

Hochschule Heilbronn  
Fakultät für Informatik

Bachelorarbeit

# **Evaluation von Web Application Vulnerability Scannern**

**vorgelegt an der Hochschule Heilbronn, Fakultät für Informatik zum Abschluss  
eines Studiums im Studiengang Angewandte Informatik**

Henning Janning

Matrikelnummer: 192972

Eingereicht am: 05.04.2019

Erstprüfer: Prof. Andreas Mayer  
Zweitprüferin: Susanne Steuer (M.Sc.)

---

## **Zusammenfassung**

In dieser Arbeit werden aktuelle Web Vulnerability Scanner (WVS) auf ihren Umfang und ihre Tauglichkeit überprüft und verglichen. Jeder WVS wird an verschiedenen Webseiten getestet, die absichtlich eingebaute Schwachstellen haben, wie z.B. Juice-Shop oder Damn Vulnerable Web Application. Kriterien für die Bewertung der Tools sind die Anzahl der gefundenen Schwachstellen, Anzahl der False Positives und die daraus resultierende Trefferquote. Zudem fließen subjektive Eindrücke wie Handhabung oder intuitive Bedienung in die Bewertung mit ein.

## **Abstract**

TODO

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>v</b>
<b>Tabellenverzeichnis</b>	<b>vi</b>
<b>1 Einführung</b>	<b>1</b>
<b>2 Grundlagen</b>	<b>5</b>
2.1 Webanwendung . . . . .	5
2.2 Schwachstellen . . . . .	7
2.3 Web Application Security . . . . .	10
2.3.1 Bedrohungen und Risiken . . . . .	10
2.3.2 Sicherheitsmaßnahmen . . . . .	10
2.3.2.1 Security Tests . . . . .	11
2.3.2.2 Penetration Testing . . . . .	13
2.3.2.3 Web Application Firewalls (WAFs) . . . . .	15
2.4 Funktionsweise WVS . . . . .	17
<b>3 Methodik</b>	<b>19</b>
3.1 Testaufbau . . . . .	19
3.2 Web Application Vulnerability Scanner (WVS) . . . . .	20
3.2.1 Auswahlkriterien . . . . .	20
3.2.2 Ausgewählte WVS . . . . .	21
3.2.2.1 Free und Open Source WVS . . . . .	21
3.2.2.2 Kommerzielle WVS . . . . .	21
3.2.3 Nicht ausgewählte WVS . . . . .	22
3.2.3.1 Free und Open Source WVS . . . . .	22
3.2.3.2 Kommerzielle WVS . . . . .	23
3.3 Verwundbare Web-Applikationen . . . . .	24
<b>4 Evaluation</b>	<b>25</b>
4.1 Gefundene Schwachstellen per Webanwendung . . . . .	25
4.2 Bedienung, Reporting und Geschwindigkeit . . . . .	26
4.2.1 Open Source . . . . .	26
4.2.2 Kommerziell . . . . .	29
4.2.3 Gesamtbewertung und Ranking . . . . .	33

<b>5</b>	<b>Diskussion</b>	<b>34</b>
5.1	Open Source WVS . . . . .	34
5.2	Kommerzielle WVS . . . . .	34
5.3	Open Source vs. kommerziell . . . . .	34
5.4	Empfehlung . . . . .	34
<b>6</b>	<b>Fazit und Ausblick</b>	<b>35</b>
	<b>Quellenverzeichnis</b>	<b>36</b>
	<b>Anhang</b>	<b>39</b>
1	Verwendete Abkürzungen . . . . .	39
2	OWASP: The Open Web Application Security Project . . . . .	41
2.1	Die OWASP Top Ten . . . . .	42
2.1.1	A1: Injection . . . . .	42
2.1.2	A2: Fehler in der Authentifizierung . . . . .	42
2.1.3	A3: Verlust der Vertraulichkeit sensibler Daten . . . . .	43
2.1.4	A4: XML External Entities . . . . .	43
2.1.5	A5: Fehler in der Zugriffskontrolle . . . . .	43
2.1.6	A6: Sicherheitsrelevante Fehlkonfiguration . . . . .	43
2.1.7	A7: Cross-Site-Scripting (XSS) . . . . .	44
2.1.8	A8: Unsichere Deserialisierung . . . . .	44
2.1.9	A9: Nutzung von Komponenten mit bekannten Schwachstellen . . . . .	44
2.1.10	A10: Unzureichendes Logging und Monitoring . . . . .	44
	<b>Eidesstattliche Erklärung</b>	<b>45</b>

# Abbildungsverzeichnis

1.1	Gefundene Schwachstellen pro Jahr laut CVE Details . . . . .	2
1.2	Bitkom Studie - Bedrohungsszenarien . . . . .	2
2.1	3-Tier-Architektur einer Webanwendung . . . . .	5
2.2	Enterprise Webanwendung auf Basis einer Microservice-Architektur . . .	6
2.3	Lebenszyklus einer Schwachstelle . . . . .	9
3.1	Testaufbau . . . . .	20
3.2	W3af: Fehlende Module . . . . .	23
4.1	Arachni WebUI 0.5.12 . . . . .	26
4.2	OpenVAS im Greenbone Security Assistant . . . . .	27
4.3	Grafische Benutzeroberfläche von ZAP . . . . .	28
4.4	Weboberfläche von Acunetix . . . . .	29
4.5	Dashboard von BurpSuite Pro . . . . .	30
4.6	Confidence-Tabelle im Report von BurpSuite Pro . . . . .	31
4.7	Grafische Benutzeroberfläche von Nessus . . . . .	31
4.8	Grafische Benutzeroberfläche von Netsparker . . . . .	32

# Tabellenverzeichnis

3.1	Ausgewählte Free und Open Source WVS . . . . .	21
3.2	Ausgewählte Kommerzielle WVS . . . . .	21
4.1	Gefundene Schwachstellen der Open Source WVS . . . . .	25
4.2	Gefundene Schwachstellen der kommerziellen WVS . . . . .	25
4.3	Bewertung der Open Source WVS . . . . .	33
4.4	Bewertung der kommerziellen WVS . . . . .	33
4.5	Ranking aller WVS . . . . .	33

# 1 Einführung

In den letzten zwei Jahrzehnten hat sich das World Wide Web von einem reinen Informationsspeicher in eine Plattform mit hochfunktionalen Anwendungen verwandelt, die nicht nur sensible Daten verarbeiten sondern auch Aktionen durchführen, die einflussreiche Auswirkungen auf die reale Welt haben. Mit jeder Weiterentwicklung bringen Webanwendungen neue Sicherheitslücken mit sich und so verändern sich auch die Art und die Anzahl der am häufigsten auftretenden Fehler. Es gibt Angriffe auf Schwachstellen, die bei der Entwicklung der Webanwendungen noch nicht bekannt waren und daher nicht berücksichtigt wurden, andere Attacken haben an Bedeutung verloren, da das Bewusstsein für sie gestiegen ist oder aufgrund von Verbesserungen der Web-Browser Software. Neue Technologien bergen jedoch auch immer das Risiko von neuen Sicherheitslücken und so ist "...in gewissem Maße die Sicherheit von Webanwendungen heute das bedeutendste Schlachtfeld zwischen Angreifern und solchen, die Daten schützen und Computerressourcen verteidigen müssen, und dies wird wahrscheinlich auf absehbare Zeit so bleiben." [1]

Mit der Anzahl der Webanwendungen steigt auch der Einfluss des World Wide Webs auf alle Lebensbereiche, sei es beim Einkaufen im Online-Shop einschließlich diverser Bezahlssysteme, dem Nachrichtenaustausch über Social-Media-Kanäle oder nur zur Informationsgewinnung auf Nachrichtenseiten - die Gesellschaft ist mehr denn je abhängig von der immer größer werdenden Anzahl an verschiedenen Webanwendungen im Internet. Fortschreitende Digitalisierung und weltweite Vernetzung verursachen jedoch auch immer mehr Sicherheitslücken: laut CVE Details hat sich die Anzahl der gefundenen Schwachstellen in den letzten zwei Jahren mehr als verdoppelt (siehe Abb. 1.1).

Zunehmende Cyberangriffe von kriminellen Hackern aber auch von politisch oder ideologisch motivierten Angreifern sind die Folge. Aktuelle Beispiele sind der Angriff auf die Münchner Firma Krauss Maffai im Dezember 2018, der die Produktion des Maschinenbauunternehmens für mehrere Tage lahmlegte oder die Attacke auf das Datennetz des Deutschen Bundestags im Oktober 2018. Das Datenleck „Collection #1“, das im Januar 2019 auftauchte, ist das Ergebnis einer Vielzahl von Cyberattacken, es enthält über 2,6 Milliarden Datensätze mit Zugangsdaten und Passwörtern für hunderte Webseiten.

Für die betroffenen Firmen ist der Schaden enorm: Laut einer Studie des Digitalverbands Bitkom lag der durch Cyberattacken verursachte wirtschaftliche Gesamtschaden für Industrieunternehmen in Deutschland innerhalb der letzten zwei Jahre bei über 43 Milli-

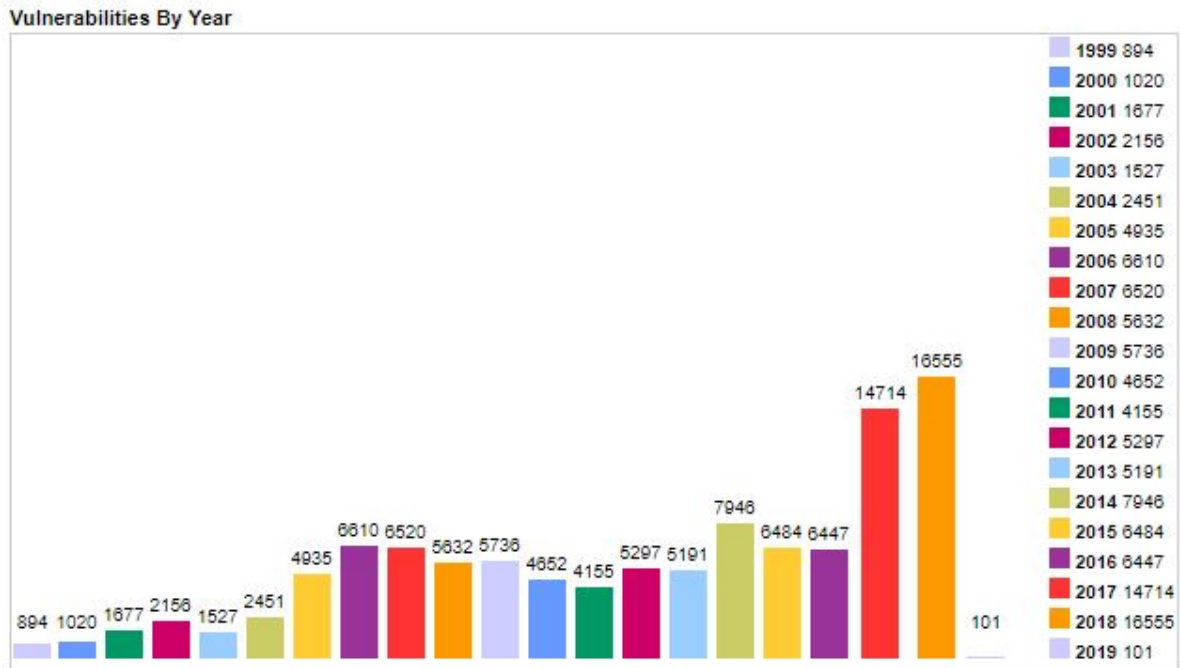
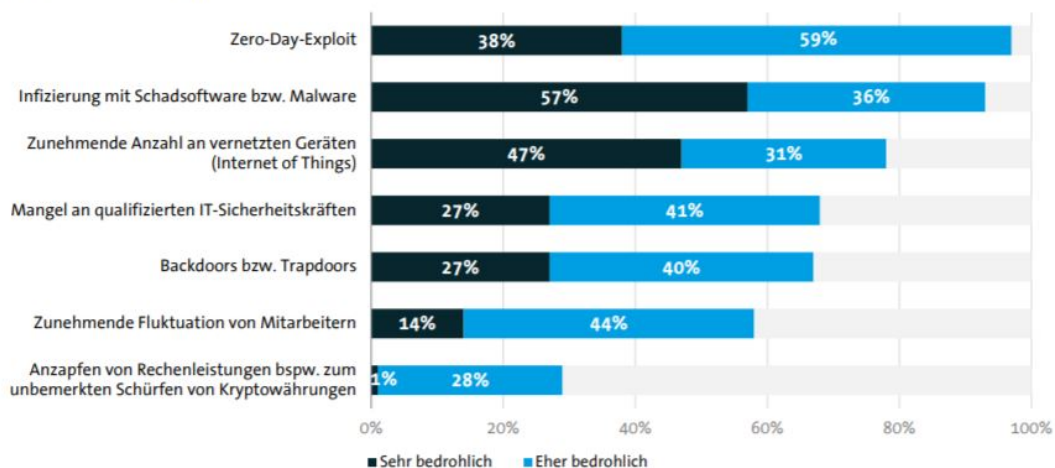


Abbildung 1.1: Gefundene Schwachstellen pro Jahr laut CVE Details [2]

arden Euro [3]. Aus der gleichen Studie geht hervor, dass unentdeckte Sicherheitslücken das größte Risiko darstellen (siehe Abb. 1.2)

## Unentdeckte Sicherheitslücken als größte Bedrohung

Inwieweit betrachten Sie die folgenden Szenarien als zukünftige Bedrohung für die IT-Sicherheit Ihres Unternehmens?



12 Basis: Alle befragten Industrieunternehmen (n=503) | Quelle: Bitkom Research

bitkom

Abbildung 1.2: Bitkom Studie - Bedrohungsszenarien [3]



Es gibt mehrere Ansätze, diesem Risiko zu begegnen: Grundsätzlich sollte der Entwickler einer Webseite von Beginn an eine Programmierung anstreben, die bereits bekannte Sicherheitslücken vermeidet und so potentiellen Angreifern möglichst wenig Angriffsfläche bietet. Hier bietet das Bundesamt für Sicherheit in der Informationstechnik (BSI) mit seinem “Leitfaden zur Entwicklung sicherer Webanwendungen” [4] Hilfestellung. Fehler in der Programmierung lassen sich jedoch nicht immer ausschließen, zudem ist auch ein Schutz gegen unentdeckte Sicherheitslücken vonnöten.

Neben dem Einsatz von Web Application Firewalls (“Web Shields”), die den Datenstrom zwischen Browser und Webapplikation überwachen, ist die Verwendung von Web Application Vulnerability Scannern (WVS) im Rahmen von Penetration Tests ein wesentlicher Bestandteil von Sicherheitskonzepten in diesem Bereich.

WVS überprüfen Webanwendungen automatisiert auf Schwachstellen und unterstützen dadurch den Penetration Tester beim Aufspüren von Sicherheitslücken.

