Masterarbeit

Sascha Kriebitzsch

Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Sascha Kriebitzsch

Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 19. August 2022

Sascha Kriebitzsch

Thema der Arbeit

Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen

Stichworte

Schwachstellenscanner für Webanwendungen, Benchmark-Anwendungen, OWASP Top 10

Kurzzusammenfassung

Diese vorliegende Arbeit beschäftigt sich mit dem Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen. Anhand der ermittelten Ergebnisse werden Empfehlungen für die Verwendung der effektivsten Scanner in den ausgewählten Schwachstellenkategorien gegeben sowie Hinweise für die erfolgreiche Durchführung von Scans erläutert. Der Vergleich wird dabei mit den Scannern Wapiti, Arachni und OWASP ZAP mit Hilfe der Benchmark-Anwendungen WAVSEP und OWASP durchgeführt. Für die Auswertung werden die Schwachstellenkategorien SQL Injection, XSS und Path Traversal herangezogen. Definierte Kriterien ermöglichen dabei die Bewertung der Ergebnisse. Der Scanner OWASP ZAP hat sich in dieser Arbeit am wirkungsvollsten beim Erkennen von Schwachstellen in den betrachteten Kategorien erwiesen. Für eine effektivere Detektion wird zudem empfohlen, nur eine Kategorie von Schwachstellen und nur ein Teil der verfügbaren URLs einer Webanwendung je Scan zu untersuchen sowie Scans mit weniger Anfragen pro Sekunde durchzuführen.

Sascha Kriebitzsch

Title of Thesis

Comparison of Open Source Scanners for detecting vulnerabilities in Web Applications

Keywords

Web Vulnerability Scanner, Benchmark Applications, OWASP Top 10

Abstract

This thesis compares open source web vulnerability scanners. Based on the results obtained, recommendations for using the most effective scanners in the selected vulnerability categories are provided, as well as advice on how to successfully perform scans. The comparison is made with the scanners Wapiti, Arachni and OWASP ZAP using the benchmark applications WAVSEP and OWASP. The vulnerability categories SQL Injection, XSS and Path Traversal are used for the evaluation. Defined criteria enable the evaluation of the results. In this work, OWASP ZAP proved to be the most effective in detecting vulnerabilities in the considered categories. For more effective detection, it is also recommended to examine only one category of vulnerabilities and only a subset of the available URLs of a web application per scan, as well as to perform scans with fewer requests per second.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Tabellenverzeichnis **viii**

Listings **ix**

1 Einleitung **1**
1.1 Motivation 1
1.2 Zielsetzung 3
1.3 Struktur der Arbeit 3

2 Schwachstellen in Webanwendungen **5**
2.1 Common Weakness Enumeration 5
2.2 OWASP Top 10 5
2.3 Schwachstellenscanner für Webanwendungen 11
2.4 Verwundbare Webanwendungen 13

3 Kriterien für die Bewertung von Schwachstellenscannern **15**

4 Versuchsumgebung für Schwachstellenscanner **18**
4.1 Auswahl Tools 18
4.2 Übersicht ausgewählter Tools 19
4.2.1 OWASP Benchmark 19
4.2.2 The Web Application Vulnerability Scanner Evaluation Project . . 21
4.2.3 OWASP Zed Attack Proxy 22
4.2.4 Arachni 23
4.2.5 Wapiti 25
4.3 Versuchsaufbau 26
4.4 Versuchsdurchführung 26

5 Auswertung **29**
5.1 Ergebnisse der Scanner WAVSEP 29
5.1.1 Wapiti 30
5.1.2 OWASP ZAP 31
5.1.3 Arachni 32
5.1.4 Vergleich der Ergebnisse WAVSEP 33
5.2 Ergebnisse der Scanner OWASP Benchmark 34
5.2.1 Wapiti 35
5.2.2 OWASP ZAP 36
5.2.3 Arachni 37
5.2.4 Vergleich der Ergebnisse OWASP Benchmark 38
5.3 Vergleich der Ergebnisse WAVSEP vs. OWASP Benchmark 39
5.3.1 SQL Injection 39
5.3.2 XSS 40
5.3.3 Path Traversal 41
5.4 Diskussion der Ergebnisse 42

6 Empfehlungen **47**

7 Zusammenfassung **50**
7.1 Fazit 50
7.2 Ausblick 52

Literaturverzeichnis **53**

Selbstständigkeitserklärung **57**

Abbildungsverzeichnis

2.1 Veränderungen der Top 10 von 2017 nach 2021 [29] 11

4.1 OWASP Benchmark Ergebnisse Interpretationsleitfaden [7] 20
4.2 OWASP ZAP Benutzeroberfläche 23
4.3 Arachni Webinterface 24
4.4 Arachni Check sql_injection 24
4.5 Ausführungsoptionen Wapiti [37] 25
4.6 Wapiti Modules 25
4.7 Aufbau der Testumgebung 26

5.1 WAVSEP Übersicht Testfälle 29
5.2 Wapiti Testfall SQL Injection 30
5.3 Wapiti Ergebnisse WAVSEP 30
5.4 OWASP ZAP Testfall XSS 31
5.5 OWASP ZAP Ergebnisse WAVSEP 32
5.6 Arachni Testfall SQL Injection 32
5.7 Arachni Ergebnisse WAVSEP 33
5.8 Vergleich TPR Ergebnisse Scanner WAVSEP 34
5.9 OWASP Benchmark Übersicht Kategorien 35
5.10 Wapiti Ergebnisse OWASP Benchmark 35
5.11 ZAP Ergebnisse OWASP Benchmark 36
5.12 Arachni Ergebnisse OWASP Benchmark 37
5.13 Vergleich TPR Ergebnisse OWASP Benchmark 38
5.14 Vergleich TPR Benchmark Ergebnisse SQL Injection 40
5.15 Vergleich TPR Benchmark Ergebnisse XSS 41
5.16 Vergleich TPR Benchmark Ergebnisse Path Traversal 42

Tabellenverzeichnis

4.1 Schwachstellen OWASP Benchmark nach Kategorien 19
4.2 Schwachstellen WAVSEP nach Kategorien 22

5.1 Wapiti Ergebnisse WAVSEP 31
5.2 OWASP ZAP Ergebnisse WAVSEP 31
5.3 Arachni Ergebnisse WAVSEP 33
5.4 Wapiti Ergebnisse OWASP Benchmark 36
5.5 ZAP Ergebnisse OWASP Benchmark 37
5.6 Arachni Ergebnisse OWASP Benchmark 37
5.7 OWASP Benchmark Vergleich Ergebnisse mit anderen Untersuchungen . . 44
5.8 WAVSEP Vergleich Ergebnisse mit anderen Untersuchungen 45

6.1 Bewertung Scanner in der Kategorie SQL Injection 47
6.2 Bewertung Scanner in der Kategorie XSS 48
6.3 Bewertung Scanner in der Kategorie Path Traversal 49

Listings

4.1 Arachni CLI-Befehl zur Auflistung der Checks 24
4.2 Wapiti CLI-Befehl zur Auflistung der Module 25
4.3 Wapiti Scan-Parameter OWASP Benchmark 27
4.4 Arachni SQL Scan-Parameter OWASP Benchmark 27
4.5 Arachni Reporter 27
4.6 Wapiti Scan-Parameter WAVSEP 28
4.7 Arachni SQL Scan-Parameter WAVSEP 28

1 Einleitung

1.1 Motivation

In den letzten Jahren hat sich der Schutz von IT-Systemen in Unternehmen zum zentralen Baustein für das betriebliche Kontinuitätsmanagement entwickelt. Über Firewalls, IDS (Intrusion Detection System), SIEM (Security Information und Event Management) und vielen weiteren Systemen werden IT-Infrastrukturen überwacht und geschützt. Dabei werden die Kontaktpunkte zu den Systemen, die Webanwendungen, aufgrund von erhöhtem Zeitdruck und schnellerer Softwareentwicklungsprozesse oft vernachlässigt. Es ist somit nicht unüblich, dass ein Netzwerk, welches mit hohem Aufwand gesichert und überwacht wird, durch eine anfällige Webanwendung zu einem Einfallstor für Angreifer werden kann. Um das Risiko eines Sicherheitsvorfalls durch eine verwundbare Webanwendung so gering wie möglich zu halten, sollte diese auf Sicherheitslücken untersucht und abgesichert werden. Denn wenn ein Sicherheitsvorfall eintritt, kann es neben einem hohen Wiederherstellungsaufwand auch zu einem wirtschaftlichen Schaden kommen, da in einem solchen Fall das Unternehmen nur eingeschränkt oder gar nicht geschäftsfähig ist. Der Ruf des Unternehmens kann dadurch auch langfristig Schaden nehmen [33].

Laut dem National Institute of Standards and Technology (NIST) wurden 2020 18.103 Schwachstellen in Computersystemen gemeldet. Von diesen konnten 68% ohne eine Benutzerinteraktion und 70% aus der Ferne ausgenutzt werden [1]. In dem Data Breach Investigations Report von Verizon aus 2019 waren 43% von 3.950 Vorfällen, bei denen es zur Offenlegung von sensiblen Informationen kam, auf verwundbare Webanwendungen zurückzuführen [5]. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) bestätigt in der aktuellen Publikation „Die Lage der IT-Sicherheit in Deutschland 2021“ die Gefahr von gravierenden Schwachstellen in Webanwendungen [4]. Als Beispiel wurden hierbei Schwachstellen bei Webanwendungen von Corona-Testzentren genannt, wodurch sensible Daten wie Testergebnisse und Anschriften über das Internet einsehbar waren.

Aufgrund solcher Gefahren sollten Webanwendungen regelmäßig auf Schwachstellen untersucht werden. Neben statischer Codeanalyse sind dynamische Sicherheitstests die wichtigsten Ansätze, die zur Aufdeckung von Schwachstellen verwendet werden [14]. Für die Durchführung solcher Tests stehen eine Reihe kommerzieller und freier, nachstehend Open Source genannter, Software zur Verfügung. Damit Sicherheitsverantwortliche die Entscheidung treffen können, welcher Scanner am effektivsten Schwachstellen erkennt, muss dies untersucht werden.

Um einen Vergleich von Scannern möglich zu machen, können Scans gegen Benchmark-Anwendungen durchgeführt werden. Solche Anwendungen enthalten absichtlich eine Reihe von Sicherheitslücken in verschiedenen Kategorien, wie Cross-Site Scripting (XSS) oder SQL Injection. Einige bringen zusätzlich negative Testfälle mit, also Testfälle die keine Schwachstellen enthalten. Diese sollen von den Scannern erkannt bzw. nicht erkannt werden. Anhand der Ergebnisse der Scans, die mit Hilfe von Metriken bewertet werden, kann so ein Vergleich stattfinden. Einige Anwendungen sind extra für die Evaluation von Schwachstellenscanner entwickelt und andere eher, um eine Plattform für Sicherheitstrainings bereitzustellen. Bekannte Anwendungen sind u. a. die Damn Vulnerable Web Application (DVWA), das Open Web Application Security Project (OWASP) Mutillidae, die Buggy Web Application (bWAPP), das OWASP Benchmark Project oder das Web Application Vulnerability Scanner Evaluation Project (WAVSEP).

In verschiedenen Studien wurden kommerzielle und Open Source Schwachstellenscanner mit unterschiedliche Benchmark-Anwendungen evaluiert. Makino und Klyuev haben in ihrer Untersuchung die Open Source Scanner OWASP ZAP und Skipfish untersucht [23]. Dabei wurden Scans gegen die Anwendungen DVWA und WAVSEP durchgeführt. In der Arbeit von Idrissi et al. sind eine ganze Reihe von kommerziellen sowie Open Source Scannern mit Hilfe von WAVSEP verglichen worden [22]. Amankwah et al. haben kommerzielle sowie Open Source Scanner benutzt und Untersuchungen gegen WebGoat und DVWA durchgeführt [15]. Al Anhar et al. haben Scanner gegen die NodeJS-basierte Benchmark-Anwendungen DVNA und NodeGoat getestet [13].

Darüber hinaus existieren weitere Studien, die die Performance von Schwachstellenscannern bewertet haben. [14, 25, 24, 34, 38, 32]

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es einen systematischen Vergleich von Open Source Scanner zum Auffinden von Schwachstellen in Webanwendungen durchzuführen.

Dafür werden Scanner sowie Benchmark-Anwendungen gesichtet und vorgestellt. Anschließend werden Probedurchläufe mit den vorgestellten Anwendungen durchgeführt und die Entscheidungen auf die in dieser Arbeit ausgewählten Tools erläutert. Um einen Vergleich durchführen zu können, müssen Kategorien von Schwachstellen in den Benchmark-Anwendungen identifiziert werden, die sich dafür eignen. Zudem müssen Kriterien für die Bewertung von Scannern definiert werden. Danach können Scans gegen die ausgewählten Kategorien durchgeführt und nach den definierten Kriterien ausgewertet werden. Daraus soll erkannt werden, welcher Scanner sich bei der Erkennung der verschiedenen Typen von Schwachstellen am besten eignet. Um die Ergebnisse einzuordnen, werden bei der Auswertung Resultate anderer Studien herangezogen. Es existieren bereits einige Untersuchungen, die Schwachstellenscanner für Webanwendungen vergleichen und bewerten. Allerdings liegen keine Studien vor, welche die drei Scanner Wapiti, OWASP ZAP und Arachni gegen OWASP Benchmark und WAVSEP evaluieren. Daher soll dies in der vorliegenden Arbeit erfolgen. Da diese Scanner sehr populär sind, oft eingesetzt und auch aktiv gepflegt werden, bietet sich ein Vergleich untereinander an, um den „Besten“ dieser Scanner zu identifizieren. Zudem eignen sich als Grundlage der Bewertung die ausgewählten Benchmark-Anwendungen, weil diese extra für den Vergleich von Schwachstellenscannern entwickelt wurden und so beim Bestimmen der Stärken und Schwächen der Scanner helfen können.

Ein weiteres Ziel ist das Definieren von Empfehlungen. Mit deren Hilfe sollen Sicherheitsverantwortliche den passenden Scanner für die jeweilige Kategorie von Schwachstellen identifizieren können. Darüber hinaus sollen die Empfehlungen Sicherheitsverantwortlichen bei der Durchführung der Scans unterstützen.

1.3 Struktur der Arbeit

Nach der Motivation und der Zielsetzung in Kapitel 1 werden in Kapitel 2 die OWASP Top 10, eine Sammlung der kritischsten Sicherheitsrisiken für Webanwendungen, verschiedene Scanner und verwundbare Webanwendungen sowie weitere Begrifflichkeiten zu Schwachstellen vorgestellt.

Kapitel 3 erläutert Kriterien für die Bewertung von Schwachstellenscannern.

Der Versuchsaufbau sowie die Auswahl der Tools und die Versuchsdurchführung werden in Kapitel 4 aufgezeigt.

Ergebnisse der Scans werden in Kapitel 5 zusammengefasst. Dabei werden die Resultate der Scanner in den einzelnen Kategorien gegenübergestellt. Außerdem werden die Ergebnisse eingeordnet und mit anderen Studien verglichen.

Empfehlungen für die Auswahl des richtigen Scanners für die jeweilige Kategorie von Schwachstellen werden in Kapitel 6 abgegeben. Zudem werden Parameter für die erfolgreiche Durchführung von Scans erläutert.

Zuletzt wird die Arbeit zusammengefasst und mit einem Fazit sowie einem Ausblick in Kapitel 7 abgeschlossen.

2 Schwachstellen in Webanwendungen

In diesem Kapitel wird auf die Klassifizierung von Schwachstellentypen eingegangen sowie die OWASP TOP 10 erläutert. Zudem wird eine Auswahl von Schwachstellenscannern und verwundbaren Webanwendungen vorgestellt.

2.1 Common Weakness Enumeration

Die Common Weakness Enumeration (CWE) ist eine gemeinschaftlich gepflegte Liste von Schwachstellen in Software und Hardware. Während Schwachstellen standardisiert durch Common Vulnerabilities and Exposures (CVE) beschrieben und mit dem Common Vulnerability Scoring System (CVSS) in ihrer Schwere bewertet werden, klassifizieren CWEs Typen von Schwachstellen. CWEs werden dabei im Format ‚CWE-IDXY' dargestellt. Beispielsweise versteckt sich hinter dem Bezeichner CWE-863 (Incorrect Authorization), dass eine Software zwar eine Berechtigungsprüfung beim Zugriff auf eine Ressource durchführt, dies aber nicht korrekt passiert, wodurch ein Angreifer diese Beschränkung umgehen kann und dadurch Zugriff auf die Ressource erlangt [3]. Das Ziel von CWEs ist, dass alle Personen, die bei der Entwicklung von Software oder Hardware eingebunden sind, u. a. Programmierer, Architekten oder Designer, für das Thema „Sicherheit" sensibilisiert werden, damit Sicherheitslücken vermieden bzw. beseitigt werden, bevor ein Produkt auf den Markt kommt.

2.2 OWASP Top 10

Das OWASP ist eine gemeinnützige Organisation, die sich für die Verbesserung der Sicherheit von Software im Internet einsetzt und eine Reihe von Projekten unterstützt und vorantreibt [29]. Ihr wohl bekanntestes Projekt ist die OWASP Top 10, bei dem es sich um ein von Sicherheitsexperten aus aller Welt zusammengestelltes Dokument handelt,

das die zehn kritischsten Sicherheitsrisiken für Webanwendungen sowie Empfehlungen zu deren Behebung beinhaltet. Dies soll Unternehmen bei der Absicherung von Webanwendungen helfen und zur Sensibilisierung von Entwicklern dienen. Die OWASP Top 10 werden alle drei bis vier Jahre veröffentlicht, wobei die aktuelle Version aus dem Jahr 2021 ist.

Zur Bestimmung der Rangliste der Top 10 werden Daten von Datenlieferanten mit einer Spezialisierung auf Anwendungssicherheit herangezogen. Im Gegensatz zu den Top 10 aus dem Jahr 2017 wurden Änderungen bei der Bestimmung der Rangliste vorgenommen. In der aktuellen Version berücksichtigte das OWASP-Team statt 30 CWEs ca. 400 CWEs, die dann Kategorien von Sicherheitsrisiken zugeordnet wurden. Im Durchschnitt besitzen die Kategorien 19,6 CWEs. Dabei enthält die Kategorie *A10:2021-Server-Side Request Forgery (SSRF)* nur eine CWE und *A04:2021-Insecure Design* 40 CWEs. Für die weitere Bestimmung wurden mit einem Tool CVEs, denen eine CWE zugeordnet ist, aus der National Vulnerability Database (NVD) einer Schwachstellen-Datenbank vom National Institute of Standards and Technology extrahiert. Anschließend wurde aus dem CVSS-Score der jeweiligen CVEs die Scores der Teilmetriken *Exploit* und *Impact* extrahiert und deren Durchschnitt bestimmt. Dabei musste beachtetet werden, dass einige Schwachstellen mit einem CVSS-Score in Version v2 und andere in v3 bewertet sind. Da dadurch abweichende Werte zustande kommen, wurde eine Gewichtung berücksichtigt, um die abweichenden Versionsunterschiede auszugleichen.

Weitere Faktoren, die bei der Berechnung berücksichtigt wurden sowie eine detaillierte Erklärung, sind auf der offiziellen Webseite zu finden [29].

