

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: 04.04.2019

Erstprüfer: Prof. Dr.-Ing. 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 Web-Seiten 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 . . . . .	9
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 Scanzeiten . . . . .	33
4.2.4 Gesamtbewertung und Ranking . . . . .	34

<b>5 Diskussion</b>	<b>35</b>
5.1 Anzahl der gefundenen Schwachstellen . . . . .	35
5.2 Bedienung, Reporting und Scan-Geschwindigkeit . . . . .	36
5.2.1 Bedienung . . . . .	36
5.2.2 Reporting . . . . .	36
5.2.3 Scan-Geschwindigkeit . . . . .	37
5.3 Ranking . . . . .	37
5.4 Open Source im Vergleich mit kommerziellen WVS . . . . .	38
<b>6 Fazit und Ausblick</b>	<b>39</b>
<b>Quellenverzeichnis</b>	<b>41</b>
 <b>Anhang</b>	
<b>A Verwendete Abkürzungen</b>	<b>44</b>
<b>B Screenshots</b>	<b>46</b>
<b>Eidesstattliche Erklärung</b>	<b>53</b>

# Abbildungsverzeichnis

1.1	Gefundene Schwachstellen pro Jahr laut CVE Details . . . . .	2
1.2	Bitkom Studie - Bedrohungsszenarien . . . . .	3
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
B.1	Bericht Acunetix . . . . .	46
B.2	Bericht BurpSuite Pro . . . . .	47
B.3	Bericht Nessus . . . . .	48
B.4	Bericht Netsparker . . . . .	49
B.5	Bericht Arachni . . . . .	50
B.6	Bericht OpenVAS . . . . .	50
B.7	Bericht Nikto . . . . .	51
B.8	Bericht Wapiti . . . . .	52
B.9	Bericht ZAP . . . . .	52

# **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	Scanzeiten der Open Source WVS in Minuten . . . . .	33
4.4	Scanzeiten der kommerziellen WVS in Minuten . . . . .	33
4.5	Bewertung der Open Source WVS . . . . .	34
4.6	Bewertung der kommerziellen WVS . . . . .	34
4.7	Ranking aller WVS . . . . .	34

# 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, S.6]

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 Bezahlsysteme, 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 ideo-logisch 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-

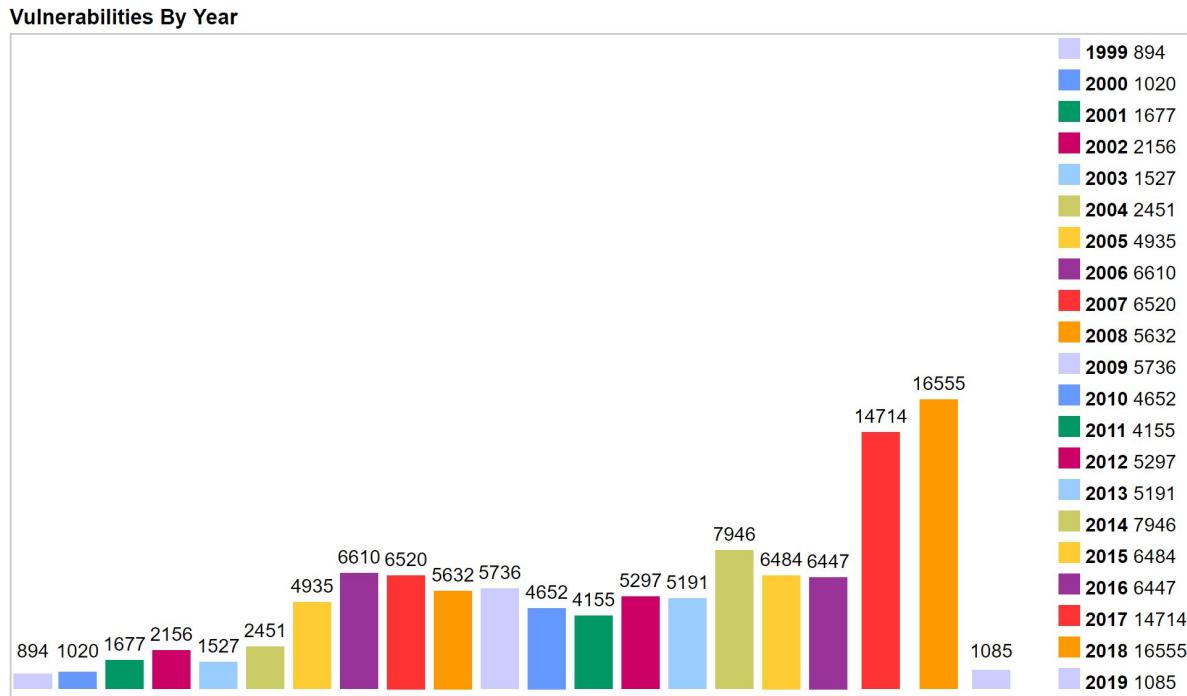


Abbildung 1.1: Gefundene Schwachstellen pro Jahr laut CVE Details [2]

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

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.

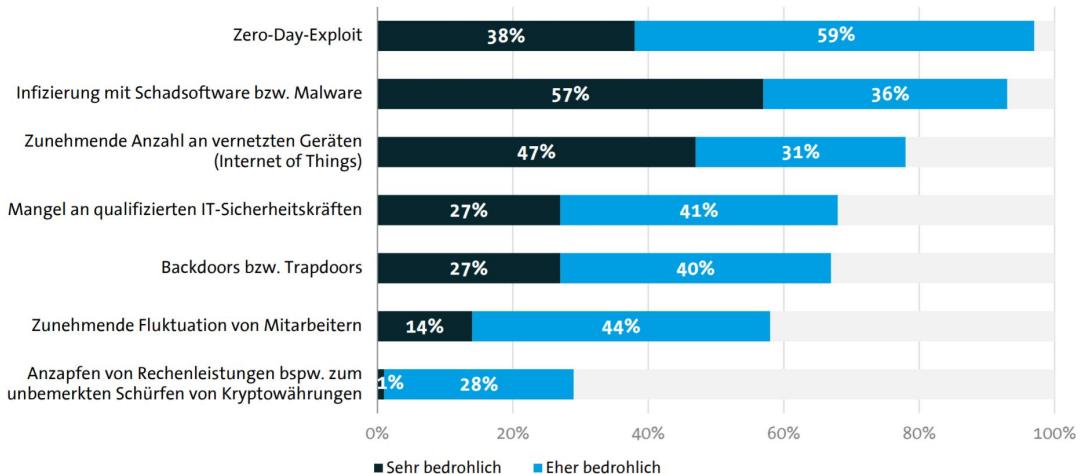
Neben dem Einsatz von Web Application Firewalls (“Web Shields”), die den Datenstrom zwischen Browser und Webapplikation überwachen, ist die Verwendung von Web Application Vulnerability Scannern (WVS) im Rahmen von Penetration Tests ein wesentlicher Bestandteil von Sicherheitskonzepten in diesem Bereich.

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

## Unentdeckte Sicherheitslücken als größte Bedrohung

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



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

**bitkom**

Abbildung 1.2: Bitkom Studie - Bedrohungsszenarien [3]

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. Wie schneiden die Open Source WVS im Vergleich mit kommerziellen WVS ab?

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 Arbeitweise 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 durchgeföhrten 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. Ob eine Anwendung als Webanwendung bezeichnet werden kann, hängt vom Einsatz von Webtechnologien ab. Daraus kann folgende Begriffsdefinition abgeleitet werden [9, S.1]:

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

Eine Webanwendung wird über einen Browser 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).

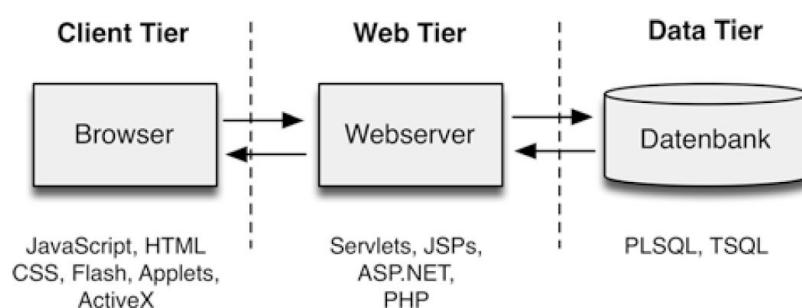


Abbildung 2.1: 3-Tier-Architektur einer Webanwendung [9]

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).

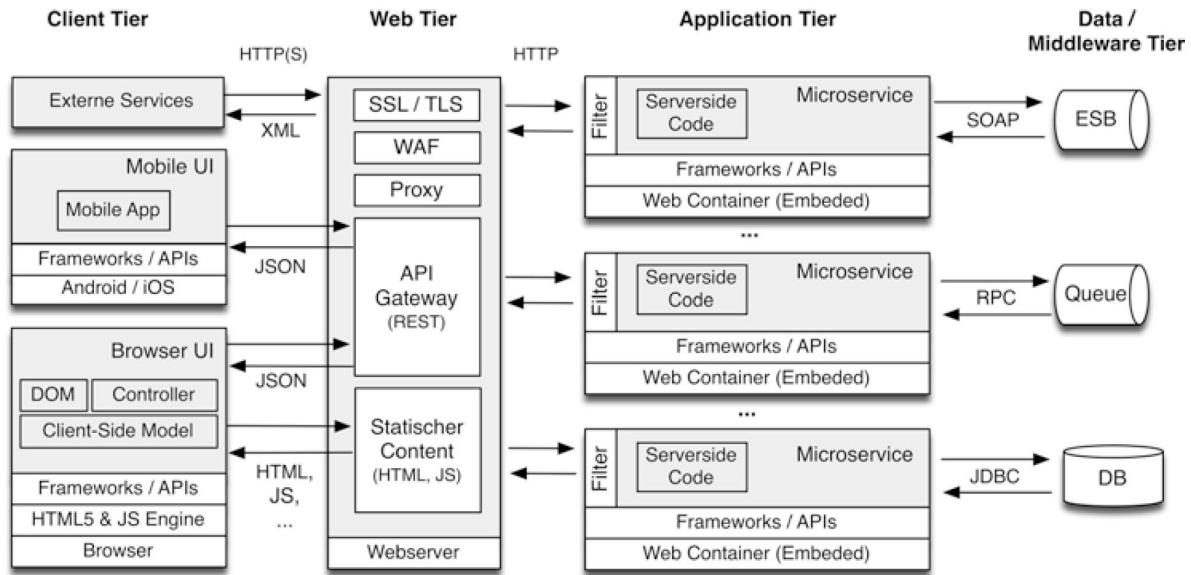


Abbildung 2.2: Enterprise Webanwendung auf Basis einer Microservice-Architektur [9]

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 resultieren in sogenannten Single Page Applications (SPAs), die nur noch aus einer einzigen HTML-Seite mit sehr viel JavaScript-Code bestehen, der im Hintergrund auf der Serverseite mit REST-Services kommuniziert und die Anzeige der Seiteninhalte steuert. Ein bekanntes Beispiel für solch eine Single Page Application ist Google Mail. [9]

## 2.2 Schwachstellen

Das 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, S.107]

Im Idealfall entdeckt ein Hersteller Schwachstellen selbst und kann sie beheben, bevor sie öffentlich bekannt werden. Sollte die Sicherheitslücke jedoch durch Dritte gefunden werden, kann ihr Lebenszyklus und die Gefährdung für die Nutzer unterschiedlich verlaufen. Der Verlauf hängt in erster Linie vom Entdecker, aber auch von der Reaktion der Hersteller ab [11]:

- Möchte der Entdecker die IT-Sicherheit verbessern, oder hat er kriminelle Absichten und benutzt die Schwachstelle für eigene Interessen?
- Inwieweit wird die Öffentlichkeit über die Schwachstelle informiert?
- Ist der Hersteller in der Lage, die Sicherheitslücke schnell zu beheben?

Um bekannte Schwachstellen herstellerübergreifend einheitlich benennen zu können, wurde der Industriestandard Common Vulnerabilities and Exposures (CVE) etabliert, der gewährleistet, dass Hersteller, Entdecker und weitere Beteiligte jeweils über dieselbe Schwachstelle diskutieren. Der CVE sammelt vorhandene Informationen über die Schwachstelle und bietet die Möglichkeit zu statistischen Auswertungen.