Die vorliegende Arbeit macht sich die Evaluation von WVS zur Aufgabe, da die wenigen bisherigen Ausarbeitungen zu diesem Thema veraltet sind und teilweise andere Ansätze verfolgen:

- Holm [5] vergleicht in seiner Studie aus dem Jahr 2011 lediglich kommerzielle Scanner und verzichtet auf Bewertungskategorien wie Bedienung und Reporting.
- Wundram [6][7] behandelt das Thema zweimal, in seinen Vergleichen von 2011 und 2012 finden sich veraltete Programme wie Watobo oder das inzwischen nicht mehr lauffähige W3af.

Zusätzliche Motivation ist die große Anzahl und Vielfalt der Scanner: Das Open Web Application Security Project (OWASP) listet allein eine Sammlung von 50 verschiedenen WVS auf [8], es gibt freie, Open Source und kommerzielle WVS jeweils für verschiedene Plattformen, reine Terminal-Anwendungen und Programme mit grafischer Benutzeroberfläche. Das Ziel dieser Arbeit ist es, aus dieser bunten Mischung diejenigen WVS herauszufiltern, für die eine nähere Betrachtung lohnenswert erscheint und diese im Hinblick auf folgende Fragestellungen zu evaluieren:

- 1. Wieviele Schwachstellen werden von den WVS gefunden?**
- 2. Wie unterscheiden sich die WVS in den Kategorien Bedienung, Reporting und Scan-Geschwindigkeit?**
- 3. Welcher WVS schneidet insgesamt am besten ab?**
- 4. Gibt es Unterschiede zwischen Open Source- und kommerziellen WVS?**

Die Evaluation soll mit Hilfe eines Punktesystems erfolgen und schließlich in einem Ranking resultieren, das die Tauglichkeit und Qualität der getesteten WVS anhand ihrer Platzierungen aufzeigt.

Die Arbeit ist in sechs Kapitel aufgeteilt:

**1. Einführung:** In der Einführung wird der Leser an das Thema herangeführt, die Motivation und die Zielsetzung sowie die zentralen Fragestellungen und der Aufbau der Arbeit werden erläutert.

**2. Grundlagen:** Das zweite Kapitel vermittelt Grundlagen über Webanwendungen, Schwachstellen, Sicherheitsmaßnahmen wie Web Application Firewalls und Penetration-testing sowie die Arbeitsweise von WVS.

**3. Methodik:** Hier wird der Testaufbau beschrieben einschließlich der Auswahlkriterien für die WVS und des Punktesystems für die Bewertung. Nicht berücksichtigte WVS werden beschrieben sowie die verwundbaren Webanwendungen, die für die Tests verwendet wurden.

**4. Evaluation:** Dieses Kapitel beinhaltet die Ergebnisse der Evaluation, zum einen die Anzahl der gefundenen Schwachstellen per WVS, zum anderen eine Beschreibung der WVS einschließlich der Bewertung in den Kategorien Bedienung, Reporting und Scangeschwindigkeit.

**5. Diskussion:** In diesem Kapitel werden die Ergebnisse der Evaluation durchleuchtet und erörtert.

**6. Fazit und Ausblick:** Zum Schluss wird die Studie nochmals zusammengefasst und auf mögliche zukünftige Arbeiten verwiesen.

Dieser Arbeit wurde eine CD-ROM beigelegt, die sämtliche Berichte aller durchgeführten Scans enthält.

## 2 Grundlagen

### 2.1 Webanwendung

Zu Beginn sollte der Begriff “Webanwendung” geklärt werden. Eine Webanwendung muss nicht zwingend über das World Wide Web erreichbar sein, auch in vielen Unternehmen kommen Webanwendungen zum Einsatz. Entscheidend dafür, dass sich eine Anwendung als Webanwendung bezeichnen lässt, ist stattdessen einzig der Einsatz von Webtechnologien. Hierüber gelangen wir zu folgender Begriffsdefinition [9]:

Eine Webanwendung ist eine Client-Server-Anwendung, die auf Webtechnologien (HTTP, HTML etc.) basiert.

Eine Webanwendung wird über einen Webbrowser aufgerufen, der den serverseitig bereitgestellten HTML-, Java-Script oder CSS-Code interpretiert und darstellt. Daneben kann auch über ein Skript oder von einer Kommandozeile aus auf Webanwendungen zugegriffen werden, man spricht hier von einem User Agent oder Client. Zur Kommunikation zwischen Browser (also Client) und Server wird das HTTP-Protokoll verwendet oder das darauf aufsetzende HTTPS-Protokoll. Serverseitig werden Webanwendungen auf Web- und Applikationsservern oder Laufzeitumgebungen ausgeführt, die dann wiederum auf Hintergrundsysteme wie Datenbanken zugreifen können.

Daraus ergibt sich eine sogenannte 3-Tier-Architektur (dreischichtige Architektur, siehe Abb. 2.1).

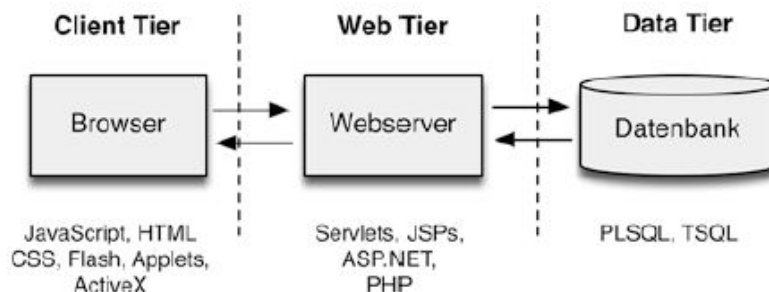


Abbildung 2.1: 3-Tier-Architektur einer Webanwendung [9]

Moderne Webanwendungen lassen sich jedoch - insbesondere im Enterprise-Umfeld - nicht als einzelne Anwendungen sehen, sondern als Zusammenschluss verschiedener eigenständiger Dienste wie REST- oder Microservices zu einer Plattform, wie z.B. einem Onlineshop (siehe Abb. 2.2).

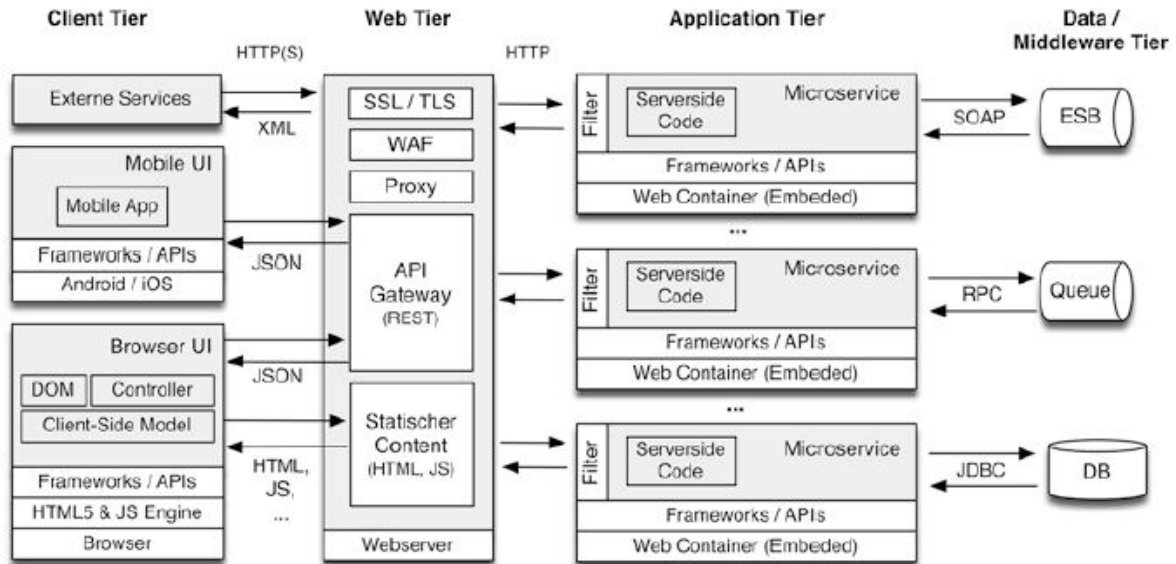


Abbildung 2.2: Enterprise Webanwendung auf Basis einer Microservice-Architektur [9]

Im Zuge der Weiterentwicklung von Webanwendungen werden immer mehr Aspekte der Benutzerschnittstelle client-seitig, vor allem über JavaScript-Code umgesetzt, der im Hintergrund serverseitige Webdienste aufruft. Diese Verlagerung von Anwendungslogik vom Server auf den Client findet ihren Höhepunkt in sogenannten Single Page Applications (SPAs), die nur noch aus einer einzigen HTML-Seite mit sehr viel JavaScript-Code bestehen, der im Hintergrund dynamisch mit serverseitigen REST-Services kommuniziert und die Anzeige von Seiteninhalten steuert. Ein bekanntes Beispiel für solch eine Single Page Application ist Google Mail. [9]

## 2.2 Schwachstellen

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) definiert eine Schwachstelle wie folgt:

“Eine Schwachstelle (englisch “vulnerability”) ist ein sicherheitsrelevanter Fehler eines IT-Systems oder einer Institution. Ursachen können in der Konzeption, den verwendeten Algorithmen, der Implementation, der Konfiguration, dem Betrieb sowie der Organisation liegen. Eine Schwachstelle kann dazu führen, dass eine Bedrohung wirksam wird und eine Institution oder ein System geschädigt wird. Durch eine Schwachstelle wird ein Objekt (eine Institution oder ein System) anfällig für Bedrohungen.” [10]

Viele Schwachstellen werden nicht öffentlich bekannt, da ein Hersteller sie im Idealfall selbst entdeckt und behebt, bevor ein Dritter sie finden und insbesondere ein Angreifer sie ausnutzen kann. Wird eine sicherheitskritische Schwachstelle jedoch nicht durch den Hersteller, sondern durch einen Dritten gefunden, gibt es unterschiedliche Möglichkeiten, wie ihr Lebenszyklus und die daraus resultierende Gefährdung für den Nutzer verlaufen können. Wichtig sind dabei Rolle und Selbstverständnis des Entdeckers. Neben Personen, die sich beruflich mit IT-Sicherheit beschäftigen, kann beispielsweise auch akademische Forschung zur Entdeckung von Schwachstellen beitragen. Es kann sich um gezielte Suche nach Schwachstellen handeln, aber auch Zufallsfunde sind möglich. Der Lebenszyklus hängt daher stark von Verhalten und Motivation des Entdeckers sowie von der Reaktion des Herstellers ab [11]:

- Ist dem Entdecker daran gelegen, die IT-Sicherheit zu verbessern, oder möchte er die Schwachstelle für eigene, möglicherweise kriminelle Zwecke nutzen?
- Wie viele Informationen über die Schwachstelle werden öffentlich bekannt?
- Schätzt der Hersteller die entstehende Gefährdung richtig ein und kann er schnell genug Abhilfe schaffen?

Um die Diskussion über Schwachstellen zwischen Entdeckern, Herstellern und weiteren Beteiligten zu systematisieren, wurde zur einheitlichen Benennung von öffentlich bekannten Schwachstellen und zur Sammlung der darüber verfügbaren Informationen mit den Common Vulnerabilities and Exposures (CVE) ein herstellerübergreifender Industriestandard geschaffen. Mit diesem Standard wird sichergestellt, dass alle Beteiligten während des gesamten Lebenszyklus tatsächlich jeweils dieselbe Schwachstelle meinen. Zudem werden statistische Auswertungen ermöglicht.

Grundsätzlich basiert der Lebenszyklus einer durch Dritte entdeckten Schwachstelle auf dem folgenden Schema [11]:

1. Die Schwachstelle wird durch einen Dritten entdeckt und untersucht.
2. Der Hersteller erlangt auf einem der folgenden Wege Kenntnis von der Schwachstelle:
  - Der Entdecker veröffentlicht sämtliche Informationen über die Schwachstelle („Full Disclosure“).
  - Der Entdecker informiert den Hersteller direkt und verzichtet zunächst auf eine detaillierte Veröffentlichung („Coordinated Disclosure“).
  - Die Schwachstelle wird für Angriffe ausgenutzt und diese werden entdeckt („Zero-Day-Exploit“).
  - Der Hersteller wird indirekt über einen sogenannten Schwachstellen-Broker über die Schwachstelle informiert.
3. Der Hersteller beginnt mit der Entwicklung eines Patches (von engl. Flicker), einer Nachbesserung der Software, mit der die Schwachstelle behoben wird.
4. Der Hersteller veröffentlicht eine Schwachstellenwarnung, ein sogenanntes Advisory. Dieses enthält üblicherweise allgemeine Informationen zur Schwachstelle, eine Bewertung der entstehenden Gefährdung sowie mögliche vorläufige Gegenmaßnahmen (sogenannte Workarounds oder Mitigations). Ein Advisory wird vor allem dann schon vor Verfügbarkeit des Patches veröffentlicht, wenn auch die Öffentlichkeit bereits Kenntnis von der Schwachstelle hat und die Gefährdung hoch ist. Häufig erscheinen Advisory und Patch jedoch zeitgleich.
5. Der Hersteller stellt zusammen mit einer entsprechenden Beschreibung (Bulletin) einen Patch zur Behebung der Schwachstelle bereit. Durch Analyse des Patches kann ein Angreifer unter Umständen so viele Details über die Schwachstelle herausfinden, dass er sie ausnutzen kann. Daher steigt mit der Veröffentlichung des Patches die Gefährdung für ungepatchte Systeme an.
6. Der Benutzer der betroffenen Software installiert den Patch, schließt dadurch die Schwachstelle und ist erst dann vor ihrer Ausnutzung geschützt. Dies unterstreicht die enorme Bedeutung eines effektiven Patchmanagements.

