/	DAST / API-Scanner	Rekursive Web-Application-Sicherheits-Scanner	Open Source
WAVSEP	https://github.com/sectooladdict/wavsep	DAST	Webanwendung Vulnerability Scanner (veraltet!)	Open Source
Sonarqube	https://www.sonarsource.com	SAST	Analyse von Codequalität und Sicherheit	Kommerziell
FindSecurityBugs	https://find-sec-bugs.github.io/	SAST	Identifizieren von Sicherheitslücken in Java- und Kotlin-Anwendungen	Open Source
APIClarity	https://github.com/openclarity/apiclarity	API-Scanner	Überwachung und Analyse von API-Traffic zur Erkennung von Sicherheitslücken	Open Source

Astra	https://github.com/flipkart-incubator/Astra	API-Scanner	Automatisiertes Sicherheitstest-Tool für REST APIs	Open Source
Automatic API Attack Tool	https://github.com/imperva/automatic-api-attack-tool	API-Scanner / API-Testing	Simuliert Angriffe auf APIs zur Erkennung von Schwachstellen	Open Source
Cherrybomb	https://blstsecurity.com/cherrybomb	API-Scanner	Analyse von API-Design zur frühzeitigen Erkennung von Sicherheitsproblemen	Kommerziell
wfuzz	https://github.com/xmendez/wfuzz	Fuzzer	Tool für das Fuzzing von Webanwendungen	Open Source
ffuf	https://github.com/ffuf/ffuf	Fuzzer	Schneller Web-Fuzzer	Open Source
Openapi3-fuzzer	https://github.com/vwt-digital/openapi3-fuzzer	Fuzzer	Fuzzing von APIs basierend auf OpenAPI-Spezifikationen	Open Source
CI Fuzz CLI	https://www.code-intelligence.com/product-ci-fuzz	DAST / Fuzzer	Dynamisches Testen und Fuzzing von Code	Kommerziell
Code Intelligence App	https://www.code-intelligence.com/guided-product-tour	DAST / Fuzzer	Automatisiertes Fuzz-Testing in CI/CD-Pipeline	Kommerziell
GraphQL Cop	https://github.com/dolevf/graphql-cop	API-Scanner	Sicherheitstool speziell für GraphQL APIs in CI/CD-Pipelines	Open Source
Pynt	https://www.pynt.io/	API-Scanner	API-Sicherheitstests und -Überwachung	Kommerziell
Akto	https://www.akto.io	API-Scanner	Sicherheitstests für APIs	Kommerziell

GoLismero	https://github.com/golismero/golismero	API Scanner/ Netzwerk- Scanner	Datenvalidierung und Informationsgewinnung für Webanwendungen, Netzwerke und Datenbanken	Open Source
Arachni	https://ecsypno.com/pages/arachni-web-application-security-scanner-framework	DAST	Dynamische Analyse von Webanwendungen	Kommerziell
StackHawk	https://stackhawk.com/	DAST	Optimiert für CI/CD-Pipelines, basiert auf ZAP	Kommerziell
Sec-Helpers	https://github.com/vwt-digital/sec-helpers/tree/master	DAST	Sammlung von dynamischen Testpraktiken	Kommerziell
OWASP Purple Team	https://owasp.org/www-project-purpleteam/	DAST	SaaS für Sicherheitsregressionstests	Kommerziell
42Crunch	https://42crunch.com/	API-Security	API-Sicherheitsplattform	Kommerziell
APISec	https://www.apisec.ai/	API-Security	Automatisierte API-Sicherheitstests	Kommerziell
API Secure	https://www.datatheorem.com/products/api-secure/	API-Security	Schutz und Überwachung von APIs	Kommerziell
API-Security	https://www.imperva.com/products/api-security/	API-Security	Schutz und Überwachung von APIs	Kommerziell
AppSentinels Full LifeCycle API-Security Platform	https://www.appsentinels.ai/	API-Security	Sicherheitsplattform für API-Lebenszyklus	Kommerziell
Aptori	https://aptori.dev/	API-Security	Sammlung von API-Sicherheitswerkzeuge	Kommerziell

BurpSuite Professional	https://portswigger.net/burp/pro	API-Testing	Tool für Sicherheitstests von Webanwendungen und APIs	Kommerziell
Beagle Security	https://beaglesecurity.com/	API-Testing / DAST	Automatisiertes Testing-Tool für APIs und Webanwendungen	Kommerziell
Bright	https://brightsec.com/			Kommerziell
Cequence Security-UAP	https://www.cequence.ai/	API-Security	API-Sicherheitsplattform	Kommerziell
Escape	https://escape.tech/	API-Security	API-Sicherheitsplattform	Kommerziell
FireTail	https://www.firetail.io/	API-Security	Monitoring und Schutz von GraphQL-APIs	Kommerziell
ImmuniWeb Neuron	https://www.immuniweb.com/products/neuron/	API-Security	KI-basierte Lösung für Sicherheits- und Compliance-Management	Kommerziell
Levo.ai	https://www.levo.ai/	API-Security	KI-gestützte Sicherheitsautomatisierung für APIs	Kommerziell
Noname Security Platform	https://nonamesecurity.com/platform	API-Security	API-Sicherheitsplattform	Kommerziell
Nuclei	https://github.com/projectdiscovery/nuclei	Netzwerk-Scanner / API-Scanner	Automatisiertes Scannen für Sicherheitslücken (vorrangig Netzwerk-Protokolle)	Open Source
Pentest-Tools.com API Scanner	https://pentest-tools.com/website-vulnerability-scanning/api-scanner	API-Scanner	Scannen von APIs auf Sicherheitslücken	Kommerziell
Probely	https://probely.com/	API-Scanner	Automatisiertes Scannen von APIs und Webanwendungen	Kommerziell
Postman	https://www.postman.com/	API-Testing	Entwicklung und Testen von APIs	Kommerziell

Rapid7 insightAppSec	In-	https://www.rapid7.com/products/insightappsec/	DAST / API-Scanner	Dynamisches Scannen von Webanwendungen und APIs	Kommerziell
Resurface		https://graylog.org/products/api-security/	API-Security	Logging und Überwachung von API-Zugriffen	Kommerziell
Salt Security API Protection Platform		https://salt.security/	API-Security	API-Sicherheitsplattform	Kommerziell
Soap UI		https://www.soapui.org/tools/soapui/	API-Testing	Testen von SOAP und REST APIs	Kommerziell
Threatspy		https://www.secureblink.com/threatspy	API-Security	Sicherheitsplattform für Bedrohungserkennung	Kommerziell
Traceable AI		https://www.traceable.ai/	API-Security	Schutz und Überwachung von APIs	Kommerziell
Wallarm		https://www.wallarm.com/	API-Security	Schutz von Webanwendungen und APIs	Kommerziell
WebInspect		https://www.opentext.com/products/fortify-webinspect	API-Scanner	Scannen von Webanwendungen und APIs	Kommerziell
SecOps Solution		https://secopsolution.com/	API-Sanning	Sicherheitsüberprüfungen für Cloud und APIs	Kommerziell
Equixly		https://equixly.com/	API-Scanner	Überwachung und Schutz von APIs	Kommerziell
Intruder		https://www.intruder.io/vulnerability-scanner/api-scanner	API-Scanner / Netzwerk-Scanner	Sicherheitsüberprüfung für Netzwerke und APIs	Kommerziell
VulnAPI		https://vulnapi.cerberauth.com/docs/	DAST / API-Scanner	Dynamisches Scannen von APIs	Open Source

Anhang B: Auswahl der Werkzeuge

Kriterien der Werkzeugauswahl:

- Runde 1: Open Source Werkzeuge Zugangsmöglichkeit mit der Kategorie API-Scanner
- Runde 2: Jene Tools dessen Systemvoraussetzungen und Voraussetzungen nicht erfüllt werden können oder keine Informationen zu den Scanergebnissen auffindbar war.

	Tool	Genau Auflistung der möglichen Schwachstellen vorhanden?	Integration	Runde 2
1	ZAPProxy	Ja	Ja (API)	X
2	Golismero and nikto2	Nein	k.A.	
3	Vega	Teilweise	Nein	Tool veraltet (2014)
4	Wapiti	Ja	Ja (Workflow)	X
5	w3af	Ja	k.A.	X
6	APIClarity	Ja	Ja (Workflow)	X
7	Astra	Ja	k.A.	Zu viele Voraussetzungen (Linux, MongoDB,...)
8	GraphQL Cop	Ja	Ja (CD/CI-Pipelines)	X
9	VulnAPI	Nein	Ja (Workflow, CI/CD)	
10	Nuclei	Nein	Ja (CD/CI-Pipelines, sowie Integration zu GitHub möglich)	
11	GoLismero	Nein	k.A.	
12	Automatic API Attack Tool	Nein	Ja (Workflow)	

- Runde 3 – genauere Analyse der Werkzeuge. Für welche OWASP Vulnerabilities könnten sie eingesetzt werden und ist diese Information auffindbar? Die Listen wurden auf den jeweiligen Doku-Seiten der Tools gefunden.