Wie in Abbildung 2.3 dargestellt, lässt sich der Lebenszyklus einer Schwachstelle, die durch Dritte entdeckt wurde, auf das folgende Schema zurückführen[11]:

1. Entdeckung der Schwachstelle durch einen Dritten.
2. Es gibt mehrere Möglichkeiten, wie der Hersteller von der Schwachstelle erfährt:
  - „Full Disclosure“: Alle Informationen über die Schwachstelle werden vom Entdecker öffentlich gemacht.
  - „Coordinated Disclosure“: Der Entdecker kontaktiert direkt den Hersteller und koordiniert mit ihm das weitere Vorgehen, hier werden zunächst keine Informationen

an die Öffentlichkeit gegeben.

- „Zero-Day-Exploit“: Niemand wird über die Schwachstelle informiert, stattdessen wird sie für Angriffe ausgenutzt. Öffentlichkeit und Hersteller erfahren erst nach erfolgreichen Angriffen von der Schwachstelle.
- Indirekte Benachrichtigung an den Hersteller über einen sogenannten Schwachstellen-Broker.

3. Der Hersteller beginnt damit, die Sicherheitslücke zu schließen, indem er die Software nachbessert und einen Patch entwickelt.

4. Veröffentlichung eines “Advisorys”, einer Schwachstellenwarnung, die Informationen über die Sicherheitslücke, eine Gefährdungsbewertung und vorläufige Gegenmaßnahmen beinhaltet. Das Advisory erscheint in der Regel zeitgleich mit dem Patch, bei hoher Gefährdung und wenn die Öffentlichkeit schon von der Schwachstelle erfahren hat, kann es auch vor der Fertigstellung des Patches veröffentlicht werden.

5. Der Patch ist fertiggestellt und wird vom Hersteller mit einer entsprechenden Beschreibung (Bulletin) zur Verfügung gestellt. Mit der Veröffentlichung des Patches und dem Bulletin steigt die Gefahr für ungepatchte Systeme, da nun auch potentielle Angreifer umfangreiche Informationen über die Schwachstelle erhalten.

6. Der Patch wird von den Benutzern der betroffenen Software installiert und die Sicherheitslücke dadurch geschlossen.

Der Ablauf kann von diesem Schema abweichen; wenn beispielsweise eine Schwachstelle vom Hersteller zunächst als nicht kritisch eingestuft und daher auf die Entwicklung eines Patches verzichtet wird, kann dies dazu führen, dass zu einem späteren Zeitpunkt schnell reagiert werden muss, wenn sich die Schwachstelle doch als ausnutzbar erweist. [11]

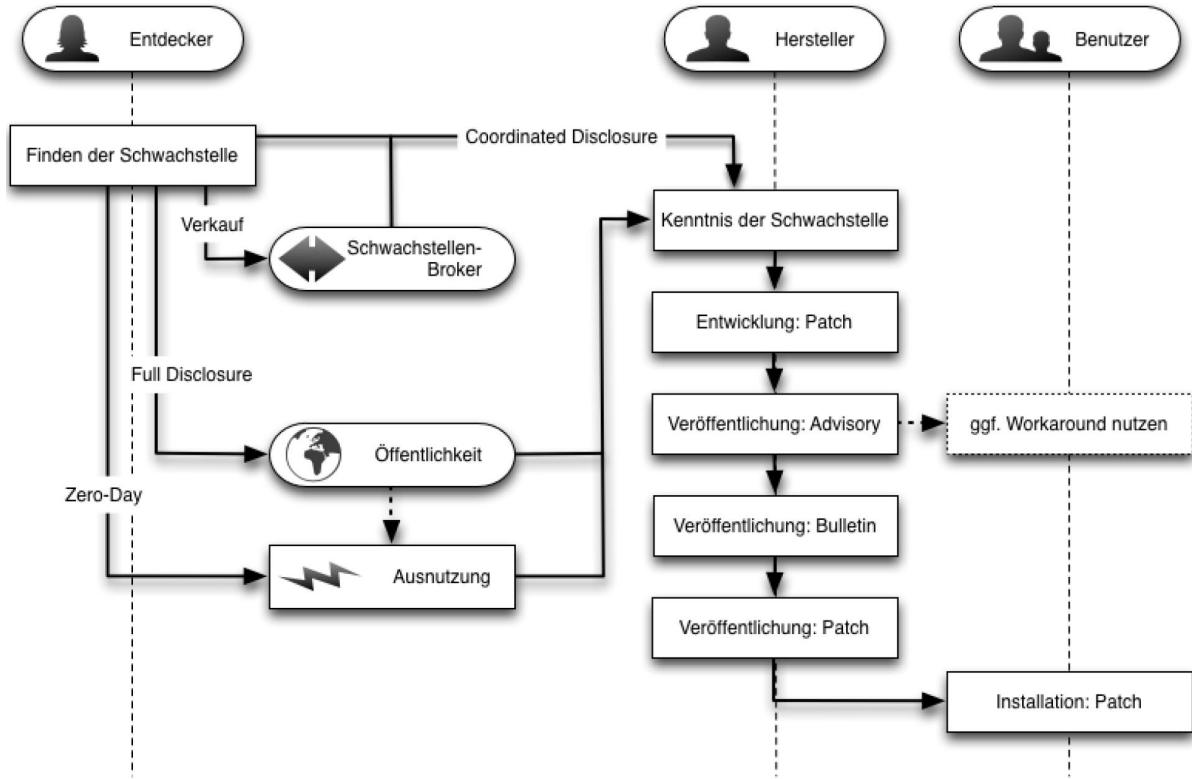


Abbildung 2.3: Lebenszyklus einer Schwachstelle [11]

## 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. Als Assets werden Bestandteile einer Webanwendung wie Daten, Systeme oder Funktionen bezeichnet, die Schutzbedarf im Hinblick auf die primären Schutzziele haben [4].

Diese primären Schutzziele lauten [4]:

- Vertraulichkeit
- Integrität
- Verfügbarkeit

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

Das Thema Webanwendungssicherheit bezieht sich auf den gesamten Lebenszyklus einer Webanwendung und ist ein wichtiger Bestandteil der Qualitätssicherung. Neben der Bedrohungsabwehr und der Vorbeugung von Sicherheitslücken befasst sie sich auch mit deren Identifizierung und Behebung. [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 versursachen (Denial-of-Service Attacken) stellen für viele Unternehmen ein sehr kritisches Ereignis dar.

Mehrere Organisationen haben es sich zur Aufgabe gemacht, Sicherheitsrisiken für Webanwendungen zu katalogisieren und zu klassifizieren. Das Web Application Security Consortium (WASC), ein Zusammenschluss von Sicherheitsexperten, die an Sicherheitsstandards für Webanwendungen arbeiten, hat mit seiner "WASC Threat Classification v.2.0" einen umfassenden Katalog mit Bedrohungen herausgegeben [12].

OWASP veröffentlicht alle drei bis vier Jahre mit den "OWASP Top Ten" [13] eine Liste mit den 10 häufigsten Sicherheitsrisiken für Webanwendungen, die durch die große Akzeptanz in der Fachwelt als Sicherheitsrichtlinie angesehen werden kann.

### 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, S.107-108]

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 mit Hilfe von 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 Penetrations-tester 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 Cases 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 sensitiven Funktionen wie beispielsweise Transaktionsmanagement, Authentisierung und Kryptographie wichtig. Die Ergebnisse des Code Reviews fließen zurück in den Implementierungsprozess.
- Statische Code Scanner: Mittels Statischer Code Analyse (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 ho-

he 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 Beispieldangriffen, 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. [14]

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

- Informationsgewinnung
- Identifizierung der Angriffsfläche
- Identifizierung von System und Anwendungen
- Schwachstellenrecherche und Priorisierung der Schwachstellen nach Ausnutzbarkeit und Auswirkung
- Angreifen 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 (gefundenen Schwachstellen werden nicht ausgenutzt) oder aggressiv (gefundenen 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 auf Anwendungsebene den Verkehr zwischen Clients und Webservern überprüfen. Sie sind in der Lage, HTTP-Traffic zu filtern und gegebenenfalls zu blockieren, um die Webanwendung zu schützen. Eine WAF überprüft dabei alle eingehenden Anfragen und die ausgehenden Antworten des Webservers. 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 wirklichen Webserver analysiert und die Websessions bei Bedarf terminiert.

Zu den Angriffen, die eine WAF üblicherweise 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, Webanwendung zu entwickeln, die ausreichend sicher sind und getestet wurden. Bei der Durchführung von Tests sollte eine WAF stets deaktiviert sein, um die Testergebnisse nicht zu verfälschen. Um aus einer WAF den größtmöglichen Nutzen zu ziehen, sollte die Konfiguration möglichst durch Fachleute auf die jeweilige Webanwendung angepasst werden. [4, 15]

## 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.

Um Webanwendungen zu analysieren, befindet sich der WVS auf einem Client-Computer, in der Regel werden bei der Analyse vier Phasen durchlaufen [16]:

- Crawl/Scan: Ähnlich einer Suchmaschine inspiziert der WVS die komplette Webanwendung, besucht dabei alle Links und füllt gegebenenfalls Formfelder mit Testwerten aus. So erlangt der WVS Kenntnis über Aufbau und Funktionsweise der Webanwendung und Inhalten von Formularseiten. Häufig wird die Webanwendung auch auf vorhandene Verzeichnisse oder Dateien durchsucht, die Informationen beinhalten. Die Scan-Phase kann durch die Verwendung von öffentlichen Suchmaschinen unterstützt werden, die gezielt nach Informationen über die anzugreifende Webanwendung sucht. Für die Automatisierung dieser Abfragen und das Erstellen von Suchmustern gibt es spezielle Tools.
- Analyse: Im zweiten Schritt analysiert der WVS die Webanwendung auf ihre Sicherheit und speichert die relevanten Erkenntnisse in einer Datenbank. Interessant sind hier zum Beispiel die Art des Webservers, welche Technologien und Tools verwendet werden (CGI, PHP, JavaScript, JSP usw.) oder Mechanismen zum Session-Tracking (Cookies).
- Audit/Penetrationstest: Die Erkenntnisse aus der Analysephase werden nun dazu verwendet, fehlerhafte und nicht zulässige Eingabemuster zu erzeugen, die dann an die Webanwendung geschickt werden. Diese Phase wird von einigen WVS in eine "schadlose" und "potentiell schadhafte" Prüfung unterteilt.
- Reporting: Am Schluss werden die Ergebnisse in Berichten zusammengefasst. Je nach WVS variiert hier der Umfang, teilweise werden nur kritische Sicherheitslücken genannt, teilweise gibt es detaillierte Beschreibungen von allen Schwachstellen mit Hinweisen zur Behebung der Probleme. Bei den meisten WVS hat sich ein System etabliert, bei dem die gefundenen Schwachstellen je nach Schweregrad in vier Kategorien eingeteilt werden [17]:

- 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. Bei den WVS, die die Scanzeit nicht im Report aufführen, wurde manuell gemessen. 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 abilden 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 Scanergebnis (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 wurden schließlich mit einem Abzug bei der Gesamtpunktzahl berücksichtigt.

Für das Scanergebnis 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.