Nachstehend werden die Kategorien der OWASP Top 10 erläutert und jeweils Beispiele für CWEs aufgeführt.

1. **A01:2021-Broken Access Control**
   Eine Zugriffskontrolle soll dafür sorgen, dass ein Benutzer nur innerhalb seiner gegebenen Berechtigungen handeln kann. Die häufigsten Sicherheitsrisiken zum Umgehen von Zugriffskontrollen sind beispielsweise die Änderungen an der URL, dem internen Anwendungsstatus oder der HTML-Webseite sowie durch die Verwendung von Werkzeugen zur Manipulation von API-Anfragen. Auch die Verletzung des Least-Privilege-Prinzips ist oft ein Problem, wobei Benutzer nur auf die von ihnen benötigten Ressourcen Zugriff haben sollten. Weitere Risiken sind die Erweiterung von Privilegien, so dass ein nicht authentifizierter Anwender als Benutzer oder Benutzer als Administrator handeln kann, sowie das Erzwingen des Aufrufens von

Seiten als nicht oder nicht ausreichend authentifizierter Benutzer. Der Zugriff auf ein anderes Benutzerkonto über Insecure Direct Object References ist ebenfalls verbreitet, wodurch über eine eindeutige URL auf das Konto zugegriffen werden kann. Genauso wie der unbeschränkte Zugriff über APIs auf POST-, PUT- und DELETE-Methoden oder eine fehlerhafte Konfiguration von Cross-Origin Resource Sharing, wodurch API-Zugriffe von nicht autorisierter bzw. nicht vertrauenswürdiger Herkunft möglich sind. Als letzter Punkt ist die Manipulation von Metadaten, wie die eines JSON Web Tokens, Zugriffskontrolltokens, Cookies oder versteckten Feldes zu nennen, womit Rechte erweitert werden können. Unzureichende Zugriffskontrollen (CWE-284) und unzureichende Beschränkungen eines Pfadnamens auf ein eingeschränktes Verzeichnis (Path Traversal) (CWE-22) sind klassische Schwachstellen dieser Kategorie [28, 29].

2. **A02:2021-Cryptographic Failures**
   Hierbei geht es um den Schutzbedarf von Daten sowohl bei der Speicherung sowie bei der Übertragung. Passwörter, Kreditkartendaten, Gesundheitsdaten, persönliche Informationen und Geschäftsgeheimnisse benötigen dabei einen besonderen Schutz, speziell wenn Datenschutzgesetze, wie z.B. die Datenschutz-Grundverordnung eingehalten werden müssen. Schwachstellen dieser Kategorie sind u. a. die schwache Kodierung eines Passworts (CWE-261) sowie die Übertragung von sensiblen Informationen im Klartext (CWE-319). Dabei müssen die folgenden Fragen geklärt werden: Findet die Speicherung von sensiblen Daten im Klartext statt? Welche Übertragungsprotokolle werden verwendet (HTTP, SMTP, FTP)? Wie werden die Daten intern weitergeleitet (Load-Balancer, Web-Server, Backend)? Welche kryptografischen Verfahren werden verwendet (schwach, alt) [28, 29]?

3. **A03:2021-Injection**
   Bei einer Anwendung, die anfällig für einen Injection-Angriff ist, werden vom Benutzer bereitgestellte Eingaben nicht oder nicht ausreichend bereinigt, geprüft oder gefiltert. Eine weitere Anfälligkeit besteht bei der direkten Verwendung von dynamischen Anfragen oder nicht parametrisierten Aufrufen ohne kontextabhängiges maskieren der Daten. Weiterhin ist eine Anwendung gegen einen Injection-Angriff verwundbar, wenn nicht verhindert wird, dass schadhafte Daten in Object Relational Mapping-Suchparametern für das Abgreifen von vertraulichen Datensätzen genutzt werden können. Auch die Verwendung von schadhaften Daten direkt oder als Teil einer dynamischen SQL-Anfrage oder Befehls stellt ein Einfallstor für einen Injection-Angriff dar. Die unzureichende Neutralisierung von Eingaben bei der Er-

stellung von Webseiten (Cross-Site Scripting) (CWE-79) oder die unzureichende Neutralisierung spezieller Elemente, die in einem SQL-Befehl verwendet werden (SQL Injection) (CWE-89), sind häufige Schwachstellen in dieser Kategorie [28, 29].

4. **A04:2021-Insecure Design**
   In dieser Kategorie werden Schwachstellen zusammengefasst, die auf fehlendes oder unwirksames Kontrolldesign zurückzuführen sind. Dabei ist zu beachten, dass es einen Unterschied zwischen unsicherem Design und unsicherer Implementierung gibt. Ein unsicheres Design bleibt auch durch eine perfekte Implementation anfällig, da Sicherheitskontrollen gegen spezifische Angriffe per Definition nicht geschaffen werden. Dagegen kann ein sicheres Design durch eine fehlerhafte Implementierung Schwachstellen erzeugen. Beispielsweise gehört die Generierung von Fehlermeldungen mit sensiblen Informationen (CWE-209) oder auch die ungeschützte Speicherung von Zugangsdaten (CWE-256) in diese Kategorie. Experten für Anwendungssicherheit sollten den Entwicklungszyklus begleitet, um sicherheits- und datenschutzbezogene Kontrollmechanismen zu bewerten und zu entwerfen. Bei der Verwendung von Bibliotheken sollten solche ausgewählt werden, die auf die Umsetzung von sicheren Entwurfsmustern geachtet haben. Threat Modeling (Bedrohungsmodellierung) sollte für kritische Authentifizierungen, Zugriffskontrollen und wichtige Geschäftprozesse durchgeführt werden. Ebenfalls wichtig ist die Integration von Sicherheitskontrollen bei der Formulierung von Software-Anforderungen (User Story). Der Ressourcenverbrauch sollte für jeden Dienst bzw. Benutzer eingeschränkt sein. Darüber hinaus ist eine Trennung auf der System- und Netzwerkebene je nach Gefährdung und Schutzbedarf sowie eine Trennung von Mandanten auf allen Ebenen anzustreben. Zudem sind Unit- und Integrationstests unverzichtbar, um alle kritischen Abläufe gegen das Bedrohungsmodell zu überprüfen [29].

5. **A05:2021-Security Misconfiguration**
   Sicherheitsrelevante Fehlkonfigurationen sind ebenfalls ein großes Problem und bergen ein hohes Sicherheitsrisiko. Die Verwendung bzw. das nicht ändern von Standardzugangsdaten oder auch aktivierte bzw. installierte, nicht benötigte Funktionen zählen zu dieser Kategorie. Auch die externe Ausgabe von Stacktraces oder internen Fehlermeldungen sollte verhindert werden. Einige Anwendungen aktivieren nach einem Upgrade die Sicherheitsfunktionen nicht oder nicht ausreichend. Das Fehlen einer angemessenen Härtung im Anwendungsstack sowie unzureichend konfigurierte Berechtigungen bei Clouddiensten sind weitere Schwachstellen. Darüber hinaus müssen die Sicherheitseinstellungen in den Anwendungsservern und

Frameworks, Datenbanken sowie Bibliotheken umgesetzt werden. Auch der Einsatz einer veralteten oder nachweislich verwundbaren Software zählt zu dieser Kategorie. Detaillierter wird darauf im nächsten Abschnitt *A06:2021-Vulnerable and Outdated Components* eingegangen. Zugehörige Schwachstellen sind die Konfiguration (CWE-16), die allgemein die Konfiguration von Software beschreibt, und unzureichende Einschränkung von Verweisen auf externe XML-Entitäten (CWE-611) [28, 29].

6. **A06:2021-Vulnerable and Outdated Components**
   Die Nutzung von Komponenten, die entweder bekannte Schwachstellen enthalten oder veraltet sind, ist ein fahrlässiges Risiko, das nicht eingegangen werden sollte. Zudem muss stets bekannt sein, welche Komponenten in welcher Version in der Anwendung benutzt werden. Dies gilt auch für eventuell verschachtelte Abhängigkeiten. Nicht mehr gepflegte Komponenten sollten rechtzeitig ausgetauscht werden. Dazu gehören beispielsweise das Betriebssystem, der Webserver oder Applikationsserver, das Datenbankmanagementsystem, APIs, Bibliotheken sowie die Laufzeitumgebung. Des Weiteren setzt ein fehlendes Patchmanagement ein Unternehmen einer unnötigen Gefahr aus, kompromittiert zu werden. Regelmäßige Schwachstellenscans und die Überwachung von Security Advisories der Hersteller von Komponenten kann das Erkennen von möglichen Bedrohungen verbessern. Die Verwendung von nicht gewarteten Komponenten von Drittanbietern (CWE-1104) sowie allgemein die Verwendung von Komponenten mit bekannten Schwachstellen (CWE-937, CWE-1035) sind hier als Schwachstellen für diese Kategorie zu nennen [28, 29].

7. **A07:2021-Identification and Authentication Failures**
   Diese Kategorie umfasst Probleme bei der Bestätigung der Identität des Benutzers, der Authentifizierung und der Sitzungsverwaltung. Hierbei können in der Anwendung hinterlegte Standardpasswörter oder auch schwache bzw. einfache Kombinationen von Passwörtern einem Angreifer Zugriff gewähren. Weiterhin sollten Brute-Force- und auch automatische Angriffe wie „Credential Stuffing“, also das Anwenden von vorher erbeuten Benutzerdaten, verhindert werden. Die Verwendung von schwachen oder ineffektive Verfahren zur Wiederherstellung von Anmeldeinformationen sollte gleichermaßen vermieden werden. Dazu zählen z. B. wissensbasierte Antworten auf Wiederherstellungsfragen. Wie in *A02:2021-Kryptografiefehler* erläutert, stellt die Speicherung von Passwörtern im Klartext, oder mit einem nicht ausreichend sicheren Hashverfahren, weitere Angriffspunkte dar. Die Offenlegung

der Sitzungskennung in der URL und auch die Wiederverwendung der Sitzungskennung nach erfolgreicher Anmeldung gehören ebenfalls in diese Kategorie. Dazu zählen auch Authentifizierungs-Tokens, die beim Abmelden oder bei Inaktivität nicht ungültig gemacht werden. Der Verzicht oder eine unzureichende Implementation einer Multi-Faktor-Authentifizierung stellt ebenso ein Sicherheitsproblem dar. Die unsachgemäße Authentifizierung (CWE-287) sowie die unsachgemäße Validierung der Zugehörigkeit eines Zertifikat zu einem Host (CWE-287) sind hier hervorzuheben [28, 29].

8. **A08:2021-Software and Data Integrity Failures**
   Unter Fehler bei der Software- und Datenintegrität sind Integritätsverletzungen aufgrund der Verwendung von Plugins, Bibliotheken oder Modulen aus nicht vertrauenswürdigen Quellen, Repositories und Content Delivery Networks zu verstehen. Beispielsweise kann die Einbindung von externen Bibliotheken Angriffsflächen schaffen, welche außerhalb des eigenen Kontrollbereichs liegen (CWE-829). Darüber hinaus kann eine Integritätsverletzung über die eigene Continuous Integration und Continuous Delivery Pipeline einem Angreifer unbefugten Zugriff auf ein System verschaffen oder ein bösartiger Programmcode kann eingeschleust werden. Im schlimmsten Fall kann dies zu einer kompletten Kompromittierung betroffener Systeme führen. Ein weiteres großes Problem sind automatische Updates ohne Integritätsprüfung, wodurch eine zuvor geprüfte Anwendung zur Gefahr wird. Solche Angriffe fallen unter CWE-294 (Herunterladen von Code ohne Integritätsprüfung). Auch die Deserialisierung von Objekten oder Daten in einer Umgebung, die ein Angreifer möglicherweise sehen oder verändern kann, ist eine Problematik, die in diese Kategorie fällt [28, 29].

9. **A09:2021-Security Logging and Monitoring Failures**
   Unzureichendes Logging und Monitoring kann dazu beitragen, Sicherheitsvorfälle nicht oder zu spät zu entdecken, wodurch weitreichende Folgen einhergehen können. Diese Kategorie umfasst aber nicht nur eine unzureichende Protokollierung (CWE-778), sondern auch die unsachgemäße Bereinigung von Protokollausgaben (CWE-117). Die fehlende Protokollierung von Anmeldevorgängen bzw. von fehlgeschlagenen Anmeldungen oder anderen wichtigen Transaktionen kann die Erkennung von Sicherheitsvorfällen verhindern. Darüber hinaus ist die Erzeugung von unzureichenden oder unklaren Logs sowie ein fehlendes zentrales Logmanagement ohne Überwachung ein weiteres Sicherheitsrisiko in diese Kategorie [28, 29].

10. **A10:2021-Server-Side Request Forgery**
    Über einen Server-Side Request Forgery (SSRF)-Angriff (CWE-918) kann eine Webanwendung serverseitig dazu veranlasst werden, im Sinne des Angreifers eine manipulierte HTTP-Anfrage an eine andere Ressource zu stellen. Dabei kann eine Verbindung zu einem beliebigen externen System aber auch zu ausschließlich intern genutzten Systemen hergestellt werden. Dadurch kann ein Angreifer Informationen ausspähen oder sogar beliebigen Programmcode ausführen. Diese Kategorie wurde aufgrund der Erstplatzierung in der Top 10 Community-Umfrage hinzugefügt [29].

Neben diesen zehn Kategorien von Sicherheitsrisiken gibt es natürlich weitere, die nicht vernachlässigt werden sollten. Abbildung 2.1 zeigt die Veränderungen der Top 10 von 2017 bis 2021. Hier ist gut zu erkennen, dass sich die Bedrohungslage ständig im Wandel befindet und dadurch neue Kategorien hinzukommen oder bereits vorhandene zusammengefasst werden.

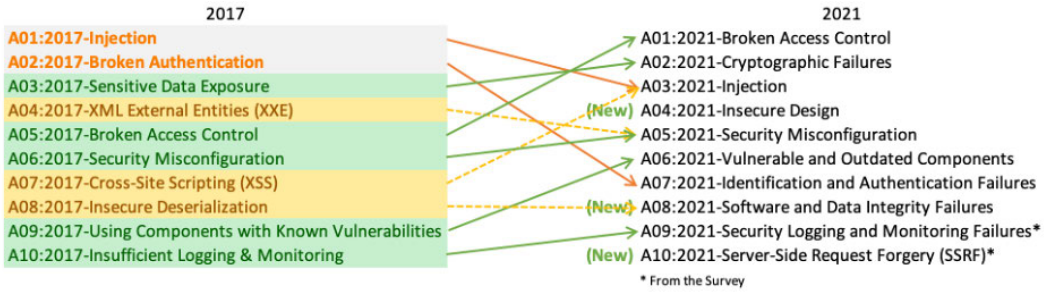

Abbildung 2.1: Veränderungen der Top 10 von 2017 nach 2021 [29]

2.3 Schwachstellenscanner für Webanwendungen

Trotz der Einhaltung von Konzepten und einer umfangreichen Teststrategie können sich Schwachstellen bei der Entwicklung in eine Webanwendung einschleichen. Zudem können Sicherheitslücken in Anwendungen oder abhängigen Bibliotheken zu einem späteren Zeitpunkt entdeckt werden. Zur Identifikation solcher Schwachstellen sollten regelmäßig Schwachstellenscans durchgeführt werden. Neben einer Reihe von kommerziellen Schwachstellenscannern stehen Open Source Produkte zur Verfügung. Diese Anwendungen bringen unterschiedliche Funktionen mit und stehen als Kommandozeilentools, Webanwendungen oder als eigenständige Software zur Verfügung.

Die meisten Schwachstellenscanner für Webanwendungen bestehen aus drei Komponenten. Ein Webcrawler durchsucht die Webanwendung und identifiziert dabei die zugehörigen Webseiten. Eine Angriffskomponente, die die Informationen von den gefunden Seiten aufarbeitet und Anfragen mit den passenden Angriffsvektoren an die Webanwendung sendet. Die dritte Komponente ist für die Auswertung der Antworten der Webanwendung und die Interpretation der Ergebnisse zuständig [15].

Für das Auffinden von Schwachstellen (AST (Application Security Testing)) stehen Werkzeuge bereit, die in drei Gruppen aufgeteilt werden. Tools für statische Codeanalysen (SAST (Static Application Security Testing)) führen sogenannte White-Box-Tests durch. Dabei ist der Quellcode während der Analyse sichtbar und wird dabei vollständig überprüft. SAST-Tools sollten auf den Bedarf der zu testenden Software angepasst werden, da sonst zu viele Stellen als Sicherheitslücke klassifiziert werden, was zu keinen guten Resultaten führt. Zudem wird kein lauffähiger Code benötigt, wodurch diese Art von Tests früh in der Entwicklung eingesetzt werden können [23, 13]. Daneben gibt es Tools für die Durchführung dynamischer Sicherheitstest (DAST (Dynamic Application Security Testing)), die aktiv Angriffe gegen eine laufende Anwendung als Black-Box-Test durchführen. Dabei ist der Quellcode während der Tests nicht sichtbar und es werden nur die erreichbaren Funktionen der laufenden Anwendung getestet. Es treten wenige Fehlalarme auf und es spielt keine Rolle in welcher Programmiersprache die Anwendung entwickelt wurde [30, 29]. Darüber hinaus gibt es noch interaktive Werkzeuge (IAST (Interactive Application Security Testing)), die versuchen eine Mischung aus SAST und DAST abzubilden. Dabei wird das Verhalten in Echtzeit sowohl von innen (SAST) und von außen (DAST) beobachtet [9].

Da in dieser Arbeit verwundbare Anwendungen verschiedener Projekte für den Vergleich der Scanner herangezogen werden und keine genauere Betrachtung des Quellcodes erfolgen soll, bieten sich hier DAST-Tools an. Nachfolgend werden verschiedene Open Source DAST-Tools kurz vorgestellt, welche für diese Untersuchung näher betrachtet werden.

1. **Skipfish** ist ein kommandozeilenbasierter und in C geschriebener Scanner. Er steht für die Plattformen Linux, FreeBSD, macOS und Windows zur Verfügung. Als Ergebnis eines Tests liefert Skipfish eine interaktive Webseite, welche die gefundenen Schwachstellen nach Schwere sortiert anzeigt [10].

2. **Nikto** ist in Perl geschrieben und wurde 2001 erstmals veröffentlicht. Der Report nach einem Scan kann in CSV, HTML, XML und als einfache Textdatei generiert werden. Dabei werden die gefundenen Schwachstellen mit einem Tag aus der Open

Source Vulnerability Database (OSVDB) versehen, welche wiederum auf CVEs passen. Das OSVDB Projekt wird aber seit 2016 nicht weiter betrieben [36].

3. **Vega** ist GUI-basiert und steht für die Plattformen Linux, macOS und Windows zur Verfügung. Der Scanner ist in Java entwickelt und bietet die Möglichkeit, weitere Angriffsmodule selber zu implementieren. Reports nach einem Scan können nur in der Anwendung ausgewertet werden [11].

4. **Wapiti** ist ein in Python geschriebener kommandozeilenbasierter Schwachstellenscanner, der verschiedene Funktionen für die Durchführung der Scans anbietet. Ein Report kann u. a. in die Formate HTML oder XML exportiert werden [37].

