



# **A Walk on the Web's Wild Side**

**STUDIENARBEIT**

für die Prüfung zum

Bachelor of Science

des Studiengangs Informatik  
Studienrichtung Angewandte Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Samuel Philipp  
Daniel Brown  
Jan-Eric Gaidusch**

19. Mai 2017

**Bearbeitungszeitraum**

6 Monate

**Matrikelnummern**

9207236, 3788021, 8296876

**Kurs**

TINF14B2

**Ausbildungsfirma**

Fiducia & GAD IT AG

**Gutachter der Studienakademie**

Dr. Martin Johns

## Abstract

Today every fifth person is a victim of cyber crime. This is, because most websites on the web are not secure and have vulnerabilities which are exploited by hackers. The standard internet user cannot see, whether a website is attacking his browser. This is the reason this paper and the related project were brought into being. We present *webifier*, an open source web application, which checks whether a website is malicious or not. It guarantees security for the user, as it runs on a webserver, and it guarantees security for the maintaining web administrators, because every test is encapsulated in an own Docker container. It identifies security issues and client side attacks ranging from simple SSL certificate checking over advanced network analysis to sophisticated virus scanning. At the time of writing there is a total of eight different tests, whereby this number can be increased by developing more tests and integrating them into the application. It is not as mature as to call it a high-class security validator, but it provides a secure, easy expandable, feature rich and user friendly alternative to existing online security scanners.

# Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29.9.2015)

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:

**„A Walk on the Web’s Wild Side“**

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 19. Mai 2017

Ort, Datum

Samuel Philipp

Karlsruhe, den 19. Mai 2017

Ort, Datum

Daniel Brown

Karlsruhe, den 19. Mai 2017

Ort, Datum

Jan-Eric Gaidusch

# Inhaltsverzeichnis

|   |             |
|---|-------------|
| <b>Abkürzungsverzeichnis</b>                        | <b>V</b>    |
| <b>Abbildungsverzeichnis</b>                        | <b>VI</b>   |
| <b>Tabellenverzeichnis</b>                          | <b>VIII</b> |
| <b>Listings</b>                                     | <b>IX</b>   |
| <b>1 Einleitung</b>                                 | <b>1</b>    |
| 1.1 Aufbau der Arbeit . . . . .                     | 1           |
| 1.2 Aufgabenstellung . . . . .                      | 2           |
| 1.3 Team . . . . .                                  | 2           |
| 1.4 webifier . . . . .                              | 3           |
| <b>2 Grundlagen</b>                                 | <b>5</b>    |
| 2.1 Frontend Technologien und Frameworks . . . . .  | 5           |
| 2.2 Backend Technologien und Frameworks . . . . .   | 9           |
| 2.3 Technologien und Frameworks der Tests . . . . . | 11          |
| 2.4 Angriffstypen . . . . .                         | 13          |
| 2.4.1 Malware . . . . .                             | 15          |
| 2.4.2 User Agent Sniffing . . . . .                 | 18          |
| 2.4.3 JavaScript Port & IP Scanning . . . . .       | 19          |
| 2.4.4 Phishing . . . . .                            | 24          |
| <b>3 Konzept</b>                                    | <b>28</b>   |
| 3.1 Gesamtkonzept . . . . .                         | 28          |
| 3.1.1 webifier Tests . . . . .                      | 32          |
| 3.1.2 webifier Tester . . . . .                     | 33          |
| 3.1.3 webifier Plattform . . . . .                  | 35          |
| 3.1.4 webifier Mail . . . . .                       | 36          |