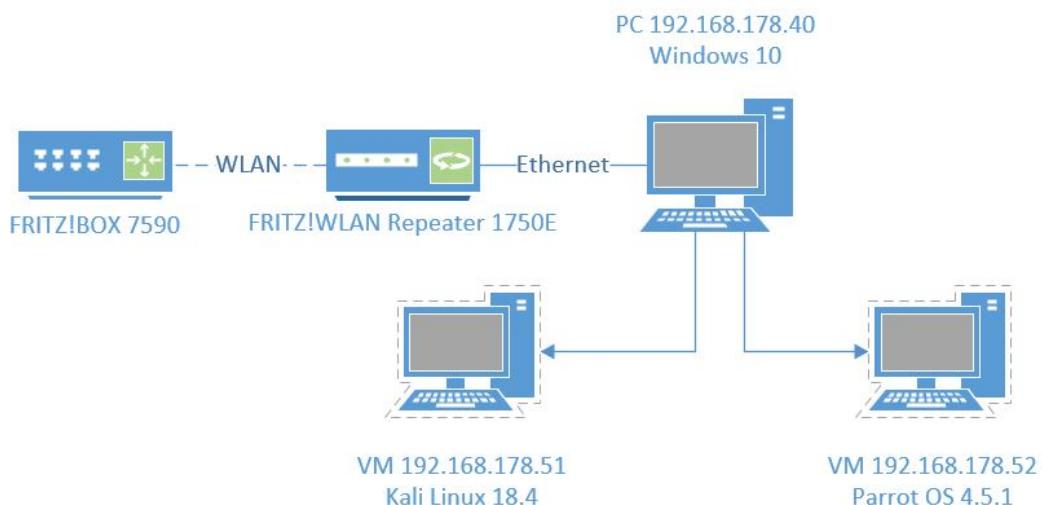


Abbildung 3.1: Testaufbau

## 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**

<b>WVS</b>	<b>Entwickler</b>	<b>Version</b>	<b>Verwendete Plattform</b>
<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**

<b>WVS</b>	<b>Anbieter</b>	<b>Version</b>	<b>Verwendete Plattform</b>
<b>Acunetix</b>	Acunetix	12.0.190206130	Windows/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**

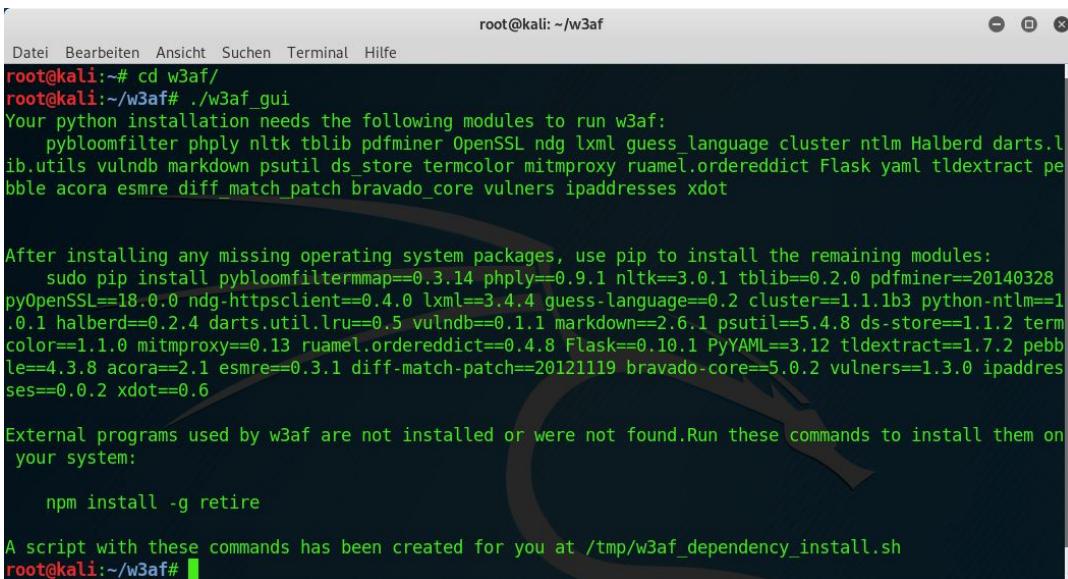
Nachfolgend werden alle WVS aufgelistet, die als Kandidaten in Erwägung gezogen wurden, bei näherer Betrachtung jedoch nicht die erforderlichen Voraussetzungen erfüllten, um in die Evaluation aufgenommen zu werden.

#### **3.2.3.1 Free und Open Source WVS**

- GoLismero [18]: GoLismero ist ein in Python geschriebenes Framework, das verschiedene Penetrationtesting-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 [19]: 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 [20]: Seit 2013 gibt es auf dem SourceForge-Repository von David Byrne keine Veränderung.
- Iron Wasp [21]: Die aktuelle Version des Windows-Programms von Lavakumar Kuppan ist ein Beta-Release aus dem Jahr 2015.
- Ratproxy [22]: Die Entwicklung wurde 2009 von Google eingestellt.
- Skipfish [23]: Ein weiteres Projekt von Google, das 2012 eingestellt wurde.
- SQLmap [24]: SQLmap ist ein beliebtes Tool zum Auffinden von SQLi Schwachstellen, ist aber darauf beschränkt.
- Vega [25]: 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 [26]: Die letzte Änderung des OpenSource Scanners stammt aus dem Jahr 2015.
- WebScarab [27]: Der Vorgänger von OWASPs Zed Attack Proxy ist veraltet (Latest

Release 2011), OWASP empfiehlt, auf ZAP umzusteigen.

- Wfuzz [28]: Der in Python geschriebene Scanner verwendet die Bibliothek Pycurl für HTTP-Requests, diese unterstützt keine SSL/TLS Verschlüsselung.
- W3af [29]: 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#
root@kali:~/w3af# ./w3af_gui
Your python installation needs the following modules to run w3af:
    pybloomfilter phply nltk tbllib pdfminer OpenSSL ndg lxml guess_language cluster ntlm Halberd darts.l
ib.utils vulndb markdown psutil ds_store termcolor mitmproxy ruamel.ordereddict 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 pybloomfilter==0.3.14 phply==0.9.1 nltk==3.0.1 tbllib==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.ordereddict==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 [30]: Wikto ist eine Windows-Portierung von Nikto und bedarf daher keiner eigenen Evaluation.
- Xenotix [31]: 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 [32]: 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 [33]. 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 [34].

- 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 [35].

- WA3: Zero Bank

Eine weitere Online-Banking Web-Applikation, entwickelt von Hewlett-Packard/Micro Focus [36].

- WA4: Bitcoin Web Site

Diese von Netsparker entwickelte Applikation ist in ASP.NET geschrieben und simuliert eine Online-Seite für Bitcoin-Transaktionen [37].

- WA5: Acuart

Eine Test-Seite von Acunetix, der einen Online-Shop für Kunstwerke simuliert, geschrieben in PHP [38].

- WA6: Crack Me Bank

Eine in PHP geschriebene Online-Banking Seite, entwickelt von Trustwave [39]:

- WA7: Security Tweets

Security Tweets ist eine von Twitter inspirierte Social Networks Applikation, die von Acunetix entwickelt wurde und HTML5 verwendet [40].

# 4 Evaluation

## 4.1 Gefundene Schwachstellen per Webanwendung

H, M, L und I: High, Medium, Low und Informational. Aus Gründen der Vergleichbarkeit wurden die Werte aus der zusätzlichen Kategorie “Critical” bei Nessus und Netsparker mit in die Kategorie High übernommen. WA1-7: Verwundbare Web-Applikationen.

	Arachni				Nikto				OpenVAS				Wapiti				ZAP			
	H	M	L	I		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				BurpSuite 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>	<b>180</b>	<b>152</b>	<b>79</b>	<b>106</b>	<b>79</b>	<b>5</b>	<b>23</b>	<b>444</b>	<b>37</b>	<b>71</b>	<b>9</b>	<b>148</b>	<b>115</b>	<b>62</b>	<b>99</b>	<b>113</b>
<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 [41]:

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**

The screenshot shows the Arachni WebUI 0.5.12 interface. At the top, it displays "Arachni v1.5.1 - WebUI v0.5.12" and a navigation bar with "Scans", "Profiles", "Dispatchers", and "Users". A blue "Administrator" button is visible. The main content area shows a URL "http://testhtml5.vulnweb.com/" being audited. Below the URL, there's a status bar indicating "Currently auditing:" followed by a link. The main panel has several sections: "Pages discovered" (20), "Requests performed" (10024), "Request concurrency" (20), and "Response times" (0.525 s). There's also a "Scanning" button with a progress bar. The "Issues" section shows 15 items, with a note about context missing during the scan. Below this, there are tabs for "All [15]", "Fixed [0]", "Verified [0]", "Pending verification [0]", "False positives [0]", and "Awaiting review [0]". The "Cross-Site Scripting (XSS)" issue is highlighted in red. The interface uses a clean, modern design with a dark header and light body.

Abbildung 4.1: Arachni WebUI 0.5.12

- Nikto [42]:

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. Der generierte Report listet alle gefundenen Schwachstellen auf, unterscheidet diese jedoch im Gegensatz zu den meisten anderen WVS nicht zwis-

schen High, Medium, Low und Informational. Die Scan-Geschwindigkeit ist überdurchschnittlich.

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

- OpenVAS [43]:

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

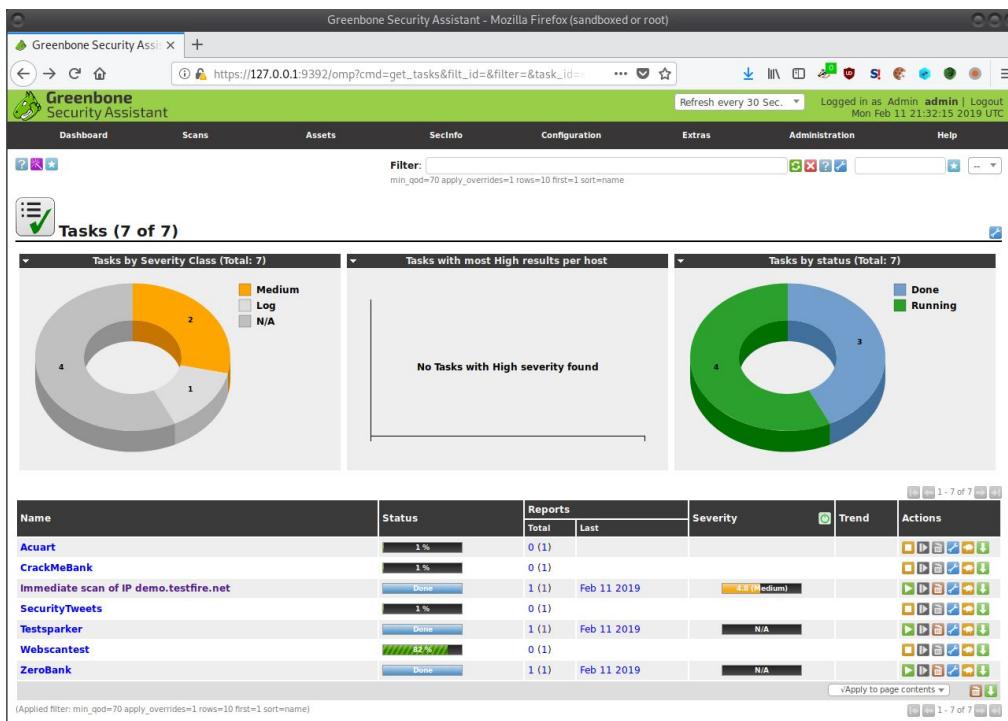


Abbildung 4.2: OpenVAS im Greenbone Security Assistant

- Wapiti [44]:

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 dritten Platz ein.

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

- Zed Attack Proxy [45]:

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 3, Bedienung 3, Geschwindigkeit 0

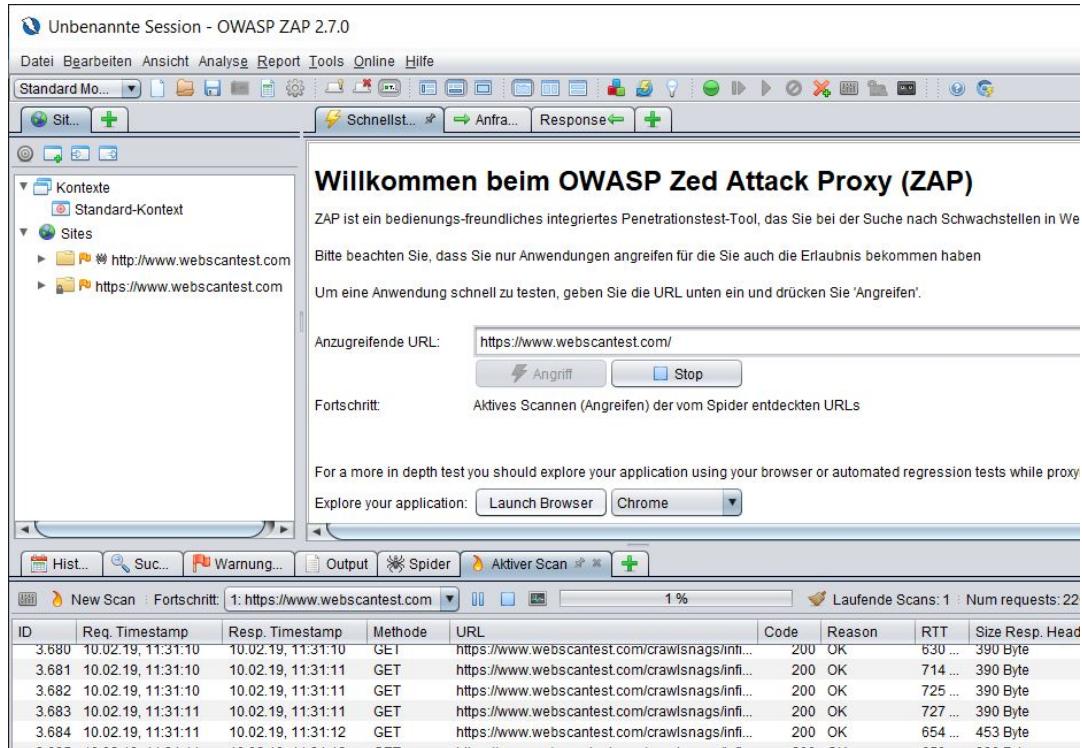


Abbildung 4.3: Grafische Benutzeroberfläche von ZAP

## 4.2.2 Kommerziell

- Acunetix [46]:

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. Auffällig ist die im Vergleich zu allen anderen WVS ungewöhnlich hohe Scan-Geschwindigkeit, die sich im Minutenbereich einordnet. 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 könnten hingegen ausführlicher sein. Aufgrund der Abstürze wurden bei der Gesamtpunktzahl 3 Punkte abgezogen.