Abweichungen von diesem Schema sind möglich, der Ablauf kann sich dynamisch ändern. So kann ein Hersteller beispielsweise eine Schwachstelle als unkritisch einschätzen und auf die Entwicklung eines Patches zunächst verzichten. Wird zu einem späteren Zeitpunkt doch eine Ausnutzungsmöglichkeit bekannt, so kann der Hersteller dadurch zu einer entsprechenden schnellen Reaktion gezwungen sein. [11]

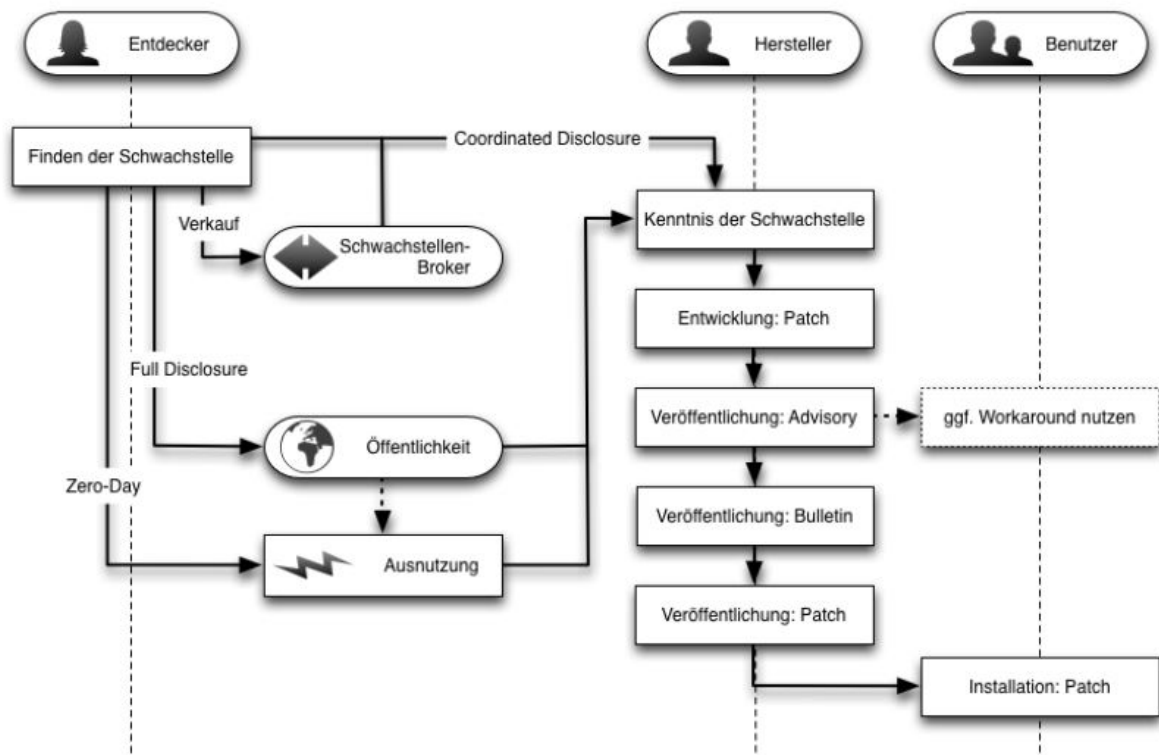


Abbildung 2.3: Lebenszyklus einer Schwachstelle [11]

## 2.3 Web Application Security

Die Webanwendungssicherheit ist ein Teilgebiet der IT-Sicherheit und befasst sich vor allem mit dem Schutz von Assets einer Webanwendung. Unter Assets versteht man in diesem Fall all jene Bestandteile (Daten, Systeme und Funktionen) einer Webanwendung, für die sich ein Schutzbedarf hinsichtlich eines der drei primären Schutzziele ableiten lässt. Diese primären Schutzziele lauten:

- Vertraulichkeit
- Integrität
- Verfügbarkeit

Aus diesen primären Schutzzielen lassen sich auch sekundäre Schutzziele wie z.B. Authentizität oder Nicht-Abstreitbarkeit ableiten.

Das Thema Webanwendungssicherheit bezieht sich auf alle Phasen des Lebenszyklus einer Webanwendung und stellt auch einen Bestandteil der Qualitätssicherung dar. Sie befasst sich sowohl mit der Abwehr von Bedrohungen und der Prävention von Schwachstellen als auch mit der Identifizierung und Behebung von Sicherheitslücken. [4]

### 2.3.1 Bedrohungen und Risiken

Die schwerwiegendsten Angriffe auf Webanwendungen sind sicherlich diejenigen, die sensible Daten verfügbar machen oder uneingeschränkten Zugriff auf die Systeme ermöglichen, auf denen die Anwendung ausgeführt wird. Hierzu gehören Angriffe per XSS oder SQL-Injection, aber auch Angriffe, die Systemausfälle verursachen (Denial-of-Service Attacken) stellen für viele Organisationen ein sehr kritisches Ereignis dar.

OWASP stellt in regelmäßigen Abständen seine OWASP Top Ten vor, eine Liste mit den 10 kritischsten Sicherheitsrisiken für Webanwendungen. Die aktuelle Version aus dem Jahr 2017 befindet sich im Anhang dieser Ausarbeitung.

### 2.3.2 Sicherheitsmaßnahmen

Das BSI definiert den Begriff wie folgt:



“Als Sicherheitsmaßnahmen werden alle Aktionen bezeichnet, die dazu dienen, Sicherheitsrisiken zu steuern und diesen entgegenzuwirken. Dies schließt sowohl organisatorische, als auch personelle, technische oder infrastrukturelle Sicherheitsmaßnahmen ein. Synonym werden auch die Begriffe Sicherheitsvorkehrung oder Schutzmaßnahme benutzt.” [10]

Zu den Maßnahmen, die bereits während der Entwicklung ergriffen werden sollten, gehören umfangreiche Security Tests. Nach Inbetriebnahme der Webanwendung kann mit Hilfe von Penetration Tests nach Sicherheitslücken gesucht werden, zusätzlichen Schutz bieten Web Application Firewalls.

### 2.3.2.1 Security Tests

Bevor eine Webanwendung ausgeliefert wird, sollte sie im Zuge eines oder mehrerer Security Assessments auf Sicherheitsmängel getestet werden. Ein Security Assessment soll dabei sicherstellen, dass die definierten Sicherheitsanforderungen in der Implementierungsphase korrekt umgesetzt worden sind. Zusätzlich soll in diesem Rahmen auch die Effektivität der gesetzten Sicherheitsanforderungen überprüft werden und so mögliche Unzulänglichkeiten in der Spezifikation selbst aufgedeckt werden. Das heißt, es soll sichergestellt werden, dass die spezifizierten Sicherheitsanforderungen auch angemessen und wirksam sind. Üblicherweise durchlaufen Security Assessments die folgenden Phasen [4]:

- **Planung:** Ein Security Assessment benötigt eine initiale Planung. Hier werden sämtliche Informationen, die für die Durchführung des Assessments notwendig sind, zusammengetragen.
- **Durchführung:** Die Hauptaufgabe dieser Phase ist das Suchen nach Schwachstellen und deren Bewertung.
- **Auswertung:** Hierbei werden die gefundenen Schwachstellen analysiert. Es werden die Ursachen und die Gegenmaßnahmen bestimmt und in einem Abschlussbericht zusammengefasst. Der Abschlussbericht wird an die Entwickler geleitet, die die identifizierten Schwachstellen beheben. Nach der Behebung werden die Schwachstellen erneut untersucht.

Die Überprüfung kann unter Verwendung mehrerer verschiedener Testverfahren erfolgen. Dabei ist zu berücksichtigen, dass unterschiedliche Testverfahren unterschiedliche Aufgaben erfüllen und unterschiedliche Ergebnisse liefern. Beispielsweise überprüfen Unit Tests die Korrektheit und die vollständige Abdeckung einer Funktionalität, während ein Code Review die korrekte Implementierung überprüft. Je nachdem was genau überprüft werden soll, können auch unterschiedliche Testrollen herangezogen werden. Ein Auditor

kann beispielsweise die Einhaltung von Vorgaben überprüfen, während ein Penetrationstester die korrekte Implementierung überprüft.

Folgende Testverfahren können bereits während der Implementierung durchgeführt werden [4]:

- **Security Test Cases:** Ein Test Case beschreibt einen Softwaretest, in dem die laut den Sicherheitsanforderungen bestimmte spezifizierte Funktionalität (funktional oder nicht funktional) der Webanwendung auf ihre Korrektheit überprüft wird. Test Cases einer Webanwendung werden hierbei in einfacher Sprache beschrieben. Ein Test Case besteht immer zumindest aus einer eindeutigen ID, einer Beschreibung und einem erwarteten Ergebnis. Im Zuge des Tests muss die Einhaltung des erwarteten Ergebnisses überprüft werden. Test Cases lassen sich in folgende Arten unterteilen:

- **Positivtest:** Das Verhalten der Webanwendung wird mit gültigen Rahmenbedingungen und Eingaben überprüft.

- **Negativtest:** Das Verhalten der Webanwendung wird mit ungültigen Rahmenbedingungen und Eingaben überprüft.

Test Cases sollten priorisiert werden. Jedem Test Case wird ein je nach Kritikalität der zugehörigen Funktion ein Prioritätslevel (z.B. Normal, Hoch, Kritisch) zugewiesen und beim Testen berücksichtigt. Test Cases müssen im Rahmen der Entwicklung durch den Entwickler erstellt werden.

- **Security Unit Tests:** Unit Tests überprüfen Softwareeinheiten (Units) einer Webanwendung auf korrekte Funktionalität. In der Regel sind das Funktionen, Methoden oder Klassen. Units werden mit verschiedenen Parametern aufgerufen und es wird überprüft, ob die Ausgabe mit den Erwartungen übereinstimmt. Unit Tests müssen im Rahmen der Entwicklung durch die Entwickler erstellt und gepflegt werden.
- **Design Review:** Bei Design Reviews geht es darum, die Architektur der Anwendung auf hoher Ebene zu untersuchen. Dabei sollen konzeptionelle Fehler und Verwundbarkeiten aufgedeckt werden und die Umsetzung der Sicherheitsanforderungen und -mechanismen auf ihre Vollständigkeit hin untersucht werden. Wenn erhebliche Mängel im Design erkannt werden, muss die Konzeption und die Planung überarbeitet werden. Die im Design Review erkannten Mängel dienen hierbei als Ansatzpunkt.
- **Code Reviews:** Beim Code Review wird der Programmcode manuell überprüft. Hierbei ist zu beachten, dass die Durchsicht durch eine andere Person erfolgt (etwa innerhalb eines Peer Reviews) als derjenigen, die den Programmcode verfasst hat, da ansonsten Fehler leicht übersehen werden können. Typische Fehler, die

entdeckt werden können, sind Abweichungen von Standards, Abweichungen gegenüber Anforderungen, Fehler im Design, Buffer Overflows etc. In erster Linie ist das bei sensiblen Funktionen wie beispielsweise Transaktionsmanagement, Authentisierung und Kryptographie wichtig. Die Ergebnisse des Code Reviews fließen zurück in den Implementierungsprozess.

- Statische Code Scanner: Mittels Statischen Code Scannern (SCA) können eine Vielzahl von Schwachstellen bereits innerhalb der Entwicklung identifiziert und so zeitnah behandelt werden. In der Praxis ist die Auswahl entsprechender Tools zur Durchführung von Sicherheitsscans sehr beschränkt. Weiterhin müssen diese Tools die eingesetzten Technologien unterstützen. Analysieren sie den Sourcecode, müssen diese zusätzlich die eingesetzten APIs kennen, da diese gewöhnlich nicht im Sourcecode vorliegen. Schließlich erzeugen SCA-Tools häufig eine sehr hohe Anzahl an False Positives, weshalb eine Verifikation der Scanergebnisse durch fachmännisches Personal erforderlich ist.
- Web Application Vulnerability Scanner: siehe 2.3
- Fuzz Testing: Fuzz Testing ist ein Testverfahren, welches automatisiert oder teilweise automatisiert durchgeführt wird. Dabei werden ungültige, unerwartete oder zufällige Werte generiert und als Eingabe an die Anwendung weitergegeben. Das Verhalten der Anwendung wird während des Tests auf Fehlverhalten überwacht. Das Prinzip ist simpel und führt dazu, dass Fehler entdeckt werden, die oft übersehen worden wären. Fuzz Testing kann nur einfache Programmfehler entdecken, die auf falsche Eingabewerte zurückzuführen sind, jedoch keine Designfehler etc. [4]

### 2.3.2.2 Penetration Testing

Nach Fertigstellung der Webanwendung kommen Penetration Tests zum Einsatz. Sie können als legaler und autorisierter Versuch definiert werden, Computersysteme anzugreifen, mit dem Ziel, diese sicherer zu machen. Der Prozess umfasst die Suche nach Schwachstellen sowie die Demonstration von Beispielangriffen, um zu zeigen, dass die Bedrohungen real sind. Ein ordnungsgemäßer Penetration Test resultiert immer in spezifischen Empfehlungen für das Beheben der während des Tests aufgetretenen Probleme. Insgesamt wird dieses Verfahren dazu verwendet, Computer und Netzwerke gegen zukünftige Angriffe abzusichern. Die allgemeine Idee besteht darin, Sicherheitslücken mithilfe der gleichen Tools und Techniken zu finden, die auch ein Angreifer benutzt. Die Lücken können so geschlossen werden, bevor ein realer Hacker sie ausnutzt. [12]

Üblicherweise lassen sich Penetration Tests in mehrere Phasen unterteilen [4]:

- Informationssammlung über das Zielsystem

- Scan nach angebotenen Diensten
- Identifikation des Systems und der Anwendungen
- Recherche nach Schwachstellen
- Ausnutzen der Schwachstellen

Penetration Tests können basierend auf unterschiedlichen Kriterien klassifiziert werden [4]:

- Informationsbasis: Hier wird festgelegt, von welchem Wissensstand der Tester ausgeht.
- Aggressivität: Hier wird festgelegt, wie aggressiv der Angreifer vorgeht. z.B. passiv (gefundene Schwachstellen werden nicht ausgenutzt) oder aggressiv (gefundene Schwachstellen werden unabhängig von den Konsequenzen ausgenutzt).
- Umfang: Hier wird festgelegt, welche Systeme getestet werden.
- Vorgehensweise: Der Angreifer kann verdeckt (Penetration wird nicht direkt als Angriff erkannt) oder offensichtlich (Penetration erfolgt offensichtlich) vorgehen.
- Technik: Es muss definiert werden, welche Techniken eingesetzt werden. Erfolgt der Angriff z.B. nur über das Netzwerk oder auch über weitere Kommunikationsnetze (Telefon, physischer Zugriff etc.)?
- Ausgangspunkt: Hier wird festgelegt, ob der Penetration Test von innen oder von außen erfolgen soll. Man unterscheidet hier zwischen drei unterschiedlichen Ansätzen, dem Black-Box, Grey-Box und White- Box Testing:

Beim Black-Box Testing befindet sich der Tester in der Rolle eines typischen Hackers von außen, der kein Wissen über die innere Arbeitsweise der Anwendung hat, weder Architektur noch Quellcode sind bekannt. Der Angreifer muss mit manuellen Methoden sowie speziellen Tools des Penetration Testings vertraut sein, um Schwachstellen zu lokalisieren und auszunutzen. Für die dynamische Analyse des anzugreifenden Netzwerks benötigt er Scanning Tools, die in der vorliegenden Arbeit evaluiert werden.