---

|          |   |           |
|----------|---|-----------|
| 3.1.5    | webifier Data . . . . .                       | 37        |
| 3.1.6    | webifier Statistics . . . . .                 | 37        |
| 3.2      | Testarten . . . . .                           | 38        |
| 3.2.1    | Virensan der Webseite . . . . .               | 38        |
| 3.2.2    | Vergleich in verschiedenen Browsern . . . . . | 39        |
| 3.2.3    | Überprüfung der Port-Nutzung . . . . .        | 39        |
| 3.2.4    | Überprüfung der IP-Nutzung . . . . .          | 40        |
| 3.2.5    | Prüfung aller verlinkten Seiten . . . . .     | 40        |
| 3.2.6    | Google Safe Browsing . . . . .                | 40        |
| 3.2.7    | Überprüfung des SSL-Zertifikats . . . . .     | 41        |
| 3.2.8    | Erkennung von Phishing . . . . .              | 42        |
| 3.2.9    | Screenshot der Seite . . . . .                | 43        |
| <b>4</b> | <b>Umsetzung</b>                              | <b>44</b> |
| 4.1      | Gesamtanwendung . . . . .                     | 44        |
| 4.1.1    | webifier Tests . . . . .                      | 48        |
| 4.1.2    | webifier Tester . . . . .                     | 48        |
| 4.1.3    | webifier Plattform . . . . .                  | 51        |
| 4.1.4    | webifier Mail . . . . .                       | 60        |
| 4.1.5    | webifier Data . . . . .                       | 61        |
| 4.1.6    | webifier Statistics . . . . .                 | 65        |
| 4.2      | Tests . . . . .                               | 67        |
| 4.2.1    | Virensan der Webseite . . . . .               | 67        |
| 4.2.2    | Vergleich in verschiedenen Browsern . . . . . | 68        |
| 4.2.3    | Überprüfung der Port-Nutzung . . . . .        | 70        |
| 4.2.4    | Überprüfung der IP-Nutzung . . . . .          | 71        |
| 4.2.5    | Prüfung aller verlinkten Seiten . . . . .     | 71        |
| 4.2.6    | Google Safe Browsing . . . . .                | 72        |
| 4.2.7    | Überprüfung des SSL-Zertifikats . . . . .     | 73        |
| 4.2.8    | Erkennung von Phishing . . . . .              | 73        |
| 4.2.9    | Screenshot der Seite . . . . .                | 76        |
| <b>5</b> | <b>Analyse</b>                                | <b>77</b> |
| 5.1      | Gesamtauswertungen . . . . .                  | 79        |
| 5.2      | Einzelauswertungen . . . . .                  | 85        |
| 5.2.1    | Virensan der Webseite . . . . .               | 86        |
| 5.2.2    | Vergleich in verschiedenen Browsern . . . . . | 87        |

---

|          |   |             |
|----------|---|-------------|
| 5.2.3    | Überprüfung der Port-Nutzung . . . . .    | 88          |
| 5.2.4    | Überprüfung der IP-Nutzung . . . . .      | 89          |
| 5.2.5    | Prüfung aller verlinkten Seiten . . . . . | 89          |
| 5.2.6    | Google Safe Browsing . . . . .            | 90          |
| 5.2.7    | Überprüfung des SSL-Zertifikats . . . . . | 91          |
| 5.2.8    | Erkennung von Phishing . . . . .          | 91          |
| 5.3      | Bewertung der Ergebnisse . . . . .        | 92          |
| <b>6</b> | <b>Ausblick</b>                           | <b>94</b>   |
| <b>7</b> | <b>Fazit</b>                              | <b>96</b>   |
|          | <b>Literaturverzeichnis</b>               | <b>XII</b>  |
|          | <b>Anhang</b>                             | <b>XVII</b> |

## Abkürzungsverzeichnis

|             |                                    |
|-------------|------------------------------------|
| <b>WWW</b>  | World Wide Web                     |
| <b>HTML</b> | Hypertext Markup Language          |
| <b>CSS</b>  | Cascading Style Sheets             |
| <b>UI</b>   | User Interface                     |
| <b>JVM</b>  | Java Virtual Machine               |
| <b>API</b>  | Application Programming Interface  |
| <b>DRY</b>  | Don't Repeat Yourself              |
| <b>REST</b> | Representational State Transfer    |
| <b>URI</b>  | Uniform Ressource Identifier       |
| <b>URL</b>  | Uniform Ressource Locator          |
| <b>NIDS</b> | Network Intrusion Detection System |
| <b>CLI</b>  | Command Line Interface             |
| <b>JSON</b> | JavaScript Object Notation         |
| <b>XML</b>  | Extensible Markup Language         |
| <b>TLD</b>  | Top-Level Domain                   |
| <b>SSH</b>  | Secure Shell                       |
| <b>HTTP</b> | Hypertext Transfer Protocol        |
| <b>SMTP</b> | Simple Mail Transfer Protocol      |
| <b>IMAP</b> | Internet Mail Access Protocol      |