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

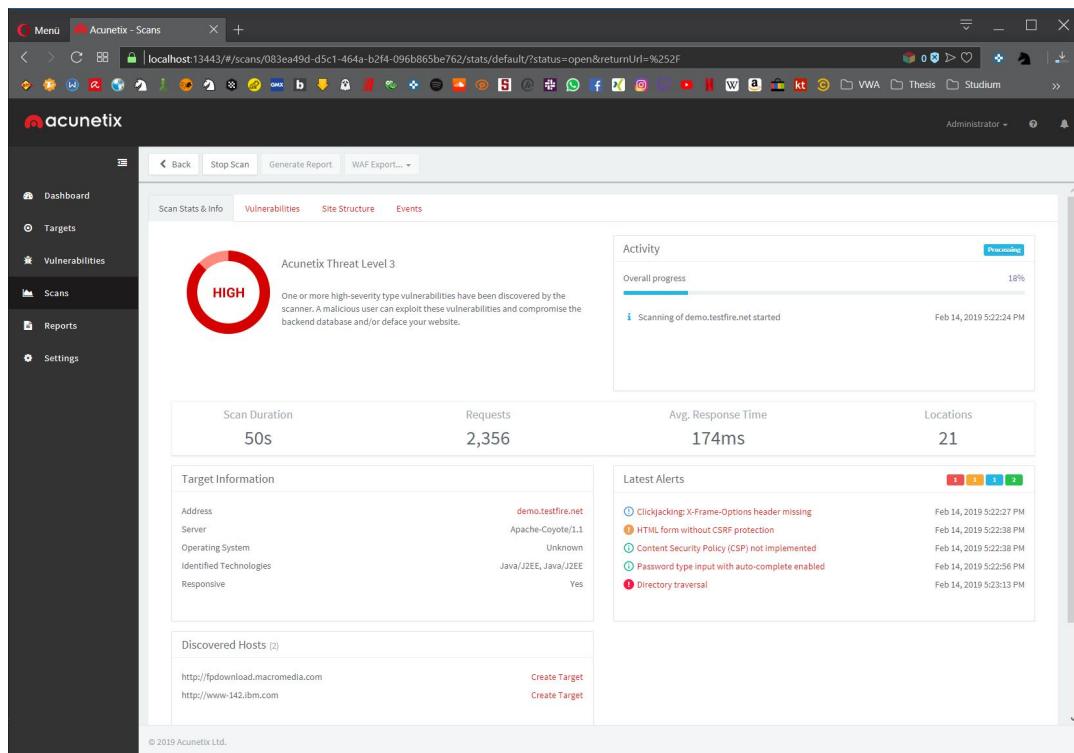


Abbildung 4.4: Weboberfläche von Acunetix

- BurpSuite Pro [47]:

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.

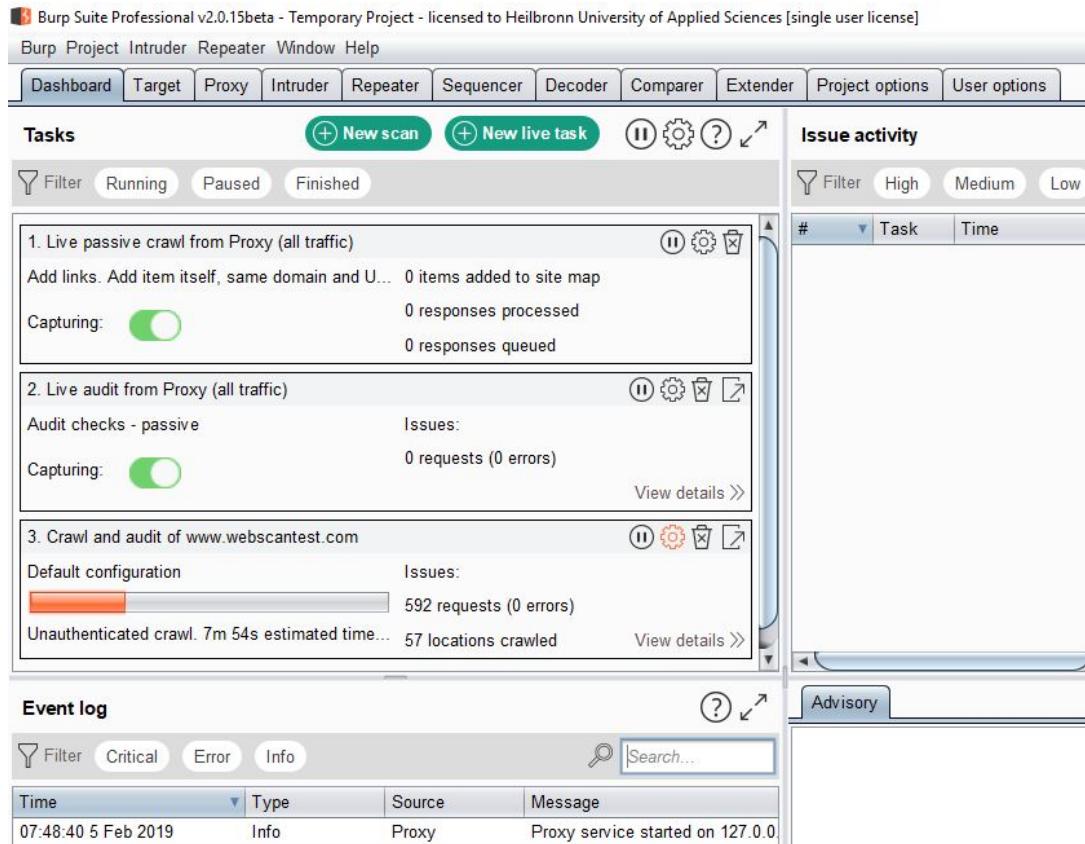


Abbildung 4.5: Dashboard von BurpSuite Pro

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**

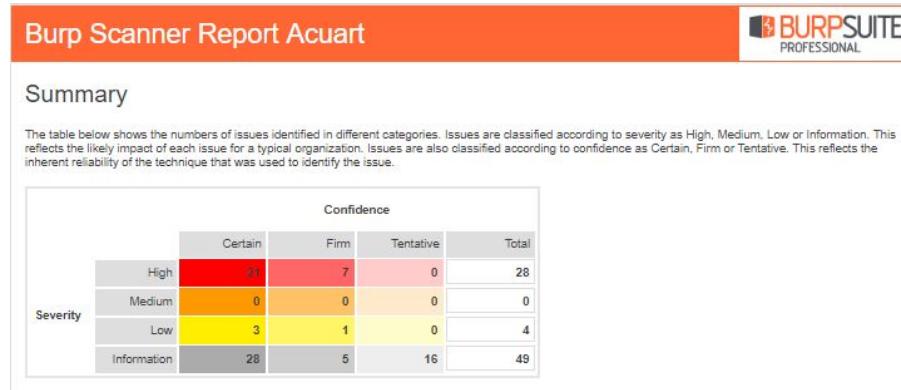


Abbildung 4.6: Confidence-Tabelle im Report von BurpSuite Pro

- Nessus [48]:

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**

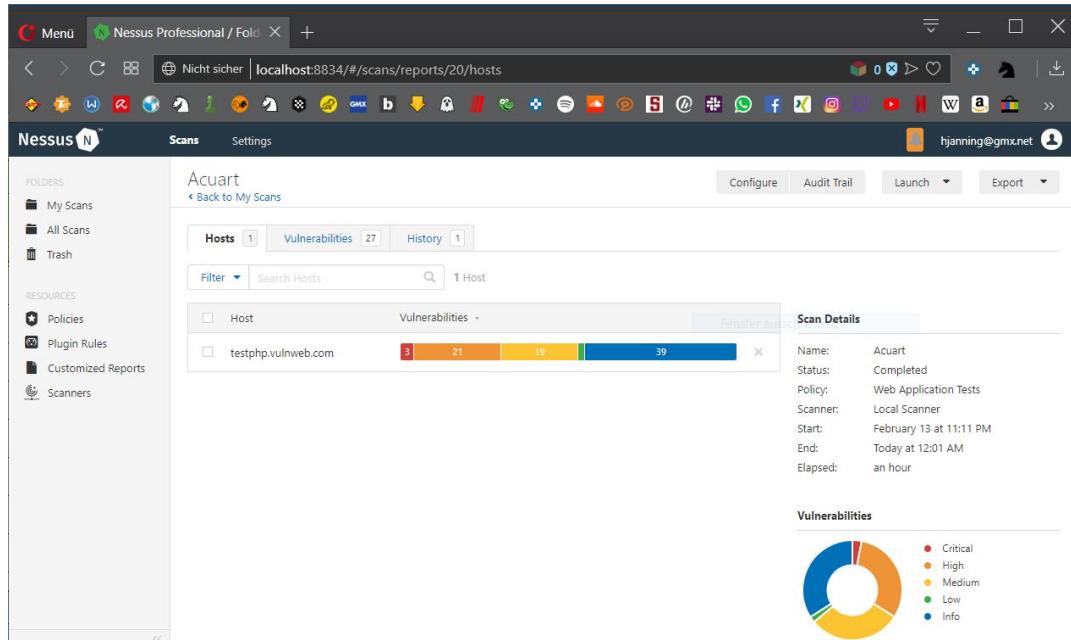


Abbildung 4.7: Grafische Benutzeroberfläche von Nessus

- Netsparker [49]:

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. Wie Nessus hat auch Netsparker nach High, Medium, Low und Informational 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

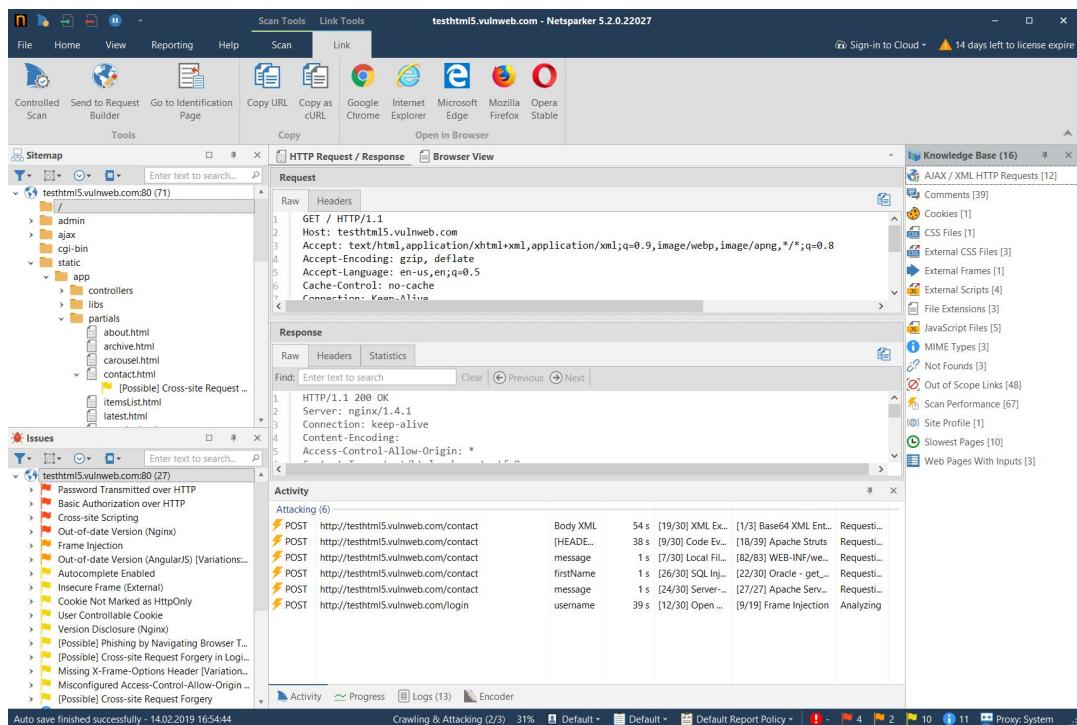


Abbildung 4.8: Grafische Benutzeroberfläche von Netsparker

### 4.2.3 Scanzeiten

In den beiden folgenden Tabellen sind die Zeiten aufgeführt, die die WVS benötigten, um die jeweiligen Webanwendungen zu scannen.