5. **Arachni** bietet neben einer Kommandozeile eine Webanwendung für die Durchführung von Scanvorgängen an. Darüber hinaus ist Arachni in Perl entwickelt und besitzt eine Komponente, das sogenannte „Trainer-Subsystem“, wodurch das System während eines Scanvorgangs sich selbständig trainiert, um gezielter Schwachstellen zu erkennen [18].

6. **OWASP ZAP** ist in Java geschrieben und als GUI-basierte Anwendung verfügbar. Es wird von OWASP betreut und entwickelt. ZAP kann auch über die Kommandozeile gestartet und über einen CLI-Client gesteuert werden. Die Auswertung der Ergebnisse kann direkt in der Anwendung erfolgen oder der Report in verschiedene Formate exportiert werden [8].

2.4 Verwundbare Webanwendungen

Damit ein Vergleich von Schwachstellenscannern möglich wird, müssen Scans gegen Webanwendungen durchgeführt werden, welche eine Vielzahl von Schwachstellen aus den gerade vorgestellten OWASP Top 10 enthalten. Denn nur, wenn eine große Anzahl verschiedener Typen von Schwachstellen abgedeckt ist, kann die Performance der Scanner bewertet werden.

Verschiedene Projekte haben solche Anwendungen mit absichtlichen Schwachstellen entworfen. OWASP Mutillidae, Damn Vulnerable Web Application (DVWA), Buggy Web Application (bWAPP), OWASP Benchmark und das Web Application Vulnerability

Scanner Evaluation Project (WAVSEP) sind populäre Vertreter dieser Plattformen. Dabei wurden die Anwendungen für verschiedene Einsatzgebiete optimiert, welche bei der nachfolgenden Vorstellung der einzelnen Tools erläutert werden.

1. **OWASP Mutillidae II** ist eine verwundbare Webanwendung, welche die OWASP TOP 10 Schwachstellen der letzten Jahre enthält. Dieses Projekt wurde mit dem Ziel umgesetzt, eine Plattform für Web-Sicherheitstrainings anzubieten [21, 39].

2. Das **Web Application Vulnerability Scanner Evaluation Project (WAVSEP)** ist eine Webanwendung, die für die Bewertung von Schwachstellenscannern entwickelt wurde. Die Anwendung enthält eine Sammlung ausnutzbarer Sicherheitslücken, die von den Scannern identifiziert werden können [35].

3. Die **Damn Vulnerable Web Application (DVWA)** ist eine verwundbare Webanwendung, die Sicherheitsexperten, Webentwicklern sowie Studenten helfen soll, ihre Fähigkeiten in einer kontrollierten Umgebung zu testen bzw. ein besseres Verständnis bei der Absicherung von Webanwendungen zu bekommen [20].

4. **bWAPP** ist ebenfalls entwickelt wurden, um Sicherheitsexperten, Webentwicklern sowie Studenten eine Plattform zu bieten, in der sie ihre Fähigkeiten zum Auffinden von Sicherheitslücken trainieren können. Über 100 Schwachstellen sind in bWAPP enthalten, die die OWASP Top 10 abdecken [2].

5. Das **OWASP Benchmark Projekt** ist mit dem Hintergrund für die Bewertung von Schwachstellenscannern entwickelt wurden. Die Anwendung enthält tausende von Testfällen, die jeweils bestimmten CWEs zugeordnet sind. Alle Testfälle sind explizit ausnutzbar, was einen Vergleich von Scannern ermöglichen soll [7].

3 Kriterien für die Bewertung von Schwachstellenscannern

Um eine Bewertung der verschiedenen Scanner durchzuführen, müssen Kriterien definiert werden. Diese Kriterien müssen messbar und vergleichbar sein.

Nachdem ein Schwachstellenscan gegen eine Webanwendung durchgeführt wurde, soll mit Hilfe der Kriterien eine Bewertung der Ergebnisse erfolgen. Dafür werden die in diesem Abschnitt vorgestellten Bewertungskriterien aus den einzelnen Scanresultaten erhoben und gegenübergestellt.

Angestrebt wird dabei eine hohe True Positve Rate (TPR) und eine geringe False Positive Rate (FPR). Einen guten Scanner zeichnet aus, dass er eine Vielzahl der vorhandenen Schwachstellen erkennt, wodurch potenziell verwundbare Angriffsflächen geschlossen werden können. Gleichzeitig sollen Fehlalarme vermieden werden, da die Überprüfung jedes einzelnen Alarms einen hohen Arbeitsaufwand bedeutet.

Die folgenden Bewertungskriterien werden für den Vergleich der Schwachstellenscanner betrachtet:

- **True Positve (TP):** Vorhandene Schwachstellen wurden korrekt als Schwachstellen erkannt.
- **False Negative (FN):** Vorhandene Schwachstellen wurden nicht als Schwachstellen erkannt.
- **True Negative (TN):** Richtig erkannt, dass keine Schwachstelle vorliegt.
- **False Positive (FP):** Es wurden fälschlicherweise Schwachstellen erkannt, obwohl keine vorhanden sind.

- **True Positive Rate (TPR):** Gibt die Rate der korrekt gemeldeten vorhandenen Schwachstellen im Verhältnis zu allen vorhandenen Schwachstellen an. Dieser Wert wird auch Recall oder Sensitivität genannt und wie folgt berechnet [17, 7].

$$TPR = \frac{TP}{(TP + FN)} \tag{3.1}$$

- **False Negative Rate (FNR):** Gibt die Rate der nicht korrekt gemeldeten vorhandenen Schwachstellen im Verhältnis zu allen vorhandenen Schwachstellen an [17].

$$FNR = \frac{FN}{(TP + FN)} \tag{3.2}$$

- **False Positive Rate (FPR):** Gibt die Rate der fälschlicherweise als echt gemeldeten negativen Schwachstellen (Fehlalarme) im Verhältnis zu allen negativen Schwachstellen an [17, 7].

$$FPR = \frac{FP}{(FP + TN)} \tag{3.3}$$

- **True Negative Rate (TNR):** Gibt die Rate der korrekt übergangenen Fehlalarme im Verhältnis zu allen negativen Schwachstellen an. Dieser Wert wird auch Spezifität genannt [17].

$$TNR = \frac{TN}{(TN + FP)} \tag{3.4}$$

- **Youden-Index:** Der Youden-Index ist eine Standardmethode, um analytische Tests zu bewerten. Dabei wird ein Wert zwischen -1 und 1 ausgegeben. Wobei ein Wert von 1 eine perfekte Erkennung aller Schwachstellen ohne False Positives bedeutet und ein Wert von -1 nur False Positives und keine tatsächlich erkannten Schwachstellen darstellt. Der erste Teil der Formel ist die True Positive Rate (TPR) oder

auch Sensitivität und der zweite Teil die True Negative Rate (TNR) oder auch Spezifität [7].

$$J = (\frac{TP}{TP+FN}) + (\frac{TN}{TN+FP}) - 1 \tag{3.5}$$

$$J = (Sensitivität + Spezifität) - 1 \tag{3.6}$$

Darüber hinaus gibt es noch weitere Metriken, wie beispielsweise die Präzision (Precision), welche das Verhältnis zwischen korrekt erkannten Schwachstellen und der Anzahl aller erkannten Schwachstellen abbildet oder die Genauigkeit (Accuracy), die das Verhältnis zwischen korrekt klassifizierten Fällen und der Gesamtzahl aller Testfälle darstellt [16, 17]. Eine weitere bekannte Metrik ist das F-Maß (F-Measure), mit dem der harmonische Mittelwert von *Precision* und *Recall* ermittelt wird. Dabei werden die richtig erkannten Schwachstellen, im Gegensatz zu den fehlerhaft erkannten Schwachstellen, doppelt gewichtet [26].

Da in dieser Arbeit zur Bewertung der Scanner ein Fokus auf einer hohen TPR und einer niedrige FPR liegt, werden nur die vorherig genannten Metriken für die Bewertung herangezogen.

4 Versuchsumgebung für Schwachstellenscanner

Die folgenden Abschnitte stellen die ausgewählten Tools für die Versuchsdurchführung sowie ihre Charakteristik vor. Zudem wird der Versuchsaufbau und die Versuchsdurchführung beschrieben.

4.1 Auswahl Tools

Vorab wurden einige Probedurchläufe gestartet, die Aufschluss darüber gegeben haben, welche Schwachstellenscanner und Webanwendungen für die Durchführung der Untersuchung in Frage kommen. Dafür wurden Scans auf die in Abschnitt 2.4 vorgestellten verwundbaren Anwendungen durchgeführt. Die Scans erfolgten dabei mit den Tools aus Abschnitt 2.3.

Die Auswertung der Resultate der Probedurchläufe hat gezeigt, dass ein Vergleich nicht ganz trivial ist. Bei den verwundbaren Webanwendungen bWAPP Bee-Box und OWASP Mutillidae existiert keine genaue Angabe über die Anzahl der vorhandenen Schwachstellen, wodurch ein Vergleich der Scanner zwar untereinander möglich ist, aber das Gesamtbild der erkannten oder fehlerhaft erkannten Schwachstellen nicht abgebildet werden kann. Die DVWA hingegen hat nur eine sehr geringe Anzahl an Schwachstellen [23], womit auch diese nicht für einen Vergleich geeignet ist. Deshalb wurden verwundbare Webanwendungen ausgewählt, die neben einem Überblick über alle Schwachstellen auch eine Vielzahl von Schwachstellen in verschiedenen Kategorien anbieten. Dafür eignen sich am besten OWASP Benchmark und WAVSEP, da beide Benchmark-Anwendungen diese Anforderungen erfüllen [31]. OWASP Benchmark bietet sogar für einige Scanner die Möglichkeit der automatischen Auswertung. Dafür wird der Report eines Scanners über ein Skript ausgewertet und eine detaillierte Übersicht über die erkannten Testfälle erstellt.

Schwachstellen Kategorie	Anzahl Tests	Pos. Tests	Negative Tests	CWE
Command Injection	251	126	125	78
Insecure Cookie	67	36	31	614
LDAP Injection	59	27	32	90
Path Traversal	268	133	135	22
SQL Injection	504	272	232	89
Trust Boundary	126	83	43	501
Weak Encryption Algorithm	246	130	116	327
Weak Hashing Algorithm	236	129	107	328
Weak Randomness	493	218	275	330
XPath Injection	35	15	20	643
XSS	455	246	209	79
Gesamtanzahl an Testfällen	2740			

Tabelle 4.1: Schwachstellen OWASP Benchmark nach Kategorien

Die Wahl der Schwachstellenscanner für die Versuchsdurchführung fiel auf die populären Scanner Wapiti, Arachni und OWASP ZAP. Diese Scanner bieten die Möglichkeit, neben einer grafischen Auswertung in HTML oder in der Anwendung, die Reports im XML-Format zu exportieren. Dies ist sehr nützlich, denn damit kann die Auswertung mit OWASP Benchmark automatisch durchgeführt werden. Auch für eine spätere Einbindung der Scanner in eine existierende Umgebung bietet sich ein Format wie XML an. Skipfish und Vega bieten keine Möglichkeit ihre Reports in ein automatisch verarbeitbares Format zu exportieren, weshalb diese nicht für diese Untersuchung in Frage kommen. Der Scanner Nikto kommt aufgrund der schlechten Ergebnisse aus den Probedurchläufen ebenfalls nicht in Frage. Ein weiterer wichtiger Grund, der für die ausgewählten Scanner spricht ist, dass alle drei aktiv gepflegt und weiter entwickelt werden.

4.2 Übersicht ausgewählter Tools

4.2.1 OWASP Benchmark

> „Das OWASP Benchmark Project ist eine Java-Testsuite zur Bewertung der Genauigkeit, Abdeckung und Geschwindigkeit automatisierter Tools zur Erkennung von Software-Schwachstellen.“ [7]

OWASP Benchmark ist eine Open Source Webanwendung, die in der aktuellen Version 1.2 2.740 Testfälle beinhaltet. Diese Testfälle sind in die Kategorien Command Injection, Weak Cryptography, Weak Hashing, LDAP Injection, Path Traversal, Secure Cookie Flag, SQL Injection, Trust Boundary Violation, Weak Randomness, XPATH Injection und XSS aufgeteilt. Tabelle 4.1 zeigt einen Überblick über alle Testfälle. Jeder Kategorie ist der Typ der Schwachstelle in Form einer CWE zugeordnet. Alle Schwachstellen, die in die Benchmark-Anwendung aufgenommen wurden, sind tatsächlich ausnutzbar, wodurch jeder Schwachstellenscanner die Möglichkeit hat, diese zu erkennen. Dies beinhaltet SAST, DAST und IAST-Tools. Darüber hinaus ist eine Liste mit allen Schwachstellen vorhanden. Diese enthält die Testfälle, die Kategorie, eine CWE und das zu erwartende Ergebnis (True, False).

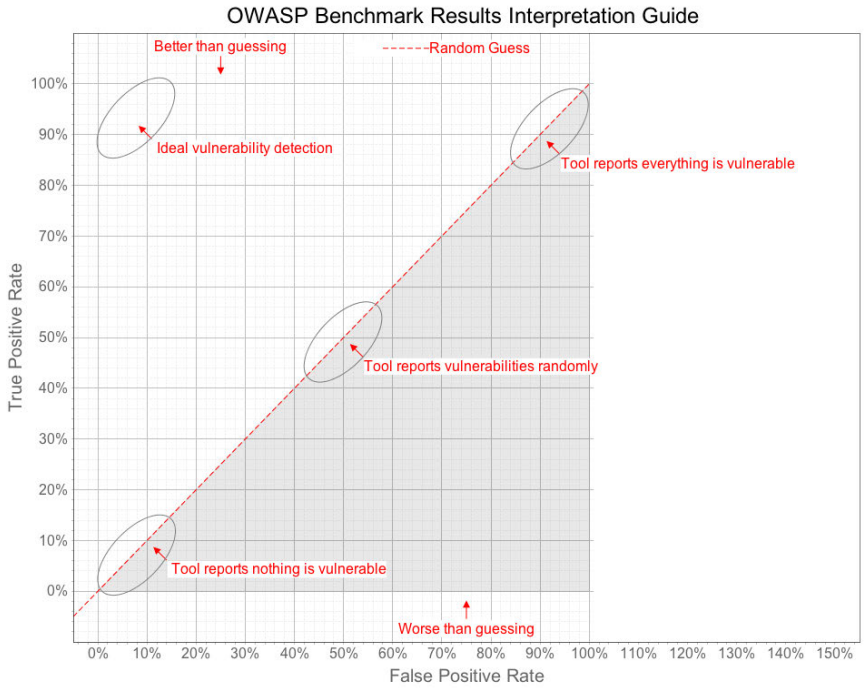

Abbildung 4.1: OWASP Benchmark Ergebnisse Interpretationsleitfaden [7]

Nachdem ein Scan gegen OWASP Benchmark durchgeführt wurde, kann über ein Skript eine Scorecard erstellt werden, welche die Ergebnisse auswertet und einen Report generiert. Dafür sollte das Ergebnis in einem vom Skript zu verarbeiteten Format vorliegen. Dies ist in der Regel XML. Die generierte Scorecard liegt in einem ausführlichen Bericht in HTML vor, indem jede Kategorie nach TP, FN, TN, FP, TPR und FPR bewertet

wird. Daneben wird zu jeder Kategorie, wie in Abschnitt 3 vorgestellt, ein Youden-Index gebildet, der von OWASP auch als Benchmark Score bezeichnet wird. Darüber hinaus wird eine CSV-Datei generiert, die das Ergebnis jedes einzelnen Testfalls wiedergibt. Das Ergebnis wird zusätzlich auf Grundlage des TPR und FPR, wie in Abbildung 4.1 zu sehen, in einer Grafik darstellt. Die Grafik zeigt den Zusammenhang zwischen TPR und FPR. Werte, die oberhalb der Geraden liegen, geben einen positiven Youden-Index an. Werte, die unterhalb der Geraden liegen, repräsentieren einen negativen Youden-Index. Da eine hohe TPR und eine geringe FPR angestrebt wird, sollte sich das Ergebnis eines guten Scanners nahe der oberen y-Achse aufhalten. Befinden sich die Ergebnisse im geringen Bereich der TPR, aber im hohen Bereich der FPR, so bedeutet das, dass der Scanner sehr schlechte Ergebnisse erzielt.

Nachfolgend eine Beispielrechnung um den Benchmark Score für einen Scanner mit einer TPR von 65% (0,65) und einer FPR von 8% (0,08) zu berechnen:

$$\begin{aligned} Sensitivität &= TPR\ (0,65) && (4.1)\\ Spezifität &= 1 - FPR\ (0,92) && (4.2)\\ J &= Sensitivität + Spezifität - 1 && (4.3)\\ J &= 0,65 + 0,92 - 1 && (4.4)\\ J &= 0,57 && (4.5) \end{aligned}$$

In diesem Beispiel ist demnach der Benchmark Score 57.

4.2.2 The Web Application Vulnerability Scanner Evaluation Project

> „WAVSEP ist eine Webanwendung zur Bewertung der Funktionen, Qualität und Genauigkeit von Schwachstellenscannern für Webanwendungen.“ [35]

WAVSEP ist eine verwundbare Webanwendung und enthält in der aktuellen Version 1.5 in den Kategorien XSS, SQL Injection, Local File Inclusion/Directory Traversal/Path Traversal, Remote File Inclusion und Unvalidated Redirect 1.226 Testfälle. Tabelle 4.2 zeigt eine Übersicht über die Anzahl der Testfälle nach Kategorie.

Ferner stehen über das Webinterface zu allen Testfällen Informationen sowie Muster zur Erkennung bzw. Nutzdaten zur Ausnutzung bereit. WAVSEP bietet keine automatische

Schwachstellen Kategorie	Anzahl Tests	Pos. Tests	Negative Tests
XSS	73	66	7
SQL Injection	146	136	10
Path Traversal/LFI	824	816	8
Remote File Inclusion	114	108	6
Unvalidated Redirect	69	60	9
Gesamtanzahl an Testfällen	1226		

Tabelle 4.2: Schwachstellen WAVSEP nach Kategorien

Auswertung an. Diese muss manuell über die generierten Reports der Scanner durchgeführt werden.

4.2.3 OWASP Zed Attack Proxy

> „OWASP Zed Attack Proxy (ZAP) ist ein einfach zu bedienendes Penetrationstest-Tool zum Auffinden von Schwachstellen in Webanwendungen.“ [8]

ZAP ist für Sicherheitsexperten und Entwickler konzipiert und bieten einen einfachen Einstieg in den Bereich der Penetrationstest. Es wird unter dem Dach des OWASP gepflegt. Im Kern ist ZAP ein Proxy, der zwischen dem Browser des Benutzers und der Webanwendung steht und Pakete abfangen, verändern und weiterleiten kann. Zusätzlich können mit ZAP automatisierte Scans durchgeführt und manuell nach Sicherheitslücken gesucht werden. ZAP steht für alle gängigen Betriebssysteme sowie als Docker-Container zur Verfügung. Neben einem Kommandozeilentool steht eine grafische Oberfläche zur Verfügung von der aus Scans durchgeführt werden können.