Während der Black-Box-Tester ein System aus der Sicht eines Außenseiters untersucht, hat ein Grey-Box Tester bereits Zugriff auf das System auf Benutzerebene, möglicherweise sogar mit erhöhten Berechtigungen eines Administrators. In der Regel liegt eine Dokumentation über Design und Architektur des Netzwerks vor,

die internen Komponenten sind bekannt. Dies hat den Vorteil, dass die Sicherheit des Netzwerks gezielter und effizienter beurteilt werden kann, der Tester kann sich sofort auf die Systeme konzentrieren, die am wichtigsten sind oder ein besonders hohes Risiko haben. Zudem kann durch das interne Benutzerkonto ein Angriff innerhalb des abgesicherten Systems mit umfassendem Zugriff auf das Netzwerk simuliert werden.

Der White-Box Tester hat schließlich uneingeschränkten Zugriff auf ein System, er besitzt alle nötigen Berechtigungen und kennt Netzwerkarchitektur und Design. Überdies hat er Zugang zum Quellcode, was es ihm erlaubt, statische Code-Analysen durchzuführen. Durch das Auffinden sowohl interner als auch externer Schwachstellen ist das White-Box Testing maximal effektiv, aber auch sehr aufwandsintensiv da der Tester sehr große Datenmengen untersuchen muss.

Penetration Tests können nicht nur auf technische Systeme, sondern auch auf die organisatorische oder personelle Infrastruktur in Form von Social Engineering erfolgen. Unabhängig von der Vorgehensweise haben Penetration Tests folgende Ziele [4]:

- Erhöhung der Sicherheit der IT-Systeme
- Identifikation von Schwachstellen
- Bestätigung der Sicherheit der IT-Systeme von unabhängigen Dritten
- Erhöhung der Sicherheit der organisatorischen und personellen Infrastruktur

In jedem Fall liefern Penetration Tests die Sicherheitslücken zwischen Planung und Implementierung. Die getroffenen Maßnahmen, um die Lücken zu schließen, müssen in einem weiteren Schritt nicht nur auf ihre korrekte Umsetzung überprüft werden, sondern auf ihre Angemessenheit, wie effektiv sie das Risiko tatsächlich senken beziehungsweise beseitigen oder ob durch sie ein falsches Sicherheitsgefühl vermittelt wird. Üblicherweise werden Penetration Tests erst dann durchgeführt, wenn die Webanwendung mit allen Komponenten fertiggestellt wird. [4]

### 2.3.2.3 Web Application Firewalls (WAFs)

Zusätzlichen Schutz von Angriffen auf Webapplikationen bieten WAFs, die den Verkehr zwischen Clients und Webservern auf Anwendungsebene überprüfen. Sie sind in der Lage, HTTP-Traffic zu filtern und gegebenenfalls zu blockieren, um die Webanwendung zu schützen. Hierzu untersucht eine WAF alle eingehenden Anfragen und die ausgehenden Antworten des Webserver. Erkennt sie dabei gefährliche Muster, verhindert sie die weitere Kommunikation mit dem Client.

Eine WAF kann zentralisiert hinter der Netzwerk Firewall und vor dem Webserver positioniert oder Host-basiert als Software-Lösung direkt auf dem Webserver installiert werden. Häufig wird der sogenannte Reverse-Proxy-Modus verwendet, bei dem der Proxy sich zwischen Webserver und Firewall befindet und Zugriffe im Namen des Clients durchführt. Im zweiten Schritt werden die Anfragen an den eigentlichen Webserver analysiert und die Websessions bei Bedarf terminiert.

Zu den üblichen Angriffen, die eine WAF verhindern kann, zählen Cross-Site-Scripting, SQL-Injection, Angriffe per Pufferüberlauf oder auch Konfigurationsfehler. Der Schutz ist hier stets nur als zusätzlicher Schutzmechanismus zu verstehen, der keinesfalls die Notwendigkeit ersetzt, eine Webanwendung mit ausreichender Sicherheit zu entwickeln und auch zu testen. Im Rahmen von Tests sollte eine WAF dabei stets deaktiviert sein, damit diese nicht die Testergebnisse verfälscht. Um den größtmöglichen Nutzen der WAF zu erhalten, muss deren Konfiguration durch fachmännisches Personal auf die Webanwendung angepasst werden. [4] [13]

## 2.4 Funktionsweise WVS

WVS sind automatisierte Werkzeuge, die Webanwendungen - in der Regel von außerhalb - nach Sicherheitslücken wie Cross-Site Scripting, SQL-Injection, Command Injection, Path Traversal und unsicheren Serverkonfigurationen absuchen. Diese Kategorie von Werkzeugen wird häufig auch als “Dynamic Application Security Testing (DAST) Tools” bezeichnet [8]. Im Gegensatz zum “Static Application Security Testing (SAST)”, bei dem der Quell-, Binär- oder Bytecode auf mögliche Implementierungs- und Konstruktionsfehler überprüft wird, testen DAST-Tools die laufende Webanwendung auf ihr Verhalten während des Betriebs. Es werden die gleichen Techniken angewendet, die auch ein realer Angreifer nutzen würde, um potentielle Sicherheitslücken zu finden.

In der Regel werden WVS auf einem Client-Computer installiert, um dann die Webanwendung zu analysieren. Im Zuge der Analyse können folgende vier Phasen durchlaufen werden [14]:

- **Crawl/Scan:** Ein Web Scanner bewegt sich ähnlich wie eine Suchmaschine durch die gesamte Webanwendung. Dabei werden sämtliche Links besucht und gegebenenfalls Formfelder mit Testwerten ausgefüllt. Der Scanner erhält so ein umfassendes Wissen über den Aufbau der Webanwendung, die Funktionsweise von Dialogschritten und den Inhalt von Formular-Seiten. Die Webanwendung wird auf häufig vorhandene Installations-, Konfigurations- oder Testverzeichnisse, sowie bekannte problematische oder informative Dateien durchsucht (Stichwörter: Directory Enumeration, File Enumeration, Directory Traversal, Forceful Browsing). Unterstützt werden kann die Scan-Phase durch Einbeziehen großer öffentlicher Suchmaschinen. Vorausgesetzt wird hierfür, dass die eigene Website bereits durch diese Suchmaschinen indexiert worden ist. Die Ergebnisse dieses Schrittes wurden dann auf den Suchmaschinen gespeichert (Index/Cache). Im Rahmen einer Scan-Phase können diese Suchmaschinen nun gezielt nach Informationen über die eigene Website abgefragt werden. Für die Zusammenstellung von Suchmustern, sowie Automatisierung der Abfragen kann die Nutzung spezieller Tools zweckmäßig sein.
- **Analyse:** In einem nächsten Schritt wird die Webanwendung einer eingehenden Sicherheitsanalyse unterzogen. Die gewonnenen Erkenntnisse können in einer Datenbank hinterlegt werden: Typ und Version des Webserver, verwendete Technologien und Tools (CGI, Servlets, JSP, JavaScript, PHP, usw.), Verwendung von Cookies oder anderer Mechanismen zum Session-Tracking, Analyse der Form-Parameter einer Seite, insbesondere auch die Verwendung von Hidden-Parametern, Extraktion von Kommentaren.
- **Audit/Penetrationstest:** Unter Einbeziehung des in der vorangegangenen Phase gewonnenen Wissens werden systematisch fehlerhafte oder unzulässige Eingabemuster erzeugt, und an die Webanwendung versandt. Einige der existierenden Scanner

unterteilen diese Phase in eine schadlose, und eine potentiell schadhafte Prüfung.

- Reporting: Die Ergebnisse der Scan- und Audit-Phase werden in Reports zusammengefasst. Der Umfang reicht dabei von Executive Summaries mit Nennung nur der größten Schwachstellen bis hin zu detaillierten Beschreibungen, in denen auch erste Hinweise für die Behebung des jeweiligen Problems gegeben werden können. Bei den meisten WVS hat sich ein System etabliert, bei dem die gefundenen Schwachstellen je nach Schweregrad in 4 Kategorien eingeteilt werden [15]:
  - High: Schwachstellen, die es Angreifern erlauben, die komplette Kontrolle über die Webanwendung einschließlich Server zu übernehmen. Angreifer können auf die Datenbank der Anwendung zugreifen, Konten ändern und vertrauliche Informationen stehlen. XSS und SQL-Injection sind Beispiele für Schwachstellen mit hohem Schweregrad, die bei der Erkennung durch einen Scanner oberste Priorität haben sollten.
  - Medium: Sicherheitslücken, die Angreifern den Zugriff auf ein angemeldetes Benutzerkonto ermöglichen, um vertrauliche Inhalte anzuzeigen. Angreifer erhalten Zugriff auf Informationen, mit denen sie zusätzlich andere Schwachstellen ausnutzen können, oder das System besser verstehen, damit sie ihre Angriffe verfeinern können. Open Redirection ist ein Beispiel für eine Schwachstelle mit mittlerem Schweregrad, durch die ein Angreifer einen Benutzer auf eine schädliche Website umleiten kann. Schwachstellen mit mittlerem Schweregrad sollten so schnell wie möglich behoben werden, wenn sie von einem Scanner erkannt werden.
  - Low: Diese Schwachstellen haben nur minimalen Einfluss oder können von einem Angreifer nicht ausgenutzt werden. Cookies, die nicht als “Http Only” gekennzeichnet sind, sind ein Beispiel für eine Sicherheitsanfälligkeit mit niedrigem Schweregrad. Das Markieren von Cookies als Http Only macht das Cookie für clientseitige Skripts unlesbar und bietet somit eine zusätzliche Schutzschicht gegen XSS-Angriffe. Schwachstellen mit geringem Schweregrad sollten untersucht und korrigiert werden, wenn Zeit und Budget dies zulassen.
  - Informational: Dies sind keine Schwachstellen, sondern lediglich Warnungen, die Informationen über die Webanwendung enthalten. Beispiele sind die erforderliche NTLM-Autorisierung und die Datenbankermittlung (MySQL). Für diese Informationsalarme ist keine Aktion erforderlich.



# 3 Methodik

## 3.1 Testaufbau

Für die Tests wurde der Ansatz des Black-Box Testings (siehe 2.2.2.2) verfolgt, der Ablauf des Testens war für jeden WVS identisch. Nach dem Scannen der 7 verwundbaren Web-Applikationen wurde jeweils ein entsprechender Bericht in Form einer HTML-Datei generiert, der die gefundenen Schwachstellen und je nach WVS auch die benötigte Zeit für den Scan auflistet. Neben diesen evidenten Daten fließen subjektive Eindrücke wie Handhabung und Bedienbarkeit der Software und die Qualität der erstellten Berichte in die Evaluation mit ein.

Um dies messbar zu machen und um am Ende ein Ranking der getesteten WVS abbilden zu können, musste ein Bewertungssystem etabliert werden. Die Bewertungskategorien wurden nach Relevanz gewichtet und für die Bewertung eine Skala mit fünf Skalenwerten von 0 bis 4 Punkten herangezogen:

- 0 **ungenügend**
- 1 **unterdurchschnittlich**
- 2 **durchschnittlich**
- 3 **überdurchschnittlich**
- 4 **überragend**

Das Gesamtergebnis setzt sich aus den Bewertungen für die Bedienung (20%), Qualität des Reportings (20%), der Geschwindigkeit (10%) und dem Scannergebnis (50%) zusammen. Die höchste zu erreichende Punktzahl ist somit 40.

Angesichts zahlreicher Abstürze der Software Acunetix unter Windows (siehe Punkt 4.2.2) wurde eine weitere Bewertungskategorie "Stabilität" in Betracht gezogen, aber im Hinblick auf die Ununterscheidbarkeit aller anderen stabil laufenden WVS wieder verworfen. Die Abstürze der Acunetix-Software fanden schließlich in der Kategorie "Bedienung" ihre Berücksichtigung.

Für das Scannergebnis wurde die reine Anzahl der gefundenen Schwachstellen zu Grunde gelegt. Für eine tiefergehende Evaluation ist eine manuelle Validierung aller Schwachstellen auf True- und False-Positives in Erwägung zu ziehen, was jedoch angesichts der Vielzahl (über 3000) an Funden den Rahmen dieser Ausarbeitung gesprengt hätte.

Für die Tests wurde folgendes System verwendet:

Intel Core i7-8700K CPU mit 32 GB RAM

Microsoft Windows 10 pro, Version 1809 (64 bit)

Als Virtuelle Maschinen innerhalb von VirtualBox: Kali Linux 18.4 und Parrot OS 4.5.1, jeweils ausgestattet mit 2 Kernen und 8 GB RAM.

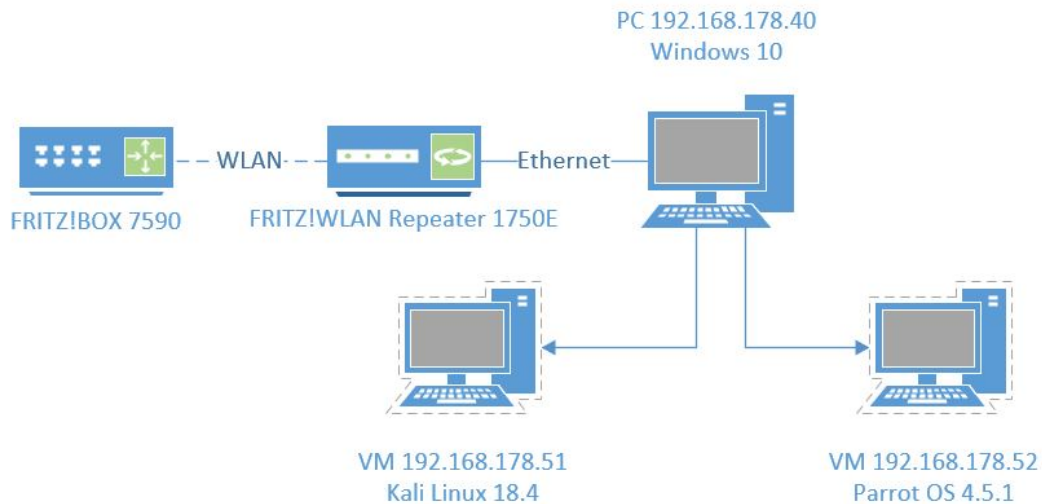


Abbildung 3.1: Testaufbau

## 3.2 Web Application Vulnerability Scanner (WVS)

### 3.2.1 Auswahlkriterien

OWASP listet 50 verschiedene Tools zum Scannen von Webanwendungen auf [8]. Die Auswahl wurde auf WVS mit folgenden Eigenschaften eingegrenzt:

- 1. Free und Open Source WVS.
- 2. Kommerzielle WVS, die eine voll funktionsfähige Testversion anbieten.
- 3. WVS, deren aktuelles Release nicht älter als 2 Jahre ist oder innerhalb der letzten 2 Jahre modifiziert wurde.
- 4. WVS, die in der Lage sind, umfassende Scans auszuführen, um möglichst viele verschiedene Schwachstellenarten aufzuspüren.