## Abbildungsverzeichnis

|    |  |    |
|----|--|----|
| 1  | Secutitysquad - Logo . . . . .   | 3  |
| 2  | webifier - Logo . . . . .  | 3  |
| 3  | Bedeutung der einzelnen Kategorien von Internetkriminalität . . . . .      | 14 |
| 4  | Malware Verbreitung in Deutschland . . . . .                               | 17 |
| 5  | Verbreitung neuer Malware in Deutschland . . . . .                         | 17 |
| 6  | 3-Step-Handshake TCP . . . . .   | 21 |
| 7  | PayPal Phishing Webseite . . . . .   | 26 |
| 8  | PayPal Original Webseite . . . . .   | 27 |
| 9  | Konzeptarchitektur der Gesamtanwendung . . . . .                           | 29 |
| 10 | Icon für den Testergebnistyp <i>CLEAN</i> . . . . .                        | 30 |
| 11 | Icon für den Testergebnistyp <i>SUSICIOUS</i> . . . . .                    | 30 |
| 12 | Icon für den Testergebnistyp <i>MALICIOUS</i> . . . . .                    | 31 |
| 13 | Icon für den Testergebnistyp <i>UNDEFINED</i> . . . . .                    | 31 |
| 14 | Screenshot einer Spam-Mail . . . . .                                       | 36 |
| 15 | Umgesetzte Architektur der Gesamtanwendung . . . . .                       | 47 |
| 16 | <i>webifier Plattform</i> - Startseite . . . . .                           | 53 |
| 17 | <i>webifier Plattform</i> - Ergebnisseite . . . . .                        | 54 |
| 18 | <i>webifier Plattform</i> - Virensan der Webseite . . . . .                | 54 |
| 19 | <i>webifier Plattform</i> - Vergleich in verschiedenen Browsern . . . . .  | 55 |
| 20 | <i>webifier Plattform</i> - Überprüfung der IP- und Port-Nutzung . . . . . | 56 |
| 21 | <i>webifier Plattform</i> - Prüfung aller verlinkten Seiten . . . . .      | 56 |
| 22 | <i>webifier Plattform</i> - Google Safe Browsing . . . . .                 | 57 |
| 23 | <i>webifier Plattform</i> - Überprüfung des SSL-Zertifikats . . . . .      | 57 |
| 24 | <i>webifier Plattform</i> - Erkennung von Phishing . . . . .               | 58 |
| 25 | <i>webifier Plattform</i> - Screenshot der Seite . . . . .                 | 59 |
| 26 | <i>webifier Mail</i> - Screenshot der Startmail . . . . .                  | 60 |
| 27 | <i>webifier Mail</i> - Screenshot der Ergebnismail . . . . .               | 61 |
| 28 | Generierte Valuebox . . . . .  | 66 |
| 29 | Tabelle der HTTP-Header auf reliply.org . . . . .                          | 69 |



|    |   |         |
|----|---|---------|
| 30 | Webifier Statistics Dashboard . . . . .                                       | 78      |
| 31 | Erkennungen anhand Top-Level-Domains . . . . .                                | 80      |
| 32 | Verteilung der getesteten Top-Level-Domains . . . . .                         | 80      |
| 33 | prozentuale Erkennungen anhand Top-Level-Domains . . . . .                    | 81      |
| 34 | Bedrohliche Funde visualisiert anhand einer Weltkarte . . . . .               | 82      |
| 35 | Testergebnisverteilung . . . . .  | 83      |
| 36 | Visualisierung der Testzusammenhänge . . . . .                                | 84      |
| 37 | Top 10: Die bedrohlichsten Webseiten . . . . .                                | 84      |
| 38 | Einzelauswertung: Vergleich in verschiedenen Browsern . . . . .               | 85      |
| 39 | Einzelauswertung: Virensan der Webseite . . . . .                             | 86      |
| 40 | Vergleich in verschiedenen Browsern - Testergebnisverteilung . . . . .        | 87      |
| 41 | Einzelauswertung: Überprüfung der Port-Nutzung . . . . .                      | 88      |
| 42 | Überprüfung der IP-Nutzung - Testergebnisverteilung . . . . .                 | 89      |
| 43 | Prüfung aller verlinkten Seiten - Testergebnisverteilung . . . . .            | 89      |
| 44 | Google Safe Browsing - Testergebnisverteilung . . . . .                       | 90      |
| 45 | Überprüfung des SSL-Zertifikats - Testergebnisverteilung . . . . .            | 91      |
| 46 | Erkennung von Phishing - Testergebnisverteilung . . . . .                     | 91      |
| 47 | <i>webifier Plattform</i> - Startseite . . . . .                              | XXV     |
| 48 | <i>webifier Plattform</i> - Ergebnisseite für reflexonature.free.fr . . . . . | XXVI    |
| 49 | <i>webifier Plattform</i> - Ergebnisseite für securitysquad.de . . . . .      | XXVII   |
| 50 | <i>webifier Plattform</i> - Ergebnisseite für google.com . . . . .            | XXVIII  |
| 51 | <i>webifier Statistics Dashboard</i> . . . . .                                | XXXVI   |
| 52 | Verteilung der getesteten Top-Level-Domains . . . . .                         | XXXVII  |
| 53 | prozentuale Erkennungen anhand Top-Level-Domains . . . . .                    | XXXVIII |
| 54 | Bedrohliche Funde visualisiert anhand einer Weltkarte . . . . .               | XXXIX   |
| 55 | Testergebnisverteilung . . . . .  | XL      |
| 56 | Visualisierung der Testzusammenhänge . . . . .                                | XLI     |
| 57 | Top 10: Die bedrohlichsten Webseiten . . . . .                                | XLII    |
| 58 | Einzelauswertung: <i>Vergleich in verschiedenen Browsern</i> . . . . .        | XLIII   |
| 59 | Einzelauswertung: <i>Virensan der Webseite</i> . . . . .                      | XLIV    |
| 60 | Einzelauswertung: <i>Überprüfung der Port-Nutzung</i> . . . . .               | XLV     |