ZAP bietet neben aktiven auch passive Scans an. Beim passiven Scan fungiert ZAP nur als Proxy und liest alle Anfragen und Antworten vom Browser an den Webserver mit und verändert nichts. Dies ist geeignet, um ein grundlegendes Bild über den Sicherheitsstatus einer Webanwendung zu bekommen und Stellen für weitere Untersuchungen zu identifizieren sowie einige Schwachstellen zu entdecken. Der aktive Scan sucht gezielt nach Schwachstellen und führt verschiedene Angriffe auf die Webanwendung aus. Dafür stehen verschiedene *Add-ons* bereit, welche zum Auffinden von Sicherheitslücken aktiviert werden können. Das *Add-on Active Scanner rules* beinhaltet u. a. Regeln um Command Injection-, XSS- oder Path Traversal-Schwachstellen zu entdecken. Zur Identifizierung der verschiedenen Seiten der Anwendung wird ein sogenannter Spider eingesetzt. Dieser sucht in den Antworten des Webservers nach weiteren URLs und Links, welche eventuell

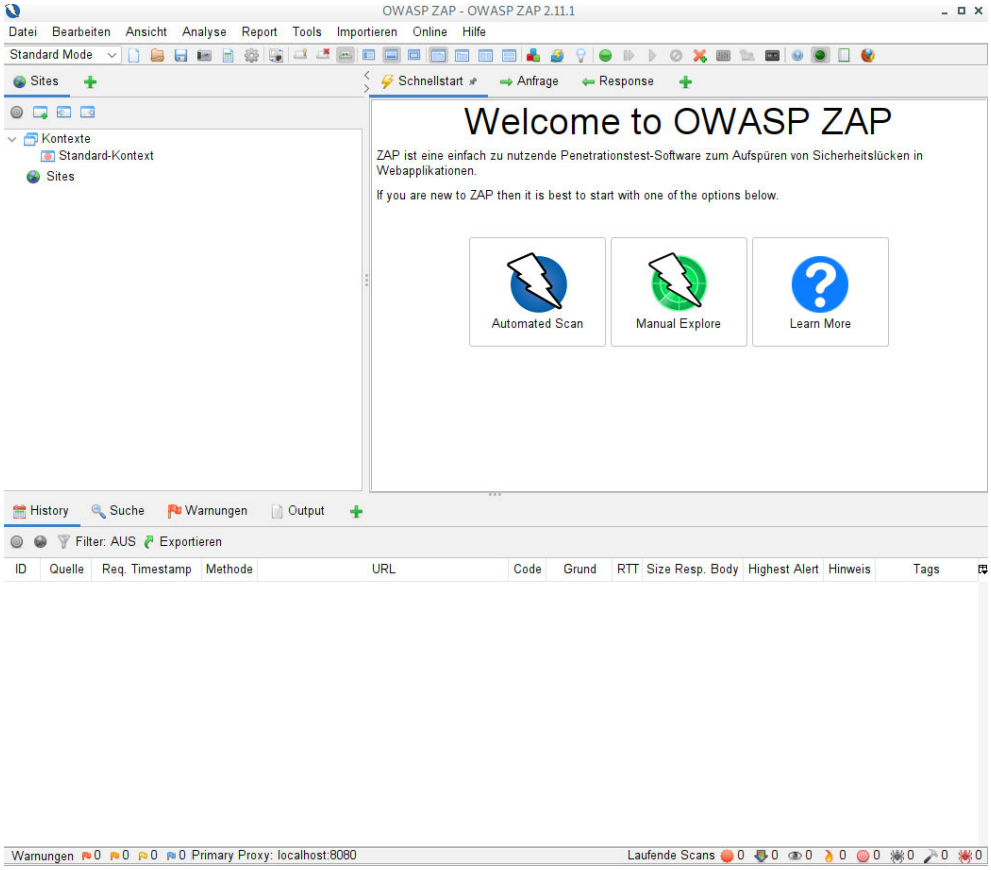

Abbildung 4.2: OWASP ZAP Benutzeroberfläche

versteckt sind, damit diese ebenfalls untersucht werden können. Für AJAX-Anwendungen steht ein AJAX-Spider zur Verfügung, der langsamer ist als der traditionelle, aber dafür effektiver arbeitet. Abbildung 4.2 zeigt die Benutzeroberfläche von OWASP ZAP.

4.2.4 Arachni

> „Arachni ist ein funktionsreiches, modulares und leistungsstarkes Ruby-Framework, das Penetrationstestern und Administratoren helfen soll, die Sicherheit von Webanwendungen zu bewerten.“ [18]

Arachni ist für die Plattformen Linux, Windows und macOS verfügbar. Darüber hinaus kann er sich während des Scanvorgangs selbst trainieren und aus dem Verhalten der Webanwendung lernen. Zudem reagiert Arachni auch auf die dynamische Veränderung einer

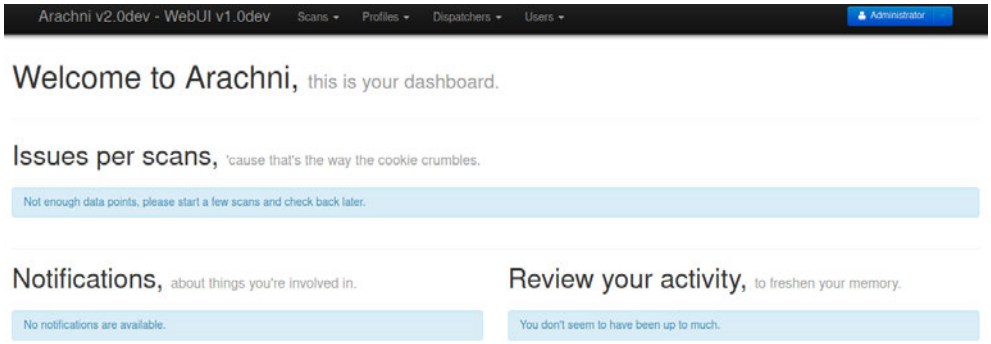

Abbildung 4.3: Arachni Webinterface

Webanwendung während des Scanvorgangs und passt diesen an. Neben einem einfach zu bedienenden Kommandozeilentool steht eine benutzerfreundliche Weboberfläche (siehe Abbildung 4.3) mit zahlreichen Funktionen zur Verfügung. Gleichzeitig bietet Arachni die Möglichkeit eigene Komponenten zu entwickeln und ins Framework einzubinden.

Für die erfolgreiche Durchführung von Scans bietet Arachni eine große Auswahl an verschiedenen Komponenten, sogenannte *checks* an, die bei einem Scan aktiviert werden können. Damit wird dann die Webanwendung mit den ausgewählten Komponenten auf Schwachstellen untersucht. Arachni bietet *checks* an, um u. a. SQL-Injection-, XSS- oder Path Traversal-Schwachstellen zu erkennen. Auflistung 4.1 zeigt den Befehl, mit dem alle verfügbaren Komponenten angezeigt werden können.

```
1 arachni --check-list
```

Auflistung 4.1: Arachni CLI-Befehl zur Auflistung der Checks

In Abbildung 4.4 ist die Beschreibung der Komponente *sql_injection* zu sehen.

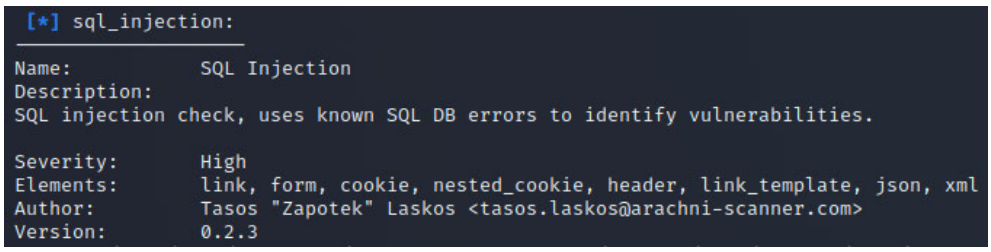

Abbildung 4.4: Arachni Check sql_injection

4.2.5 Wapiti

Wapiti ist ein in Python geschriebener Schwachstellenscanner. Nachdem alle URLs durch einen Crawler identifiziert wurden, führt der Scanner Black-Box-Tests durch. Dabei agiert Wapiti wie ein Fuzzer und schleust Nutzdaten ein, um festzustellen, ob ein Formular verwundbar ist. Wapiti steht als Kommandozeilentool zur Verfügung, worüber eine Vielzahl von Einstellungen (siehe Abbildung 4.5) vorgenommen werden können.

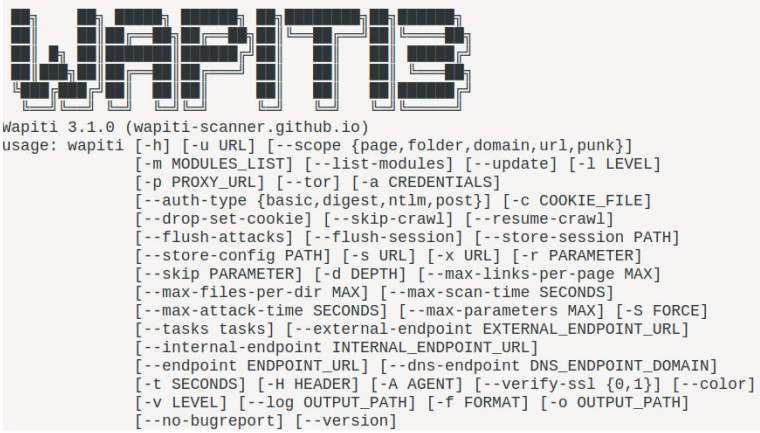

Abbildung 4.5: Ausführungsoptionen Wapiti [37]

Wapiti bietet, wie Arachni, ebenfalls eine Auswahl an Komponenten, hier *modules* genannt, zum Auffinden von Schwachstellen an. Über den Befehl in Auflistung 4.2 können alle Module angezeigt werden.

```
1 wapiti --list-modules
```

Auflistung 4.2: Wapiti CLI-Befehl zur Auflistung der Module

Abbildung 4.6 zeigt eine kleine Auswahl der zur Verfügung stehenden Module.

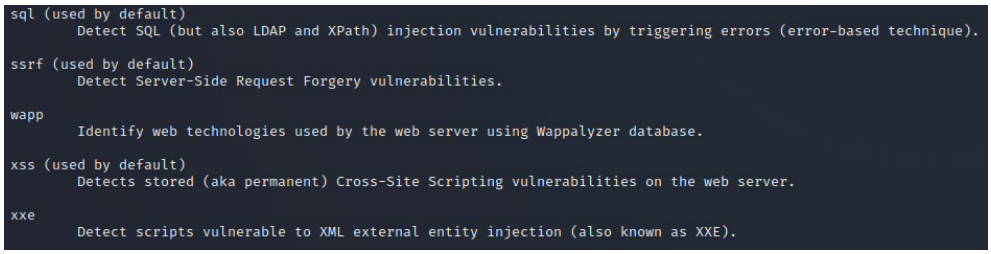

Abbildung 4.6: Wapiti Modules

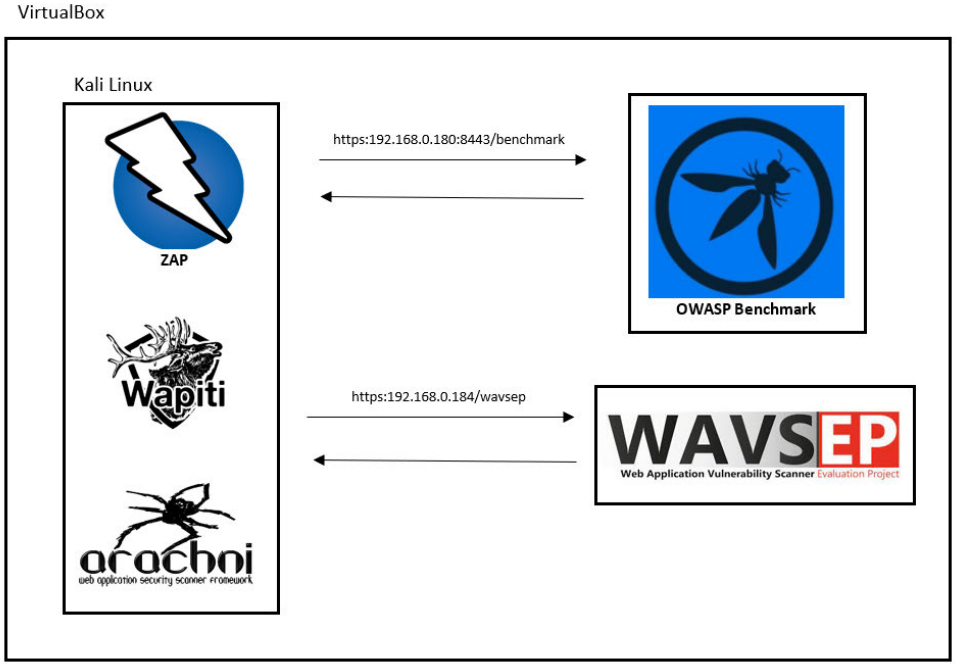

Abbildung 4.7: Aufbau der Testumgebung

4.3 Versuchsaufbau

Der Aufbau der Testumgebung erfolgte in einer virtuellen Umgebung mit VirtualBox. Für die Installation von OWASP Benchmark wurde eine virtuelle Maschine (VM) mit Lubuntu 20.04 LTS mit 2 CPUs und 6 GB Arbeitsspeicher konfiguriert. WAVSEP hingegen stand über ein vorinstallierte VM des OWASP Broken Web Applications Projekts, das neben WAVSEP auch noch andere verwundbare Webanwendungen für unterschiedliche Zwecke bereitstellt, zur Verfügung. Für die Durchführung der Scans wurde Kali Linux in Version 2021.4 ausgewählt. Eine vorinstallierte VM stand auf *kali.org* zur Verfügung, welche 2 CPUs und 4 GB Arbeitsspeicher zugewiesen bekam [6]. Abbildung 4.7 zeigt den Aufbau der Testumgebung.

4.4 Versuchsdurchführung

Die Scans gegen OWASP Benchmark Version 1.2 bzw. WAVSEP Version 1.5 wurden mit Arachni in Version 2.0dev, OWASP ZAP Version 2.11.1 und Wapiti Version 3.0.4 durch-

geführt. Die Realisierung der Scans erfolgt über das interne Netzwerk von VirtualBox. Dies gewährleistet eine nahezu störungsfreie Verbindung.

Im ersten Schritt wurden die Scans gegen OWASP Benchmark durchgeführt. Der Scan mit OWASP ZAP erfolgte dabei direkt über die Benutzeroberfläche als automatisierter Scan und dauerte ca. 10 Stunden. Wapiti wurde mit den Befehl in Auflistung 4.3 ausgeführt und war nach 8 Stunden fertig.

```
1 wapiti -u https://192.168.0.174:8443/benchmark/ -f xml -o
    ↪ wapitiBenchmark2
```

Auflistung 4.3: Wapiti Scan-Parameter OWASP Benchmark

Die Durchführung eines Scans mit Arachni hatte mehrere Schwierigkeiten, da Arachni ab einer gewissen Laufzeit das Trainer-Subsystem aktiviert, wodurch sich der Scan drastisch verlangsamt hat und auch nach 7 Tagen noch nicht abgeschlossen war. Deshalb wurde jede Schwachstellenkategorie einzeln und auch nur mit den benötigten Komponenten untersucht. Auflistung 4.4 zeigt den Befehl zum Auffinden von SQL Injection-Schwachstellen. Mit diesen Parametern konnte der Scan in ca. 4 Stunden abgeschlossen werden.

```
1 ./arachni https://192.168.0.174:8443/benchmark/sqli-Index.html
    ↪ --checks=sql_injection,sql_injection_differential
    ↪ --report-save-path=arachniAllSQLChecks.afr
```

Auflistung 4.4: Arachni SQL Scan-Parameter OWASP Benchmark

Im zweiten Schritt folgte die Konvertierung der Ergebnisse von OWASP ZAP und Arachni ins XML-Format. Der Report von Wapiti wurde nach einem erfolgreichen Scan direkt im XML-Format gespeichert. Bei OWASP ZAP konnte der Report über die Anwendung direkt exportiert werden. Arachni benötigte noch ein zusätzliches Skript (siehe Auflistung 4.5), um den Report umzuwandeln.

```
1 arachni_reporter arachniAllSQLChecks.afr
    ↪ --report=xml:outfile=arachniSQLI.xml
```

Auflistung 4.5: Arachni Reporter

Abschließend mussten alle Reports in das Verzeichnis *results* von OWASP Benchmark kopiert werden. Danach folgte die Generierung der Scorecards über das Skript *createScorecard.sh*.

Die Scans gegen WAVSEP haben sich mit allen drei Scannern etwas schwieriger gestaltet, da es zu sehr langen Laufzeiten gekommen ist. Deshalb wurden die Scans nur gegen die jeweilige Schwachstellenkategorie mit den benötigten Modulen ausgeführt und somit in überschaubarer Zeit beendet. Auflistung 4.6 zeigt den Befehl für einen Scan von Wapiti und Auflistung 4.7 den Befehl für einen Scan von Arachni mit den jeweiligen Parametern. Die Durchführung des Scans mit OWASP ZAP wurde über die grafische Benutzeroberfläche durchgeführt.

```
1 wapiti -u http://192.168.0.184/wavsep/active/index-xss.jsp --start
    ↪ "http://192.168.0.184/wavsep/active/index-xss.jsp" --scope
    ↪ "domain" -m xss,permanentxss --color --verbose 2 -f html -o
    ↪ wapitiWavsepXSS
```

Auflistung 4.6: Wapiti Scan-Parameter WAVSEP

```
1 ./arachni https://192.168.0.174:8443/benchmark/sqli-Index.html
    ↪ --checks=sql_injection,sql_injection_differential
    ↪ --report-save-path=arachniAllSQLChecks.afr
```

Auflistung 4.7: Arachni SQL Scan-Parameter WAVSEP

5 Auswertung

In diesem Abschnitt werden die Ergebnisse der individuellen Scanner gegen die Benchmark-Anwendungen vorgestellt. Zuerst erfolgt die Auswertung gegen WAVSEP und OWASP Benchmark und danach werden die Resultate verglichen und bewertet. Die Ergebnisse werden im Nachgang mit Hilfe weiterer Studien diskutiert.

5.1 Ergebnisse der Scanner WAVSEP

In dieser Auswertung wird auf vier von den in Abschnitt 4.2.2 vorgestellten fünf Schwachstellenkategorien von WAVSEP eingegangen, da die Scanner in diesen Kategorien aussagekräftige Ergebnisse erzielt haben. Dies sind XSS, SQL Injection, Path Traversal/LFI und Remote File Inclusion. Die Testfälle der einzelnen Kategorien konnten über explizite URLs, wie in Abbildung 5.1 dargestellt, gescannt werden.