ZAPProxy:

Threats (unter anderem):	OWASP
Directory Browsing, Source Code Disclosure, Information Disclosure	API8
Cookie No HttpOnly Flag, Cookie Without Secure Flag	API2
Password Autocomplete in Browser	API2
Incomplete or No Cache-control and Pragma HTTP Header Set	API8
Web Browser XSS Protection Not Enabled, Cross Site Scripting	API5
Cross-Domain JavaScript Source File Inclusion, Cross-Domain Misconfiguration	API7
X-Frame-Options Header Scanner, X-Content-Type-Options Header Missing	API8
HTTP Parameter Override, HTTP Parameter Pollution scanner	API8/API4
Absence of Anti-CSRF Tokens, Anti CSRF Tokens Scanner	API2
Remote Code Execution (Shell Shock), Remote OS Command Injection, Server Side Code Injection	Mehrere – API1/API3/API5/API8
SQL Injection, XPath Injection	API1/API3
Buffer Overflow, Format String Error, Integer Overflow Error	API4
CRLF Injection, Parameter Tampering	API8
Session ID in URL Rewrite	API2
External Redirect	API5
Insecure HTTP Method	API8
API1, API2, API3, API4, API5, API7, API8, API9, API10	

Wapiti:

Threat	
	OWASP
SQL Injections (Error based, boolean based, time based) and XPath Injections	API1/API3
Cross Site Scripting (XSS) reflected and permanent	API2
File disclosure detection (local and remote include, require, fopen, readfile...)	API8
Command Execution detection (eval(), system(), passtru()...)	API8
XXE (Xml eXternal Entity) injection	API7
CRLF Injection	API8
Search for potentially dangerous files on the server	API8
Bypass of weak htaccess configurations	API8
Search for copies (backup) of scripts on the server	API8
Shellshock	API8
Folder and file enumeration (DirBuster like)	API9
Server Side Request Forgery	API7
Open Redirects	API8
Detection of uncommon HTTP methods (like PUT)	API8
Basic CSP Evaluator	API8
Brute Force login form (using a dictionary list)	API2
Checking HTTP security headers	API8
Checking cookie security flags (secure and httponly flags)	API2/API8
Cross Site Request Forgery (CSRF) basic detection	API8/API2
Fingerprinting of web applications using the Wappalyzer database	API9
Enumeration of Wordpress and Drupal modules	API9
Detection of subdomain takeovers vulnerabilities	API9
Log4Shell vulnerability detection	API8

Check for TLS misconfiguration and vulnerabilities	API8
API1, API2, API3, API7, API8, API9, API10	

API-Clarity:

Tool	O-WASP
Trace Analyzer	API1/ API2/ API3
Spec Diffs This module compares the API traces with the OAPI specifications provided by the user	API9
API endpoints that are observed but not documented in the specs, i.e. <i>Shadow APIs</i> ;	API9
API endpoints that are observed but marked as deprecated in the specs, i.e. <i>Zombie APIs</i> ;	API9
difference between of the APIs observed and their documented specification.	API9
BFLA Detector	API5
API1, API3, API5, API9	

➔ Nur mit ausgewählten Technologien kombinier und demnach für diesen Toolvergleich nicht einsetzbar

GraphQL Cop:

Threat	O-WASP
Alias Overloading (DoS)	API4
Batch Queries (DoS)	API4
GET based Queries (CSRF)	API2
POST based Queries using urlencoded payloads (CSRF)	API2

GraphQL Tracing / Debug Modes (Info Leak)	API9
Field Duplication (DoS)	API4
Field Suggestions (Info Leak)	API3
GraphiQL (Info Leak)	API9
Introspection (Info Leak)	API9
Directives Overloading (DoS)	API4
Circular Query using Introspection (DoS)	API4
Mutation support over GET methods (CSRF)	API2
API2, API3, API4, API9	

w3af:

Threats (unter anderem)	
	OWASP
Cookie without HttpOnly, Secure Flag Missing in HTTPS Cookie	API2/API8
SQL Injection	API8/API1/API2
Remote Code Execution & Remote File Inclusion	API10
OS Commanding Vulnerability	API8
Cross-Site Request Forgery (CSRF) Vulnerability	API5
Local File Inclusion Vulnerability	API8
LDAP Injection Vulnerability	API8
Blind SQL Injection Vulnerability	API8
Access-Control-Allow-Origin set to '*'	API6
Cross Side-Scripting	API8
API1, API2, API5, API8, API10	

➔ Installation zum Zeitpunkt der Analyse der Sicherheitswerkzeuge nicht möglich, da eine Installation für Windows nicht empfohlen wird und scheiterte.

Finale Werkzeuge für Analyse/Vergleich der Sicherheitswerkzeuge:

Name	GUI	API	Fokussierte Schwachstellen
ZAPProxy	Ja	Ja	API1, API2, AP3, API4, API5
GraphQL Cop	Nein	Nein	API1, API3, API4, API9
Wapiti	Nein	Nein	API1, API2, API3

Anhang C: Analyse der Sicherheitswerkzeuge

ZAPProxy

Basisinformationen	
Toolname	ZAPProxy
Entwickler/Anbieter	Crash Override Open Source Fellowship
Version/Datum der letzten Aktualisierung	2.14.0 (12.10.2023)
Usability	
Grafische Benutzeroberfläche vorhanden? (0/1)	1 (wurde beim Test aber nicht verwendet)
Einfachheit der Installation und Konfiguration? (1= sehr einfach; 5 = sehr komplex)	1
Qualität der Dokumentation (1=hervorragend; 5 = unzureichend)	2
Anpassungsfähigkeit (Customization) (1=vollständig anpassbar; 5=nicht anpassbar)	1 (Plug-ins, Skripte)
Funktionalität & Integration	
Umfang der API-Unterstützung	REST, SOAP, GraphQL
Art der Scans	Aktiver und Passiver API Scan möglich*, Plug-in für GraphQL
Möglichkeiten der Datenbereitstellung	Text, JSON, XML, HTML

Integration mit anderen Tools/Systemen (1=volle Integration, 5=keine Integration möglich)	1 (via API)
Abdeckung der OWASP Top 10 API Security-Kategorien	
AP1: Broken Object Level Authentication	Ja
AP2: Broken Authentication	Ja
AP3: Broken Object Property Level Authorization	Ja
AP4: Unrestricted Resource Consumption	Ja
API5: Broken Function Level Authorization	Ja
API6: Unrestricted Access to Sensitive Business Flows	-
API7: Server Side Request Forgery	Ja
API8: Security Misconfiguration	Ja
API9: Improper Inventory Management	-
API10: Unsafe Consumption of APIs	-
Einbindung von Authentifizierungsmechanismen	
Einbindung verschiedener Authentifizierungsarten (z.B. Basic Auth, OAuth, API Keys usw.)	1 (alle gängigen Authentifizierungsmechanismen sind vertreten)
Einfachheit der Einbindung von Authentifizierungsmechanismen (1=sehr einfach; 5=sehr komplex)	2 (Docker)
Berichterstattung	
Abrufen der Berichte (Konsole, HTML, Dashboard, API,...)	Konsole, HTML, API
Übersichtlichkeit der Berichte (1=hervorragend, 5=unzureichend)	2
Automatisierung der Berichterstattung (1=vollautomatisiert verwendbar; 5=manuell/keine Automatisierung)	1 (z.B. mit GitHub Actions, oder Jenkins)
Praktische Anwendung: Analyse einer Beispiel-API	
1. Beispiel-API	VAmPI
Anzahl der URLs	<ul style="list-style-type: none"> • Passive: 4 • Active: 15
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • Low (Passiv): 2 • Information (Passiv): 1 • Low (Active): 4

	<ul style="list-style-type: none"> • Information (Active): 4
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
<ul style="list-style-type: none"> • Passiver Scan: <ul style="list-style-type: none"> - Server Leaks Version Information via "Server" HTTP Response Header Field (L) – 3x – API8 - X-Content-Type-Options Header Missing (L) – 1: API8 - Storable and Cacheable Content (I) – 3x: API3 • Aktiver Scan: <ul style="list-style-type: none"> - A Server Error response code was returned by the server (L) - 5x – API8/API10 - Server Leaks Version Information via "Server" HTTP Response Header Field (L) – 13x: API8 - Unexpected Content-Type was returned (L) – 5x: API8 - X-Content-Type-Options Header Missing (L) – 8x: API8 - Storable and Cacheable Content (I) – 7x: API3 - Non-Storable Content (I) – 5x: API3 - Authentication Request Identified (I) – 1x: API2 - A Client Error Response Case was returned by surver (I) – 119x: API8/API2 	
Gesamtvorkommen der OWASP Kategorien	<p>Passiver Scan:</p> <ul style="list-style-type: none"> • API3: • API8: <p>Aktiver Scan:</p> <ul style="list-style-type: none"> • API2: 2x • API3: 2x • API8: 4x • API10: 1x
2. Beispiel-API	crAPI
Anzahl der URLs	<ul style="list-style-type: none"> • Passiv: 14 (AUTH/ o. AUTH) • Active: 437 (426 AUTH)
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • Medium (Passive): 2 • Low (Passive): 5 • Information (Passive): 2 • High (Active): 1 • Medium (Active): 5 • Low (Active): 5 • Information (Active): 4 • Medium (Active/AUTH): 5 • Low (Active/AUTH): 5 • Info (Active/AUTH): 4
Aufschlüsselung der erfassten Schwachstellen nach OWASP (AUTH/o.AUTH)	
<ul style="list-style-type: none"> • Passiver Scan: <ul style="list-style-type: none"> - Content Security Policy (CSP) Header Not Set (M) – 2x: API8 - Missing Anti-clickjacking Header (M) – 2x: API8 - Dangerous JS Functions (L) – 1x: API1 (potentially exposes sensitive data or functions) - Permissions Policy Header Not Set (L) – 4x: API8 - Server Leaks Version Information via "Server" HTTP Response Header Field (L) – 10x: API8 - Timestamp Disclosure – Unix (L) – 44x: API8 - X-Content-Type-Options Header Missing (L) – 10x: API8 - Information Disclosure - Suspicious Comments (I) – 2x: API8 - Modern Web Application (I) – 2x: No threat! 	