## Tabellenverzeichnis

|   |  |    |
|---|--|----|
| 1 | Beschreibung der einzelnen Tests . . . . .       | 33 |
| 2 | Gewichtungen der einzelnen Tests . . . . .       | 34 |
| 3 | Zuordnung Testergebnis zu Ergebniswert . . . . . | 35 |

## Listings

|    |  |      |
|----|--|------|
| 1  | Beispiel.html . . . . .  | 6    |
| 2  | Beispiel.css . . . . .   | 7    |
| 3  | Beispiel eines simplen Java PortScanners . . . . .   | 20   |
| 4  | Beispiel eines simplen JavaScript PortScanners . . . . .   | 23   |
| 5  | Phishing Lockmail . . . . .  | 24   |
| 6  | Result JSON . . . . .  | 48   |
| 7  | <i>webifier Tester</i> - Hilfe . . . . .   | 48   |
| 8  | <i>webifier Tester</i> - Standardausgabe . . . . .   | 49   |
| 9  | <i>webifier Tester</i> - Ausschnitt Konfigurationsdatei . . . . .  | 50   |
| 10 | <i>webifier Tester</i> - Ausschnitt Ergebnisberechnung . . . . .   | 51   |
| 11 | <i>webifier Plattform</i> - Konfigurationsdatei . . . . .  | 52   |
| 12 | <i>webifier Data</i> - Ausschnitt Inhalt push-Request . . . . .  | 62   |
| 13 | <i>webifier Data</i> - Inhalt check-Request . . . . .  | 62   |
| 14 | <i>webifier Data</i> - Verarbeitung check Methode . . . . .  | 63   |
| 15 | <i>webifier Data</i> - Inhalt check-Response . . . . .   | 64   |
| 16 | <i>webifier Data</i> - Beispieldokument <i>webifierTestResultData</i> . . . . .  | 65   |
| 17 | <i>webifier Statistics</i> - Beispiel R-Grafik . . . . .   | 65   |
| 18 | Beispiel für Browserkonfiguration und gespeicherte Header . . . . .  | 68   |
| 19 | Ergebnisberechnung der Erkennung von Phishing . . . . .  | 75   |
| 20 | <i>webifier Tester</i> - Vollständige Konfigurationsdatei . . . . .  | XXI  |
| 21 | <i>webifier Tester</i> - Vollständige Ergebnisberechnung . . . . .   | XXIV |
| 22 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : <i>Virens-</i><br><i>can der Webseite</i> . . . . .            | XXIX |
| 23 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : <i>Vergleich</i><br><i>in verschiedenen Browsern</i> . . . . . | XXX  |
| 24 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : <i>Überprü-</i><br><i>fung der Port-Nutzung</i> . . . . .      | XXXI |
| 25 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : <i>Überprü-</i><br><i>fung der IP-Nutzung</i> . . . . .        | XXXI |

---

|    |  |        |
|----|--|--------|
| 26 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : Prüfung<br>aller verlinkten Seiten . . . . .   | XXXII  |
| 27 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : Google<br>Safe Browsing . . . . .              | XXXIII |
| 28 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : Überprü-<br>fung des SSL-Zertifikats . . . . . | XXXIV  |
| 29 | <i>webifier Data</i> - Beispieldokument <i>webifierSingleTestResultData</i> : Erken-<br>nung von Phishing . . . . .          | XXXV   |

# 1 Einleitung

Anbieter von zwielichtigen Web-Angeboten greifen ihre User mit diversen Client-seitigen Methoden an. Beispiele für solche Angriffe sind Malware Downloads, Phishing, JavaScript Intranet Angriffe, oder Browser Exploits.

## „Jeder Fünfte Opfer von Internetkriminalität“<sup>1</sup>

Dieses Zitat macht deutlich welche Bedrohung vom Internet und insbesondere von Webseiten ausgeht. Normale Nutzer sind heutzutage im World Wide Web ein gefragtes Angriffsziel für webbasierte Angriffe. Häufig wird hierfür der Nutzer auf maliziose Webseiten gelockt. Diese Webseiten nutzen dann unter anderem Sicherheitslücken im Browser des Nutzers um Schadsoftware zu verbreiten oder den Anwender auszuspähen. Die nachfolgende Studienarbeit beschäftigt sich mit diesen Webseiten und analysiert deren Bedrohungspotenzial.