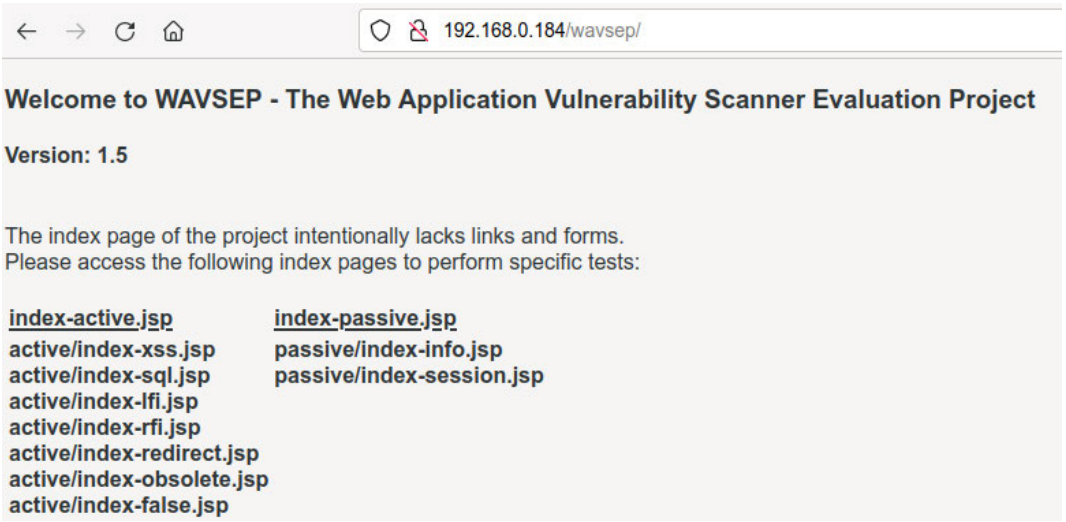

Abbildung 5.1: WAVSEP Übersicht Testfälle

5.1.1 Wapiti

Die Ergebnisse von Wapiti wurden über den generierten HTML-Report ausgewertet. Abbildung 5.2 zeigt einen gefundenen Testfall aus diesem in der Kategorie SQL Injection. Dabei ist zu sehen, dass über den Parameter *msg* eine Blind SQL Injection möglich ist.

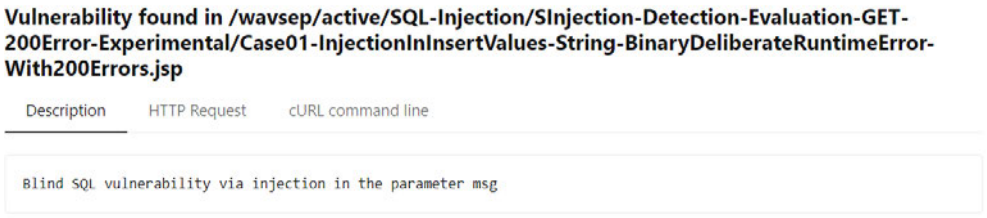

Abbildung 5.2: Wapiti Testfall SQL Injection

In Abbildung 5.3 sowie Tabelle 5.1 sind die Ergebnisse des Scans dargestellt. Die grünen Säulen stellen dabei die TPR und die roten die FNR dar. FP sind nicht aufgetreten. Hierbei ist gut zu sehen, dass Wapiti in der Kategorie SQL Injection das beste Ergebnis erzielt hat. Dort wurden 85,29% aller Schwachstellen erkannt.

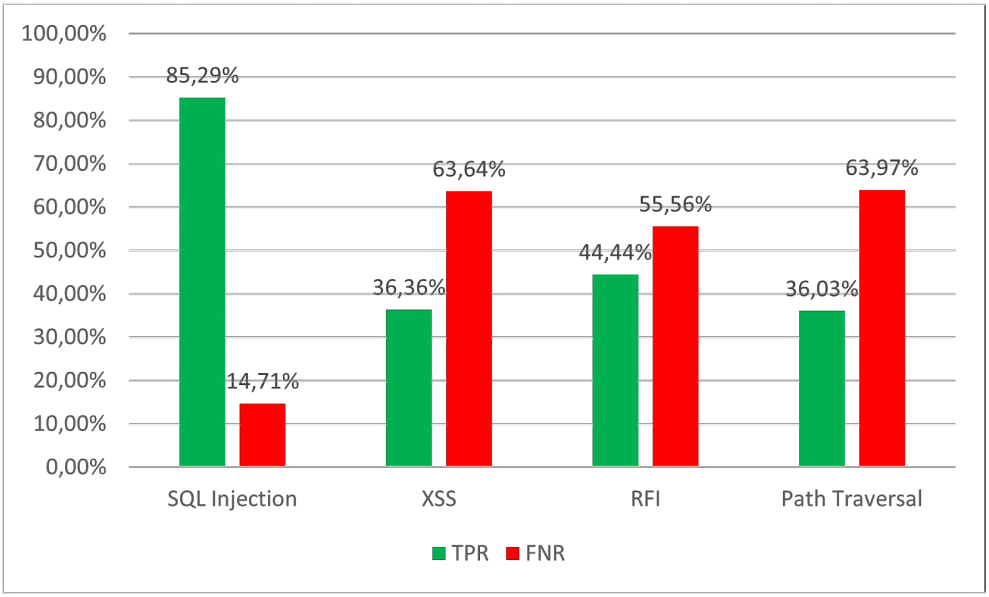

Abbildung 5.3: Wapiti Ergebnisse WAVSEP

In der Kategorie XSS waren es 36,36%, in der Kategorie Remote File Inclusion 44,44% und in der Kategorie Path Traversal 36,03%. Wapiti konnte in drei von vier Kategorien weniger als die Hälfte der vorhanden Schwachstellen erkennen.

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	146	116	20	10	0	85,29%	0%	85,29%
XSS	73	24	42	7	0	36,36%	0%	36,36%
Remote File Inclusion	114	48	60	6	0	44,44%	0%	44,44%
Path Traversal	824	294	522	8	0	36,03%	0%	36,03%

Tabelle 5.1: Wapiti Ergebnisse WAVSEP

5.1.2 OWASP ZAP

Die Ergebnisse von OWASP ZAP konnten direkt über die Benutzeroberfläche der Anwendung ausgewertet werden. In Abbildung 5.4 wurde in den Testfall *RXSS-Detection-Evaluation-GET Case15* über den Parameter *userinput* die Payload *;alert(1)* eingeschleust. Dadurch wird eine Alert-Box geöffnet, sobald der Programmcode ausgeführt wird.

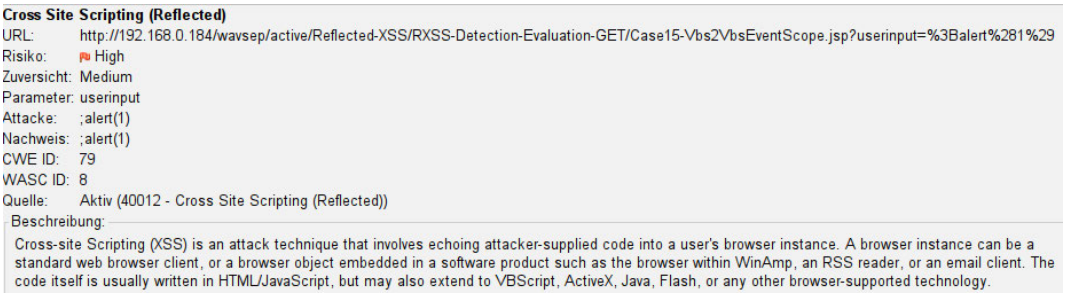

Abbildung 5.4: OWASP ZAP Testfall XSS

In Tabelle 5.2 und Abbildung 5.5 sind die Ergebnisse der verschiedenen Kategorien zu sehen. Besonders ragt die Erkennung von XSS-Schwachstellen mit 100% heraus. Auch bei SQL Injection ist die Datektionsrate mit 83,09% sehr gut, gefolgt von RFI mit 77,82% und Path Traversal mit 58,82%.

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	146	113	23	10	0	83,09%	0%	83,09%
XSS	73	66	0	7	0	100%	0%	100%
Remote File Inclusion	114	84	24	6	0	77,78%	0%	77,78%
Path Traversal	824	480	336	8	0	58,82%	0%	58,82%

Tabelle 5.2: OWASP ZAP Ergebnisse WAVSEP

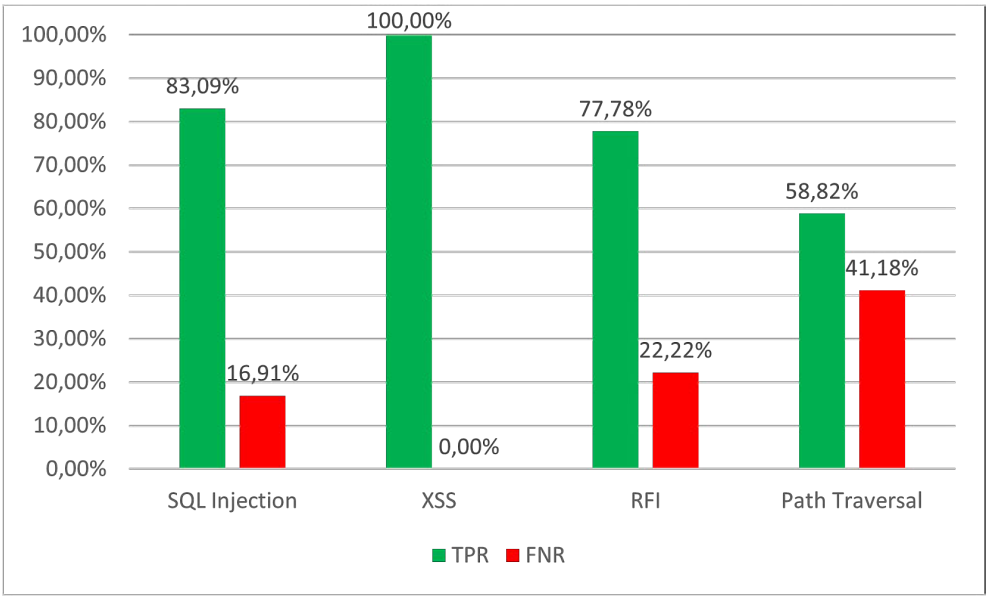

Abbildung 5.5: OWASP ZAP Ergebnisse WAVSEP

5.1.3 Arachni

Arachni bietet die Möglichkeit den Report in verschiedenen Formaten zu erzeugen. Die Auswertung für WAVSEP wurde über den HTML-Report durchgeführt. In Abbildung 5.6 ist ein Testfall für die Erkennung einer SQL Injection-Schwachstelle zu sehen.

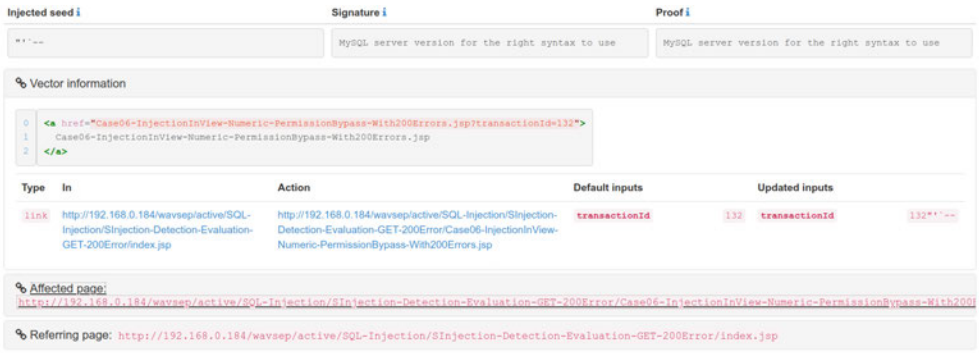

Abbildung 5.6: Arachni Testfall SQL Injection

Dabei wird eine Payload übergeben und die Webanwendung gibt eine Fehlermeldung der Datenbank zurück. Abbildung 5.7 und Tabelle 5.3 zeigen die Ergebnisse des Scans. Herausragend ist hierbei die Detektion von RFI-Schwachstellen (100%). In der Kategorie

SQL Injection konnten 66,18%, in der Kategorie XSS 69,70% und in der Kategorie Path Traversal 25% der Schwachstellen identifiziert werden.

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	146	90	46	10	0	66,18%	0%	66,18%
XSS	73	46	20	7	0	69,7%	0%	69,7%
Remote File Inclusion	114	108	0	6	0	100%	0%	100%
Path Traversal	824	204	612	8	0	25%	0%	25%

Tabelle 5.3: Arachni Ergebnisse WAVSEP

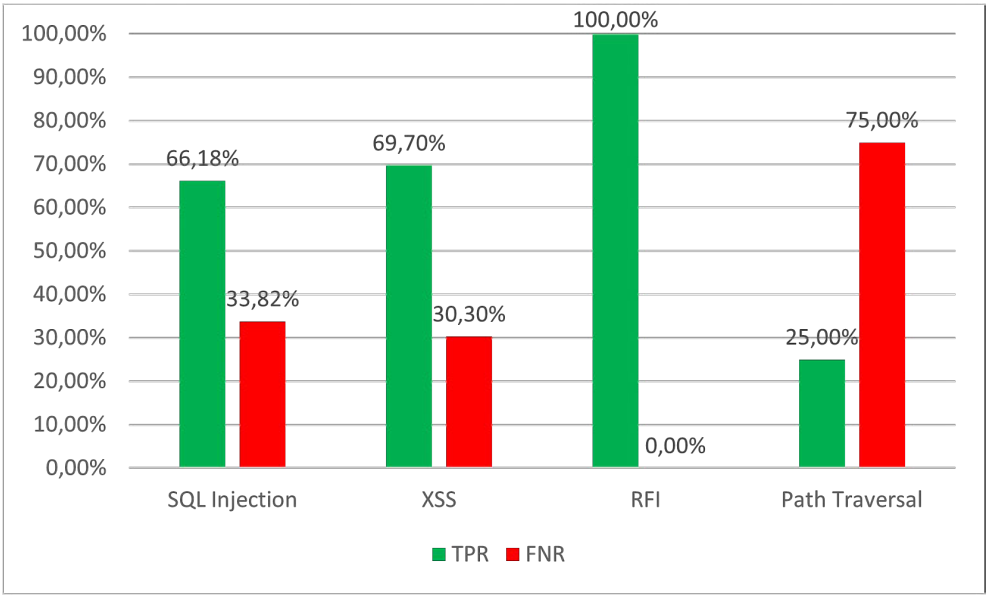

Abbildung 5.7: Arachni Ergebnisse WAVSEP

5.1.4 Vergleich der Ergebnisse WAVSEP

Die Ergebnisse der einzelnen Kategorien werden in diesem Abschnitt miteinander verglichen und bewertet. Abbildung 5.8 zeigt die TPR der Scanner sortiert nach Kategorien. Dabei ist gut zu erkennen, dass Arachni in der Kategorie RFI mit 100% das beste Ergebnis aller Scanner erzielt hat. OWASP ZAP konnte wiederum alle XSS-Schwachstellen detektieren. Wapiti hat das beste Ergebnis in der Kategorie SQL Injection mit 85,29%, dicht gefolgt von OWASP ZAP mit 83,09%, erreicht. In der Kategorie Path Traversal haben alle Scanner etwas schlechter abgeschlossen. ZAP hat hier aber immer noch 58,82% der Schwachstellen erkannt. Diese Kategorie hatte mit 816 die meisten Testfälle.

Dieser Vergleich zeigt bereits, dass sich einige Scanner sehr gut zur Detektion bestimmter Schwachstellen eignen. Es ist jedoch zu erwähnen, dass keiner dieser betrachteten Scanner in der Lage ist, alle Schwachstellen zu erkennen.

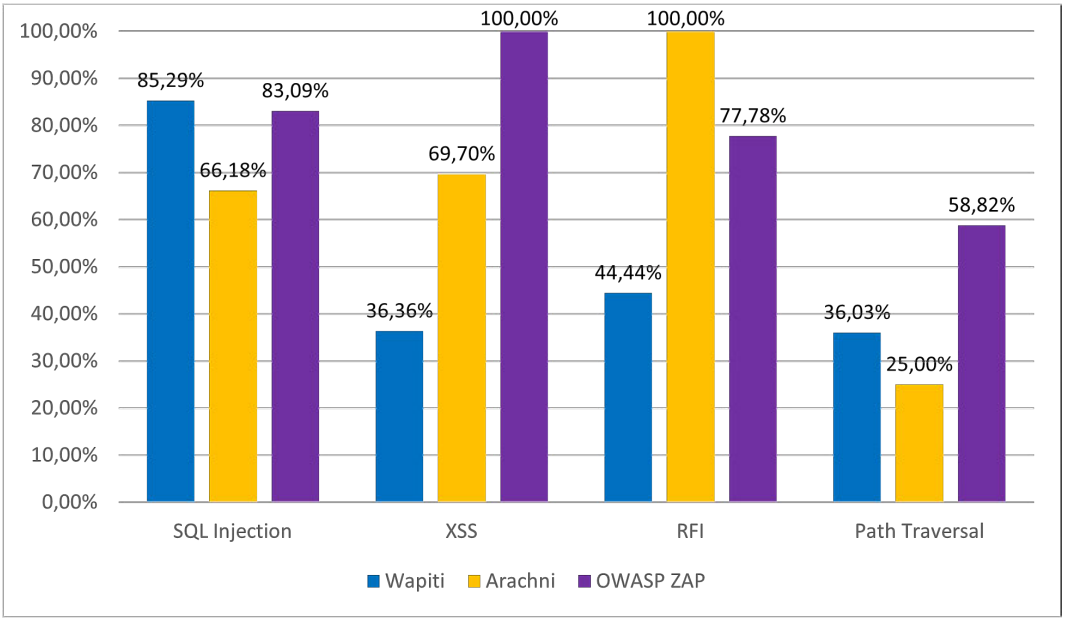

Abbildung 5.8: Vergleich TPR Ergebnisse Scanner WAVSEP

5.2 Ergebnisse der Scanner OWASP Benchmark

OWASP Benchmark hat die beinhalteten Schwachstellen in elf Kategorien aufgeteilt, wobei vier dieser Kategorien hier näher beleuchtet werden. Das sind Command Injection, Path Traversal, SQL Injection und XSS. Diese Kategorien wurden ausgewählt, da die Scanner in diesen aussagekräftige Ergebnisse erzielt haben. Ein Scan ist auch hier über explizite URLs siehe Abbildung 5.9 möglich. Die Auswertung der Resultate musste bei OWASP Benchmark nicht wie bei WAVSEP manuell durchgeführt werden, sondern ließ sich über das Skript *createScorecards.sh* generieren. Damit wird ein HTML-Report zur Begutachtung der Ergebnisse sowie eine CSV-Datei mit Details zu jedem einzelnen Testfall erstellt.