<ul style="list-style-type: none"> - Non-Storable Content (I) – 10x: API3 • Aktiver Scan: <ul style="list-style-type: none"> - SQL Injection (H) – 1x: API1/API3 - .env Information Leak (M) – 1x: API8 - Bypassing 403 (M) – 1x: API1/API5 - CORS Misconfiguration (M) – 45x: API8 - Content Security Policy (CSP) Header Not Set (M) – 7x: API8 - Hidden File Found (M) – 4x: API9 - A Server Error response code was returned by the server (L) – 742 x: AP10/API8 - Application Error Disclosure (L) – 4x: API10/API8 - Permission Policy Header not Set (L) – 7x: API8 - Server Leaks Version Information via "Server" HTTP Response Header Field (L) – 12x: API8 - Unexpected Content-Type was returned (L) – 874 x: API8 - A Client Error response code was returned by the server (I) – 1714x: API2/API8 - Authentication Request Identified (I) – 4x: API2 - Non-Storable Content (I) – 12x: API3 - Session Management Response Identified (I) – 1x: API2 	
Gesamtvorkommen der OWASP Kategorien	<p>Passiver Scan:</p> <ul style="list-style-type: none"> • API1: 1x • API3: 1x • API8: 7x <p>Aktiver Scan:</p> <ul style="list-style-type: none"> • API1: 2x • API2: 3x • API3: 2x • API5: 1x • API8: 9x • API9: 1x • API10: 2x
3. Beispiel-API	REST API GOAT
Anzahl der URLs	4
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • Medium (Passive): 2 • Low (Passive): 3 • Information (Passive): 1 • Medium (Passive/AUTH): 2 • Low (Passive/AUTH): 3 • Info (Passive/AUTH): 1
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
<ul style="list-style-type: none"> - Content Security Policy (CSP) Header Not Set (M) – 3x: API8 - Missing Anti-clickjacking Header (M) – 1x: API8 - Permissions Policy Header Not Set (L) – 3x: API8 - Server Leaks Version Information via "Server" HTTP Response Header Field (L) – 3x: API8 - X-Content-Type-Options Header Missing (L) – 1x: API8 - Storable and Cacheable Content (I) – 3x: API3 	
Gesamtvorkommen der OWASP Kategorien	<ul style="list-style-type: none"> • API3: 1x • API8: 5x
4. Beispiel-API (GraphQL)	Damn Vulnerable GraphQL
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • Medium: 2

	<ul style="list-style-type: none"> • Low: 4 • Information: 7
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
<ul style="list-style-type: none"> - Content Security Policy (CSP) Header Not Set (M) – 20x: API8 - Missing Anti-clickjacking Header (M) – 13x: API8 - Cookie No HttpOnly Flag (L) – 2x: API8 - Cookie without SameSite Attribute (L) – 2x: API8 - Private IP Discloser (L) – 3x: API8 - X-Content-Type-Options Header Missing (L) – 25x: API8 - Authentication Request Identified (I) – 2x: API2 - GraphQL Endpoint Supports Introspection (I) – 1x: API8/9 - GraphQL Server Implementation Identified (I) – 1x: No Threat - Information Discloser – Suspicious Comments (I) – 8x: API8 - Loosley Scoped Cookie (I) – 2x: API8 - Modern Web Application (I) – 13x: No threat - Session Management Response Identified (I) – 5x: API2 	
Gesamtvorkommen der OWASP Kategorien	<ul style="list-style-type: none"> • API2: 2x • API8: 9x
Performance und Effizienz der Analyse (Durchschnitt aller Scans)	<ul style="list-style-type: none"> • Active: 528 Sekunden • Passive: 60 Sekunden
Fazit	
Gesamteindruck des Werkzeugs	<p>Der ZAPProxy API Scanner unterstützt sowohl eine Ausführung der aktiven und passiven Scans über die Benutzeroberfläche, sowie als CLI oder Docker Command. Das Set-Up ist somit ein sehr einfaches. Die Funktionen des Werkzeugs sind sehr vielfältig, aufgrund der integrierten API sehr gut in bereits bestehende Systeme einbindbar sowie für den jeweiligen Zweck anpassbar. Der API-Scan ist für alle gängen API-Typen und Authentifizierungsmechanismen geeignet und stellt die gefundenen Schwachstellen sowohl in der Konsole als auch in verschiedenen Formaten wie HTML oder XML zur Verfügung. Besonders der aktive Scan, wofür die Schnittstellen-Spezifikationen, wie beispielsweise im Format openapi, zur Verfügung gestellt werden muss, kann das Tool Schwachstellen im Falle einer der getesteten APIs 9 der 10 OWASP Kategorien abdecken. Im Falle des passiven Scans derselben API konnten nur Schwachstellen aus 3 Kategorien erkannt werden. Ein Scan mit und ohne Authentifizierung machte hierbei keinen Unterschied. Mit steigender Anzahl der gescannten URLs stieg jedoch auch die Scanningzeit exponentiell. Während ein passiver Scan bei 4 gescannten URLs etwa eine Minute brauchte, waren es bei einem aktiven Scan von 434 URLs über 16 Minuten.</p>
Empfehlung für den Einsatz im API Analysis Dashboard	<p>Für das Dashboard ist der sehr leicht integrierbare ZAPProxy API-Scan eine gute Wahl. Aufgrund von Faktoren wie Zeit aber auch Computer-</p>

	ressourcen und der oft fehlenden Verfügbarkeit von API-Spezifikationen wäre hier jedoch ein ausschließlicher Einsatz von passiven Scans sinnvoller.
--	---

Zeiten: 20 (GraphQL) 67 (p VAmPI), 82 (a VamPI), 95 (p/AUTHp CrAPI), 16 min 14 (a/AUTHa crAPI), 65 (p GOAT), 54 (p AUTH GOAT)

Wapiti

Basisinformationen	
Toolname	Wapiti
Entwickler/Anbieter	Nicolas Surribas
Version/Datum der letzten Aktualisierung	3.1.8 (16.05.2024)
Usability	
Grafische Benutzeroberfläche vorhanden? (0/1)	0
Einfachheit der Installation und Konfiguration? (1= sehr einfach; 5 = sehr komplex)	2 (In Linux), 3 (in Windows)
Qualität der Dokumentation (1=hervorragend; 5 = unzureichend)	3
Anpassungsfähigkeit (Customization) (1=vollständig anpassbar; 5=nicht anpassbar)	3 (setzten von Scanoptionen, allerdings keine Plug-Ins o.ä.)
Funktionalität & Integration	
Umfang der API-Unterstützung	REST, SOAP, GraphQL
Art der Scans	Aktive Scans (könnte als Angreifer von der gescannten API wahrgenommen werden)
Möglichkeiten der Datenbereitstellung	HTML, XML, JSON, TXT, CSV
Integration mit anderen Werkzeuge/Systemen (1=volle Integration, 5=keine Integration möglich)	4 (in CD/CI Pipelines über CL Commands, gibt aber keine API o.ä.)
Abdeckung der OWASP Top 10 API Security-Kategorien	

AP1: Broken Object Level Authentication	Ja
AP2: Broken Authentication	Ja
AP3: Broken Object Property Level Authorization	Ja
AP4: Unrestricted Resource Consumption	-
API5: Broken Function Level Authorization	-
API6: Unrestricted Access to Sensitive Business Flows	-
API7: Server Side Request Forgery	Ja
API8: Security Misconfiguration	Ja
API9: Improper Inventory Management	Ja
API10: Unsafe Consumption of APIs	Ja
Einbindung von Authentifizierungsmechanismen	
Einbindung verschiedener Authentifizierungsarten (z.B. Basic Auth, OAuth, API Keys usw.)	3 (Einfache wie Basic Auth ja)
Einfachheit der Einbindung von Authentifizierungsmechanismen (1=sehr einfach; 5=sehr komplex)	3 (siehe oben – bei Oauth o.ä. kann eine Einbindung mitunter schnell komplex werden)
Berichterstattung	
Abrufen der Berichte (Konsole, HTML, Dashboard, API,...)	HTML
Übersichtlichkeit der Berichte (1=hervorragend, 5=unzureichend)	3
Automatisierung der Berichterstattung (1=vollautomatisiert verwendbar; 5=manuell/keine Automatisierung)	1 (in CI/CD Pipeline eingebunden)
Praktische Anwendung: Analyse einer Beispiel-API	
1. Beispiel-API	VAmPI
Anzahl an URLs	1
Anzahl der gefunden Schwachstellen	2
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
<ul style="list-style-type: none"> - Content Security Policy Configuration – 1x („CSP is not set“): API8 - HTTP Secure Headers – 2x („X-Frame-Options is not set“; „X-Content-Type-Options is not set“): API8 	
Gesamtvorkommen der OWASP Kategorien	• API8: 2x

2. Beispiel-API	crAPI
Anzahl der URLs	21
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • o. AUTH: 2 • AUTH: 2
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
- Content Security Policy Configuration – 1x („CSP is not set“): API8 - HTTP Secure Headers – 2x („X-Frame-Options is not set“; „X-Content-Type-Options is not set“): API8	
Gesamtvorkommen der OWASP Kategorien	<ul style="list-style-type: none"> • API8: 2x
3. Beispiel API	Rest API Goat
Anzahl der URLs	1 (AUTH/o.AUTH)
Anzahl der gefunden Schwachstellen	1 (AUTH/o.AUTH)
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
- Content Security Policy Configuration – 1x („CSP is not set“): API8 - HTTP Secure Headers – 2x („X-Frame-Options is not set“; „X-Content-Type-Options is not set“): API8	
Gesamtvorkommen der OWASP Kategorien	<ul style="list-style-type: none"> • API8: 2x
Performance und Effizienz der Analyse (Durchschnitt aller Scans)	12.7 Sekunden
Fazit	
Gesamteindruck des Werkzeugs	<p>Wapiti ist ein sehr leichtgewichtiges auf Python basierendes Tool welche, nach Aussagen der Entwickler, vorrangig für CD/CI-Pipelines eingesetzt werden kann. Es ist nicht explizit für das Scannen von APIs entwickelt, deckt diese aber mit ab. Wapiti besitzt keine Benutzeroberfläche und wurde aufgrund der primären Unterstützung von Linux über ein Dockerimage ausgeführt. Das Tool arbeitet mit aktiven Scans und benötigt dabei lediglich die URL und bei Bedarf den Authentifizierungstoken. Wapiti kann nur beschränkt mit bereits bestehenden Systemen integriert werden und auch ein Customizing ist nicht vollständig möglich. Wapiti unterstützt dabei alle gängigen API-Typen und kann die Ergebnisse in vielerlei Formaten zur Verfügung stellen. Die Einsicht der Ergebnisse geschieht über einem HTML-Seite. Die von Wapiti in der Dokumentation genannten Schwachstellen, welche das Tool extrahieren kann können 7 von 10 OWASP Kategorien zugeordnet werden. Die Ergebnisse der Tests zeigten hier jedoch ein anderes Bild. Wapiti konnte lediglich Schwachstelle der Kategorie 8 finden, und meist handelte</p>

	es sich hierbei um dieselben 2-3 Schwachstellen. Die durchschnittliche Scanzeit lag bei etwa 13 Sekunden. Diese erhöhte sich auch bei einer größeren Anzahl an URLs nicht exponentiell.
Empfehlung für den Einsatz im API Analysis Dashboard	Aufgrund der beschränkten OWASP-Abdeckung der gefundenen Schwachstellen wird dieses Tool nicht für die Einbindung in das API Analysis Dashboard verwendet.