## 1.1 Aufbau der Arbeit

Die Studienarbeit ist in fünf Kapitel unterteilt. Das erste Kapitel ist die *Einleitung*. Hier werden die Rahmenbedingungen für die Arbeit erläutert und es wird ein Einblick in die Hintergründe gegeben. Das nächste Kapitel *Grundlagen* behandelt Tools, die maßgeblich zur Entwicklung der Lösung verwendet werden. Weiterhin werden clientseitige Probleme bzw. Angriffspunkte fachlich aufbereitet, die in der Lösung angesprochen werden. Im dritten Kapitel *Konzeption* werden die Ergebnisse unseres Entwurfsprozesses dargestellt und begründet. Dabei wird grundsätzlich zwischen der Gesamtanwendung und den Tests unterschieden. Diese Aufteilung findet sich auch im darauffolgenden Kapitel der *Umsetzung* wieder. Dort wird hingegen auf die konkrete Implementierung der Entwürfe eingegangen und bedeutende Anwendungslogik anhand von

<sup>1</sup> Zitat Rieckmann/ Kraus (2015), S. 297

Codebeispielen erklärt. In Kapitel fünf *Analyse* werden die Erkenntnisse der Arbeit präsentiert und danach differenziert beurteilt. In den letzten beiden Kapiteln *Ausblick* und *Fazit* werden Ideen und Verbesserungsvorschläge für mögliche Folgeprojekte vortragen und die Arbeit abschließen retrospektiv betrachtet.

## 1.2 Aufgabenstellung

Ziel der Arbeit ist eine systematische Untersuchung der Aktivitäten von semi-legalen Webseiten im World Wide Web (WWW). Das erwartete Ergebnis ist ein Prüfportal, auf dem jene Webseiten automatisiert analysiert werden und Ergebnisse präsentiert werden sollen.

Nach dem ersten Schaffen einer Übersicht von interessanten Zielen, wie z.B. One-Click-Hoster oder File-sharing Sites sollen ausgewählte Webseiten manuell untersucht werden. Außerdem sollen verschiedene Angriffsszenarien zur weiteren Prüfung ausgewählt werden. Der Untersuchungsprozess der Webseiten soll im Verlauf dieser Arbeit stückweise automatisiert und in den Rahmen einer Prüfanwendung gebracht werden.

Abschließend sollen eine Vielzahl von Webseiten mit der Anwendung getestet und die Ergebnisse ausgewertet und dokumentiert werden.

## 1.3 Team

Das Entwicklerteam besteht aus drei Studenten der angewandten Informatik: Samuel Philipp, Daniel Brown und Jan-Eric Gaidusch. Der Name der Arbeitsgruppe ist *SecuritySquad*.<sup>2</sup>

---

<sup>2</sup> Der Name *SecuritySquad* ist angelehnt an den Titel des US-amerikanischen Actionfilms *Suicide Squad*.



Abbildung 1: Secutitysquad - Logo

Die Studienarbeit wird von Dr. Martin Johns betreut, der an der DHBW Karlsruhe die Vorlesung Datensicherheit hält. Hauptberuflich ist er Forscher eben dieses Gebietes am CEC Karlsruhe der SAP SE.<sup>3</sup>

## 1.4 webifier



Abbildung 2: webifier - Logo

webifier ist eine Anwendung mit der Webseiten auf deren Seriosität und mögliche clientseitige Angriffe auf den Nutzer geprüft werden können. Sie besteht aus mehreren eigenständigen Teilanwendungen. Im Zentrum steht der Tester, welcher die einzelnen Tests verwaltet, ausführt und anschließend die Ergebnisse auswertet. Jeder einzelne Test ist eine weitere isolierte Teilanwendung des Testers. So kann jeder Test unabhängig von allen anderen betrieben werden.

<sup>3</sup> Vgl. Johns (2017)

Die Plattform ist eine Webanwendung welche den Endnutzern eine grafische Oberfläche zur Verfügung stellt, um Webseiten zu überprüfen. Im Hintergrund setzt die Plattform auf den Tester auf. webifier Mail ist ein Dienst mit dem Links aus E-Mails überprüft werden können. Anschließend erhält der Sender eine E-Mail mit den Resultaten zurück.

Eine weitere Teilanwendung von webifier ist das Data-Modul. Es stellt eine Schnittstelle für den Tester bereit, um alle Testergebnisse sammeln zu können. Das Statistik-Modul ist die letzte Teilanwendung von webifier. Es setzt auf das Data-Modul auf und stellt Funktionen zur Auswertung aller Testergebnisse bereit.