#### Open Source WVS

	<b>Arachni</b>	<b>Nikto</b>	<b>OpenVAS</b>	<b>Wapiti</b>	<b>Zed Attack Proxy</b>
<b>Altoro Mutual</b>	41	96	38	88	615
<b>Webscantest</b>	86	80	90	82	821
<b>Zero Bank</b>	42	29	85	31	786
<b>Aspnet Testspark</b>	28	20	29	14	579
<b>Acuart</b>	39	9	43	23	660
<b>Crack Me Bank</b>	38	32	45	28	704
<b>Security Tweets</b>	28	19	33	27	523
<b>Total</b>	<b>302</b>	<b>285</b>	<b>363</b>	<b>293</b>	<b>4688</b>

Tabelle 4.3: Scanzeiten der Open Source WVS in Minuten

#### Kommerzielle WVS

	<b>Acunetix</b>	<b>Burp Suite Pro</b>	<b>Nessus</b>	<b>Netsparker</b>
<b>Altoro Mutual</b>	15	56	44	46
<b>Webscantest</b>	60	102	82	241
<b>Zero Bank</b>	26	71	68	47
<b>Aspnet Testspark</b>	11	30	27	52
<b>Acuart</b>	6	45	61	37
<b>Crack Me Bank</b>	28	39	31	71
<b>Security Tweets</b>	8	41	51	20
<b>Total</b>	<b>154</b>	<b>384</b>	<b>364</b>	<b>514</b>

Tabelle 4.4: Scanzeiten der kommerziellen WVS in Minuten

#### 4.2.4 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	3
Bedienung (20%) *2	3	2	1	2	3
Geschwindigkeit (10%)	3	3	2	3	0
Gesamt	32	14	28	14	12

Tabelle 4.5: Bewertung der Open Source WVS

##### Kommerzielle WVS

	Acunetix	BurpSuite Pro	Nessus	Netsparker
Scanergebnis (50%) *5	4	4	2	3
Reporting (20%) *2	3	4	2	3
Bedienung (20%) *2	3	3	3	2
Geschwindigkeit (10%)	4	2	2	1
Gesamt	36	36	22	26

Tabelle 4.6: Bewertung der kommerziellen WVS

##### Ranking aller WVS

	WVS	Gesamt	Scanergebnis	Reporting	Bedienung	Geschwindigkeit
1	BurpSuite Pro	36	4	4	3	2
2	Acunetix	33 <sup>1</sup>	4	3	3	4
3	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	14	1	1	2	3
7	Wapiti	14	1	1	2	3
9	ZAP	12	0	3	3	0

Tabelle 4.7: Ranking aller WVS

<sup>1</sup>Ergebnis nach Abzug von 3 Punkten

# 5 Diskussion

## 5.1 Anzahl der gefundenen Schwachstellen

Die Tabellen mit den gefundenen Schwachstellen zeigen eine große Bandbreite an unterschiedlichen Ergebnissen. Während Zed Attack Proxy insgesamt nur 63 Schwachstellen findet, sind es bei BurpSuite Pro mit 551 Funden ungefähr neun mal so viele. Die beiden Terminalprogramme Nikto und Wapiti liegen fast gleich auf mit 97 und 93 Funden und sind damit nur geringfügig besser als das Schlusslicht Zed Attack Proxy. Mit 265 Funden hat Nessus bei den kommerziellen Scannern das schlechteste Ergebnis, überraschend ist hier vor allem, dass die Open Source Software und Nessus-Abspaltung OpenVAS fast 200 Schwachstellen mehr findet. OpenVAS lässt mit diesem Ergebnis auch noch den kommerziellen WVS Netsparker (389 Funde) hinter sich und belegt insgesamt den dritten Platz bei der Anzahl gefundener Schwachstellen. Arachni findet 319 Schwachstellen und reiht sich damit zwischen Nessus und Netsparker ein. Acunetix muss sich mit 517 Funden nur BurpSuite Pro geschlagen geben und belegt hier den zweiten Platz.

Auch wenn die Werte in die Kategorien High, Medium, Low und Informational aufgeteilt werden, gibt es eine breite Palette an Ergebnissen, es gibt fast keinerlei Übereinstimmungen oder erkennbare Muster. Bei den WVS, die bei den insgesamt gefundenen Schwachstellen ungefähr gleich auf liegen, lässt sich dies veranschaulichen: Acunetix hat zum Beispiel mehr als 330 Funde in High und Medium eingeteilt und ca. 180 in Low und Informational, BurpSuite Pro stuft die Schwachstellen als nicht so schwerwiegend ein, gerade einmal 84 Funde finden sich in High und Medium, aber 467 in Low und Informational. Arachni hat 134 Funde in die Kategorie High eingestuft, Nessus nur 37 und OpenVAS gar nur neun.

Selbst wenn man die Ergebnisse nach den verwundbaren Webanwendungen aufteilt, gibt es keine Wiedererkennungswerte, weder gibt es eine Webseite, bei der von allen WVS besonders viele Schwachstellen gefunden wurden, noch sticht eine mit besonders wenigen Funden heraus. Auch die Anzahl der Funde pro Webanwendung ist von WVS zu WVS sehr verschieden. So findet zum Beispiel Acunetix auf der Seite Zero Bank (WA3) 68 Schwachstellen, BurpSuite Pro nur 14, andererseits findet BurpSuite Pro auf Webscantest (WA2) 169 Schwachstellen und Acunetix nur 86. OpenVAS findet auf Security Tweets (WA7) 110 Schwachstellen, Netsparker nur 31. Auf Webscantest (WA2) hingegen

findet Netsparker doppelt so viele (99) Schwachstellen wie OpenVAS (49). Beziiglich der Webanwendungen gibt es eine interessante Beobachtung: Acuart (WA5) und Security Tweets (WA7) wurden beide von Acunetix entwickelt, die naheliegende Vermutung, dass der WVS von Acunetix hier besonders viele Schwachstellen finden müsste, hat sich nicht bestätigt. Ebensowenig findet auch Netsparker auf der selbst entwickelten Bitcoin Web Site (WA4) relevant mehr Schwachstellen als die anderen WVS, Acunetix und BurpSuite Pro finden sogar deutlich mehr.

Zusammenfassend muss festgehalten werden, dass bei der Anzahl der gefundenen Schwachstellen keinerlei Regelmäßigkeiten erkennbar sind, und so lässt sich die erste Forschungsfrage **Wie viele Schwachstellen werden von den WVS gefunden?** nur mit einem Blick auf die Tabelle beantworten - in absoluten Zahlen pro WVS.

## 5.2 Bedienung, Reporting und Scan-Geschwindigkeit

Dieses Kapitel widmet sich der Fragestellung **Wie unterscheiden sich die WVS in den Kategorien Bedienung, Reporting und Scan-Geschwindigkeit?**

### 5.2.1 Bedienung

In dieser Kategorie gibt es keinen WVS, der die volle Punktzahl erreicht, es gibt aber auch keinen mit null Punkten. Den beiden Terminalprogrammen Nikto und Wapiti fehlt es naturgemäß am Komfort, den eine grafische Benutzeroberfläche bietet, trotzdem schneiden sie noch besser ab als OpenVAS, das erst nach umfangreicher Einarbeitungszeit sinnvoll genutzt werden kann. Das Streben nach Automatisierung geht hier zu Lasten der Bedienbarkeit. Netsparker befindet sich mit seiner überladenen Benutzeroberfläche ebenfalls nur im Mittelfeld, Acunetix, BurpSuite Pro und Nessus sind - erwartungsgemäß für kommerzielle WVS - sehr übersichtlich und benutzerfreundlich. Die Open Source Programme Arachni und Zed Attack Proxy stehen dem jedoch in nichts nach und sind ebenfalls leicht verständlich und intuitiv zu bedienen.

Insgesamt gibt es in dieser Kategorie keinen klaren Sieger, mit Ausnahme von OpenVAS sind alle WVS mühelos und unproblematisch zu bedienen.

### 5.2.2 Reporting

Nikto und Wapiti schneiden in dieser Kategorie unterdurchschnittlich ab, da sie die gefundenen Schwachstellen nicht nach Schweregrad unterteilen. Das Reporting von Nes-

sus und OpenVAS könnte ausführlicher sein, OpenVAS verzichtet auf die Nennung von gefundenen Schwachstellen der Kategorie “Informational”, bei Nessus sind die Beschreibungen der Schwachstellen und Lösungsvorschläge nur online abzurufen, das bedeutet zusätzlichen Aufwand, wenn der Auftraggeber einen gedruckten Bericht wünscht. Überdurchschnittlich schneiden Zed Attack Proxy, Acunetix und Netsparker ab, die alle sehr umfangreiche Beschreibungen der Schwachstellen liefern, Netsparker ist hier hervorzuheben für die zusätzliche Kategorie “Critical” und die besondere Kennzeichnung gesicherter Ergebnisse als “Confirmed”. Überragende Ergebnisse liefern Arachni und BurpSuite Pro, Arachni sticht neben der Unterscheidung zwischen gesicherten und noch zu überprüfenden Ergebnissen mit seinen ausführlichen Statistikfunktionen heraus, BurpSuite Pro mit seiner “Confidence”-Tabelle, die die Schwachstellen noch genauer zwischen gesichert, wahrscheinlich und tentativ differenziert. Zudem bieten beide äußerst detaillierte Beschreibungen der Schwachstellen und sind trotzdem sehr übersichtlich gestaltet.

### 5.2.3 Scan-Geschwindigkeit

Mit einer Ausnahme liegen die Scan-Geschwindigkeiten alle im vertretbaren Bereich, Lediglich Zed Attack Proxy weicht hier mit Scanzeiten von 10 bis 14 Stunden pro Webanwendung ungewöhnlich weit von der Norm ab. Selbst die Werte von Netsparker, der auf dem vorletzten Platz landet, sind bis auf einen Ausreißer bei Webscantest (WA2) akzeptabel. Die Zeiten von OpenVAS, BurpSuite Pro und Nessus liegen alle nah beieinander und ordnen sich im Mittelfeld ein. Arachni sowie die beiden Terminalprogramme Nikto und Wapiti scannen überdurchschnittlich schnell, der beste WVS in dieser Kategorie ist mit großem Vorsprung Acunetix, der teilweise keine zehn Minuten für einen Scan benötigt.