Zeiten: 10 (VAmPI), 17 (crapAPI), 11 (GOAT)

GraphQL Cop

Basisinformationen	
Toolname	GraphQL Cop
Entwickler/Anbieter	GitHub-User: dolevf
Version/Datum der letzten Aktualisierung	1.12 (03.11.2022)
Usability	
Grafische Benutzeroberfläche vorhanden? (0/1)	0
Einfachheit der Installation und Konfiguration? (1= sehr einfach; 5 = sehr komplex)	2 (Linux), 3 (Windows)
Qualität der Dokumentation (1=hervorragend; 5 = unzureichend)	4
Anpassungsfähigkeit (Customization) (1=vollständig anpassbar; 5=nicht anpassbar)	3 (keine Plugins, nur Options in den Commands)
Funktionalität & Integration	
Umfang der API-Unterstützung	GraphQL
Art der Scans	Aktive Scans
Möglichkeiten der Datenbereitstellung	Text, JSON
Integration mit anderen Werkzeuge/Systemen (1=volle Integration, 5=keine Integration möglich)	4 (in CD/CI Pipelines über CL Commands, gibt aber keine API o.ä.)
Abdeckung der OWASP Top 10 API Security-Kategorien	
AP1: Broken Object Level Authentication	-

AP2: Broken Authentication	Ja
AP3: Broken Object Property Level Authorization	Ja
AP4: Unrestricted Resource Consumption	Ja
API5: Broken Function Level Authorization	-
API6: Unrestricted Access to Sensitive Business Flows	-
API7: Server Side Request Forgery	-
API8: Security Misconfiguration	-
API9: Improper Inventory Management	Ja
API10: Unsafe Consumption of APIs	-
Einbindung von Authentifizierungsmechanismen	
Einbindung verschiedener Authentifizierungsarten (z.B. Basic Auth, OAuth, API Keys usw.)	3 (Einfache wie Basic Auth ja)
Einfachheit der Einbindung von Authentifizierungsmechanismen (1=sehr einfach; 5=sehr komplex)	3 (siehe oben – bei OAuth o.ä. kann eine Einbindung mitunter schnell komplex werden)
Berichterstattung	
Abrufen der Berichte (Konsole, HTML, Dashboard, API,...)	Konsole
Übersichtlichkeit der Berichte (1=hervorragend, 5=unzureichend)	3
Automatisierung der Berichterstattung (1=vollautomatisiert verwendbar; 5=manuell/keine Automatisierung)	1 (in CI/CD Pipeline eingebunden)
Praktische Anwendung: Analyse anhand von Beispiel-APIs	
1. Beispiel-API	Damn Vulnerable GraphQL
Anzahl der URLs	1 (/graphql)
Anzahl der gefunden Schwachstellen	<ul style="list-style-type: none"> • High: 6 • Medium: 2 • Low: 3 • Info: 1
Aufschlüsselung der erfassten Schwachstellen nach OWASP	
<ul style="list-style-type: none"> - Alias Overloading (H): API4 - Array-based Query Batching (H): API4 - Directive Overloading (H) – 2x: API4 	

<ul style="list-style-type: none"> - Field Duplication (H): API4 - Introspection (H): API9 - Introspection-Based Circular Query (H): API4 - GET Method Query Support (M): API2 - POST Based URL-Encoded Query (possible CSRF) (M): API2 - Field Suggestions (L) – 2x: API3 - GraphQL IIDE (L): API9 - Unhandled Errors Detection (I): API9 	
Performance und Effizienz der Analyse	5 Sekunden
Gesamtvorkommen der OWASP Kategorien	<ul style="list-style-type: none"> • API2: 2x • API3: 1x • API4: 5x • API9: 3x
Fazit	
Gesamteindruck des Werkzeugs	Aufgrund der beschränkten Testmöglichkeit des Scanningwerkzeugs kann kein direkter Vergleich zu einer zweiten API hergestellt werden. Im Beispielszenario zeigt sich das Scanning-Tool aber durchaus positiv. Das Tool spezialisiert sich hierbei ausschließlich auf GraphQL-APIs und zeigt auch während des Scans einen Fokus auf GraphQL-spezifischen Schwachstellen. Dabei unterstützt GraphQL-Cop keine Benutzeroberfläche und kann über ein via GitHub zur Verfügung gestelltes Repository mittels Python Programm gestartet werden. Der Output beschränkt sich auf die Konsolenausgabe, hierbei werden aber die Risikolevel der jeweiligen Schwachstelle mitangedruckt. Die Effizienz des Werkzeugs wirkt sehr solide, was bei einer steigenden Anzahl an scannenden URLs passiert, konnte bei dieser Analyse der Sicherheitswerkzeuge nicht überprüft werden. Das Customizing, sowie die Integration in bestehende Systeme ist eingeschränkt. Eine Einbettung in die CD/CI-Pipeline wird aber seitens der Entwickler als potentielles Einsatzgebiet explizit genannt. In Bezug auf die Abdeckung der OWASP Kategorien konnte nur 4 der 10 Kategorien während der Analyse extrahiert werden. Auch die Dokumentation des Werkzeugs lassen das Erkennen von Schwachstellen im Bereich dieser 4 Kategorien vermuten.
Empfehlung für den Einsatz im API Analysis Dashboard	Da der Fokus des API Analysis Dashboards nicht rein auf GraphQL-APIs liegen soll wird dieses Tool nicht für die Einbindung in das Analysis Dashboard miteinbezogen.

Test-APIS:

API	Github-Projekt	GraphQL	AUTH	Docs	Einsetzbar
VAmPI	erev0s/VAmPI	Nein	Nein	Ja	Ja

Set-Up:

```
docker run -d -p 5001:5000 erev0s/vampi:latest
```

- ZAP-passiv Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:5001/ -r api-passive-scan-report_vampi.html
```

- ZAP-active Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-weekly zap-api-scan.py -t vampi.yaml -f openapi -r api-active-scan-report_vampi.html
```

- WAPITI Scan:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenaCode/masterthesis:latest wapiti -u http://host.docker.internal:5000/
```

Pixi	DevSlop/Pixi	Nein	Nein	Nein	Ja
------	--------------	------	------	------	----

Set-Up:

- Download des Github-Repositories
- docker compose up -d

- ZAP-passiv Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:8090/ -r api-passive-scan-report_pixi.html
```

- WAPITI Scan:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenaCode/masterthesis:latest wapiti -u "http://host.docker.internal:8090/"
```

crAPI	OWASP/crAPI	Nein	Ja	Ja	Ja
-------	-------------	------	----	----	----

Set-Up:

- curl.exe -o docker-compose.yml https://raw.githubusercontent.com/OWASP/crAPI/main/deploy/docker/docker-compose.yml
- docker-compose pull
- docker-compose -f docker-compose.yml --compatibility up -d

- ZAP-passiv Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:8888/ -r api-passive-scan-report_crapi.html
```

- ZAP-active Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-weekly zap-api-scan.py -t crapi.json -f openapi -r api-active-scan-report_crapi.html
```

- ZAP-passive Scan mit AUTH:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g /zap/wrk/auth.conf -t http://host.docker.internal:8888/ -r api-passive-scan-report_crapi_auth.html -z "auth.header=Authorization: Basic $(echo -n 'lomlabupsa@gufum.com:Test1234!' | base64)"
```

- ZAP-active Scan mit AUTH:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-weekly zap-api-scan.py -t crapi.json -f openapi -r api-active-scan-report_crapi_auth.html -z "auth.header=Authorization: Basic $(echo -n 'lomlabupsa@gufum.com:Test1234!' | base64)"
```

- WAPITI Scan:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenacode/masterthesis:latest wapiti -u http://host.docker.internal:8888/
```

- WAPITI Scan mit AUTH:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenacode/masterthesis:latest wapiti -u http://host.docker.internal:8888/ --header "Authorization: Basic $(echo -n 'lomlabupsa@gufum.com:Test1234!' | base64)"
```

vulnAPI	tkisason/vulnapi	Nein	Nein	Ja	Ja
---------	------------------	------	------	----	----

Set-Up:

- Download des Git-Repositories
- docker build --tag vulnapi .
- docker run -it --rm -p8000:8000 vulnapi
- bei run.sh mussten die Zeilenenden in ein UNIX/OSX Format gebracht werden

- ZAP-passive Scan:

ZAP-passiv Scan: docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:8000 / -r api-passive-scan-report_vulnapi.html

- ZAP-active Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-weekly zap-api-scan.py -t vulnapi.yaml -f openai -r api-active-scan-report_cvulnapi.html
```

- WAPITI Scan:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenaicode/masterthesis:latest wapiti -u http://host.docker.internal:8000/
```

Rest API Goat	optiv/rest-api-goat	Nein	Ja	Nein	Ja
----------------------	---------------------	------	----	------	----

Set-Up:

- Download des Git-Repositories
- docker build -t rest-api-goat:latest .
- docker run -d -p 5000:5000 rest-api-goat
- (Anpassung im requirements.txt war nötig)

- ZAP-passiv Scan:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:5000/ -r api-passive-scan-report_goat.html
```