Um die Techniken und Algorithmen von webifier verstehen zu können sind einige Grundlagen erforderlich, welche nun im nächsten Kapitel genauer vorgestelt werden.



## 2 Grundlagen

In diesem Kapitel werden die Grundlagen, welche für das weitere Verständnis der Arbeit und der gesamten Anwendung notwendig sind, näher beschrieben. Zunächst werden die verschiedenen Technologien und Frameworks, sowohl des Frontends, als auch des Backends dargestellt. Anschließend werden einige gängige Angriffstypen im WWW erläutert.

### 2.1 Frontend Technologien und Frameworks

Dieser Abschnitt behandelt diejenigen Technologien, die die grafische Darstellung von Webinhalten und Interaktionen der Benutzers in seinem Browser ermöglichen. Da es sich bei webifier um eine Webanwendung handelt, sind dies ausschließlich Webtechnologien, welche von grafischen Browsern unterstützt werden.

Die grundlegende Informationssprache des WWW heißt Hypertext Markup Language (HTML). Sie wurde ursprünglich entwickelt um wissenschaftliche Dokumente semantisch zu beschreiben (engl. 'to mark up'). Heute wird sie jedoch in weitaus größerem Umfang genutzt.<sup>4</sup> HTML-Dateien bestehen aus zwei Arten von Informationen: Textdaten und Markupinformationen. Erstere sind verantwortlich für den textuellen Inhalt der Webseite. Dazu zählen alle abgebildeten Texte wie sie auch in Überschriften, Abschnitten, Menüs, usw. stehen. Sie sind die Informationen, die Betrachter der Webseite direkt über das grafische Browserfenster lesen kann. Markupinformationen hingegen definieren den Aufbau und die Semantik der Inhalte. Diese sind für den normalen Betrachter nicht unbedingt sichtbar. Hierbei handelt es sich um sogenannte *Tags*, die im Quellcode in spitzen Klammern stehen und aus einer Menge von bestimmten Werten stammen. Tags treten immer in Paaren auf, wobei der zweite Tag (Endtag) zusätzlich einen Backslash zwischen aufgehender spitzer Klammer und Tagnamen hat (s. Listing 1 Zeile 5). Innerhalb dieser beiden Tags können wiederum neue Tags, aber

<sup>4</sup> Vgl. World Wide Web Consortium (W3C) (2014)

auch einfache Textdaten stehen (s. Zeile 4). Diese Verschachtelung führt dazu, dass meist ein komplexer Baum von Tagelernen entsteht. Zu den wichtigsten Tags zählt der `<html>`-Tag. Er ist der äußerste Wurzeltag, der alle anderen Tags umschließt. Der `<head>`- und `<body>`-Tag stehen beide eine Ebene tiefer und beinhalten Metadaten für das gesamte Dokument bzw. den Seiteninhalt.<sup>5</sup>

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Testseite</title>
5 </head>
6 <body>
7     <h1>Überschrift</h1>
8     <p>Abschnitt 1</p>
9     <p>Abschnitt 2</p>
10 </body>
11 </html>
```

Listing 1: Beispiel.html

Um den in HTML definierten Inhalt optisch ansprechend zu gestalten verwenden Webseiten Cascading Style Sheets (CSS). CSS ist eine deklarative Sprache, mit der sich alle Optik-relevanten Eigenschaften von HTML-Elementen definieren lassen, die dann vom Browser entsprechend angezeigt werden. Die einfachste Möglichkeit ist es, die gewünschten Änderungen direkt über das `style`-Attribut des jeweiligen Elements anzugeben. Dazu wird die zu ändernde Eigenschaft angegeben, gefolgt von einem Doppelpunkt und dem neuen Eigenschaftswert. Alle Anweisungen sind durch ein Semikolon zu trennen. Um die gleichen Änderungen auf mehrere Elemente anzuwenden, sollten diese Anweisungen in einem globalen Stylesheet zu einem Regelset gebündelt werden. Stylesheets können direkt im HTML in ein Style-Element als Textwert angegeben (s. Listing 2 Zeilen 5 - 9), oder über ein Link-Element zu einer CSS-Datei nachgeladen werden. In beiden Fällen ist es notwendig anzugeben, auf welche Elemente diese Regeln angewandt werden sollen. Dies geschieht über den CSS-Selektor. Er wird vom Browser interpretiert und kann verschiedene Einschränkungen auf die Menge der betroffenen Elemente machen. Häufige Kriterien der Selektoren sind Elementnamen, IDs und Klassennamen. Elementnamen werden unformatiert angegeben. IDs werden durch eine vorangestellte Raute und Klassennamen durch einen vorangestellten Punkt markiert (s. Listing 2 Zeile 7ff). Mehrere Selektoren werden durch Kommas getrennt. Weiterhin können Kindschaftsverhältnisse, Pseudoklassen und Nachbarschaften in Selektoren verwendet werden um komplexere Regelwerke zu definieren.<sup>6</sup>

<sup>5</sup> Vgl. Jackson (2007), S. 57

<sup>6</sup> Vgl. ebenda, S. 125 f.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Testseite</title>
5     <style>
6         body { background-color: grey }
7         #datum { text-decoration: underline }
8         .abschnitt { font-size: 20px }
9     </style>
10 </head>
11 <body>
12     <h1 style="color: red">Überschrift</h1>
13     <span id="datum">08.07.2017</span>
14     <p class="abschnitt">Abschnitt 1</p>
15     <p class="abschnitt">Abschnitt 2</p>
16 </body>
17 </html>
```