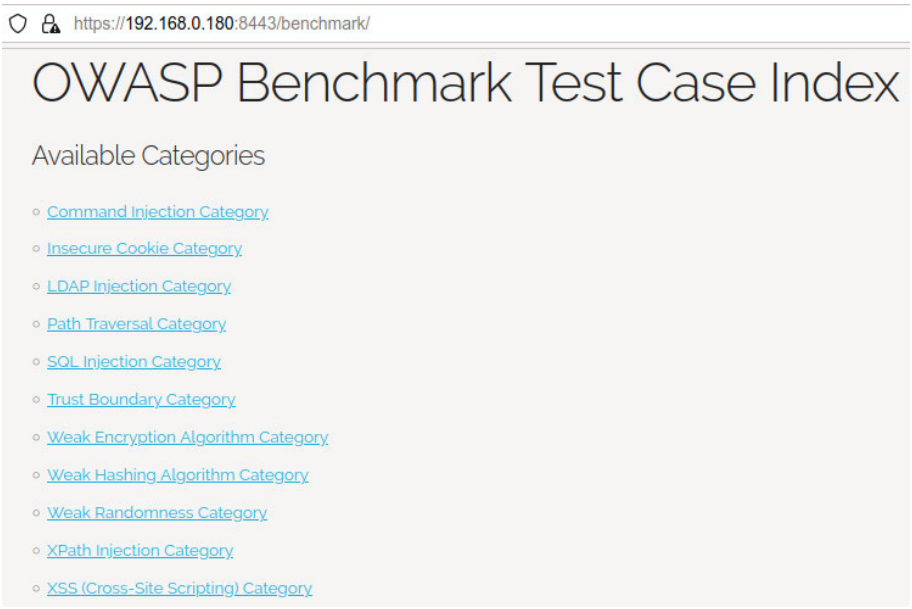

Abbildung 5.9: OWASP Benchmark Übersicht Kategorien

5.2.1 Wapiti

Tabelle 5.4 und Abbildung 5.10 zeigen die Ergebnisse, die aus der Scorecard entnommen werden konnten.

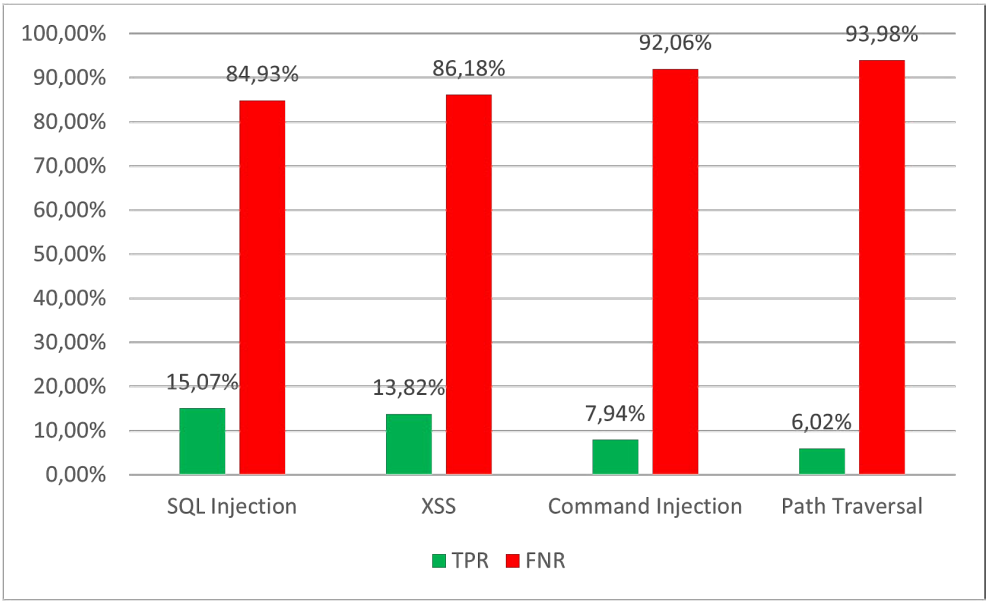

Abbildung 5.10: Wapiti Ergebnisse OWASP Benchmark

Dabei ist die geringe Erkennung von Schwachstellen gut zu sehen. Am besten wurden noch Testfälle von SQL Injection erkannt, aber auch hier sind dies nur 15,07%.

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	504	41	231	232	0	15,07%	0%	15,07%
XSS	455	34	212	209	0	13,82%	0%	13,82%
Command Injection	251	10	116	125	0	7,94%	0%	7,94%
Path Traversal	268	8	125	135	0	6,02%	0%	6,02%

Tabelle 5.4: Wapiti Ergebnisse OWASP Benchmark

In der Kategorie XSS sind 13,82%, in Command Injection 7,94% und in Path Traversal 6,02% richtig bestimmt wurden. FPs sind keine aufgetreten.

5.2.2 OWASP ZAP

Die Auswertung der Ergebnisse von OWASP ZAP zeigen, dass die besten Ergebnisse in der Kategorie SQL Injection mit 57,72% und XSS mit 55,69% erzielt wurden.

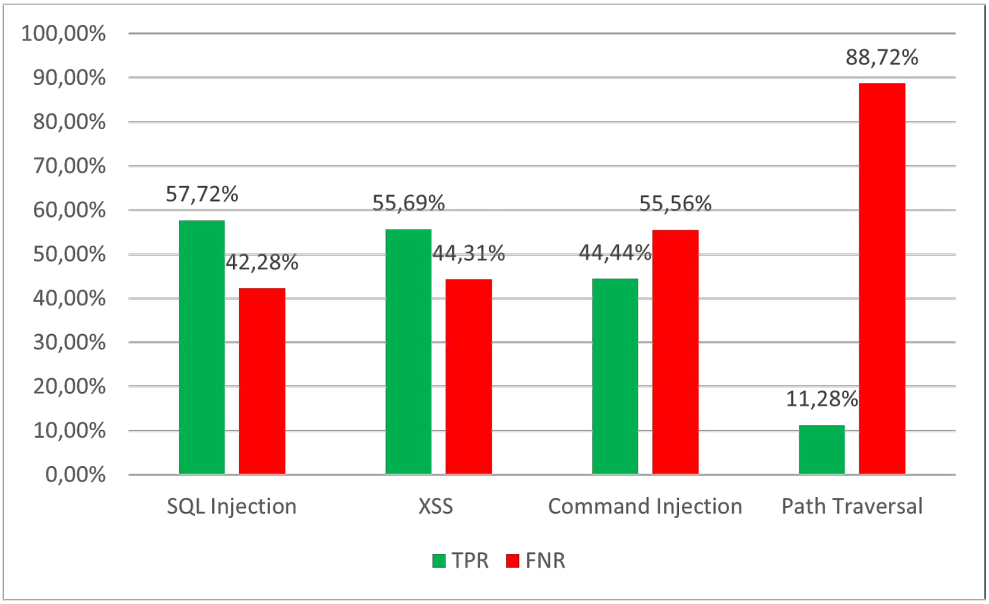

Abbildung 5.11: ZAP Ergebnisse OWASP Benchmark

Danach kommt Command Injection mit 44,44% und Path Traversal mit nur 11,28%. Die Ergebnisse sind in Abbildung 5.11 und Tabelle 5.5 dargestellt. Hier ist noch zu erwähnen, dass ein FP in der Kategorie SQL Injection aufgetreten ist.

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	504	157	115	231	1	57,72%	0,43%	57,29%
XSS	455	137	109	209	0	55,69%	0%	55,69%
Command Injection	251	56	70	125	0	44,44%	0%	44,44%
Path Traversal	268	15	118	135	0	11,28%	0%	11,28%

Tabelle 5.5: ZAP Ergebnisse OWASP Benchmark

5.2.3 Arachni

Nach der Auswertung der Scorecard zeigen sich folgende Ergebnisse in den einzelnen Kategorien: XSS 62,20%, SQL Injection 50%, Command Injection 34,13% und Path Traversal 11,29%. In der Tabelle 5.6 sowie der Abbildung 5.12 sind die Ergebnisse nochmals dargestellt.

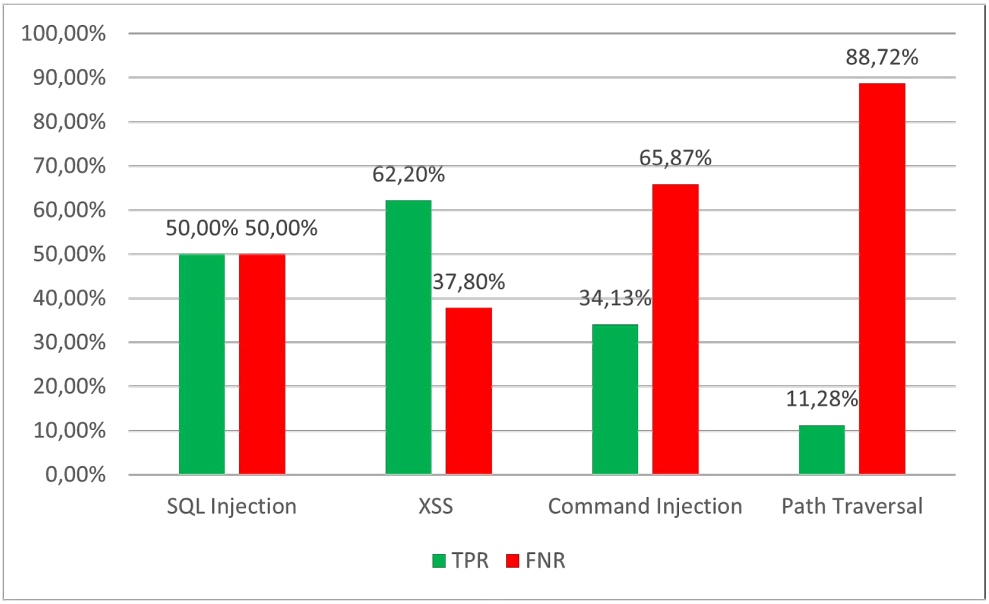

Abbildung 5.12: Arachni Ergebnisse OWASP Benchmark

Schwachstellen Kategorie	Anzahl Testfälle	TP	FN	TN	FP	TPR	FPR	Score
SQL Injection	504	136	136	232	0	50%	0%	50%
XSS	455	153	93	209	0	62,2%	0%	62,2%
Command Injection	251	43	83	125	0	34,13%	0%	34,13%
Path Traversal	268	15	118	135	0	11,28%	0%	11,28%

Tabelle 5.6: Arachni Ergebnisse OWASP Benchmark

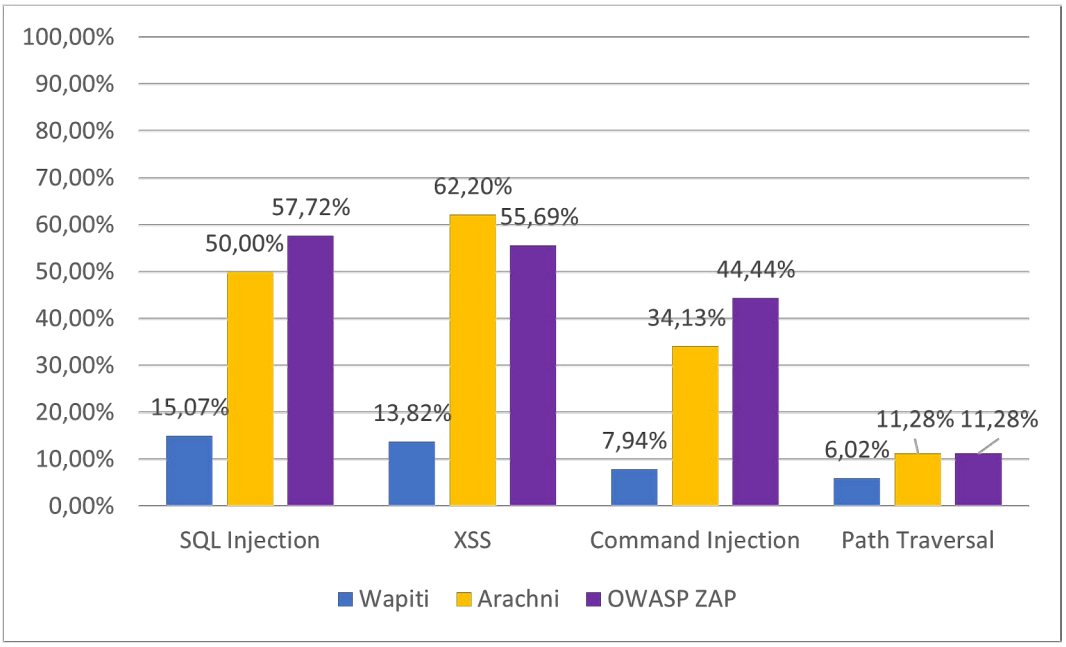

Abbildung 5.13: Vergleich TPR Ergebnisse OWASP Benchmark

5.2.4 Vergleich der Ergebnisse OWASP Benchmark

Nachfolgenden werden die Ergebnisse der vier Kategorien gegenübergestellt und bewertet. In Abbildung 5.13 ist der TPR der Scanner nach Kategorien sortiert dargestellt. Hierbei ist gut zu sehen, dass Wapiti in allen Kategorien das schlechteste Ergebnis erzielt hat. Darüber hinaus haben alle Scanner in der Kategorie Path Traversal sehr schlecht abgeschnitten. Wapiti hat dort 6,02%, Arachni 11,28% und ZAP ebenfalls 11,28% der Schwachstellen erkannt. Das beste Ergebnis bei der Detektion von SQL Injection- und Command Injection-Schwachstellen erreichte ZAP mit 57,72% bzw. 44,44%. In der Kategorie XSS konnte dagegen Arachni mit 62,20% am meisten Testfälle erkennen.

Bei der Betrachtung der Ergebnisse fällt auf, dass Wapiti im Schnitt nur 10,7% der Schwachstellen erkennen konnte. Arachni und ZAP hingegen konnten 39,4% bzw. 42,5% detektieren. Wenn nur die Kategorien SQL Injection, XSS und Command Injection betrachet werden, sind es ca. 50%.

5.3 Vergleich der Ergebnisse WAVSEP vs. OWASP Benchmark

In diesem Abschnitt wird ein Vergleich der einzelnen Kategorien der Benchmark-Anwendungen durchgeführt. Dafür werden die drei Kategorien SQL Injection, XSS und Path Traversal genauer betrachtet, da diese bei beiden Benchmark-Anwendungen vorhanden sind.

Es ist erforderlich Faktoren bei der Diskussion zu betrachten. Dies ist zum einen, dass eine unterschiedliche Anzahl an Testfällen in den Kategorien vorhanden sind. Zum anderen können die Testfälle unterschiedlich komplex gestaltet sein. Darüber hinaus kann ein Scanner auch so programmiert und angepasst sein, dass er Testfälle in einer Benchmark-Anwendung besser erkennt als in der anderen.

5.3.1 SQL Injection

Die Kategorie SQL Injection besitzt in WAVSEP 136 und in OWASP Benchmark 272 positive Testfälle. WAVSEP hat also genau die Hälfte an positiven Testfällen. Zudem stehen in WAVSEP nur 10 und in OWASP Benchmark 232 negative Testfälle zur Verfügung. Nachfolgend werden nur die Ergebnisse der positiven Testfälle verglichen. Das ist vertretbar, da bei den Scans in dieser Kategorie nur bei ZAP 1 FP (0,43%) aufgetreten ist.

Wapiti hat in dieser Kategorie bei WAVSEP mit 116 (85,29%) von 136 erkannte Fällen am besten abgeschlossen, aber bei OWASP Benchmark mit 41 (15,07%) von 272 Fälle am schlechtesten. Das kann an der Schwierigkeit der Tests liegen und zeigt sehr gut, dass die Scanner beim Auffinden von Schwachstellen unterschiedlicher Komplexität Probleme haben. Die 20 nicht detektierten Schwachstellen von Wapiti in WAVSEP wurden auch von keinem der anderen Scanner erkannt. Warum das so ist, konnte nicht bestimmt werden. Auch bei genauer Betrachtung der Testfälle ist keine Besonderheit zu anderen Testfällen aufgefallen. ZAP liegt nur knapp hinter Wapiti mit 113 (83,09%) von 136 bei WAVSEP und hat bei OWASP Benchmark 157 (57,72%) von 272 Schwachstellen erkannt. Damit erreicht ZAP auch das beste Ergebnis der drei Scanner bei OWASP Benchmark in dieser Kategorie. Arachni erkennt im Gegensatz zu den anderen beiden Scannern bei WAVSEP nur 90 (66,18%) von 136 Schwachstellen und bei OWASP Benchmark mit 136 (50%) genau die Hälfte. Die Ergebnisse sind in Abbildung 5.14 dargestellt.

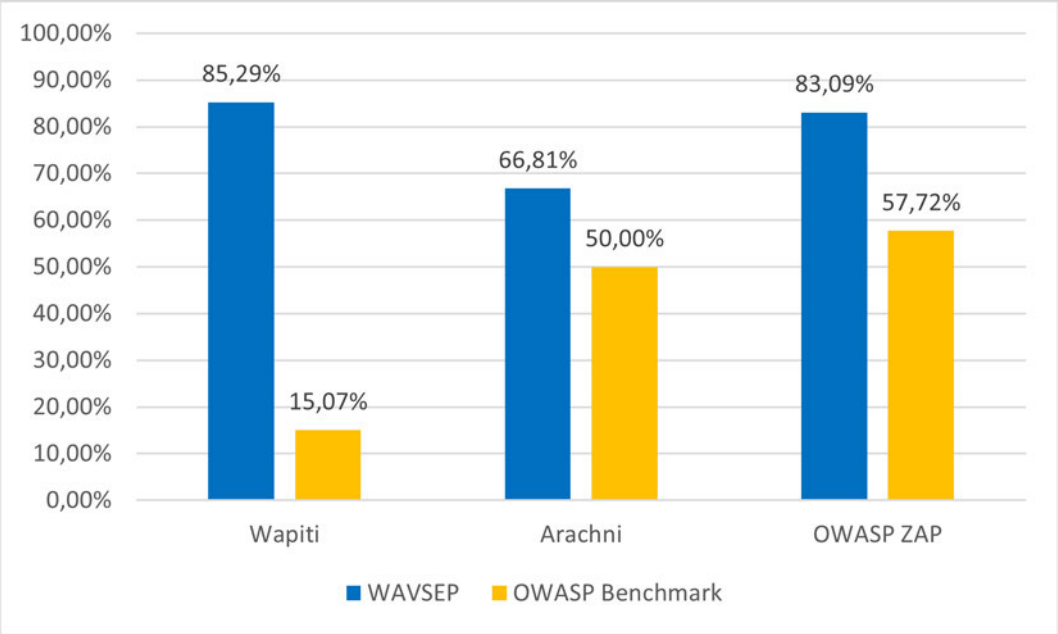

Abbildung 5.14: Vergleich TPR Benchmark Ergebnisse SQL Injection

Bei der Analyse der Ergebnisse fällt auf, dass Wapiti bei der Detektion der Schwachstellen in OWASP Benchmark im Gegensatz zu WAVSEP Probleme hatte und dadurch eine um 70% schlechtere Erkennungsrate erzielt hat. Arachni ist ebenfalls abgefallen, aber nur um 16,18% und ZAP um 25,37%. Das deutet darauf hin, dass die Mechanismen zum Erkennen dieser Testfälle nicht oder nicht ausreichen implementiert sind.

Darüber hinaus haben Wapiti und Arachni jeweils eine Teilmenge der von ZAP in OWASP Benchmark gefunden Schwachstellen erkannt. Bei WAVSEP konnte das gleiche beobachtet werden. Dort hat ZAP und Arachni jeweils eine Teilmenge der von Wapiti detektierten Testfälle bestimmt. Daraus kann geschlossen werden, dass die Scanner ähnliche Verfahren zum Detektieren der Schwachstellen verwenden.