- ZAP-passive Scan mit AUTH:

```
docker run -v /c/Users/König/zap-wrk:/zap/wrk -t owasp/zap2docker-stable zap-baseline.py -g api-passive-scan.conf -t http://host.docker.internal:5000/ -r api-passive-scan-report_goat_auth.html -z "auth.header= Authorization: Bearer jyrvml4k9tvdivesxwgku"
```

- WAPITI Scan:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenaicode/masterthesis:latest wapiti -u "http://host.docker.internal:5000/"
```

- WAPITI Scan mit AUTH:

```
docker run -it --rm -v "C:\Users\König\OneDrive\Dokumente\Uni\linz\WIN_Master\3.Semester\Masterarbeit\Toolanalyse\wapiti\Reports:/root/.wapiti/generated_report" magdalenaicode/masterthesis:latest wapiti -u "http://host.docker.internal:5000/" --header "Authorization: Bearer jyrvml4k9tvdivesxwgku"
```

Damn Vulnerable GraphQL	dolevf/Damn-Vulnerable-GraphQL-Application	Ja	Nein	Nein	Ja
--------------------------------	--	----	------	------	----

Set-Up:

- | |
|---|
| <ul style="list-style-type: none"> • docker pull dolevf/dvga • docker run -t -d -p 5013:5013 -e WEB_HOST=0.0.0.0 dolevf/dvga |
| <ul style="list-style-type: none"> • ZAP-GraphQL-Scan (via active Scan and Spider) • GraphQL-Cop Scan: graphql-cop>python graphql-cop.py -t http://localhost:5013/ |

Vulnerable Graphql Api	ivision-research/ vulnerable-graphql-api	Ja	Nein	Nein	Nein
-------------------------------	--	----	------	------	------

Set-Up:

- Download des Git-Repositories
- npm install
- npm run tsc
- npm run sequelize db:migrate
- npm run sequelize db:seed:all
- (Linux) ausführen von run.sh
- (Windows) erstellen von run.bat und ausführen der Batch-Datei