Listing 2: Beispiel.css

JavaScript ist die Sprache, die den Inhalt von Webseiten lokal interaktiv macht. Sie ist eine leichtgewichtige, objektorientierte und dynamisch schwach typisierte Skriptsprache, die von Browsern interpretiert wird. Ihre Syntax erinnert stark an C und C++. Ähnlich wie bei CSS gibt es die Möglichkeit JavaScript direkt im HTML-Dokument zu schreiben. Dies geschieht über das `<script>`-Element. Der auszuführende Code wird direkt als Textwert angegeben. Soll hingegen eine externe JavaScript-Datei eingebunden werden, so wird sie über das `src`-Attribut referenziert.<sup>7</sup> Die Hauptaufgaben, die in JavaScript erledigt werden sind die Reaktion von Usereingaben durch Events, Modifikation des Webseiteninhalts und die asynchrone Kommunikation mit dem Webserver.

Um diese Aufgaben für mehrere Browser einheitlich und pragmatisch zu lösen, wurde die Bibliothek *jQuery* entwickelt.<sup>8</sup> Die Einstiegspunkt von *jQuery* ist die `$`-Funktion. Wird sie mit einem CSS-Selektorstring als Parameter aufgerufen, so interpretiert sie ihn und liefert alle passenden HTML-Elemente als Menge von *jQuery*-Objekten zurück. Durch die Struktur der *jQuery*-Application Programming Interface (API) können auf dieser Ergebnismenge direkt neue Funktionen aufgerufen werden, die intern auf jedes Objekt einzeln angewandt werden. Dieses Verhalten findet sich in vielen weiteren Funktionen von *jQuery* wieder und erlaubt viele verkettete Funktionsaufrufe in einer einzigen Anweisung. Weiterhin vereinfacht *jQuery* die Erstellung von Elementen. Ruft

<sup>7</sup> Vgl. Jackson (2007), S. 198

<sup>8</sup> Vgl. The jQuery Foundation (2017)

man die `$`-Funktion mit einem HTML-Tagstring auf (z.B. `$ ("<div>")`), so wird dadurch ein *jQuery*-Objekt erstellt, das das entsprechende Element repräsentiert. Dieses Objekt kann nun weiter durch Funktionen modifiziert werden, indem ihm Attributwerte gesetzt werden und Routinen für Interaktionsereignisse definiert werden. Beim Einfügen des Elements in das Dokument, kann wieder die `$`-Funktion verwendet werden um den Vaterknoten zu bestimmen und ihm das generierte Element anzuhängen. Die Schlichtheit, die Verkettungsmöglichkeit und das Prinzip der Mengenoperationen von *jQuery* spart Entwicklern Platz und macht zusammengehörige Anweisungen besser lesbar.<sup>9</sup> Aus diesen Gründen setzen viele Webentwickler auf diese Bibliothek. In den neuen Versionen von JavaScript werden diese Features allerdings auch teilweise implementiert, sodass sich der Mehrwert etwas verringert und kritischer überlegt werden sollte, ob die Nachteile der Abhängigkeit gegenüber dem Mehrwert klein genug sind.