Bei den bekanntesten Anbietern kommerzieller WVS wurde jeweils eine Testversion angefragt, um ein realistisches Abbild der aktuell meistgenutzten Tools zu erhalten und um die Ergebnisse der kostenlosen denen der kommerziellen Scanner gegenüber zu stellen. Es handelt sich um folgende Firmen:

Acunetix, Beyond Security (WSSA), Beyond Trust (Retina), Netsparker, N-Stalker, Portswigger (BurpSuite Pro), Rapid 7 (Nexpose) und Tenable (Nessus).

Von den angefragten Firmen stellten Acunetix, Netsparker, N-Stalker, Portswigger und Tenable jeweils eine Testversion zur Verfügung.

### 3.2.2 Ausgewählte WVS

Aus der Vielzahl der ursprünglich in Betracht kommenden WVS haben sich am Ende 9 herauskristallisiert, 5 mit einer OpenSource-Lizenz und 4 kommerzielle Produkte. Unter Punkt 4.2 werden sie im Zuge der Evaluation näher beschrieben.

#### 3.2.2.1 Free und Open Source WVS

WVS	Entwickler	Version	Verwendete Plattform
<b>Arachni</b>	Tasos Laskos	0.5.12 (WebUI)	Windows
<b>Nikto</b>	cirt.net	2.1.6	Kali
<b>OpenVAS</b>	Greenbone	7.0.3	Parrot
<b>Wapiti</b>	devloop	3.0.1	Kali
<b>Zed Attack Proxy</b>	OWASP	2.7.0	Parrot

Tabelle 3.1: Ausgewählte Free und Open Source WVS

#### 3.2.2.2 Kommerzielle WVS

WVS	Anbieter	Version	Verwendete Plattform
<b>Acunetix</b>	Acunetix	12.0.190206130	Kali
<b>Burp Suite Pro</b>	Portswigger	2.0.15	Windows
<b>Nessus</b>	Tenable	8.2.2	Windows
<b>Netsparker</b>	Netsparker	5.2.0.22027	Windows

Tabelle 3.2: Ausgewählte Kommerzielle WVS

### 3.2.3 Nicht ausgewählte WVS

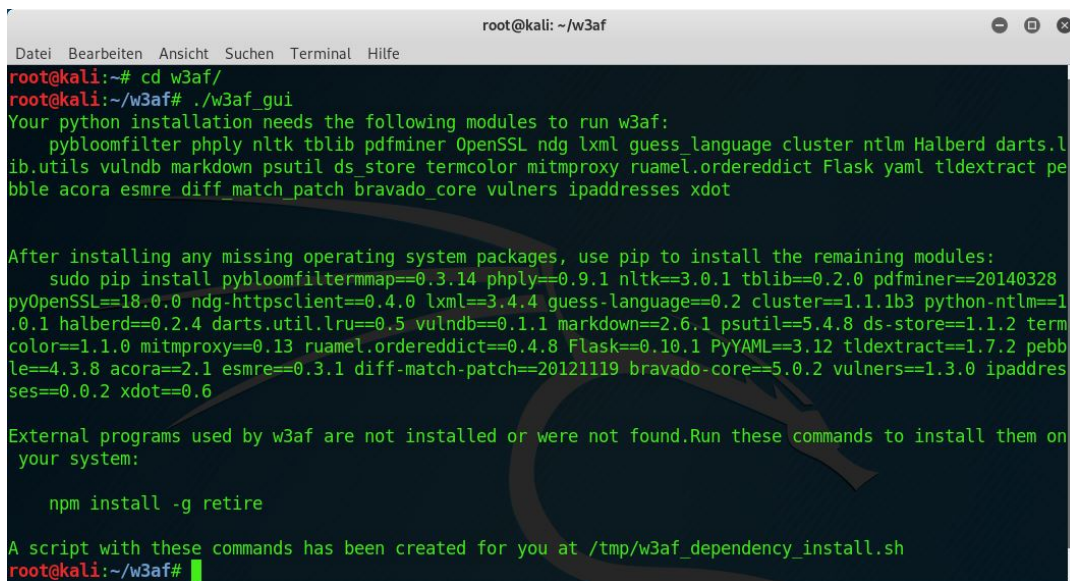
Nachfolgend werden alle WVS aufgelistet, die als Kandidaten in Erwägung gezogen wurden, bei näherer Betrachtung jedoch nicht die erforderlichen Voraussetzungen erfüllten, um in die Evaluation aufgenommen zu werden.

#### 3.2.3.1 Free und Open Source WVS

- GoLismero [16]: GoLismero ist ein in Python geschriebenes Framework, das verschiedene Penetrationstesting-Tools in sich vereint. Theoretisch sollten bei einem Angriff alle Tools angewendet und die jeweiligen Ergebnisse in einem einzigen Report gebündelt werden. In der Praxis hängte sich das Programm jedoch jedes Mal nach einer Weile auf, sowohl unter Kali-Linux und Parrot OS, als auch unter Windows. Es werden zwar Teilergebnisse auf dem Bildschirm angezeigt, dies reicht aber nicht aus, um in die Evaluation aufgenommen zu werden.
- Grabber [17]: Das von Romain Gaucher entwickelte Programm ist zwar noch Bestandteil von Kali-Linux, ist aber schon über 12 Jahre alt (Latest Release 2006).
- Grendel-Scan [18]: Seit 2013 gibt es auf dem SourceForge-Repository von David Byrne keine Veränderung.
- Iron Wasp [19]: Die aktuelle Version des Windows-Programms von Lavakumar Kuppan ist ein Beta-Release aus dem Jahr 2015.
- Ratproxy [20]: Die Entwicklung wurde 2009 von Google eingestellt.
- Skipfish [21]: Ein weiteres Projekt von Google, das 2012 eingestellt wurde.
- SQLmap [22]: SQLmap ist ein beliebtes Tool zum Auffinden von SQLi Schwachstellen, ist aber darauf beschränkt.
- Vega [23]: Das aktuelle Release ist aus dem Jahr 2014, seit diesem Jahr wird in regelmäßigen Abständen angekündigt, ein Feature zum Exportieren der Ergebnisse hinzuzufügen, aber die Firma Subgraph scheint das Projekt nicht weiter zu verfolgen.
- Watobo [24]: Die letzte Änderung des OpenSource Scanners stammt aus dem Jahr 2015.
- Webscarab [25]: Der Vorgänger von OWASPs Zed Attack Proxy ist veraltet (Latest

Release 2011), OWASP empfiehlt, auf ZAP umzusteigen.

- Wfuzz [26]: Der in Python geschriebene Scanner verwendet die Bibliothek Pycurl für HTTP-Requests, diese unterstützt keine SSL/TLS Verschlüsselung.
- W3af [27]: W3af wird zwar noch sporadisch mit Bibliotheks-Aktualisierungen gepflegt, das aktuelle Release ist jedoch aus dem Jahr 2014 und es ist weder auf Kali-Linux noch auf Parrot OS gelungen, alle für den Programmstart benötigten Dependencies zu installieren. Das bei der Installation generierte Script versucht, teils veraltete Module zu installieren, manuelles Nachinstallieren der aktuellen Versionen brachte keinen Erfolg.



```
root@kali: ~/w3af
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
root@kali:~# cd w3af/
root@kali:~/w3af# ./w3af_gui
Your python installation needs the following modules to run w3af:
    pybloomfilter phply nltk tblib pdfminer OpenSSL ndg lxml guess_language cluster ntlm Halberd darts.l
ib.utils vulndb markdown psutil ds_store termcolor mitmproxy ruamel.orderdict Flask yaml tldextract pe
bble acora esmre diff_match_patch bravado_core vulners ipaddresses xdot

After installing any missing operating system packages, use pip to install the remaining modules:
    sudo pip install pybloomfiltermmmap==0.3.14 phply==0.9.1 nltk==3.0.1 tblib==0.2.0 pdfminer==20140328
pyOpenSSL==18.0.0 ndg-httpsclient==0.4.0 lxml==3.4.4 guess-language==0.2 cluster==1.1.1b3 python-ntlm==1
.0.1 halberd==0.2.4 darts.util.lru==0.5 vulndb==0.1.1 markdown==2.6.1 psutil==5.4.8 ds-store==1.1.2 term
color==1.1.0 mitmproxy==0.13 ruamel.orderdict==0.4.8 Flask==0.10.1 PyYAML==3.12 tldextract==1.7.2 pebb
le==4.3.8 acora==2.1 esmre==0.3.1 diff-match-patch==20121119 bravado-core==5.0.2 vulners==1.3.0 ipaddres
ses==0.0.2 xdot==0.6

External programs used by w3af are not installed or were not found.Run these commands to install them on
your system:

    npm install -g retire

A script with these commands has been created for you at /tmp/w3af_dependency_install.sh
root@kali:~/w3af#
```

Abbildung 3.2: W3af: Fehlende Module

- Wikto [28]: Wikto ist eine Windows-Portierung von Nikto und bedarf daher keiner eigenen Evaluation.
- Xenotix [29]: Xenotix wurde von OWASP für das Auffinden von Cross-Site-Scripting Schwachstellen entwickelt und ist darauf beschränkt.

#### 3.2.3.2 Kommerzielle WVS

- N-Stalker [30]: Die angebotene “7-Day Evaluation Licence” erlaubt nur das Scannen einer einzigen, vorher festgelegten URL und ist daher für den geplanten Testaufbau nicht geeignet.

### 3.3 Verwundbare Web-Applikationen

Bei der Auswahl der Web-Applikationen musste darauf geachtet werden, dass sie für WVS geeignet sind. Auf ursprünglich in die Auswahl aufgenommene Applikationen wie WebGoat, JuiceShop (beide von OWASP) oder Damn Vulnerable Web Application (Bestandteil von Metasploitable) wurde am Ende verzichtet, da hier beim Scannen keine hinreichenden Ergebnisse hervorgebracht wurden. Diese Anwendungen sind zwar sehr gut dokumentiert, aber hauptsächlich für das Erlernen von manuellen Angriffen entwickelt worden. Es wurde bei der Auswahl auf das OWASP Vulnerable Web Applications Directory Project zurückgegriffen, das eine Reihe von verwundbaren Web-Applikationen auflistet [31]. Die Auswahl deckt mehrere Technologien wie PHP, ASP.Net oder HTML5 ab.

Nachfolgend werden die für die Auswertung genutzten Web-Applikationen kurz vorgestellt.

- **WA1: Altoro Mutual**  
Altoro Mutual ist eine in C# .NET geschriebene Online-Banking Web-Applikation, die von IBM entwickelt wurde, um WVS zu testen [32].
- **WA2: Webscantest**  
Diese Web-Applikation ist in PHP geschrieben und wurde von NTOSpider entwickelt, um WVS zu testen. Die Schwachstellen sind direkt auf der Seite dokumentiert [33].
- **WA3: Zero Bank**  
Eine weitere Online-Banking Web-Applikation, entwickelt von Hewlett-Packard/Micro Focus [34].
- **WA4: Bitcoin Web Site**  
Diese von Netsparker entwickelte Applikation ist in ASP.NET geschrieben und simuliert eine Online-Seite für Bitcoin-Transaktionen [35].
- **WA5: Acuart**  
Eine Test-Seite von Acunetix, der einen Online-Shop für Kunstwerke simuliert, geschrieben in PHP [36].
- **WA6: Crack Me Bank**  
Eine in PHP geschriebene Online-Banking Seite, entwickelt von Trustwave [37]:
- **WA7: Security Tweets**  
Security Tweets ist eine von Twitter inspirierte Social Networks Applikation, die von Acunetix entwickelt wurde und HTML5 verwendet [38].

## 4 Evaluation

### 4.1 Gefundene Schwachstellen per Webanwendung

Die Buchstaben H, L, M und I stehen für die Kategorien High, Medium, Low und Informationale (siehe Punkt 2.3). WA1 - WA7 stehen für die verwundbaren Web-Applikationen (siehe Punkt 3.3)

	Arachni				Nikto	OpenVAS				Wapiti	ZAP			
	H	M	L	I		H	M	L	I		H	M	L	I
<b>WA1</b>	9	4	2	5	12	0	2	0	24	10	1	2	5	0
<b>WA2</b>	4	10	5	29	18	0	8	0	41	12	1	4	11	0
<b>WA3</b>	4	6	4	4	18	6	51	2	34	2	0	1	2	0
<b>WA4</b>	22	6	8	27	15	0	4	0	24	32	3	4	6	0
<b>WA5</b>	56	6	10	24	18	2	31	1	77	31	4	1	2	0
<b>WA6</b>	29	10	4	10	9	0	11	1	29	6	4	3	3	0
<b>WA7</b>	10	2	3	6	7	1	31	1	77	0	0	1	5	0
<b>Subt.</b>	134	44	36	105	97	9	138	5	306	93	13	16	34	0
<b>Total</b>	<b>319</b>				<b>97</b>	<b>458</b>				<b>93</b>	<b>63</b>			

Tabelle 4.1: Gefundene Schwachstellen der Open Source WVS

	Acunetix				Burp Suite Pro				Nessus				Netsparker			
	H	M	L	I	H	M	L	I	H	M	L	I	H	M	L	I
<b>WA1</b>	4	10	2	34	10	0	7	18	0	3	2	21	4	6	10	15
<b>WA2</b>	25	41	6	14	5	2	4	160	1	7	1	25	37	18	24	20
<b>WA3</b>	19	23	20	24	0	0	1	13	14	31	1	23	7	6	16	14
<b>WA4</b>	73	26	6	7	17	1	4	119	1	6	1	20	21	8	16	28
<b>WA5</b>	39	34	9	19	28	0	4	49	21	18	1	21	24	18	12	12
<b>WA6</b>	6	14	27	5	11	0	1	74	0	5	1	22	14	4	11	13
<b>WA7</b>	14	4	9	3	8	2	2	11	0	1	2	16	8	2	10	11
<b>Subt.</b>	180	152	79	106	79	5	23	444	37	71	9	148	115	62	99	113
<b>Total</b>	<b>517</b>				<b>551</b>				<b>265</b>				<b>389</b>			

Tabelle 4.2: Gefundene Schwachstellen der kommerziellen WVS

## 4.2 Bedienung, Reporting und Geschwindigkeit

### 4.2.1 Open Source

- Arachni [39]:  
Arachni gibt es als reine Terminal-Anwendung oder als Ruby on Rails Framework mit Web-Interface. Die Web-Oberfläche ist übersichtlich und verständlich aufgebaut, der User findet sich schnell zurecht und kann sofort mit dem Scannen einer Seite beginnen, die Scangeschwindigkeit ist überdurchschnittlich. Als Hilfestellung gibt es ein umfangreiches Wiki mit Erklärungen und Screenshots. Der Report ist sehr umfangreich, verschiedene Balken- und Kuchendiagramme geben Statistiken über Art und Schweregrad der Funde wieder, zudem gibt es Verlinkungen zu den OWASP Top 10 und detaillierte Ausführungen über die Schwachstellen. Arachni unterscheidet bei den Funden außerdem zwischen gesicherten (“Trusted”) und noch zu überprüfenden Ergebnissen (“Untrusted”).