- | |
|--|
| <ul style="list-style-type: none"> • Funktioniert leider nicht! |
|--|

Anhang D: API Security Analysis-Dashboard Set-Up

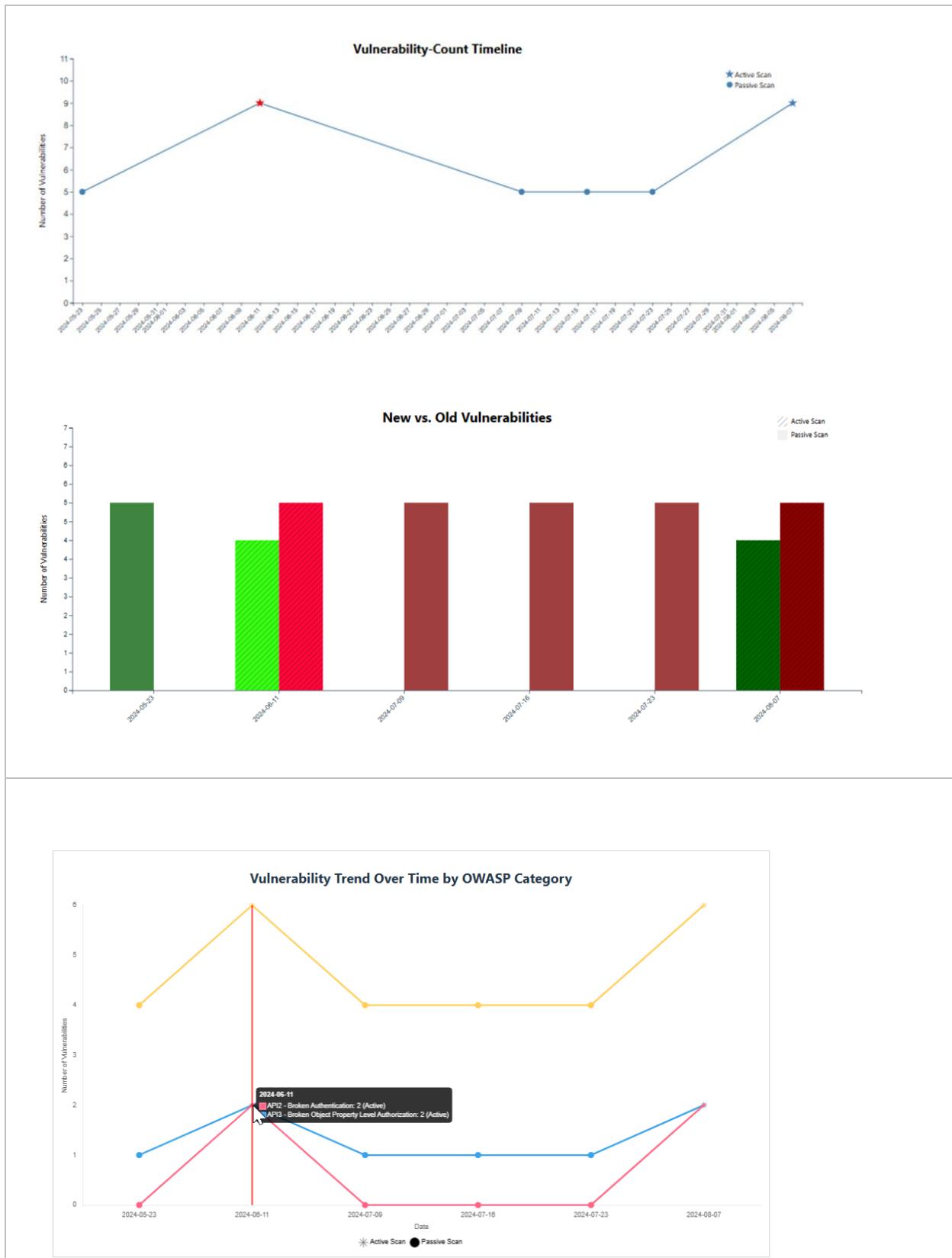
Set up:

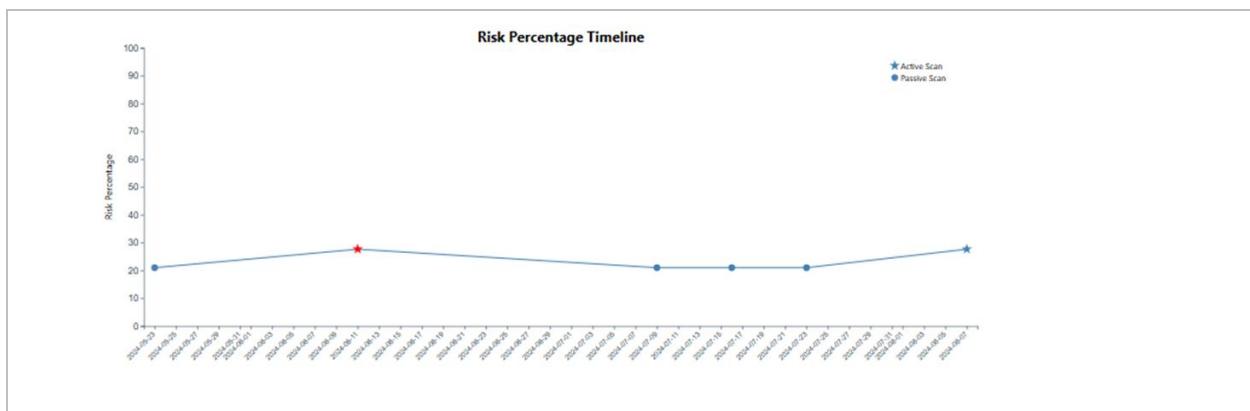
- Download the front- and backend from GitHub (https://github.com/Magdalena-code/API_Security_Analysis_Dashboard)
- Start both in appropriate IDEs
- Set up the virtual environment for the Python backend and import the necessary packages
- Also install:
 - pip install python-dotenv
 - pip install openpyxl
 - the package flask-cors may need to be installed manually depending on the IDE
- Adjust the .env files regarding paths
- Pull and run the docker „master-postgres“
 - docker pull magdalenacode/masterthesis:latest
 - docker run --name master-postgres -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d magdalenacode/masterthesis:latest
- Run the python-script „Initialize_Database.py“ in the dashboard_backend folder
- Run the start_services.bat
 - Check the logs if an issue occurs.
- Open <http://localhost:5173/>
- Open <http://localhost:5000/apidocs> for the API documentation

To test the application run the python-script „Insert_Script_Dummy.py“ in the dashboard_backend folder.

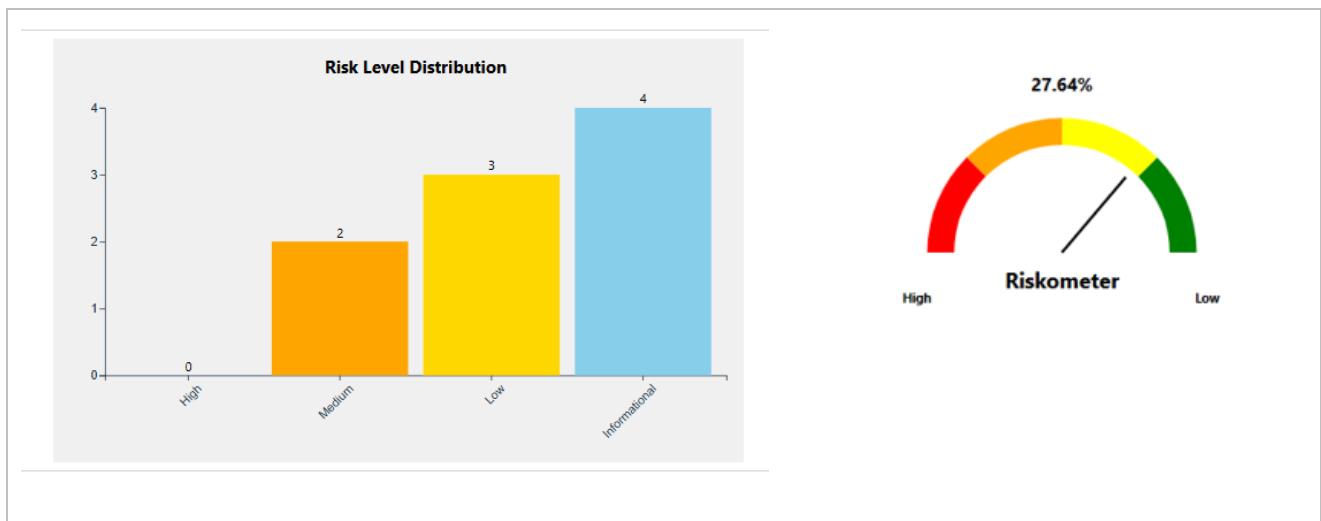
Anhang E: Visualisierungstypen

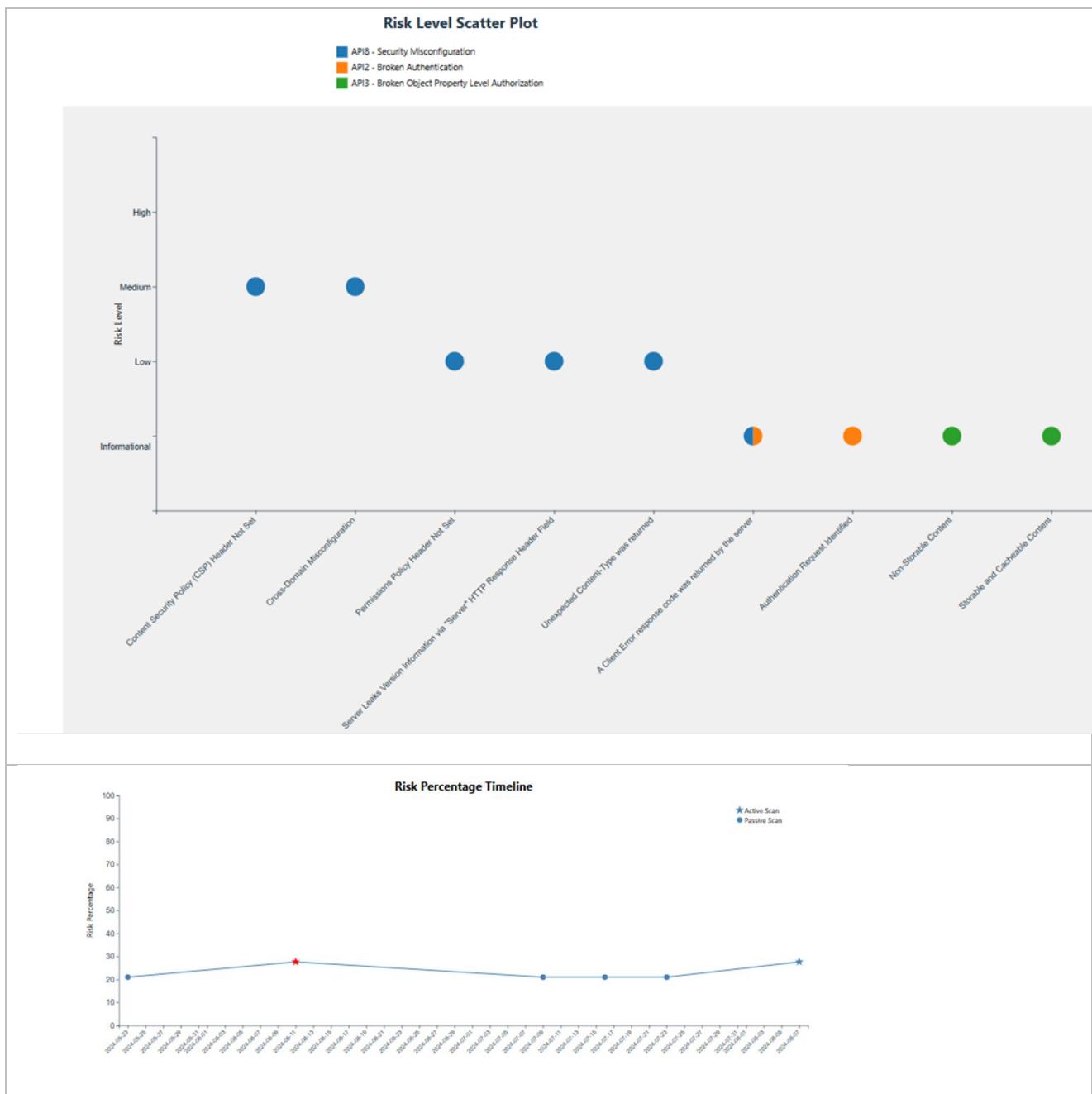
Zeitstrahlbasierte Visualisierungen:



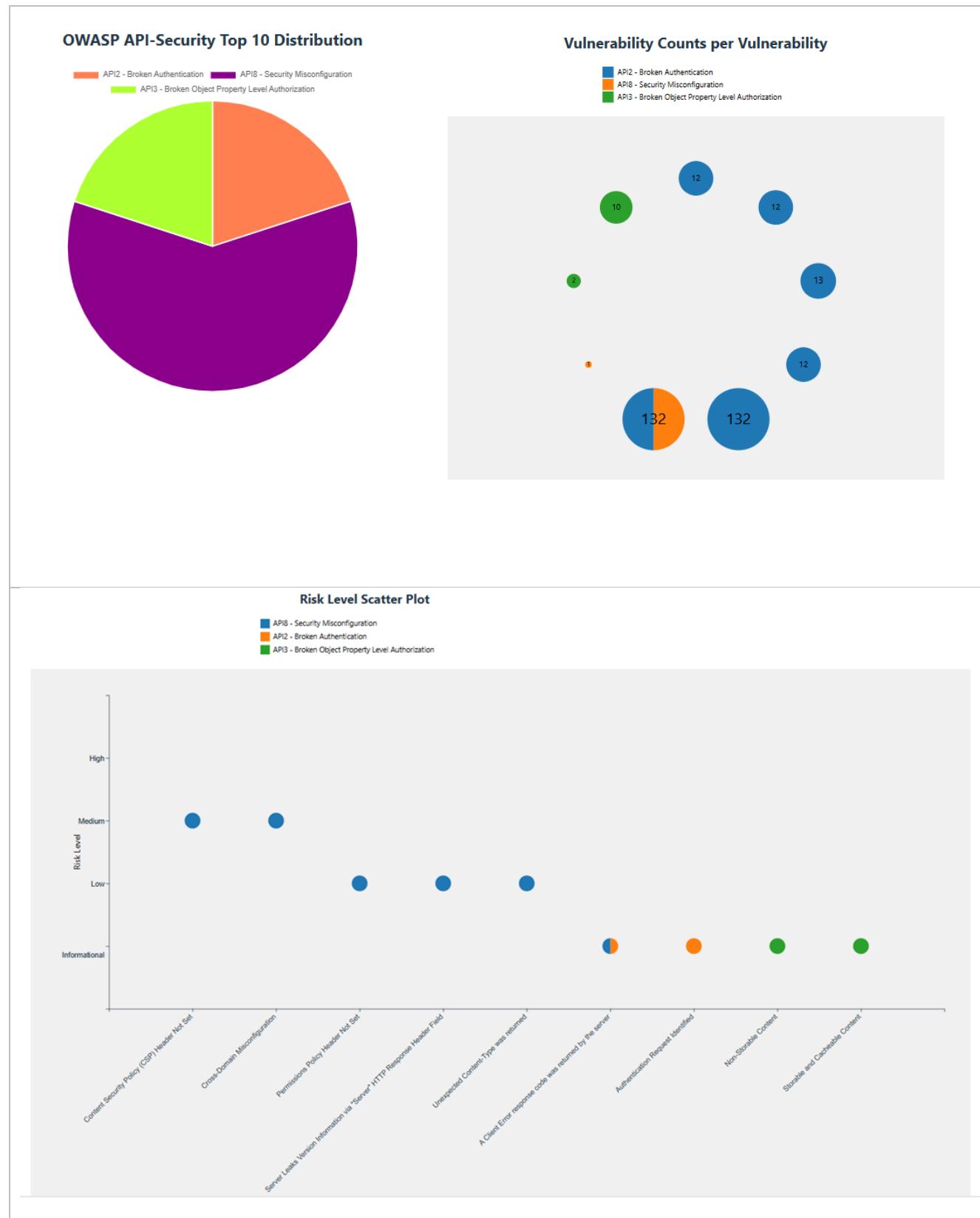


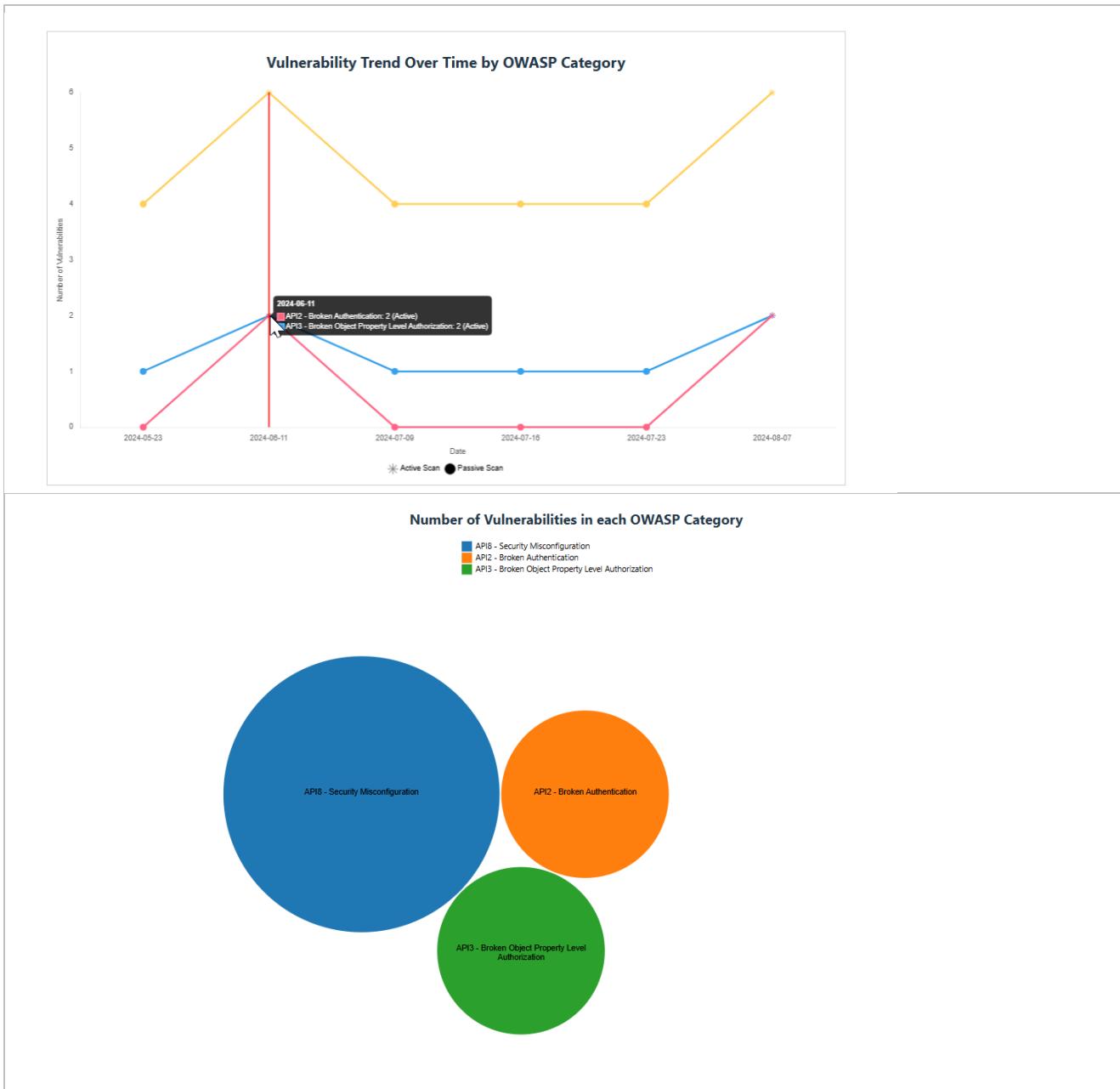
Risikobasierte Visualisierungen:





Visualisierungen basierend auf den Verteilungen der OWASP-Kategorien:





Anhang F: Beispielhaftes Dashboard

Start New API Scan

Active Scan

Keine Datei ausgewählt

Passive Scan

Previous scans

- Sun, 25 Aug 2024 17:22:44 GMT (<http://host.docker.internal:8888>)
- Sun, 18 Aug 2024 20:19:02 GMT (<https://www.example.com>)
- Sat, 17 Aug 2024 20:01:51 GMT (<https://www.example.com>)
- Wed, 07 Aug 2024 14:16:02 GMT (<http://host.docker.internal:5000>)
- Tue, 23 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 16 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 09 Jul 2024 19:30:00 GMT (<http://host.docker.internal:5000>)
- Tue, 11 Jun 2024 14:13:24 GMT (<http://host.docker.internal:5000>)
- Thu, 23 May 2024 19:58:53 GMT (<http://host.docker.internal:5000>)
- Sat, 27 Jul 2024 19:20:53 GMT (<https://www.example.com>)
- Wed, 17 Jul 2024 19:19:53 GMT (<https://www.example.com>)

[Customize](#)

[Go Back to Scan Page](#)

Found Vulnerabilities

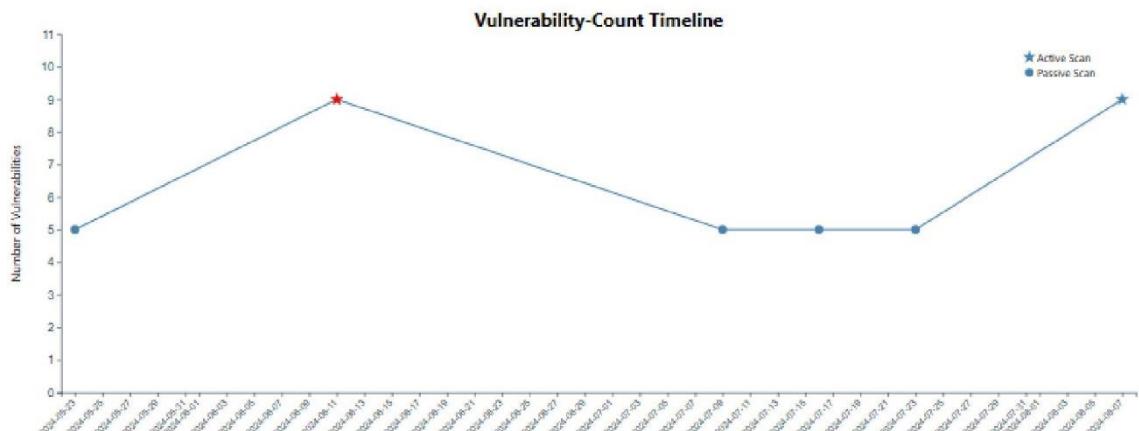
Quick Overview	
Scan-Date:	Tue, 11 Jun 2024 14:13:24 GMT
Scan-URL:	http://host.docker.internal:5000
Scan Type:	Active
Overall Vulnerability Count:	9
Number One Category:	API8 - Security Misconfiguration
Most Severe Vulnerability:	Content Security Policy (CSP) Header Not Set, Cross-Domain Misconfiguration

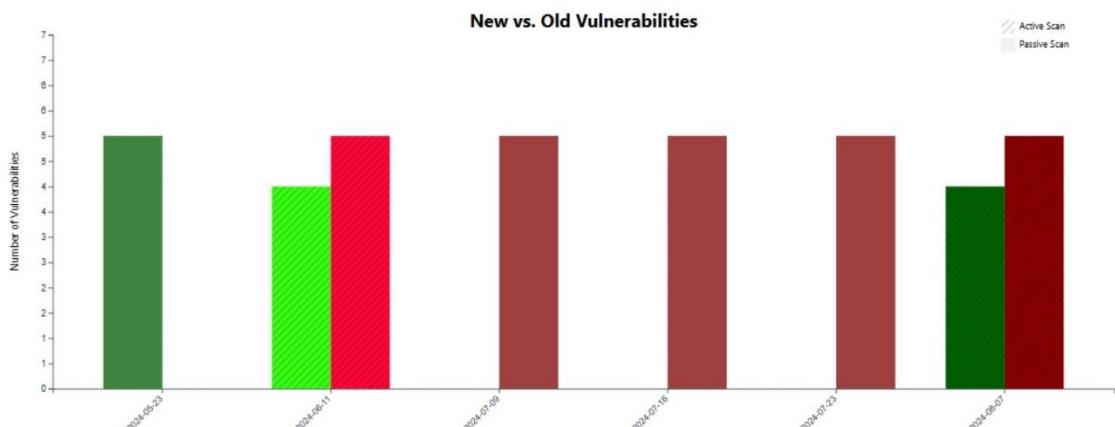
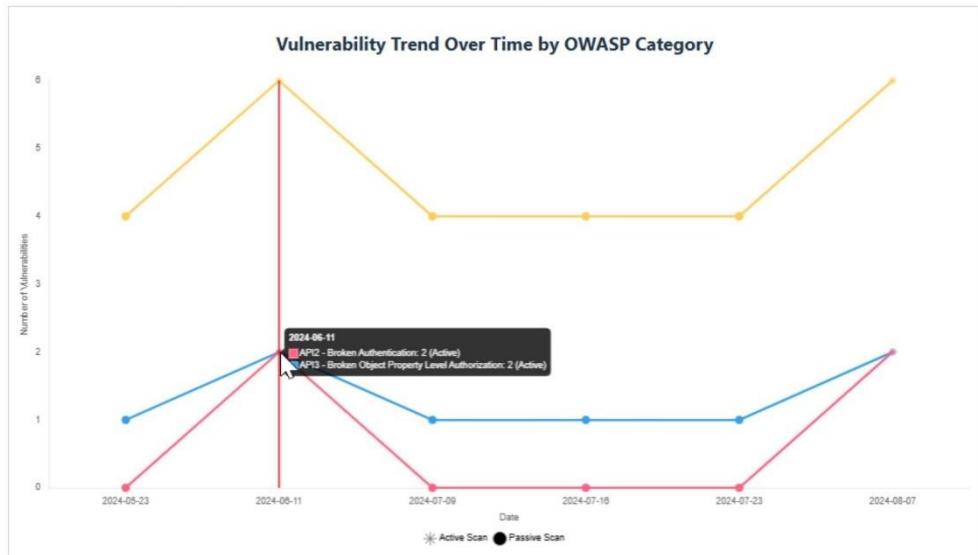
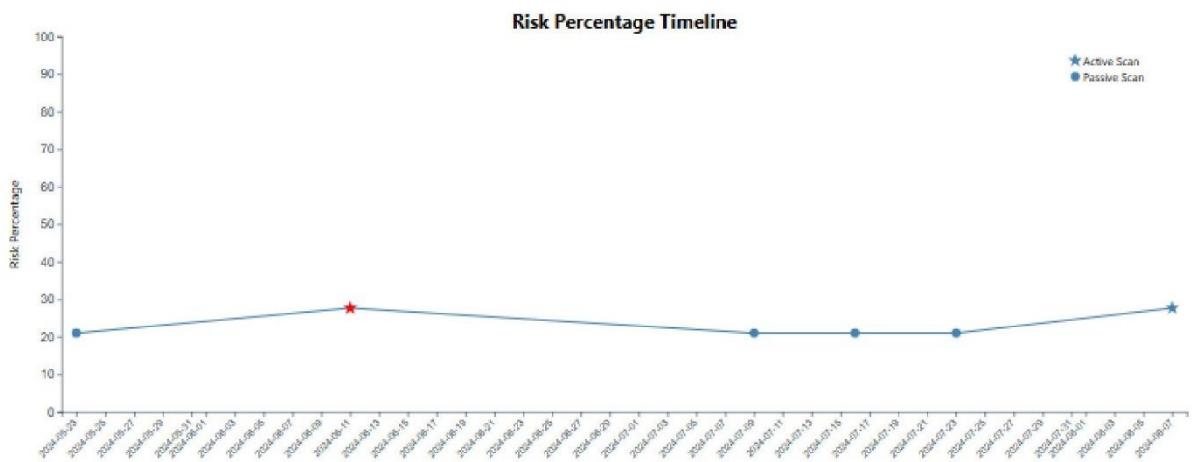


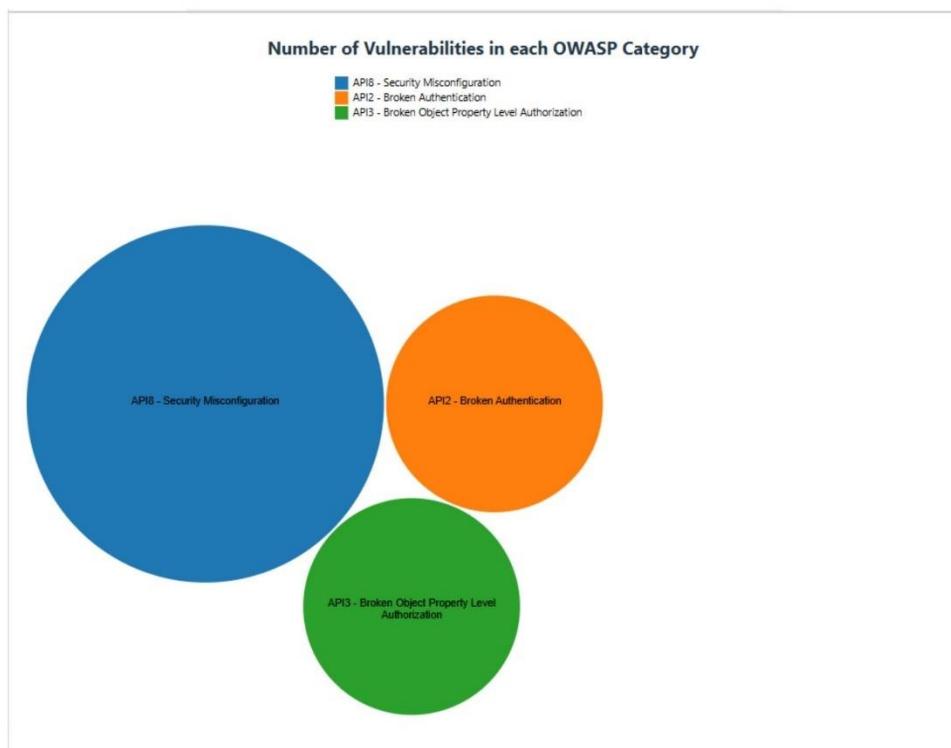
Vulnerability ID	Vulnerability Name	Vulnerability Count	Risk Level	OWASP Category	Description
216	Content Security Policy (CSP) Header Not Set	12	Medium	API8 - Security Misconfiguration	View