5.3.2 XSS

Die Kategorie XSS hat bei OWASP Benchmark 246 und bei WAVSEP 66 positive Testfälle. ZAP hat bei WAVSEP das beste Ergebnis der Scanner mit 100% erkannten Schwachstellen erzielt. Bei OWASP Benchmark allerdings nur 137 (55,69%) detektiert. Arachni liegt bei OWASP Benchmark mit 153 (62,2%) gefunden Schwachstellen knapp vor ZAP. Bei WAVSEP wurden 46 (69,7%) Testfälle richtig bestimmt. Wapiti hat bei beiden

Benchmarks am schlechtesten abgeschnitten. Bei WAVSEP wurden 24 (36,36%) und bei OWASP Benchmark 34 (13,82%) der Schwachstellen erkannt. In Abbildung 5.15 sind die Ergebnisse grafisch dargestellt.

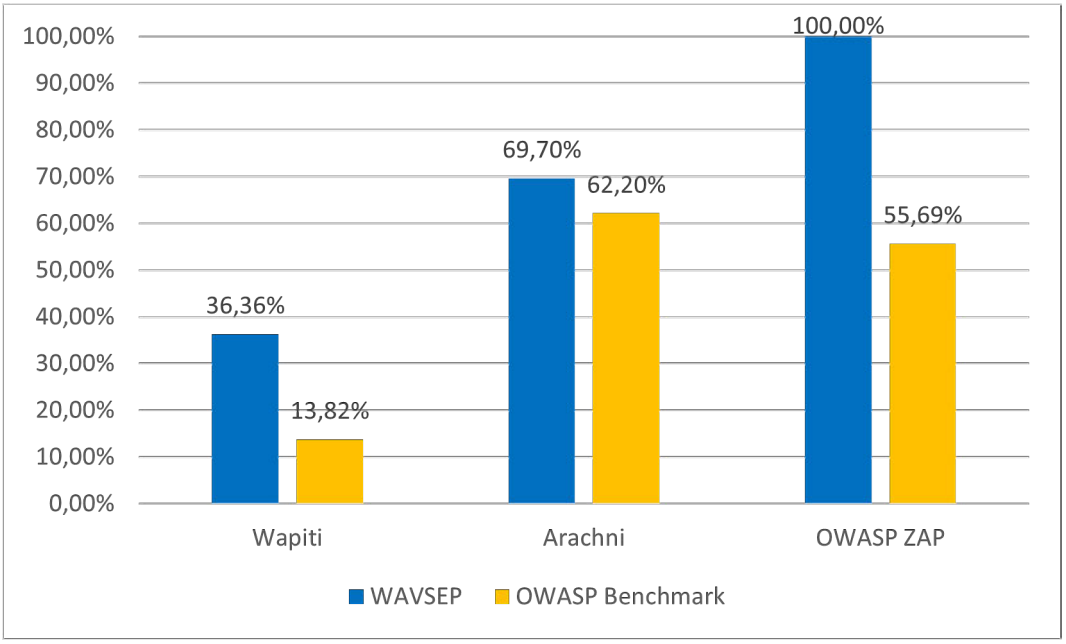

Abbildung 5.15: Vergleich TPR Benchmark Ergebnisse XSS

Auch bei dieser Kategorie haben die Scanner bis auf ein paar kleine Abweichungen die gleichen Schwachstellen gefunden. In WAVSEP waren es sogar wieder Teilmengen der besser platzierten Scanner. Das deutet wie bei SQL Injection auf ein ähnliches Detektionsverfahren hin.

5.3.3 Path Traversal

Abschließend werden noch die Ergebnisse der letzten Kategorie Path Traversal verglichen. OWASP Benchmark hat in dieser Kategorie 133 und WAVSEP 816 positive Testfälle. Arachni hat bei WAVSEP 204 (25%) Fälle richtig erkannt und damit am schlechtesten in dieser Kategorie abgeschlossen. Bei OWASP Benchmark hat Arachni genau wie ZAP 15 (11,28%) Schwachstellen richtig bestimmt und damit das beste Ergebnis erzielt. Gegen WAVSEP konnte Arachni 204 (25%) und WAVSEP 480 (58,82%) der Fälle detektieren. Dieses Resultat lässt vermuten, dass bei ZAP die Path Traversal Erkennung besser umgesetzt ist.

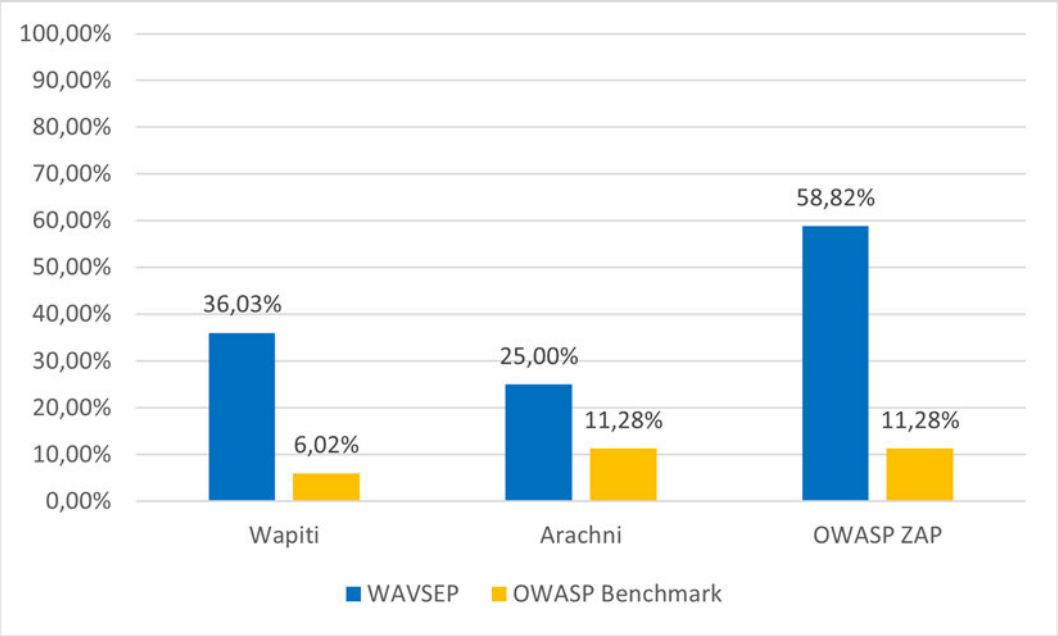

Abbildung 5.16: Vergleich TPR Benchmark Ergebnisse Path Traversal

Wapiti konnte bei WAVSEP 294 (36,03%) und bei OWASP Benchmark 8 (6,02%) Testfälle richtig erkennen. Damit hat Wapiti das schlechteste Ergebnis bei OWASP Benchmark erzielt. Abbildung 5.16 zeigt die Ergebnisse gegen die Benchmark-Anwendungen.

5.4 Diskussion der Ergebnisse

Beim Vergleich der Ergebnisse ist klar zu sehen, dass alle Scanner beim OWASP Benchmark weniger Schwachstellen finden konnten als bei WAVSEP. Daraus kann geschlossen werden, dass die Testfälle von OWASP Benchmark für die Scanner schwieriger zu erkennen waren. ZAP hat die besten Ergebnisse der drei Scanner gezeigt. Wapiti hingegen hat im Durchschnitt die schlechtesten Ergebnisse erzielt, auch wenn der Scanner bei WAVSEP in der Kategorie SQL Injection am besten abgeschnitten hat. Es gab auch keine Kategorie in der ein Scanner alle und die anderen Scanner keine Schwachstellen erkennen konnten. Hervorzuheben ist noch, dass bei den Untersuchungen nur bei ZAP gegen OWASP Benchmark in der Kategorie SQL Injection ein FP aufgetreten ist.

Um die Resultate besser einordnen zu können, müssen die Ergebnisse aus dieser Untersuchung mit anderen Studien verglichen werden. Dafür werden einerseits Arbeiten

herangezogen, welche ältere Versionen dieser Scanner untersucht haben. Zusätzlich werden Resultate weiterer Scanner gegen OWASP Benchmark und WAVSEP betrachtet, womit die Performance der verwendeten Scanner bewertet werden kann.

Das OWASP Benchmark Projekt stellt Ergebnisse zu Vergleichszwecken bereit. Dafür werden Resultate von ZAP mit einer Version auf dem Stand von 09.06.2016 sowie des Scanners Find Security Bugs in Version 1.4.6 herangezogen. In der Kategorie XSS wurden bei ZAP in dieser Arbeit ein um ca. 27% bessere TPR erzielt. Darüber hinaus ist die Detektion von Path Traversal-Schwachstellen in der älteren Version schlechter. Bei der Betrachtung der Resultate von Find Security Bugs fällt neben der hohen TPR auch die sehr hohe FPR auf. Da OWASP Benchmark sehr viele FP-Testfälle bereitstellt, ist dieses Ergebnis sehr kritisch zu sehen. Insgesamt sollten FP vermieden werden, da jeder Fall geprüft werden muss, wodurch ein hoher Arbeitsaufwand anfällt. Dieses Ergebnis ist deshalb als sehr schlecht einzustufen.

In der Arbeit von Mburano und Si wurden Scans gegen OWASP Benchmark mit ZAP 2.7 und Arachni 1.5.1 durchgeführt [24]. Die Ergebnisse sind dabei vergleichbar zu den Ergebnissen, die in dieser Arbeit erzielt wurden. Aufgefallen ist, dass bei der Kategorie Path Traversal keine Testfälle erkannt wurden. Dafür hat ZAP bei XSS einen um 20% besseren TPR erzielt. Die weiteren Resultate haben sich nur um ein paar Prozentpunkte unterschieden. In Tabelle 5.7 sind die Ergebnisse der Untersuchungen dargestellt. Die TPR ist dabei grün und die FPR, wenn diese größer als 0% ist, rot hinterlegt. Zusätzlich ist für jeden Scanner die Version angegeben.

Zur Einordnung der Ergebnisse von WAVSEP werden Resultate von der offiziellen Webseite des Entwicklers der Benchmark-Anwendung Shay Chen und aus der Arbeit von Idrissi et al. herangezogen [12, 22]. Dabei werden neben den Scannern Wapiti, Arachni und ZAP noch Ergebnisse von Skipfish, Vega, IronWASP und W3AF betrachtet. Die Scans wurden dabei ebenfalls gegen die Version 1.5 von WAVSEP durchgeführt. Bei den Ergebnissen von Shay Chen ist sehr auffällig, dass fast alle betrachteten Scanner 100% der Schwachstellen bei SQL Injection detektiert haben. Nur Skipfish hat hier lediglich 76.47% erzielt. Bei XSS hat Chen mit Arachni ein um 20% und mit Wapiti ein um 30% bessere TPR erzielt als in dieser Arbeit. In der Kategorie Path Traversal hat Chen ebenfalls mit allen Scannern bessere Ergebnisse erreicht. Diese guten Ergebnisse können daran liegen, dass Shay Chen WAVSEP selbst entwickelt hat und so die Parameter für die Scans perfekt anpassen konnte.

Scanner mit Version	SQL Injection			XSS			Path Traversal		
	TP	FN	FP	TP	FN	FP	TP	FN	FP
	272		232	246		209	133		135
diese Arbeit									
Wapiti 3.0.4	41	231	0	34	212	0	8	125	0
	15,07%		0%	13,82%		0%	6,02%		0%
Arachni 2.0dev	136	136	0	153	93	0	15	118	0
	50%		0%	62,2%		0%	11,28%		0%
ZAP 3.11.0	157	115	1	137	109	0	15	118	0
	57,72%		0,43%	55,69%		0%	11,28%		0%
Benchmark Projekt									
ZAP vD-2016-09-05	158	114	3	71	175	0	0	133	0
	58,09%		1,29%	28,86%		0%	0%		0%
FindSecBugs v1.4.6	272	0	210	246	0	131	128	5	117
	100%		90,52%	100%		62,68%	96,24%		86,67%
Mburano und Si									
Arachni 1.5.1	136	136	5	157	89	0	0	133	0
	50%		2,16%	63,82%		0%	0%		0%
ZAP 2.7	158	114	8	186	60	0	0	133	0
	58,09%		3,45%	75,62%		0%	0%		0%

Tabelle 5.7: OWASP Benchmark Vergleich Ergebnisse mit anderen Untersuchungen

Idrissi et al. haben in ihrer Untersuchung gleichermaßen Scans gegen WAVSEP 1.5 u. a. mit Wapiti, ZAP, Arachni, IronWASP und W3AF durchgeführt [22]. Dabei konnten ähnlich gute Werte wie von Shay Chen erzielt werden. Bei der Detektion von SQL Injection-Schwachstellen wurden gleichermaßen von fast allen Scannern 100% der Schwachstellen erkannt. Nur W3AF hat 59,55% detektiert. Dafür sind auch einige FP aufgetreten. Da WAVSEP aber im Vergleich zu OWASP Benchmark nur sehr wenige FP-Testfälle mitbringt, ist es schwierig diese Ergebnisse einzuordnen. Eine größere Anzahl an Testfällen würde helfen dieses Verhalten besser bewerten zu können. Bei der Kategorie XSS unterscheidet sich die TPR von Arachni und ZAP nur jeweils um ca. 5%. Größere Unterschiede sind bei Path Traversal aufgetreten. In der Studie von Shay Chen sind dort mit Arachni 100%, bei Idrissi et al. 19,8% und in dieser Arbeit 25% der Schwachstellen detektiert wurden. In Tabelle 5.8 sind die Ergebnisse dieser Untersuchung mit der von Shay Chen und von Idrissi et al. gegenübergestellt.

Scanner mit Version	SQL Injection			XSS			Path Traversal		
	TP	FN	FP	TP	FN	FP	TP	FN	FP
	136		10	66		7	816		8
diese Arbeit									
Wapiti 3.0.4	116	20	0	24	42	0	294	522	0
	85,29%		0%	36,36%		0%	36,03%		0%
Arachni 2.0dev	90	46	0	46	20	20	204	612	0
	61,18%		0%	69,7%		0%	25%		0%
ZAP 3.11.0	113	20	0	66	0	0	480	0	0
	83,09%		0%	100%		0%	58,82%		0%
Shay Chen									
Wapiti 2.3.0	136	0	0	44	22	3	420	396	1
	100%		0%	66,67		42,86%	51,47		12,5%
Arachni 1.1	136	0	0	60	6	0	816	0	0
	100%		0%	90,91%		0%	100%		0%
ZAP 2.2.2	136	0	0	66	0	0	612	204	0
	100%		0%	100%		0%	75%		0%
SkipFish 2.10	104	22	0	62	4	0	672	144	2
	76,47%		0%	93,94%		0%	82,35%		25%
Vega 1.0	136	0	2	66	0	0	768	48	5
	100%		20%	100%		0%	94,12%		62,50%
Idrissi et al.									
Wapiti 2.3.0	136	0	2	44	22	3	414	402	1
	100%		20%	66,67%		42,85%	50,73		12,5%
Arachni 1.2.1	136	0	5	63	3	0	162	654	0
	100%		50%	95,45%		0%	19,85%		0%
ZAP 2.3.1	136	0	0	63	3	0	590	226	0
	100%		0%	95,45%		0%	72,3%		0%
IronWASP 0.9.7.1	136	0	5	52	14	0	288	528	1
	100%		50%	78,78%		0%	35,29%		12,5%
W3AF 1.2	81	55	3	19	47	3	461	355	1
	59,55%		30%	28,78%		42,85%	56,49%		12,5%

Tabelle 5.8: WAVSEP Vergleich Ergebnisse mit anderen Untersuchungen

Der Vergleich und die Analyse der Ergebnisse hat gezeigt, dass sich die betrachteten Open Source Scanner dafür eignen um Schwachstellen in Webanwendungen zu erkennen. Die Detektion ist dabei nicht bei allen Kategorien perfekt, aber ein Großteil der Schwachstellen konnte erkannt werden, ohne dabei viele Fehlalarme auszulösen. Die geringen Abweichungen mit den Vergleichsergebnissen sind dabei auf die unterschiedlichen

Parametern bei der Ausführung, den verschiedenen Versionen der Scanner und den jeweiligen Testumgebungen zurückzuführen.

6 Empfehlungen

Nach der Diskussion und Bewertung der Ergebnisse werden in diesem Abschnitt Empfehlungen vorgestellt, die eine erfolgreiche Durchführung von Schwachstellenscans wahrscheinlicher machen. Das heißt, eine hohe True Positive Rate und eine niedrige False Positive Rate erzielen. Zudem sollen Sicherheitsverantwortlichen Informationen für die Auswahl des für sie passenden Scanners mitgegeben werden. Dabei werden die Resultate, die im Abschnitt 5.4 diskutiert wurden, herangezogen. Die Ergebnisse dieser Arbeit haben gezeigt, dass es nicht den „einen Scanner“ für alle Probleme gibt. Vielmehr sollte der passende Scanner für die Kategorie von Schwachstelle, die untersucht werden soll, beispielsweise SQL Injection, XSS, Path Traversal, Remote File Inclusion oder Command Injection, ausgewählt werden.

Scanner	Schwachstellen Kategorie	Benchmark	++	+	0	-	--
Wapiti	SQL Injection	OWASP					x
		WAVSEP	x				
Arachni	SQL Injection	OWASP			x		
		WAVSEP		x			
ZAP	SQL Injection	OWASP			x		
		WAVSEP	x				
Wapiti	SQL Injection		1	-	-	-	1
Arachni	SQL Injection		-	1	1	-	-
ZAP	SQL Injection		1	-	1	-	-

Tabelle 6.1: Bewertung Scanner in der Kategorie SQL Injection

Es hat sich gezeigt, dass es sinnvoll ist, je nach Komplexität der Webanwendung, nur einen Teil dieser zu scannen. Das heißt, dass bei einem Scan nur ein Bruchteil der vorhandenen URLs berücksichtigt werden sollten und die weiteren URLs in anschließenden Scans. Zudem hat sich die Untersuchung auf nur einen Typ von Schwachstelle bewährt. Dafür bietet sich die zielgerichtete Aktivierung der jeweiligen Scan-Module der Schwachstellenscanner an. Durch die vorher genannten Maßnahmen kann die Laufzeit der einzelnen Scans stark reduziert werden. Darüber hinaus hat sich herausgestellt, dass ein

Scanner	Schwachstellen Kategorie	Benchmark	++	+	0	-	--
Wapiti	XSS	OWASP					x
		WAVSEP				x	
Arachni	XSS	OWASP		x			
		WAVSEP		x			
ZAP	XSS	OWASP			x		
		WAVSEP	x				
Wapiti	XSS		-	-	-	1	1
Arachni	XSS		-	2	-	-	-
ZAP	XSS		1	-	1	-	-

Tabelle 6.2: Bewertung Scanner in der Kategorie XSS

Scan mit weniger Anfragen pro Sekunde bessere Ergebnisse liefert. Der Webserver wird dadurch nicht überlastet und kann alle Anfragen beantworten, wodurch die Detektion von Schwachstellen besser ist.

Die Ergebnisse der Scanner sind in den Tabellen in diesem Abschnitt dargestellt. Dabei bedeutet ++ eine 80-100%ige, + eine 60-80%ige, **0** eine 40-60%ige, **-** eine 20-40%ige und **--** eine 0-20%ige Erkennungsrate (TPR). Im fortlaufenden Text wird dies mit sehr gut, gut, befriedigend, ausreichend und mangelhaft beschrieben.