## 5.3 Ranking

Die Ranking-Tabelle zeigt einen klaren Verlierer dieser Evaluation: Zed Attack Proxy ist der einzige WVS, der mit null Punkten bewertet werden musste, und das gleich zweimal, in den Kategorien Scanergebnis und Geschwindigkeit. Zed Attack Proxy ist damit als unbrauchbar zu bezeichnen, zumindest in der Funktion als WVS; die jeweils drei Punkte in den Kategorien Reporting und Bedienung sind wertlos, da kaum Schwachstellen gefunden werden. Nikto und Wapiti liegen mit jeweils 14 Punkten auf dem vorletzten Platz, mit den dürftigen Scanergebnissen und den Mängeln im Reporting sind diese Tools für professionelles Penetration Testing nur sehr bedingt tauglich. Nessus befindet sich mit 22 Punkten im unteren Mittelfeld und kann nur in der Kategorie Bedienung überzeugen, erfüllt aber insgesamt seinen Zweck. Netsparker hat gute Werte im Scanergebnis und

im Reporting und belegt mit 26 Punkten den 5. Platz. OpenVAS erreicht 28 Punkte und lässt damit den kommerziellen WVS Nessus, aus dem OpenVAS hervorgegangen ist, hinter sich. Auf den ersten drei Plätzen befinden sich mit Arachni, Acunetix und BurpSuite Pro diejenigen WVS, die durchweg überdurchschnittliche und überragende Werte erreicht haben und somit fast uneingeschränkt empfehlenswert sind. Arachni belegt mit 32 Punkten den 3. Platz und ist damit der beste Open Source WVS. Acunetix belegt mit 33 Punkten den 2. Platz, allerdings nur durch den Punktabzug aufgrund der Abstürze beim Testen unter Windows. Ohne Abstürze hätte Acunetix sich den ersten Platz mit BurpSuite Pro geteilt.

Testsieger und damit die Antwort auf die Frage **Welcher WVS schneidet insgesamt am besten ab?** ist BurpSuite Pro mit 36 Punkten. Ausschlaggebend für das Ergebnis sind in erster Linie das Scanergebnis und das Reporting, in beiden Kategorien hat BurpSuite Pro am besten von allen WVS abgeschnitten.

## 5.4 Open Source im Vergleich mit kommerziellen WVS

Nachfolgend wird die Frage **Wie schneiden die Open Source WVS im Vergleich mit kommerziellen WVS ab?** erörtert.

Von den fünf getesteten Open Source WVS finden sich im Ranking drei auf den letzten Plätzen wieder. Nikto und Wapiti haben als Terminalprogramme naturgemäß Nachteile in der Handhabung, sie schneiden aber auch beim Scanergebnis und im Reporting unterdurchschnittlich ab. Zed Attack Proxy auf dem letzten Platz ist zwar als WVS ungeeignet, wird in der Regel aber auch als “Intercepting Proxy” verwendet, der den Browsertraffic abfangen, überprüfen und verändern kann. Der Name deutet darauf hin, dass dies die Hauptfunktion von Zed Attack Proxy ist, vom schlechten Abschneiden als WVS sollte hier also nicht auf die übrigen Funktionen geschlossen werden.

Arachni und OpenVAS auf dem 3. und 4. Platz im Gesamtranking schneiden besser ab, als die kommerziellen WVS Netsparker und Nessus und stellen damit zwei kostengünstige Alternativen dar. Nach Bewältigung der Einarbeitungszeit lassen sich mit OpenVAS gute Ergebnisse erzielen, insbesondere wenn man einen hohen Automatisierungsgrad anstrebt. Arachni überzeugt mit leichter Handhabung, sehr gutem Reporting und überdurchschnittlicher Geschwindigkeit, bei der Anzahl der gefundenen Schwachstellen setzen sich jedoch die kommerziellen WVS BurpSuite Pro und Acunetix durch.

# 6 Fazit und Ausblick

Das Ziel dieser Arbeit war es, aus der großen Anzahl von WVS die für eine Evaluation geeigneten herauszufiltern und auf ihre Tauglichkeit und Qualität zu testen. Nach eingehender Prüfung der in Frage kommenden Kandidaten wurden schließlich neun in die Evaluation aufgenommen, fünf Open Source- und vier kommerzielle WVS. Zum Testen wurden sieben verwundbare Webanwendungen mit verschiedenen Technologien ausgewählt, die von den WVS gescannt und auf Schwachstellen überprüft wurden. Anhand eines Punktesystems wurden die WVS in den Kategorien Scanergebnis, Reporting, Bedienung und Geschwindigkeit verglichen.

Neben dem Aufzeigen der Unterschiede in den einzelnen Kategorien, sollten die Fragen beantwortet werden, welcher WVS das beste Resultat erzielt und wie die Open Source WVS im Vergleich mit den kommerziellen Scannern abschneiden.

Durch die Evaluation gelang es, für jede Fragestellung Antworten zu finden:

**Anzahl gefundener Schwachstellen:** Mit Hilfe einer Tabelle wurden die großen Unterschiede bei der Anzahl gefundener Schwachstellen sichtbar gemacht, hier schnitten BurpSuitePro und Acunetix mit über 500 Funden am besten ab.

**Bedienung, Reporting und Geschwindigkeit:** Die Scan-Geschwindigkeit wurde den Berichten entnommen oder manuell ermittelt. Die subjektiven Eindrücke für die Bedienung und das Reporting wurden für jeden WVS einzeln beschrieben und mit Hilfe des Punktesystems bewertet. Die Bewertung resultiert in einem Gesamtranking.

**Ranking:** Aus dem Gesamtranking lässt sich der WVS ermitteln, der am besten abschnitten hat, es ist BurpSuite Pro mit 36 Punkten.

**Open Source im Vergleich mit kommerziellen WVS:** Anhand des Rankings wurde auch das Abschneiden der Open Source WVS aufgezeigt, mit Arachni und OpenVAS an dritter und vierter Stelle der Rangliste gibt es vollwertige, kostenlose Alternativen zu den kommerziellen WVS.

Ungeachtet der Resultate soll hier jedoch durch Reflexion der gewählten Methodik auch auf die Grenzen der Evaluation eingegangen werden.

So hat die Gesamtanzahl der gefundenen Schwachstellen zum Beispiel nur eingeschränkte Aussagekraft; zum einen müsste jeder Fund noch durch manuelles Testen verifiziert werden, um False-Positives auszuschließen, zum anderen ist die durch die WVS vorgenommene Einteilung in die Kategorien “High”, “Medium”, “Low” und “Informational” problematisch, da die “Informational”-Funde, wie der Name schon sagt, nur der Information dienen und gar kein Eingreifen erfordern. Würde man diese Werte ausblenden, ergäbe sich im Ranking ein komplett anderes Bild, der Testsieger BurpSuite Pro käme zum Beispiel nur noch auf 107 gefundene Schwachstellen und würde damit in dieser Kategorie den vorletzten Rang belegen. In diesem Zusammenhang muss rückblickend auch die Gewichtung der einzelnen Kategorien hinterfragt werden, angesichts der zweifelhaften Aussagekraft des Scanergebnisses ist die Gewichtung von 50% vielleicht zu hoch angesetzt.

Die Ergebnisse dieser Thesis sind somit als Grundlage für weitergehende Evaluationen einzuordnen.

Für künftige Studien sind mehrere Ansätze möglich:

- Durch Verifizierung der False-Positives werden die Scan-Ergebnisse genauer, hier müsste allerdings aufgrund des hohen Aufwands die Anzahl der WVS und der Webanwendungen reduziert werden.
- Durch Weglassen der “Informational”-Funde werden nur relevante Schwachstellen erfasst, die auch tatsächlich ein Eingreifen erfordern.
- Bei Auswahl und Gewichtung der Bewertungskategorien können andere Prioritäten gesetzt werden, indem etwa die Scan-Geschwindigkeit vernachlässigt oder bei den kommerziellen WVS der Preis mit einbezogen wird.

Des Weiteren sei hier für tiefergehende Analysen auf das Projekt OWASP-Benchmark [50] verwiesen, eine von OWASP entwickelte, kostenlose Testsuite, mit der WVS in mehrtägigen Scans anhand von tausenden von Testfällen untersucht werden können.

# 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 26.03.2019).
- [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. *IT-Grundschutzkatalog*. BSI, 2016.
- [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] Web Application Security Consortium. *The WASC Threat Classification v2.0*. 2010. URL: <http://projects.webappsec.org/w/page/13246978/Threat%20Classification> (besucht am 16.03.2019).
- [13] 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).

- [14] Patrick Engebretson. *The Basics of Hacking and Penetration Testing*. Elsevier Inc., 2013.
- [15] 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).
- [16] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheit von Webanwendungen, Maßnahmenkatalog und Best Practices*. BSI, 2006.
- [17] Ron Lepofsky. *The Manager's Guide to Web Application Security*. Apress, Berkeley, CA, 2014.
- [18] Golismero Project. *Golismero*. 2019. URL: <http://www.golismero.com> (besucht am 19.02.2019).
- [19] Romain Gaucher. *Grabber*. 2006. URL: <http://rgaucher.info/beta/grabber/> (besucht am 19.02.2019).
- [20] David Byrne. *Grendel-Scan*. 2015. URL: <https://sourceforge.net/projects/grendel/> (besucht am 19.02.2019).
- [21] Lavakumar Kuppan. *IronWasp*. 2014. URL: <https://ironwasp.org> (besucht am 19.02.2019).
- [22] Google.com. *ratproxy*. 2009. URL: <https://code.google.com/archive/p/ratproxy/> (besucht am 19.02.2019).
- [23] Google.com. *skipfish*. 2012. URL: <https://code.google.com/archive/p/skipfish/> (besucht am 19.02.2019).
- [24] Bernardo Damele. *sqlmap*. 2019. URL: <http://sqlmap.org> (besucht am 19.02.2019).
- [25] Subgraph. *Vega*. 2014. URL: <https://subgraph.com/vega/> (besucht am 19.02.2019).
- [26] siberas. *Watobo*. 2015. URL: <https://sourceforge.net/projects/watobo/> (besucht am 19.02.2019).
- [27] 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).
- [28] sectools.org. *Wfuzz*. 2019. URL: <http://www.edge-security.com/wfuzz.php> (besucht am 19.02.2019).
- [29] w3af.sourceforge. *w3af*. 2013. URL: <http://w3af.org> (besucht am 19.02.2019).
- [30] sensepost. *Wikto*. 2015. URL: <https://github.com/sensepost/wikto> (besucht am 19.02.2019).
- [31] OWASP. *OWASP Xenotix XSS Exploit Framework*. 2019. URL: <https://xenotix.in> (besucht am 19.02.2019).
- [32] nstalker.com. *N-Stalker*. 2019. URL: <http://www.nstalker.com> (besucht am 20.02.2019).

- [33] 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).
- [34] IBM. *Altoro Mutual*. 2019. URL: <https://demo.testfire.net> (besucht am 21.02.2019).
- [35] NTOSpider. *Web Scanner Test Site*. 2019. URL: <https://www.webscantest.com> (besucht am 21.02.2019).
- [36] Micro Focus Development Company. *Zero Bank*. 2018. URL: <http://zero.webappsecurity.com> (besucht am 21.02.2019).
- [37] Netsparker. *Bitcoin Web Site*. 2018. URL: <http://aspnet.testsparker.com> (besucht am 21.02.2019).
- [38] Acunetix. *Acunetix Acuart*. 2018. URL: <http://testphp.vulnweb.com> (besucht am 21.02.2019).
- [39] Micro Focus Development Company. *Crack Me Bank*. 2018. URL: <http://crackme.cenzic.com/kelev/view/home.php> (besucht am 21.02.2019).
- [40] Acunetix. *Security Tweets*. 2013. URL: <http://testhtml5.vulnweb.com> (besucht am 21.02.2019).
- [41] arachni scanner.com. *arachni web application security scanner framework*. 2017. URL: <http://www.arachni-scanner.com> (besucht am 19.02.2019).
- [42] cirt.net. *Nikto2*. 2019. URL: <https://cirt.net/Nikto2> (besucht am 19.02.2019).
- [43] Greenbone. *OpenVAS - Open Vulnerability Assessment System*. 2019. URL: <http://www.openvas.org/index-de.html> (besucht am 19.02.2019).
- [44] Nicolas Surribas. *Wapiti - The web-application vulnerability scanner*. 2018. URL: <http://wapiti.sourceforge.net> (besucht am 20.02.2019).
- [45] OWASP. *The OWASP Zed Attack Proxy*. 2019. URL: <https://www.zaproxy.org> (besucht am 20.02.2019).
- [46] Acunetix. *acunetix*. 2019. URL: <https://www.acunetix.com> (besucht am 20.02.2019).
- [47] Portswigger. *Burp Suite Editions*. 2019. URL: <https://portswigger.net/burp> (besucht am 20.02.2019).
- [48] Tenable. *nessus Professional*. 2019. URL: <https://www.tenable.com/products/nessus/nessus-professional> (besucht am 20.02.2019).
- [49] Netsparker. *netsparker*. 2019. URL: <https://www.netsparker.com> (besucht am 20.02.2019).
- [50] OWASP. *Benchmakr*. 2018. URL: <https://www.owasp.org/index.php/Benchmark#tab>Main> (besucht am 25.03.2019).