**Bewertung: Reporting 4, Bedienung 3, Geschwindigkeit 3**

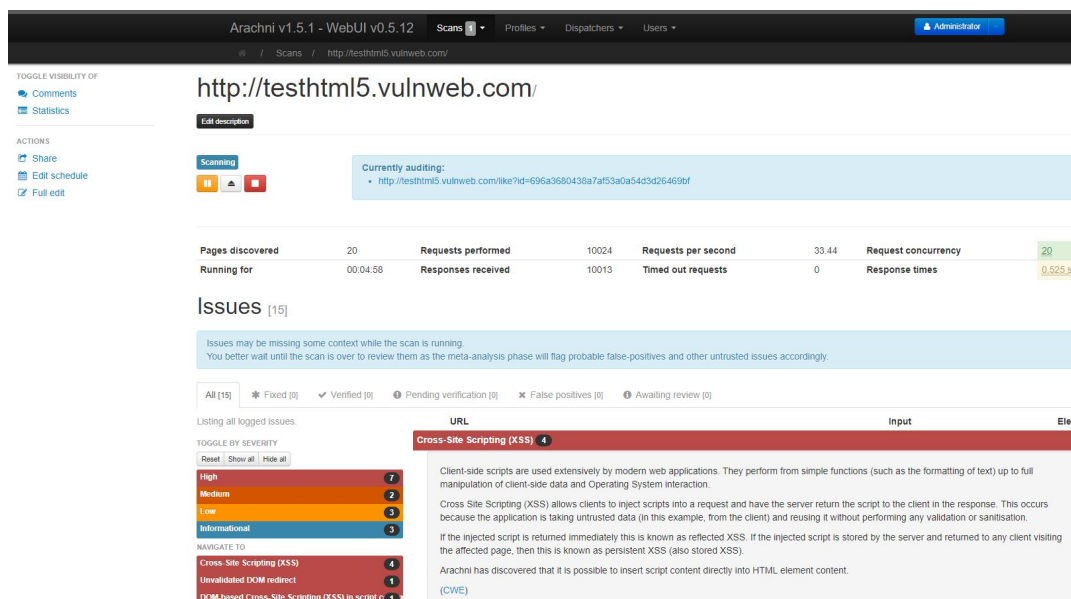


Abbildung 4.1: Arachni WebUI 0.5.12

- Nikto [40]:  
Nikto ist ein gut dokumentiertes Terminal-Programm, nach kurzer Einarbeitung hat ein ungeübter User die benötigten Befehle und Optionen gefunden, um einen Scan zu starten. Auffällig ist die im Vergleich zu allen anderen WVS ungewöhnlich hohe Scan-Geschwindigkeit, die sich im Minutenbereich einordnet. Der generierte



Report listet alle gefundenen Schwachstellen auf, unterscheidet diese jedoch im Gegensatz zu den meisten anderen WVS nicht zwischen High, Medium, Low und Informationnal.

### Bewertung: Reporting 1, Bedienung 2, Geschwindigkeit 4

- OpenVAS [41]:

OpenVAS ist aus der Software Nessus hervorgegangen, als diese im Jahr 2005 von Open Soucre zu einer kommerziellen Lizenz wechselte, und wird seitdem auf Basis der letzten freien Nessus-Version 2.2 von Greenbone Networks weiterentwickelt. Der Scanner ist eingebettet in den Greenbone Security Assistant, der über ein Web-Interface bedient wird. Die Oberfläche ist nicht selbsterklärend, es bedarf etwas an Recherche, bis sich dem User der logische Aufbau des Programms erschließt. Hier ist das “Tech Doc-Portal” von Greenbone sehr hilfreich. Zuerst muss unter Configuration/Targets ein Ziel definiert werden, dann kann der User für dieses Ziel unter dem Punkt “Scans” einen Task erstellen und diesen entweder sofort oder per Schedule starten. Die umständliche Handhabung hat jedoch den Vorteil, dass sich leicht mehrere Tasks automatisieren lassen. Der Bericht präsentiert sich etwas sparsam, die gefundenen Schwachstellen der Kategorie “Informational” werden nicht aufgelistet.

### Bewertung: Reporting 2, Bedienung 1, Geschwindigkeit 2

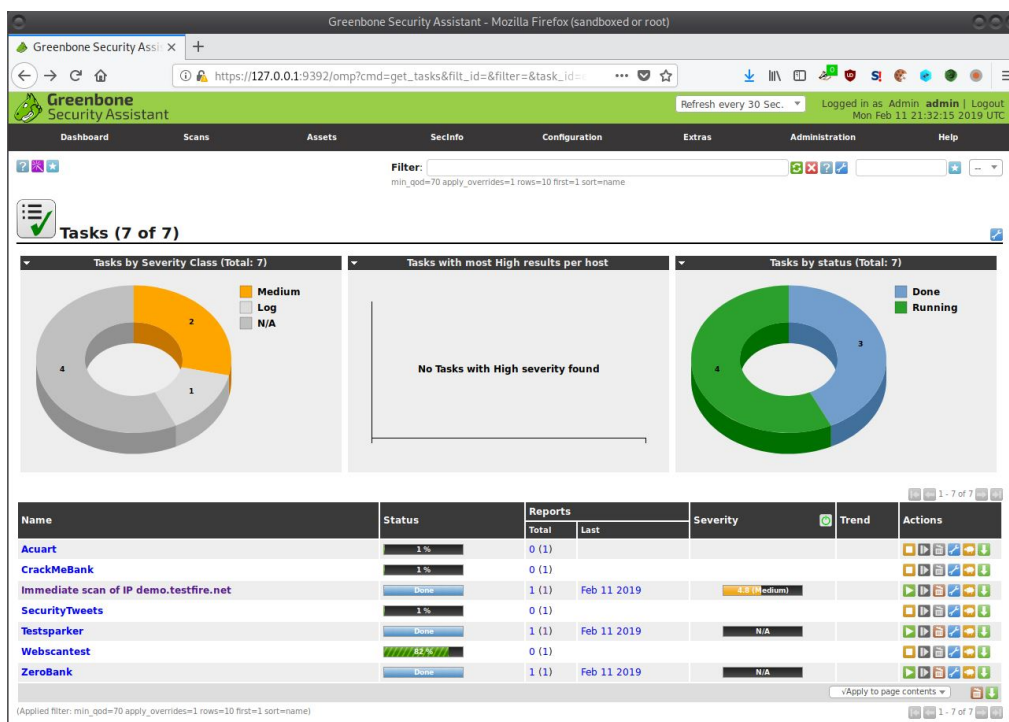


Abbildung 4.2: OpenVAS im Greenbone Security Assistant

- Wapiti [42]:

In der Handhabung und im Reporting ähnelt Wapiti dem anderen reinen Terminal-Programm Nikto. Die auswählbaren Optionen sind ähnlich, und die gefundenen Schwachstellen werden auch hier nicht in Kategorien eingeteilt. Die Dokumentation fällt etwas spartanischer aus, ist aber ausreichend, um sich schnell zurecht zu finden. Wapiti scannt fast so schnell wie Nikto und reiht sich hier auf dem zweiten Platz ein.

**Bewertung: Reporting 1, Bedienung 2, Geschwindigkeit 4**

- Zed Attack Proxy [43]:

Der von OWASP entwickelte Scanner hat eine übersichtliche Benutzeroberfläche und lässt sich intuitiv bedienen. Auf der Startseite lässt sich direkt die anzugreifende URL eingeben und ohne weitere Konfiguration angreifen. Es gibt umfangreiche Hilfestellung in Form eines Handbuchs und einem Online-Wiki, zudem gibt es mit der OWASP ZAP User Group ein gut frequentiertes Benutzerforum, auf dem ein reger Austausch zwischen den Benutzern stattfindet. Auffällig sind die langen Scan-Zeiten von mehreren Stunden, die sich jedoch nicht in einer höheren Anzahl an gefundenen Schwachstellen widerspiegeln. Die wenigen Funde werden im Report ausführlich beschrieben einschließlich umfangreicher Empfehlungen zur Behebung.

**Bewertung: Reporting 2, Bedienung 3, Geschwindigkeit 0**

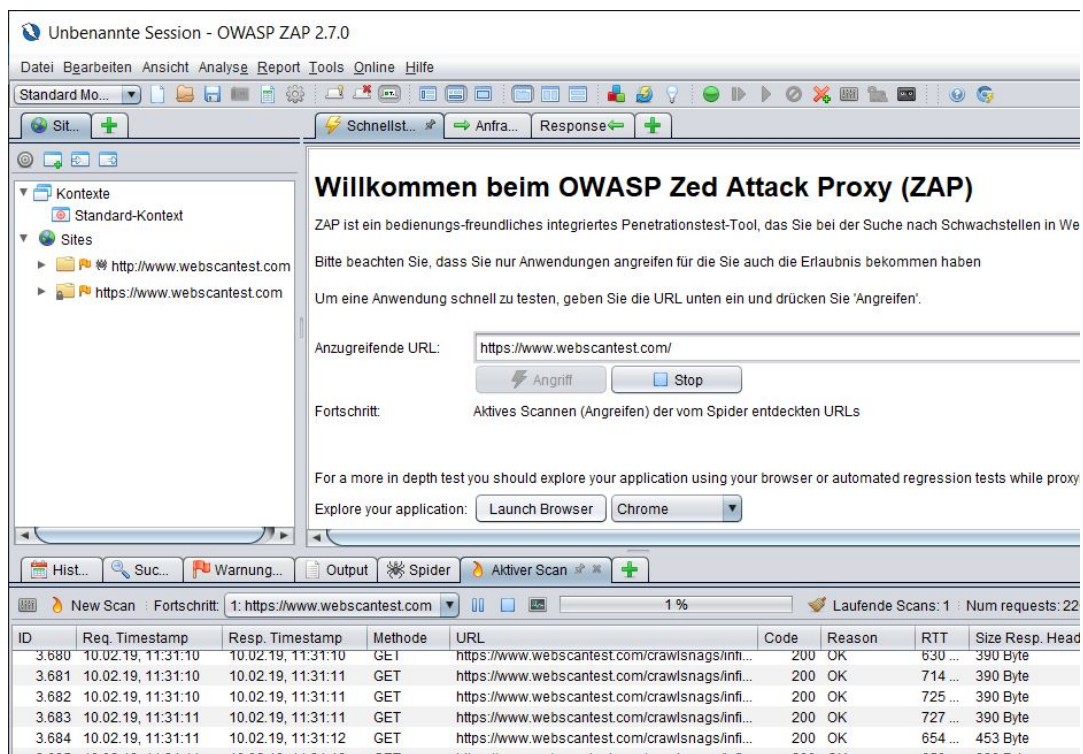


Abbildung 4.3: Grafische Benutzeroberfläche von ZAP

## 4.2.2 Kommerziell

- Acunetix [44]:  
Acunetix stellte eine 14-tägige Vollversion zur Verfügung. Die graphische Oberfläche ist sehr übersichtlich und lässt sich intuitiv bedienen. Online gibt es ein Support-Portal mit ausführlicher Dokumentation und Hilfestellung. Unter Windows brachte Acunetix das System mehrmals zum Absturz (BSOD), so dass auf die Linux-Version ausgewichen wurde. Hier lief das Programm stabil und scannte von den kommerziellen WVS am schnellsten, von allen WVS muss sich Acunetix hier nur den Terminalprogrammen Wapiti und Nikto geschlagen geben. Der Report listet zu jeder gefundenen Schwachstelle sehr detailliert die vollständigen GET- und POST-Requests sowie teilweise seitenlange Code-Passagen auf. Die Empfehlungen zur Behebung der Funde könnte hingegen ausführlicher sein. Für die Abstürze wurden bei der Bewertung für die Bedienung 2 Punkte abgezogen.

**Bewertung: Reporting 3, Bedienung 1, Geschwindigkeit 4**

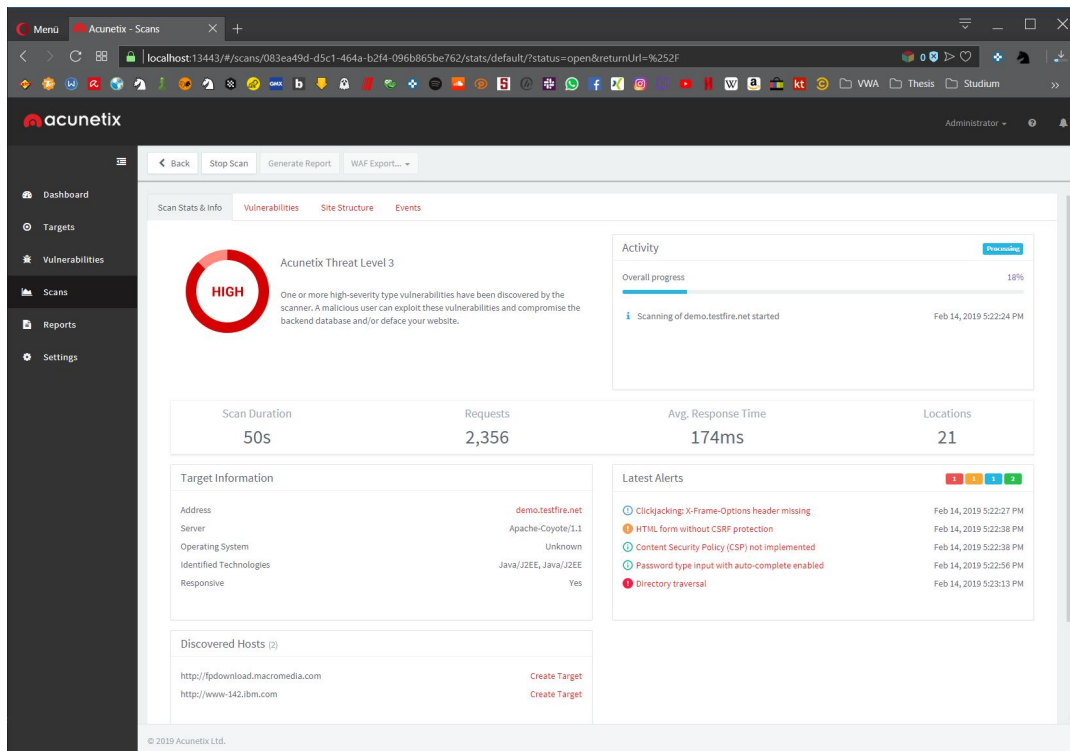


Abbildung 4.4: Weboberfläche von Acunetix

- BurpSuite Pro [45]:

Die Firma Portswigger stellte auf Anfrage eine 30-tägige unbeschränkte Testversion zur Verfügung. BurpSuite Pro enthält eine umfangreiche Dokumentation mit zahlreichen Hilfestellungen für verschiedene Anwendungsszenarien. Das Dashboard ist sehr übersichtlich und intuitiv zu bedienen: mit Hilfe des Buttons “New Scan” lässt sich direkt ohne größeren Konfigurationsaufwand eine Website erfolgreich und in durchschnittlicher Geschwindigkeit scannen.