In der Kategorie SQL Injection konnten alle drei getesteten Scanner gute bzw. sehr gute Ergebnisse gegen die Benchmark-Anwendung WAVSEP erzielen (siehe Tabelle 6.1). Arachni ist zwar in dieser Arbeit etwas abgefallen, hat jedoch in anderen Untersuchungen sehr gute Resultate erzielt [22, 12]. Beim OWASP Benchmark haben ZAP und Arachni mittelmäßig und Wapiti mangelhaft abgeschnitten. Die Resultate haben sich in den Vergleichsergebnissen bestätigt [24, 27]. Für diese Kategorie kann ZAP empfohlen werden.

Die Erkennung von XSS-Schwachstellen war gegen beide Benchmark-Anwendungen von den Scannern Arachni und ZAP besser als von Wapiti. Tabelle 6.2 ist zu entnehmen, dass Wapiti nur unzureichende bzw. mangelhafte Resultate erzielen konnte. Arachni hingegen hat gegen beiden Benchmark-Anwendungen gut und ZAP sehr gut und befriedigend abgeschlossen. Bei der Diskussion der Ergebnisse hat sich ZAP am geeignetsten für diesen Typ herausgestellt. Es muss aber erwähnt werden, dass die Ergebnisse, die Arachni bei diesem Typ erzielt hat, nur leicht hinter denen von ZAP lagen.

In der Kategorie Path Traversal hatten alle drei Scanner Schwierigkeiten, wie Tabelle 6.3 zu entnehmen ist. ZAP konnte bei WAVSEP etwas über die Hälfte der Schwachstellen erkennen und konnte so befriedigend abschließen. Alle anderen erzielten Ergebnisse der drei

Scanner	Schwachstellen Kategorie	Benchmark	++	+	0	-	--
Wapiti	Path Traversal	OWASP					x
		WAVSEP				x	
Arachni	Path Traversal	OWASP					x
		WAVSEP				x	
ZAP	Path Traversal	OWASP					x
		WAVSEP			x		
Wapiti	Path Traversal		-	-	-	1	1
Arachni	Path Traversal		-	-	-	1	1
ZAP	Path Traversal		-	-	1	-	1

Tabelle 6.3: Bewertung Scanner in der Kategorie Path Traversal

Scanner waren unzureichend oder mangelhaft. Wie schon in Abschnitt 5.4 erwähnt, hatten andere Untersuchungen ebenfalls Probleme bei der Identifizierung von Schwachstellen in OWASP Benchmark aufgezeigt. Arachni hatte in einer Untersuchung gegen WAVSEP, die von dem Entwickler des Benchmarks selber durchgeführt wurde, alle Schwachstellen erkannt [19]. Dies konnte in dieser Arbeit sowie in einer Studie, welche bei der Diskussion der Ergebnisse herangezogen wurde, nicht bestätigt werden [22]. Deshalb wird auch bei dieser Kategorie ZAP zur Detektion empfohlen.

Nach der Analyse der Ergebnisse dieser Arbeit hat sich gezeigt, dass sich ZAP bei den betrachteten Kategorien sehr gut zur Detektion von Schwachstellen eignet. Gleichzeitig treten nur sehr selten FPs auf. Es ist zusätzlich zu erwähnen, dass in Kategorien, die in dieser Arbeit nicht untersucht wurden, ZAP keine bzw. nur sehr wenige Schwachstellen erkennen konnte. Dies konnte bei den Voruntersuchungen zur Auswahl der richtigen Scanner und Benchmark-Anwendungen sowie den Ergebnissen der Vergleichsstudien, die in Abschnitt 5.4 zur Diskussion der Ergebnisse herangezogen wurden, beobachtet werden. Darüber hinaus haben die Abweichungen der Ergebnisse dieser Arbeit sowie den anderen betrachteten Studien hervorgebracht, dass beim Vorhandensein von genügend Ressourcen, der Einsatz eines weiteren Scanners sich vorteilhaft erweisen kann. Dadurch können Probleme bzw. Abweichungen bei der Erkennung kompensiert werden.

7 Zusammenfassung

7.1 Fazit

Das Ziel dieser Arbeit war es einen systematischen Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen durchzuführen. Dafür wurden mehrere Scanner gesichtet und nach einer Vorauswahl die populären Scanner Wapiti, Arachni und OWASP ZAP für diese Arbeit ausgewählt. Die Entscheidung fiel auf diese Scanner, da alle drei aktiv gepflegt und weiterentwickelt werden. Zudem können bei allen Reports in verschiedenen Formaten erstellt werden, wodurch manuelle und automatische Auswertungen möglich sind.

Für den Vergleich wurden Bewertungskriterien herausgearbeitet. Mit Hilfe dieser Kriterien konnten die Scanner gegenübergestellt werden. Als Grundlage der Bewertung wurden die Benchmark-Anwendungen WAVSEP und OWASP Benchmark ausgewählt, da beide eine Vielzahl von Schwachstellen in verschiedenen Kategorien anbieten und neben positiven auch negative Testfälle vorhanden sind. Darüber hinaus geben beide Benchmark-Anwendungen die genaue Anzahl der Testfälle, eine detaillierte Beschreibung und die genauen Pfade an. Für die Versuchsdurchführung wurde eine virtuelle Umgebung aufgesetzt, in der Scans gegen die Benchmark-Anwendungen durchgeführt wurden. Nach erfolgreichem Abschluss der Scanvorgänge wurden die Ergebnisse ausgewertet.

Zuerst wurden die Resultate bei OWASP Benchmark in den Schwachstellenkategorien SQL Injection, XSS, Path Traversal und Command Injection und bei WAVSEP in den Kategorien SQL Injection, XSS, Path Traversal und Remote File Inclusion bewertet und verglichen. Dabei wurden alle Kategorien der Benchmark-Anwendungen berücksichtigt, in denen die Scanner aussagekräftige Ergebnisse erzielt haben, um möglichst viele Daten für den Vergleich nutzen zu können. Anschließend wurden für den Vergleich der Ergebnisse der beiden Benchmark-Anwendungen die Schwachstellenkategorien SQL Injection,

XSS und Path Traversal herangezogen, da diese in beiden Anwendungen vorhanden waren. Angestrebt war eine hohe True Postive Rate (TPR) und eine niedrige False Positive Rate (FPR), um von einem guten Ergebnis eines Scans zu sprechen. Bei der Analyse der Resultate ist aufgefallen, dass alle drei Scanner bei der Erkennung von Schwachstellen in OWASP Benchmark Probleme hatten. Wapiti hat in den untersuchten Kategorien sogar nur mangelhaft abgeschnitten. ZAP konnte befriedigende und Arachni befriedigende sowie gute Ergebnisse bei SQL Injection und XSS erzielen. In der Katgorie Path Traversal haben ZAP und Arachni ebenfalls nur mangelhaft abgeschnitten. Bei WAVSEP konnten in den Kategorien XSS und SQL Injection hingegen gute bis sehr gute Ergebnisse von den Scannern erzielt werden. Nur Wapiti hat in der Kategorie XSS ausreichend abgeschlossen. Die Erkennung von Path Traversal-Schwachstellen fiel den Scannern bei WAVSEP ebenfalls schwer. Hier war das Ergebnis von ZAP befriedigend und von Arachni und Wapiti ausreichend. Abschließend wurden die Ergebnisse mit anderen Arbeiten verglichen, um die Ergebnisse einzuordnen. Die Resultate dieser Untersuchung waren dabei vergleichbar zu den anderen Studien.

Ein weiteres Ziel dieser Arbeit war es, Empfehlungen für Scanner in den untersuchten Kategorien abzugeben. Auf Grundlage der Ergebnisse dieser Untersuchung kann eine Empfehlung für ZAP für die Erkennung von SQL Injection-Schwachstellen gegeben werden. In der Kategorie XSS fiel die Empfehlung ebenfalls auf ZAP, wobei die Resultate von Arachni nur leicht dahinter lagen. Bei der Detektion von Path Traversal-Schwachstellen hatten alle drei Scanner ihre Probleme, wobei ZAP noch die besten Ergebnisse erzielt hat. Nach der Analyse der Ergebnisse in dieser Arbeit erwies sich der Einsatz von ZAP zum Auffinden von Schwachstellen als geeignet, wenn die Wahl auf einen einzigen Schwachstellenscanner fällt. Falls genug Ressourcen in einem Unternehmen vorhanden sind, sollten aber zwei Scanner eingesetzt werden, um eventuell fehlende Detektionsmechnismen eines Scanners abzufangen.

Darüber hinaus wurden Empfehlungen abgegeben, die eine erfolgreiche Durchführung von Schwachstellenscans gewährleisten sollen. Dazu gehört die Untersuchung einer Webanwendung bei einem Scanvorgang auf nur einen Typ von Schwachstelle. Besser wäre sogar nur einen Teil der verfügbaren URLs und nicht die ganze Anwendung in einem Durchlauf zu scannen. Ebenfalls hat sich ein etwas langsamerer Scan, weniger Anfragen pro Sekunde, als vorteilhaft erwiesen, um das Verwerfen oder sogar einen Zusammenbruch des Webservers zu verhindern. In einer Produktivumgebung wird deshalb von der Durchführung von Schwachstellenscans abgeraten. Diese sollten in abgesonderten Netzen einer Testumgebung erfolgen.

Bei der Durchführung dieser Studie muss beachtet werden, dass die Scans gegen zwei bekannte Benchmark-Anwendungen durchgeführt wurden. Den Entwicklern der Scanner waren die Testfälle somit bekannt. Zudem basieren die Testfälle auf Java Servlets und decken keine APIs oder Frameworks ab. Des Weiteren wurden, um den Rahmen dieser Untersuchung nicht zu sprengen, nur wenige Typen von Schwachstellen verglichen, weshalb keine allgemeine Empfehlung für einen Scanner abgegeben werden kann, sondern nur für die betrachteten Kategorien.

7.2 Ausblick

Da in dieser Arbeit nur wenige Kategorien von Schwachstellen untersucht wurden, sollten in anschließenden Untersuchungen weitere für die Bewertung betrachtet werden.

Da die Benchmark-Anwendungen begrenzte Testfälle aufweisen, kann die Betrachtung einer oder mehrerer realen Anwendungen weitere Erkenntnisse zu der Performance der Scanner beitragen. Dafür könnten gleichermaßen Open Source Anwendungen verwendet werden. Hierbei ist natürlich schwierig, ohne Kenntnisse möglicher Sicherheitslücken, einen Vergleich der Scanner durchzuführen. Dazu müsste die Anwendung anschließend oder vorher, beispielsweise mit statischen Codeanalysen, weiter untersucht werden. Eine weitere Möglichkeit ist die Verwendung einer älteren Version einer Webanwendung bei der Schwachstellen bekannt geworden sind. Dabei ist egal, ob es sich um eine Fremd- oder Eigenentwicklung handelt.

Die aufwendigste Möglichkeit ist es eine Benchmark-Anwendung oder wenigstens einige Testfälle selbst zu entwickeln. Der Vorteil, der sich hierbei ergibt ist, dass der Technologie-Stack, der im Unternehmen verwendet wird, dafür herangezogen werden kann. Der Aufwand könnte durch die Implementation von Testfällen in eine bereits fertige Anwendung reduziert werden. Damit kann der Scanner identifiziert werden, der für die eigenen Anwendungen die besten Ergebnisse liefert.

Literaturverzeichnis

[1] *A Redscan report NIST security vulnerability trends in 2020: an analysis.* https://www.redscan.com/media/Redscan_NIST-Vulnerability-Analysis-2020_v1.0.pdf. – [abgerufen am 29.05.2022]

[2] *bWAPP - an extremely buggy web app.* http://www.itsecgames.com/index.htm. – [abgerufen am 10.02.2022]

[3] *Common Weakness Enumeration - About CWE.* https://cwe.mitre.org/about/index.html. – [abgerufen am 13.04.2022]

[4] *Das Bundesamt für Sicherheit in der Informationstechnik (BSI): Die Lage der IT-Sicherheit in Deutschland 2021.* https://www.bmi.bund.de/SharedDocs/downloads/DE/publikationen/themen/it-digitalpolitik/bsi-lagebericht-cybersicherheit-2021.pdf. – [abgerufen am 10.05.2022]

[5] *Data Breach Investigations Report 2020.* https://www.verizon.com/business/resources/reports/2020/2020-data-breach-investigations-report.pdf. – [abgerufen am 29.05.2022]

[6] *Kali Linux.* https://www.kali.org. – [abgerufen am 26.04.2022]

[7] *OWASP Benchmark Project.* https://owasp.org/www-project-benchmark. – [abgerufen am 10.03.2022]

[8] *OWASP Zed Attack Proxy (ZAP).* https://www.zaproxy.org. – [abgerufen am 26.04.2022]

[9] *SAST, DAST, IAST, and RASP: how to choose?* https://www.ptsecurity.com/ww-en/analytics/knowledge-base/sast-dast-iast-and-rasp-how-to-choose/. – [abgerufen am 10.05.2022]

[10] *skipfish - web application security scanner.* https://code.google.com/archive/p/skipfish/. – [abgerufen am 10.02.2022]

[11] *Vega*. https://subgraph.com/vega/documentation/index.en.html. – [abgerufen am 10.02.2022]

[12] *WAVSEP - List of Tested Web Application Scanners*. http://sectoolmarket.com/list-of-tested-web-application-scanners-unified-list.html. – [abgerufen am 10.03.2022]

[13] Al Anhar, Azwar ; Suryanto, Yohan: Evaluation of Web Application Vulnerability Scanner for Modern Web Application. In: *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)* IEEE (Veranst.), 2021, S. 200–204

[14] Alsaleh, Mansour ; Alomar, Noura ; Alshreef, Monirah ; Alarifi, Abdulrahman ; Al-Salman, AbdulMalik: Performance-based comparative assessment of open source web vulnerability scanners. In: *Security and Communication Networks* 2017 (2017)

[15] Amankwah, Richard ; Chen, Jinfu ; Kudjo, Patrick K. ; Towey, Dave: An empirical comparison of commercial and open-source web vulnerability scanners. In: *Software: Practice and Experience* 50 (2020), Nr. 9, S. 1842–1857

[16] Antunes, Nuno ; Vieira, Marco: Assessing and comparing vulnerability detection tools for web services: Benchmarking approach and examples. In: *IEEE Transactions on Services Computing* 8 (2014), Nr. 2, S. 269–283

[17] Antunes, Nuno ; Vieira, Marco: On the metrics for benchmarking vulnerability detection tools. In: *2015 45th Annual IEEE/IFIP international conference on dependable systems and networks* IEEE (Veranst.), 2015, S. 505–516

[18] Arachni: *Arachni - Web Application Security Scanner Framework*. https://github.com/Arachni/arachni. – [abgerufen am 26.04.2022]

[19] Chen, Shay: *Evaluation of Web Application Vulnerability Scanners in Modern Pentest/SSDLC Usage Scenarios*. http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html. – [abgerufen am 10.02.2022]

[20] digininja: *Damn Vulnerable Web Application (DVWA)*. https://github.com/digininja/DVWA. – [abgerufen am 10.02.2022]

[21] Druin, Jeremy: *Introduction to the OWASP Mutillidae II Web Pen-Test Training Environment*. https://sansorg.egnyte.com/dl/fcnjG7IQE6. 2021. – [abgerufen am 29.05.2022]

[22] Idrissi, SE ; Berbiche, N ; Guerouate, F ; Shibi, M: Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. In: *International Journal of Applied Engineering Research* 12 (2017), Nr. 21, S. 11068–11076

[23] Makino, Yuma ; Klyuev, Vitaly: Evaluation of web vulnerability scanners. In: *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)* Bd. 1 IEEE (Veranst.), 2015, S. 399–402

[24] Mburano, Balume ; Si, Weisheng: Evaluation of web vulnerability scanners based on owasp benchmark. In: *2018 26th International Conference on Systems Engineering (ICSEng)* IEEE (Veranst.), 2018, S. 1–6

[25] McQuade, Kinnaird: Open source web vulnerability scanners: the cost effective choice? In: *Proceedings of the Conference for Information Systems Applied Research* Bd. 2167, 2014, S. 1508

[26] Nunes, Paulo ; Medeiros, Ibéria ; Fonseca, José C ; Neves, Nuno ; Correia, Miguel ; Vieira, Marco: Benchmarking static analysis tools for web security. In: *IEEE Transactions on Reliability* 67 (2018), Nr. 3, S. 1159–1175

[27] OWASP Benchmark: *OWASP Benchmark Project*. https://github.com/OWASP-Benchmark/BenchmarkJava. – [abgerufen am 10.02.2022]

[28] OWASP Foundation: *OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks*. https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_(en).pdf.pdf. – [abgerufen am 10.02.2022]

[29] OWASP Foundation: *OWASP Top 10 - 2021*. https://owasp.org/Top10/A00_2021_Introduction/. – [abgerufen am 10.02.2022]

[30] Pan, Yuanyuan: Interactive application security testing. In: *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)* IEEE (Veranst.), 2019, S. 558–561

[31] Parizi, Reza M. ; Qian, Kai ; Shahriar, Hossain ; Wu, Fan ; Tao, Lixin: Benchmark requirements for assessing software security vulnerability testing tools. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* Bd. 1 IEEE (Veranst.), 2018, S. 825–826

[32] Parvez, Muhammad ; Zavarsky, Pavol ; Khoury, Nidal: Analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)* IEEE (Veranst.), 2015, S. 186–191

[33] Sagar, Deepika ; Kukreja, Sahil ; Brahma, Jwngfu ; Tyagi, Shobha ; Jain, Prateek: Studying open source vulnerability scanners for vulnerabilities in web applications. In: *IIOAB JOURNAL* 9 (2018), Nr. 2, S. 43–49

[34] Sarpong, Pious ; Larbi, Lawrence ; Paa, Daniel ; Abdulai, Issah ; Amankwah, Richard ; Amponsah, Akwasi: Performance Evaluation of Open Source Web Application Vulnerability Scanners based on OWASP Benchmark. In: *International Journal of Computer Applications* 174 (2021), 02, S. 15–22

[35] sectooladdict: *The Web Application Vulnerability Scanner Evaluation Project - WAVSEP*. `https://github.com/sectooladdict/wavsep`. – [abgerufen am 02.05.2022]

[36] sullo: *Nikto web server scanner*. `https://github.com/sullo/nikto/`. – [abgerufen am 10.02.2022]

[37] Surribas, Nicolas: *Wapiti - The web-application vulnerability scanner*. `https://wapiti-scanner.github.io/`. – [abgerufen am 10.02.2022]

[38] Vieira, Marco ; Antunes, Nuno ; Madeira, Henrique: Using web security scanners to detect vulnerabilities in web services. In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks* IEEE (Veranst.), 2009, S. 566–571

[39] webpwnized: *OWASP Mutillidae II*. `https://github.com/webpwnized/mutillidae`. – [abgerufen am 10.02.2022]

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

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

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: [redacted]

Vorname: [redacted]

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

Vergleich von Open Source Scannern zum Auffinden von Schwachstellen in Webanwendungen

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

Ort	Datum	Unterschrift im Original