# A Verwendete Abkürzungen

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

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

**CGI** - Common Gateway Interface

**CPU** - Central Processing Unit

**CVE** - Common Vulnerabilities and Exposures

**DAST** - Dynamic Application Security Testing

**DoS** - Denial of Service

**HTML** - Hypertext Markup Language

**HTTP** - Hypertext Transfer Protocol

**HTTPS** - Hypertext Transfer Protocol Secure

**JSP** - JavaServer Pages

**LDAP** - Lightweight Directory Access Protocol

**NTLM** - New Technology Lan Manager

**OS** - Operating System

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

**RAM** - Random-Access Memory

**REST** - Representational State Transfer

**SAST** - Static Application Security Testing

**SCA** - Static Code Analysis

**SPA** - Single Page Application

**SSL** - Secure Sockets Layer

**SQL** - Structured Query Language

**TLS** - Transport Layer Security

**URI** - Uniform Resource Identifier

**URL** - Uniform Resource Locator

**WAF** - Web Application Firewall

**WASC** - Web Application Security Consortium

**WVS** - Web Application Vulnerability Scanner

**WWW** - World Wide Web

**XML** - Extensible Markup Language

**XSS** - Cross Site Scripting

# B Screenshots

Generated by Acunetix

## Scan of testphp.vulnweb.com

### Scan details

Scan information	
Start time	14/02/2019, 16:50:27
Start url	http://testphp.vulnweb.com/
Host	testphp.vulnweb.com
Scan time	5 minutes, 35 seconds
Profile	Full Scan
Server information	nginx/1.4.1
Responsive	True
Server OS	Unknown
Server technologies	PHP

### Threat level

#### Acunetix Threat Level 3

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

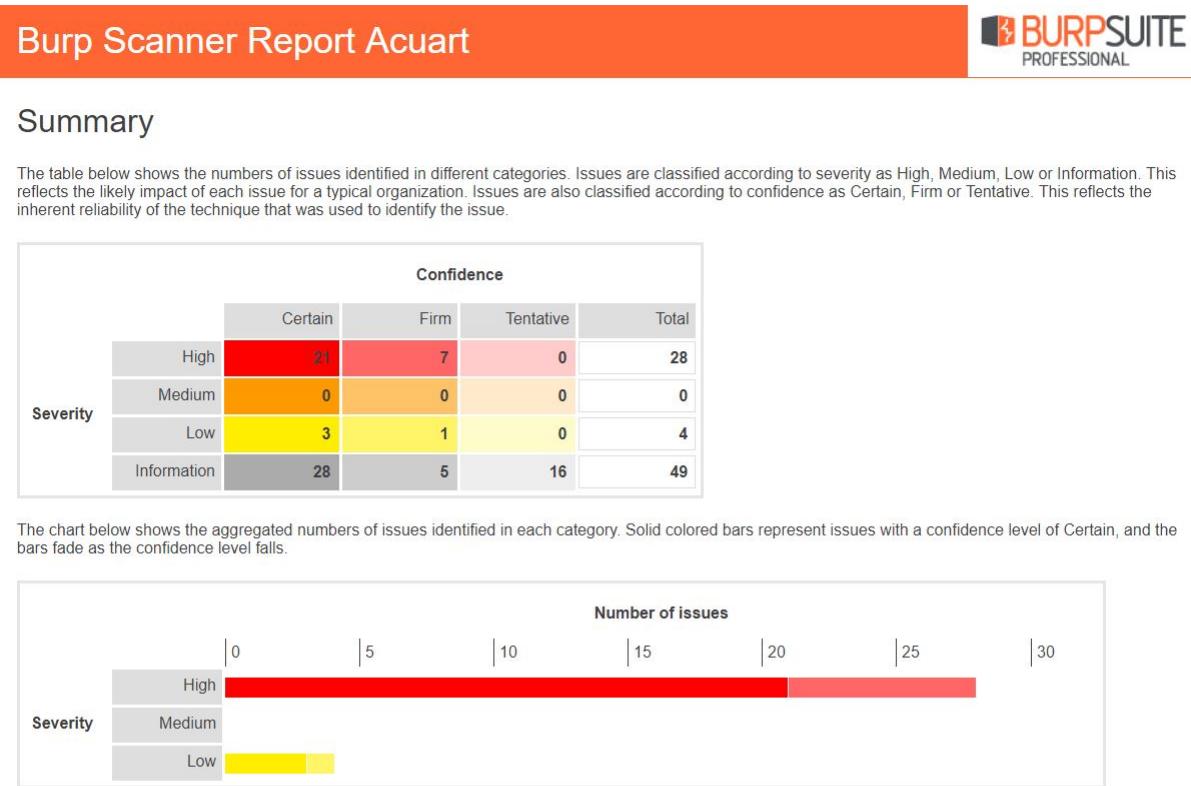
### Alerts distribution

Total alerts found	101
High	39
Medium	34
Low	9
Informational	19

### Affected items

/search.php	
Alert group	<b>Blind SQL Injection</b>
Severity	High
Description	SQL injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements that control a web application's database server.
Recommendations	Use parameterized queries when dealing with SQL queries that contains user input. Parameterized queries allows the database to understand which parts of the SQL query should be considered as user input, therefore solving SQL injection.

Abbildung B.1: Bericht Acunetix



## Contents

### 1. SQL injection

- 1.1. <http://testphp.vulnweb.com/artists.php> [artist parameter]
- 1.2. <http://testphp.vulnweb.com/listproducts.php> [artist parameter]
- 1.3. <http://testphp.vulnweb.com/listproducts.php> [cat parameter]
- 1.4. <http://testphp.vulnweb.com/product.php> [ip parameter]
- 1.5. <http://testphp.vulnweb.com/search.php> [test parameter]
- 1.6. <http://testphp.vulnweb.com/secured/newuser.php> [uname parameter]
- 1.7. <http://testphp.vulnweb.com/userinfo.php> [pass parameter]
- 1.8. <http://testphp.vulnweb.com/userinfo.php> [uname parameter]

### 2. Out-of-band resource load (HTTP)

### 3. Cross-site scripting (reflected)

- 3.1. <http://testphp.vulnweb.com/guestbook.php> [name parameter]