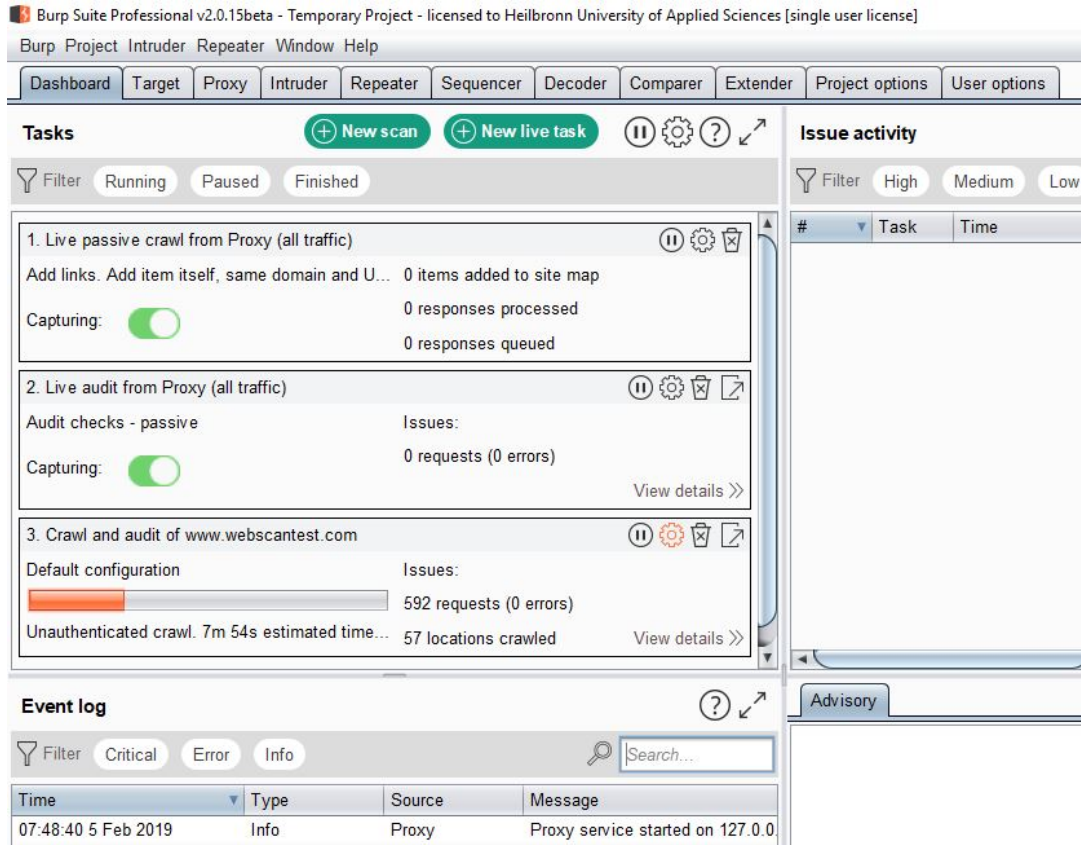



Abbildung 4.5: Dashboard von BurpSuite Pro

Der von BurpSuite generierte Report ist von allen getesteten WVS der umfangreichste und ist trotzdem sehr übersichtlich. Ähnlich wie Arachni unterscheidet BurpSuite zwischen gesicherten und ungesicherten Ergebnissen, allerdings noch genauer: in einer “Confidence”-Tabelle wird zwischen gesicherten (Certain), wahrscheinlichen (Firm) und möglichen (Tentative) Schwachstellen unterschieden.

**Bewertung: Reporting 4, Bedienung 3, Geschwindigkeit 2**

Burp Scanner Report Acuart



Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	21	7	0	28
	Medium	0	0	0	0
	Low	3	1	0	4
	Information	28	5	16	49

Abbildung 4.6: Confidence-Tabelle im Report von BurpSuite Pro

- Nessus [46]:

Die Firma Tenable bietet zwei Testversionen an: Nessus Home für das Testen des heimischen Netzwerks, gültig für ein Jahr sowie eine unbeschränkte Version von Nessus Pro, gültig für sieben Tage. Nessus präsentiert sich auf einer übersichtlich gestalteten Web-Schnittstelle und ist selbsterklärend bedienbar. Einen Hilfe-Button sucht man vergeblich, die umfangreiche Dokumentation mit Anleitungen findet man online durch Recherchieren. Die Scangeschwindigkeit ist durchschnittlich, der Report übersichtlich aber eher spartanisch. Nur durch Anklicken der Funde gibt es online Beschreibungen und Lösungsvorschläge.

**Bewertung: Reporting 2, Bedienung 3, Geschwindigkeit 2**

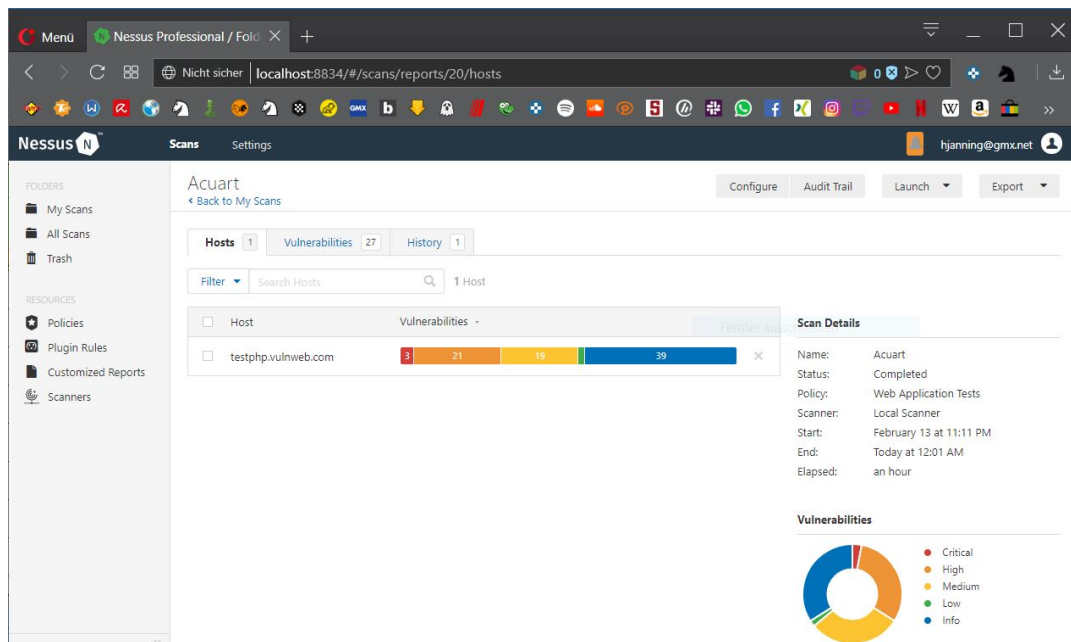


Abbildung 4.7: Grafische Benutzeroberfläche von Nessus



- Netsparker [47]:

Netsparker stellte nach Rücksprache eine 14-tägige Testversion zur Verfügung, die auf 8 zu scannende Web-Applikationen begrenzt ist. Die grafische Benutzeroberfläche ist etwas unruhig und überladen, der User kann aber direkt über den Button “New” eine Web-Seite scannen. Die Scangeschwindigkeit ist im unteren Mittelfeld angesiedelt. Der Report ist äußerst umfangreich aber etwas unübersichtlich. Neben Nessus hat auch Netsparker nach High, Medium, Low und Informationale eine fünfte Kategorie “Critical” eingeführt, die auf besonders kritische Schwachstellen hinweist. Wie bei Arachni und BurpSuite werden gesicherte Ergebnisse gesondert gekennzeichnet, hier als “Confirmed”.

**Bewertung: Reporting 3, Bedienung 2, Geschwindigkeit 1**

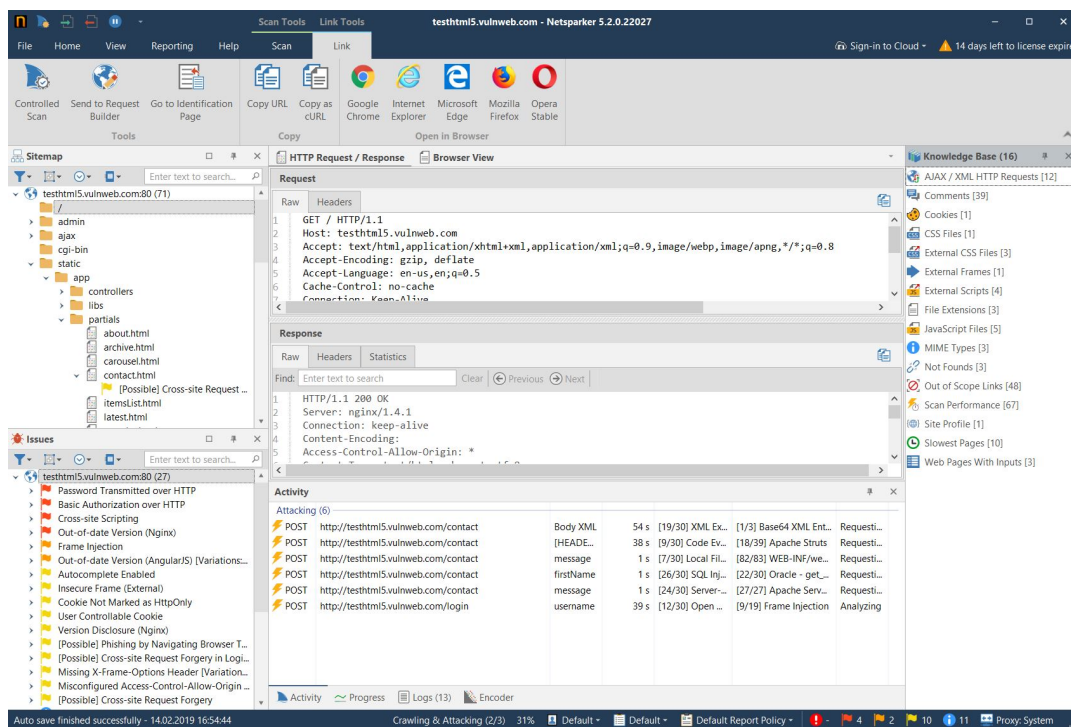


Abbildung 4.8: Grafische Benutzeroberfläche von Netsparker

### 4.2.3 Gesamtbewertung und Ranking

Open Source WVS	Arachni	Nikto	OpenVAS	Wapiti	ZAP
Scanergebnis (50%) *5	3	1	4	1	0
Reporting (20%) *2	4	1	2	1	2
Bedienung (20%) *2	3	2	1	2	3
Geschwindigkeit (10%)	3	4	2	4	0
<b>Gesamt</b>	<b>32</b>	<b>15</b>	<b>28</b>	<b>15</b>	<b>10</b>

Tabelle 4.3: Bewertung der Open Source WVS

Kommerzielle WVS	Acunetix	Burp Suite Pro	Nessus	Netsparker
Scanergebnis (50%) *5	4	4	2	3
Reporting (20%) *2	3	4	2	3
Bedienung (20%) *2	1	3	3	2
Geschwindigkeit (10%)	4	2	2	1
<b>Gesamt</b>	<b>32</b>	<b>36</b>	<b>22</b>	<b>26</b>

Tabelle 4.4: Bewertung der kommerziellen WVS

	WVS	Gesamt	Scanergebnis	Reporting	Bedienung	Geschwindigkeit
1	BurpSuite Pro	36	4	4	3	2
2	Acunetix	32	4	3	1	4
2	Arachni	32	3	4	3	3
4	OpenVAS	28	4	2	1	2
5	Netsparker	26	3	3	2	1
6	Nessus	22	2	2	3	2
7	Nikto	15	1	1	2	4
7	Wapiti	15	1	1	2	4
9	ZAP	10	0	2	3	0

Tabelle 4.5: Ranking aller WVS

# **5 Diskussion**

## **5.1 Open Source WVS**

- 

## **5.2 Kommerzielle WVS**

TODO

## **5.3 Open Source vs. kommerziell**

TODO

## **5.4 Empfehlung**



## **6 Fazit und Ausblick**

# Quellenverzeichnis

- [1] Dafydd Stuttard und Marcus Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition*. John Wiley & Sons, Inc., 2011.
- [2] CVE Details. *Vulnerabilities by Year*. 2019. URL: <https://www.cvedetails.com/browse-by-date.php> (besucht am 07.01.2019).
- [3] Bitkom. *Wirtschaftsschutz in der Industrie*. 2018. URL: <https://www.bitkom.org/sites/default/files/file/import/Bitkom-PK-Wirtschaftsschutz-Industrie-13-09-2018-2.pdf> (besucht am 02.03.2019).
- [4] Bundesamt für Sicherheit in der Informationstechnik. *Leitfaden zur Entwicklung sicherer Webanwendungen*. BSI, 2013.
- [5] Hannes Holm. *A quantitative evaluation of vulnerability scanning*. Royal Institute of Technology, Stockholm, 2011.
- [6] Martin Wundram. „Die schwächste Stelle, Drei Webapplikations-Scanner im Vergleich“. In: *iX* (2011), 72–77.
- [7] Martin Wundram. „Gut Gesucht, Werkzeuge für das Aufspüren von Schwachstellen“. In: *iX* (2012), 92–98.
- [8] OWASP. *Category: Vulnerability Scanning Tools*. 2019. URL: [https://www.owasp.org/index.php/Category:Vulnerability\\_Scanning\\_Tools](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) (besucht am 26.01.2019).
- [9] Matthias Rohr. *Sicherheit von Webanwendungen in der Praxis, 2. Auflage*. Springer Fachmedien Wiesbaden GmbH, 2018.
- [10] Bundesamt für Sicherheit in der Informationstechnik. *Glossar und Begriffsdefinitionen*. 2013. URL: [https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar\\_node.html](https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html) (besucht am 16.03.2019).
- [11] Bundesamt für Sicherheit in der Informationstechnik. *Lebenszyklus einer Schwachstelle*. 2018. URL: [https://www.allianz-fuer-cybersicherheit.de/ACS/DE/\\_/downloads/BSI-CS\\_027.pdf?\\_\\_blob=publicationFile&v=4](https://www.allianz-fuer-cybersicherheit.de/ACS/DE/_/downloads/BSI-CS_027.pdf?__blob=publicationFile&v=4) (besucht am 16.03.2019).
- [12] Patrick Engebretson. *The Basics of Hacking and Penetration Testing*. Elsevier Inc., 2013.