217	Cross-Domain Misconfiguration	12	Medium	API8 - Security Misconfiguration	View
218	Permissions Policy Header Not Set	13	Low	API8 - Security Misconfiguration	View
219	Server Leaks Version Information via "Server" HTTP Response Header Field	12	Low	API8 - Security Misconfiguration	View
220	Unexpected Content-Type was returned NEW	132	Low	API8 - Security Misconfiguration	View
221	A Client Error response code was returned by the server NEW	132	Informational	API2 - Broken Authentication	View
222	Authentication Request Identified NEW	1	Informational	API2 - Broken Authentication	View
223	Non-Storable Content NEW	2	Informational	API3 - Broken Object Property Level Authorization	View
224	Storable and Cacheable Content	10	Informational	API3 - Broken Object Property Level Authorization	View

Disclaimer: A vulnerability is labeled as NEW when it hasn't occurred in a scan in the span of one month.

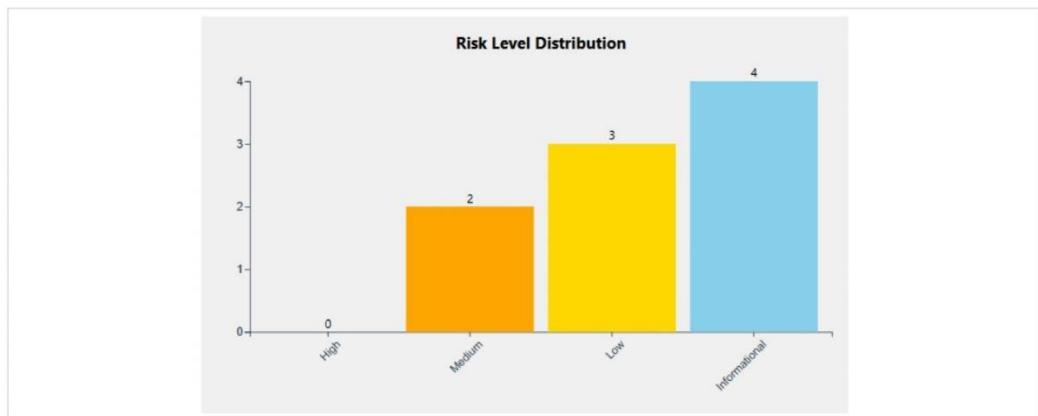
Timelines



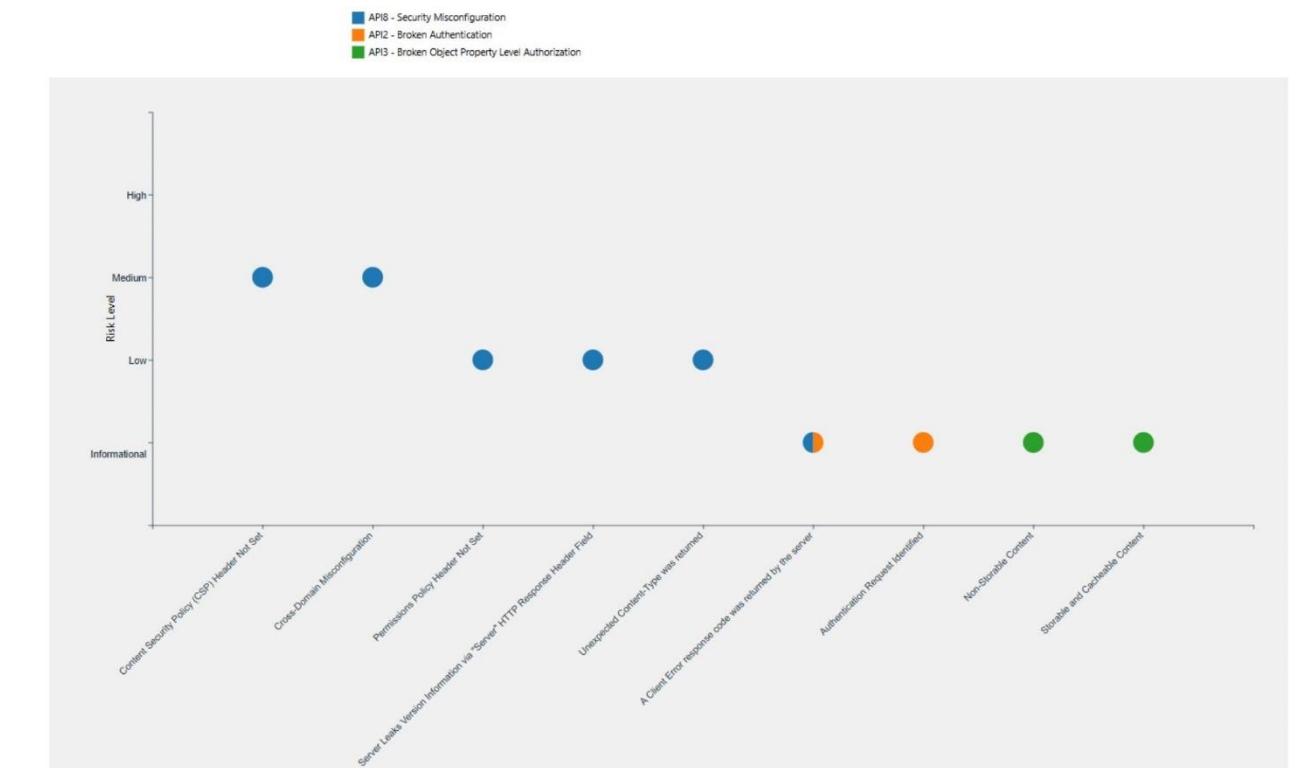




Risklevels



Risk Level Scatter Plot



[Go Back to Scan Page](#)

Customisation

Riskometer

- API1 - Broken Object Level Authorization
- API2 - Broken Authentication
- API3 - Broken Object Property Level Authorization
- API4 - Unrestricted Resource Consumption
- API5 - Broken Function Level Authorization
- API6 - Unrestricted Access to Sensitive Business Flows
- API7 - Server Side Request Forgery
- API8 - Security Misconfiguration
- API9 - Improper Inventory Management
- API10 - Unsafe Consumption of APIs

Total weight used: 100 / 100

[Save Changes](#)

[Reset to Default](#)