Abbildung B.2: Bericht BurpSuite Pro

## B Screenshots

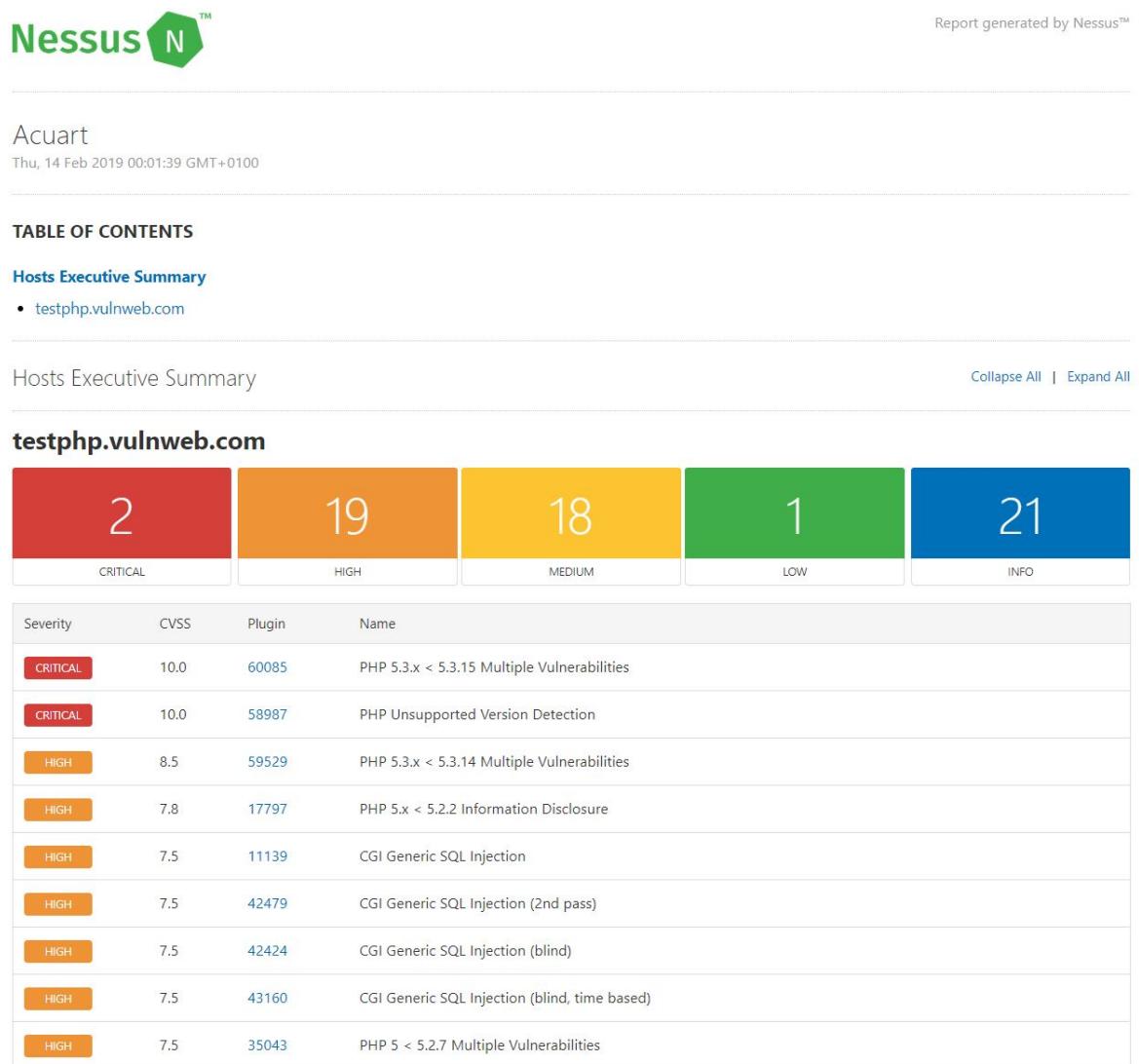


Abbildung B.3: Bericht Nessus

## B Screenshots

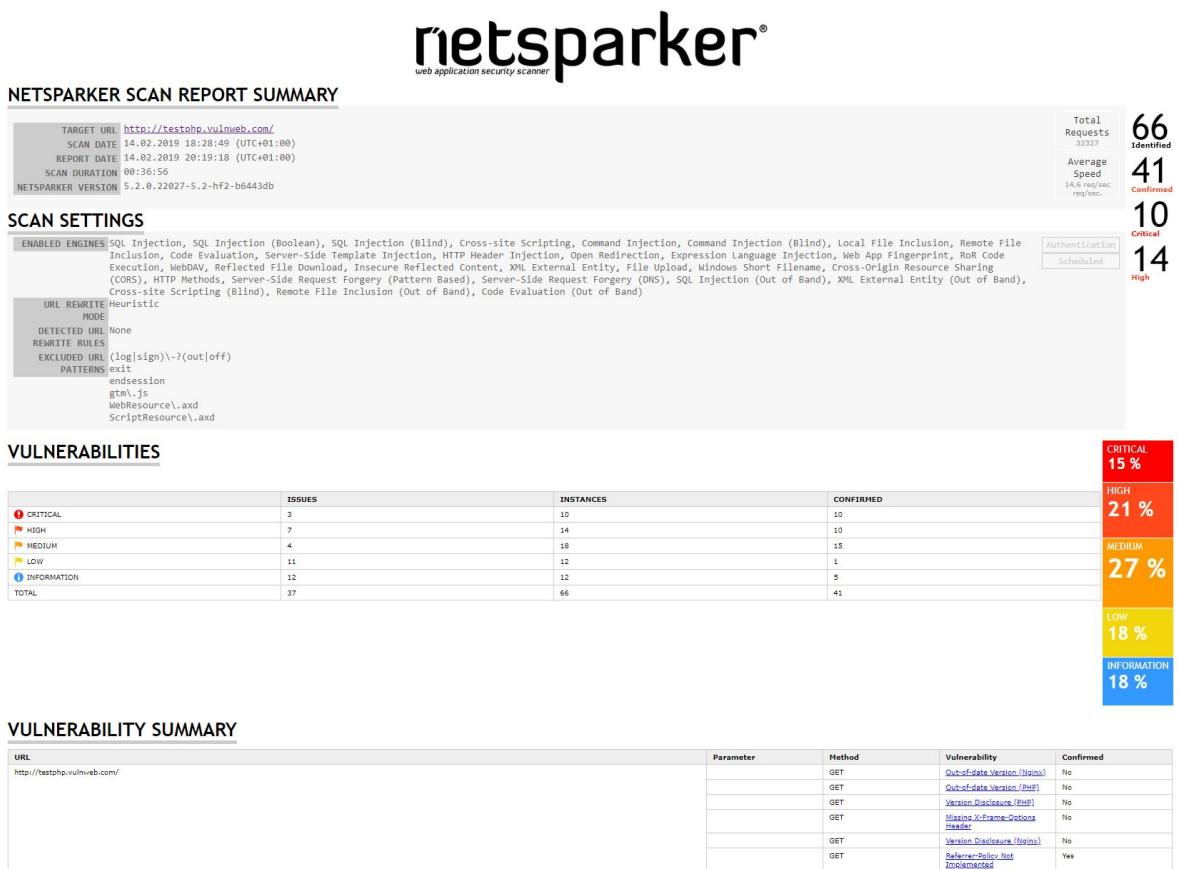


Abbildung B.4: Bericht Netsparker

## B Screenshots

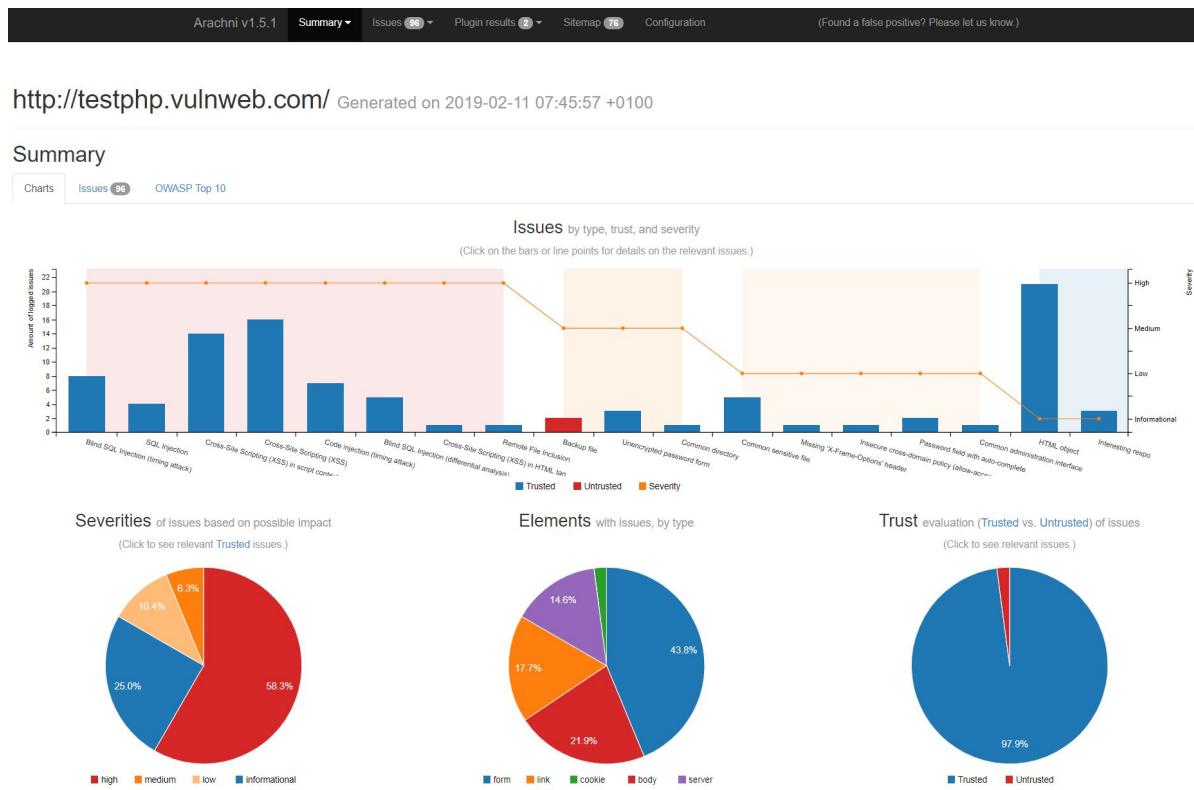


Abbildung B.5: Bericht Arachni

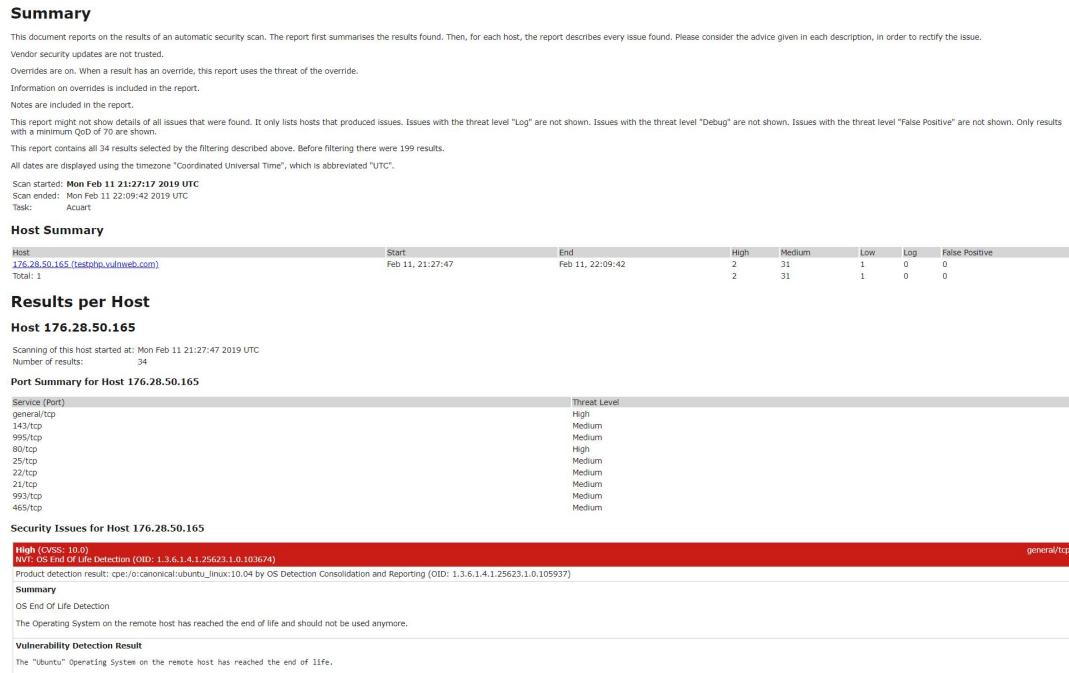


Abbildung B.6: Bericht OpenVAS

## B Screenshots

---

testhtml5.vulnweb.com / 176.28.50.165 port 80	
Target IP	176.28.50.165
Target hostname	testhtml5.vulnweb.com
Target Port	80
HTTP Server	nginx/1.4.1
Site Link (Name)	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a>
Site Link (IP)	<a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
URI	/
HTTP Method	GET
Description	The anti-clickjacking X-Frame-Options header is not present.
Test Links	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a> <a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/
HTTP Method	GET
Description	The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
Test Links	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a> <a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/
HTTP Method	GET
Description	The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
Test Links	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a> <a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/favicon.ico
HTTP Method	GET
Description	Server leaks inodes via ETags, header found with file /favicon.ico, fields: 0x51e79f63 0x37e
Test Links	<a href="http://testhtml5.vulnweb.com:80/favicon.ico">http://testhtml5.vulnweb.com:80/favicon.ico</a> <a href="http://176.28.50.165:80/favicon.ico">http://176.28.50.165:80/favicon.ico</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/
HTTP Method	GET
Description	Retrieved x-powered-by header: PHP/5.3.10-1~lucid+2uwsgi2
Test Links	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a> <a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/
HTTP Method	OPTIONS
Description	Allowed HTTP Methods: HEAD, OPTIONS, GET
Test Links	<a href="http://testhtml5.vulnweb.com:80/">http://testhtml5.vulnweb.com:80/</a> <a href="http://176.28.50.165:80/">http://176.28.50.165:80/</a>
OSVDB Entries	<a href="#">OSVDB-0</a>
URI	/samples/
HTTP Method	GET
Description	/samples/: This might be interesting...
Test Links	<a href="http://testhtml5.vulnweb.com:80/samples/">http://testhtml5.vulnweb.com:80/samples/</a> <a href="http://176.28.50.165:80/samples/">http://176.28.50.165:80/samples/</a>
OSVDB Entries	<a href="#">OSVDB-3092</a>
Host Summary	
Start Time	2019-02-14 17:15:35
End Time	2019-02-14 17:34:20
Elapsed Time	1125 seconds
Statistics	8354 requests, 20 errors, 7 findings
Scan Summary	
Software Details	<a href="#">Nikto 2.1.6</a>
CLI Options	-h <a href="http://testhtml5.vulnweb.com/">http://testhtml5.vulnweb.com/</a> -o /root/SecurityTweets_nikto.html -Format HTM
Hosts Tested	1
Start Time	Thu Feb 14 17:15:35 2019
End Time	Thu Feb 14 17:34:20 2019
Elapsed Time	1125 seconds

Abbildung B.7: Bericht Nikto

## B Screenshots

---

**Wapiti vulnerability report**

Target: <http://testphp.vulnweb.com/>

Date of the scan: Wed, 13 Feb 2019 21:59:38 +0000. Scope of the scan: folder

---

**Summary**

Category	Number of vulnerabilities found
<a href="#">SQL Injection</a>	8
<a href="#">Blind SQL Injection</a>	7
<a href="#">File Handling</a>	2
<a href="#">Cross Site Scripting</a>	13
CRLF Injection	0
Commands execution	0
Htaccess Bypass	0
Backup file	0
Potentially dangerous file	0
<a href="#">Server Side Request Forgery</a>	1
Internal Server Error	0
Resource consumption	0

---

### SQL Injection

#### Description

SQL injection vulnerabilities allow an attacker to alter the queries executed on the backend database. An attacker may then be able to extract or modify informations stored in the database or even escalate his privileges on the system.

#### Vulnerability found in /artists.php

- [Description](#)
- [HTTP Request](#)
- [cURL command line](#)

MySQL Injection via injection in the parameter artist

Abbildung B.8: Bericht Wapiti

## ZAP Scanning Report

### Summary of Alerts

Risk Level	Number of Alerts
<a href="#">High</a>	4
<a href="#">Medium</a>	1
<a href="#">Low</a>	2
<a href="#">Informational</a>	0

### Alert Detail

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	<a href="http://testphp.vulnweb.com/userinfo.php">http://testphp.vulnweb.com/userinfo.php</a>
Method	POST
Parameter	uname
Attack	ZAP' OR '1='1' --
URL	<a href="http://testphp.vulnweb.com/listproducts.php?cat=4+AND+1%3D1+--+">http://testphp.vulnweb.com/listproducts.php?cat=4+AND+1%3D1+--+</a>
Method	GET
Parameter	cat
Attack	4 OR 1=1 --
URL	<a href="http://testphp.vulnweb.com/secured/newuser.php">http://testphp.vulnweb.com/secured/newuser.php</a>
Method	POST
Parameter	username
Attack	ZAP' OR '1='1' --
URL	<a href="http://testphp.vulnweb.com/userinfo.php">http://testphp.vulnweb.com/userinfo.php</a>
Method	POST
Parameter	pass
Attack	ZAP' OR '1='1' --
URL	<a href="http://testphp.vulnweb.com/artists.php?artist=5-2">http://testphp.vulnweb.com/artists.php?artist=5-2</a>
Method	GET

Abbildung B.9: Bericht ZAP

# **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.

Löwenstein, den 04.04.2019

---

Henning Janning