- [13] Vogel Communications Group. *Was ist eine Web Application Firewall?* 2017. URL: <https://www.security-insider.de/was-ist-eine-web-application-firewall-a-627220/> (besucht am 13.03.2019).
- [14] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheit von Webanwendungen, Maßnahmenkatalog und Best Practices*. BSI, 2006.
- [15] Ron Lepofsky. *The Manager's Guide to Web Application Security*. Apress, Berkeley, CA, 2014.
- [16] Golismero Project. *Golismero*. 2019. URL: <http://www.golismero.com> (besucht am 19.02.2019).
- [17] Romain Gaucher. *Grabber*. 2006. URL: <http://rgaucher.info/beta/grabber/> (besucht am 19.02.2019).
- [18] David Byrne. *Grendel-Scan*. 2015. URL: <https://sourceforge.net/projects/grendel/> (besucht am 19.02.2019).
- [19] Lavakumar Kuppan. *IronWasp*. 2014. URL: <https://ironwasp.org> (besucht am 19.02.2019).
- [20] Google.com. *ratproxy*. 2009. URL: <https://code.google.com/archive/p/ratproxy/> (besucht am 19.02.2019).
- [21] Google.com. *skipfish*. 2012. URL: <https://code.google.com/archive/p/skipfish/> (besucht am 19.02.2019).
- [22] Bernardo Damele. *sqlmap*. 2019. URL: <http://sqlmap.org> (besucht am 19.02.2019).
- [23] Subgraph. *Vega*. 2014. URL: <https://subgraph.com/vega/> (besucht am 19.02.2019).
- [24] siberas. *Watobo*. 2015. URL: <https://sourceforge.net/projects/watobo/> (besucht am 19.02.2019).
- [25] OWASP. *OWASP WebScarab Project*. 2014. URL: [https://www.owasp.org/index.php/Category:OWASP\\_WebScarab\\_Project](https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project) (besucht am 19.02.2019).
- [26] sectools.org. *Wfuzz*. 2019. URL: <http://www.edge-security.com/wfuzz.php> (besucht am 19.02.2019).
- [27] w3af.sourceforge. *w3af*. 2013. URL: <http://w3af.org> (besucht am 19.02.2019).
- [28] sensepost. *Wikto*. 2015. URL: <https://github.com/sensepost/wikto> (besucht am 19.02.2019).
- [29] OWASP. *OWASP Xenotix XSS Exploit Framework*. 2019. URL: <https://xenotix.in> (besucht am 19.02.2019).
- [30] nstalker.com. *N-Stalker*. 2019. URL: <http://www.nstalker.com> (besucht am 20.02.2019).
- [31] OWASP. *OWASP Vulnerable Web Applications Directory Project*. 2018. URL: [https://www.owasp.org/index.php/OWASP\\_Vulnerable\\_Web\\_Applications\\_Directory\\_Project#tab=On-Line\\_apps](https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project#tab=On-Line_apps) (besucht am 26.02.2019).

- [32] IBM. *Altoro Mutual*. 2019. URL: <https://demo.testfire.net> (besucht am 21.02.2019).
- [33] NTOSpider. *Web Scanner Test Site*. 2019. URL: <https://www.webscantest.com> (besucht am 21.02.2019).
- [34] Micro Focus Development Company. *Zero Bank*. 2018. URL: <http://zero.webappsecurity.com> (besucht am 21.02.2019).
- [35] Netsparker. *Bitcoin Web Site*. 2018. URL: <http://aspnet.testsparker.com> (besucht am 21.02.2019).
- [36] Acunetix. *Acunetix Acuart*. 2018. URL: <http://testphp.vulnweb.com> (besucht am 21.02.2019).
- [37] Micro Focus Development Company. *Crack Me Bank*. 2018. URL: <http://crackme.cenzic.com/kelev/view/home.php> (besucht am 21.02.2019).
- [38] Acunetix. *Security Tweets*. 2013. URL: <http://testhtml5.vulnweb.com> (besucht am 21.02.2019).
- [39] arachni scanner.com. *arachni web application security scanner framework*. 2017. URL: <http://www.arachni-scanner.com> (besucht am 19.02.2019).
- [40] cirt.net. *Nikto2*. 2019. URL: <https://cirt.net/Nikto2> (besucht am 19.02.2019).
- [41] Greenbone. *OpenVAS - Open Vulnerability Assessment System*. 2019. URL: <http://www.openvas.org/index-de.html> (besucht am 19.02.2019).
- [42] Nicolas Surribas. *Wapiti - The web-application vulnerability scanner*. 2018. URL: <http://wapiti.sourceforge.net> (besucht am 20.02.2019).
- [43] OWASP. *The OWASP Zed Attack Proxy*. 2019. URL: <https://www.zaproxy.org> (besucht am 20.02.2019).
- [44] Acunetix. *acunetix*. 2019. URL: <https://www.acunetix.com> (besucht am 20.02.2019).
- [45] Portswigger. *Burp Suite Editions*. 2019. URL: <https://portswigger.net/burp> (besucht am 20.02.2019).
- [46] Tenable. *nessus Professional*. 2019. URL: <https://www.tenable.com/products/nessus/nessus-professional> (besucht am 20.02.2019).
- [47] Netsparker. *netsparker*. 2019. URL: <https://www.netsparker.com> (besucht am 20.02.2019).
- [48] OWASP. *About The Open Web Application Security Project*. 2018. URL: [https://www.owasp.org/index.php/About\\_The\\_Open\\_Web\\_Application\\_Security\\_Project](https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project) (besucht am 28.12.2018).
- [49] OWASP. *OWASP Top 10 Application Security Risks*. 2017. URL: [https://www.owasp.org/index.php/Top\\_10-2017\\_Top\\_10](https://www.owasp.org/index.php/Top_10-2017_Top_10) (besucht am 28.12.2018).

# Anhang

## 1 Verwendete Abkürzungen

**WVS** - Web Application Vulnerability Scanner

**CVE** - Common Vulnerabilities and Exposures

**BSI** - Bundesamt für Sicherheit in der Informationstechnik

**DoS** - Denial of Service

**WAF** - Web Application Firewall

**OWASP** - The Open Web Application Security Project

**DAST** - Dynamic Application Security Testing

**SAST** - Static Application Security Testing

**NTLM** - New Technology Lan Manager

**BSoD** - “Blue Screen of Death” (Kritischer Systemfehler unter Microsoft-Windows)

**XSS** - Cross Site Scripting



## 2 OWASP: The Open Web Application Security Project

Das “Open Web Application Security Project” (OWASP) ist eine unabhängige, weltweite Community mit dem Ziel, die Bedeutung der Sicherheit von Webanwendungen »sichtbar zu machen«, Fachwissen zur Entwicklung und den Betrieb sicherer Webanwendungen zu verbreiten und frei zur Verfügung zu stellen. OWASP ist mit keinem Technologieunternehmen verbunden, obwohl der Einsatz kommerzieller Sicherheitstechnologien unterstützt wird. Sämtliche OWASP-Instrumente, wie Dokumente, Videos, Slides, Podcasts etc. sind kollaborativ produziert worden und sind zudem kostenlos unter einer freien Lizenz verwendbar. Die OWASP Foundation ist eine Non-Profit-Organisation, die sich vier Grundwerten verschrieben hat [48]:

- Offenheit: Von den Finanzen bis zum Code ist alles radikal transparent.
- Innovation: OWASP fördert und unterstützt Innovationen und Experimente zur Lösung von Software-Sicherheits Herausforderungen.
- Globalität: Jeder auf der ganzen Welt kann sich an der OWASP-Community beteiligen.
- Integrität: OWASP ist eine ehrliche und aufrichtige, herstellerneutrale, globale Gemeinschaft.

Zudem gibt es einen Verhaltenskodex mit folgenden Prinzipien:

- Führe alle beruflichen Tätigkeiten und Pflichten in Übereinstimmung mit allen anwendbaren Gesetzen und den höchsten ethischen Grundsätzen aus.
- Fördere die Umsetzung von Normen, Verfahren und Kontrollen für die Anwendungssicherheit und deren Einhaltung.
- Bewahre eine angemessene Vertraulichkeit gegenüber geschützter oder anderweitig sensibler Informationen, die im Rahmen einer beruflichen Tätigkeit auftreten.
- Übernimm die berufliche Verantwortung mit Sorgfalt und Ehrlichkeit.
- Kommuniziere offen und ehrlich.
- Vermeide Aktivitäten, die einen Interessenskonflikt hervorrufen oder anderweitig den Ruf des Arbeitgebers, den Informationssicherheitsberuf oder die Vereinigung beeinträchtigen könnten.
- Bewahre und verstärke unsere Objektivität und Unabhängigkeit.

- Weise unangemessenen Druck der von Seiten der Industrie oder anderen zurück.
- Verletze oder bestreite nicht absichtlich den Ruf von Kollegen, Kunden oder Arbeitgebern.
- Handle jeden mit Respekt und Würde.
- Vermeide Beziehungen, die die Objektivität und Unabhängigkeit von OWASP beeinträchtigen könnten.

## 2.1 Die OWASP Top Ten

Eines der bekanntesten und wichtigsten Projekte von OWASP sind die OWASP Top Ten. Alle drei bis vier Jahre wird eine Liste mit den 10 häufigsten Sicherheitsrisiken für Webanwendungen veröffentlicht, um Unternehmen und Organisationen für Websicherheit zu sensibilisieren. Die Liste wird von einem weltweiten Team von IT-Sicherheitsexperten zusammengestellt, und kann durch die große Akzeptanz in der Fachwelt als Sicherheitsrichtlinie angesehen werden.

Nachfolgend werden die aktuellen OWASP Top Ten aus dem Jahr 2017 aufgelistet [49]:

### 2.1.1 A1: Injection

Injection-Schwachstellen, wie beispielsweise SQL-, OS- oder LDAP-Injection, treten auf, wenn nicht vertrauenswürdige Daten von einem Interpreter als Teil eines Kommandos oder einer Abfrage verarbeitet werden. Ein Angreifer kann Eingabedaten dann so manipulieren, dass er nicht vorgesehene Kommandos ausführen oder unautorisiert auf Daten zugreifen kann.

### 2.1.2 A2: Fehler in der Authentifizierung

Anwendungsfunktionen, die im Zusammenhang mit Authentifizierung und Sessionmanagement stehen, werden häufig fehlerhaft implementiert. Dies erlaubt es Angreifern, Passwörter oder Session-Token zu kompromittieren oder die entsprechenden Schwachstellen so auszunutzen, dass sie die Identität anderer Benutzer vorübergehend oder dauerhaft annehmen können.



### **2.1.3 A3: Verlust der Vertraulichkeit sensibler Daten**

Viele Anwendungen schützen sensible Daten, wie personenbezogene Informationen und Finanzoder Gesundheitsdaten, nicht ausreichend. Angreifer können diese Daten auslesen oder modifizieren und mit ihnen weitere Straftaten begehen (Kreditkartenbetrug, Identitätsdiebstahl etc.). Vertrauliche Daten können kompromittiert werden, wenn sie nicht durch Maßnahmen, wie Verschlüsselung gespeicherter Daten und verschlüsselte Datenübertragung, zusätzlich geschützt werden. Besondere Vorsicht ist beim Datenaustausch mit Browsern angeraten.

### **2.1.4 A4: XML External Entities**

Viele veraltete oder schlecht konfigurierte XML Prozessoren berücksichtigen Referenzen auf externe Entitäten innerhalb von XML-Dokumenten. Dadurch können solche externen Entitäten dazu eingesetzt werden, um mittels URI Datei-Handlern interne Dateien oder File-Shares offenzulegen oder interne Port-Scans, Remote-Code-Executions oder Denial-of-Service Angriffe auszuführen.

### **2.1.5 A5: Fehler in der Zugriffskontrolle**

Häufig werden die Zugriffsrechte für authentifizierte Nutzer nicht korrekt um- bzw. durchgesetzt. Angreifer können entsprechende Schwachstellen ausnutzen, um auf Funktionen oder Daten zuzugreifen, für die sie keine Zugriffsberechtigung haben. Dies kann Zugriffe auf Accounts anderer Nutzer sowie auf vertrauliche Daten oder aber die Manipulation von Nutzerdaten, Zugriffsrechten etc. zur Folge haben.

### **2.1.6 A6: Sicherheitsrelevante Fehlkonfiguration**

Fehlkonfigurationen von Sicherheitseinstellungen sind das am häufigsten auftretende Problem. Ursachen sind unsichere Standardkonfigurationen, unvollständige oder ad-hoc durchgeführte Konfigurationen, ungeschützte Cloud-Speicher, fehlkonfigurierte HTTP-Header und Fehlerausgaben, die vertrauliche Daten enthalten. Betriebssysteme, Frameworks, Bibliotheken und Anwendungen müssen sicher konfiguriert werden und zeitnah Patches und Updates erhalten.

### **2.1.7 A7: Cross-Site-Scripting (XSS)**

XSS tritt auf, wenn Anwendungen nicht vertrauenswürdige Daten entgegennehmen und ohne Validierung oder Umkodierung an einen Webbrowser senden. XSS tritt auch auf, wenn eine Anwendung HTML- oder JavaScript-Code auf Basis von Nutzereingaben erzeugt. XSS erlaubt es einem Angreifer, Scriptcode im Browser eines Opfers auszuführen und so Benutzersitzungen zu übernehmen, Seiteninhalte verändert anzuzeigen oder den Benutzer auf bösartige Seiten umzuleiten.

### **2.1.8 A8: Unsichere Deserialisierung**

Unsichere, weil unzureichend geprüfte Deserialisierungen können zu Schwachstellen in der Remote-Code-Execution führen. Aber auch wenn das nicht der Fall ist, können Deserialisierungsfehler Angriffsmuster wie Replay-Angriffe, Injections und Erschleichung erweiterter Zugriffsrechte ermöglichen.

### **2.1.9 A9: Nutzung von Komponenten mit bekannten Schwachstellen**

Komponenten wie Bibliotheken, Frameworks etc. werden mit den Berechtigungen der zugehörigen Anwendung ausgeführt. Wird eine verwundbare Komponente ausgenutzt, kann ein solcher Angriff von Datenverlusten bis hin zu einer Übernahme des Systems führen. Applikationen und APIs, die Komponenten mit bekannten Schwachstellen einsetzen, können Schutzmaßnahmen unterlaufen und so Angriffe mit schwerwiegenden Auswirkungen verursachen.

### **2.1.10 A10: Unzureichendes Logging und Monitoring**

Unzureichendes Logging und Monitoring führt zusammen mit fehlender oder uneffektiver Reaktion auf Vorfälle zu andauernden oder wiederholten Angriffen. Auch können Angreifer dadurch in Netzwerken weiter vordringen und Daten entwenden, verändern oder zerstören. Viele Studien zeigen, dass die Zeit bis zur Aufdeckung eines Angriffs bei ca. 200 Tagen liegt sowie typischerweise durch Dritte entdeckt wird und nicht durch interne Überwachungs- und Kontrollmaßnahmen.

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorgelegte Bachelorarbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

---

Henning Janning, am 17. März 2019