Da die Webanwendung *webifier* sich an unerfahrene Internetnutzer wendet liegt ein großer Schwerpunkt auf einer professionellen User Interface (UI), einer optimalen Usability und auf einer User Experience, die dem Nutzer ein Gefühl von Sicherheit gibt. Um diese nichtfunktionalen Anforderungen an die Oberfläche einheitlich und korrekt umzusetzen verwendet die Webanwendung das UI-Framework *Bootstrap*. *Bootstrap*. Das Framework wurde 2010 von einem Designer und einem Entwickler der Social Media Plattform Twitter erschaffen.<sup>10</sup> Damals wurde es als ein interner Styleguide zur Standardisierung von Oberflächen verwendet. Heute ist es ein sehr bekanntes Open Source Projekt und eines der führenden Webframeworks in ebendiesem Bereich. *Bootstrap* liefert neben den grundlegenden CSS-Regeln für Standardelemente auch ein Strukturierungssystem mit, das es Entwicklern ermöglicht ihre Webseiten für alle möglichen Endgeräte mit verschiedenen Auflösungen zu optimieren. Dieses Prinzip nennt sich *Responsive Design* und lässt sich in *Bootstrap* durch CSS-Regeln an seine eigenen Anforderungen anpassen.<sup>11</sup> Im Kern wird *Bootstrap* so verwendet, dass die zu vereinheitlichenden Elemente mit Framework-spezifischen Klassennamen versehen werden.<sup>12</sup> Zusätzlich zu der Einheitlichen UI und dem Responsive Design können mit dem Framework auch häufig gebrauchte Interaktionsmuster durch HTML-Code angegeben werden. Dazu muss dann das JavaScript-Paket mit als Ressource angegeben werden, wobei diese wiederum eine Kompatible Version von *jQuery* voraussetzt.

---

<sup>9</sup> Vgl. Flanagan (2010), S. 3 ff.

<sup>10</sup> Vgl. Spurlock (2013), S. 1

<sup>11</sup> Vgl. ebenda, S. 7 f.

<sup>12</sup> Vgl. ebenda, S. 4

## 2.2 Backend Technologien und Frameworks

In diesem Abschnitt werden nun alle Technologien und Frameworks vorgestellt welche in den Backends der einzelnen Teilanwendungen zum Einsatz kamen.

Wohl am häufigsten kam die Programmiersprache Java zum Einsatz. Java ist eine universal einsetzbare, nebenläufige, klassenbasierte und objektorientierte Programmiersprache. Sie wurde möglichst einfach gestaltet um von vielen Entwicklern genutzt zu werden. In ihrer Syntax ähnelt sie den Programmiersprachen C und C++. Außerdem ist sie stark und statisch typisiert. Vorallem aber zeichnet sich Java durch seine plattformunabhängigkeit aus. Diese wird dadurch umgesetzt, dass Java-Quellcode in plattformunabhängigen Byte-Code kompiliert wird, welcher von einer Java Virtual Machine (JVM) ausgeführt wird. Java ist eine Hochsprache, die mit Hilfe des so genannten „Garbage Collectors“ eine automatische Speicherverwaltung bereitstellt.<sup>13</sup>

In einigen Teilprojekten wurde das auf Java basierende *Spring*-Framework verwendet. *Spring* stellt eine vereinfachte Möglichkeit auf den Zugriff auf viele API der Standard-Version zur Verfügung. Ein weiterer wesentlicher Bestandteil des *Spring*-Frameworks ist die *Dependency Injection*. Hierbei suchen sich Objekte ihre Referenzen nicht selbst, sondern bekommen diese Anhand einer Konfiguration injiziert. Dadurch sind sie eigenständig und können in verschiedenen Umgebungen eingesetzt werden. Des weiteren bringt *Spring* eine Unterstützung für aspektorientierte Programmierung mit, wodurch mit verschiedenen Abstraktionsschichten einzelne Module abgekapselt werden können.<sup>14</sup>

Aufbauend auf dem *Spring* Basis-Modul werden noch weitere Module, wie beispielsweise Spring Security, Spring Boot, Spring Integration, Spring Data, Spring Session oder Spring MVC.<sup>15</sup> Im folgenden werden die *Spring*-Module näher erläutert, die für das weitere Verständnis der Arbeit notwendig sind.

### Spring Boot

Mit Spring Boot können Anwendungen, welche das *Spring*-Framework nutzen, einfacher entwickelt und ausgeführt werden, da dadurch eigenständig lauffähig-

---

<sup>13</sup> Vgl. Gosling u. a. (2014), S. 1

<sup>14</sup> Vgl. Wolff (2011), S. 2

<sup>15</sup> Vgl. Cosmina (2016), S. 2

ge Programme erzeugt werden können, welche nicht von externen Services abhängig sind. Hierfür bringt Spring Boot einen Integrierten Server mit, auf welchem die Anwendung bereitgestellt wird.<sup>16</sup>

### Spring MVC

Spring MVC ist sehr gut geeignet um Webanwendungen zu implementieren.<sup>17</sup> Hierfür können die diese in mehrere Abstraktionsschichten gegliedert werden. Beispielsweise in das UI, die Geschäftslogik und die Persistenzschicht.<sup>18</sup>

### Spring Data

Spring Data vereinfacht Datenbankzugriffe ungemein. Das Modul stellt APIs für fast alle gängigen Datenbankzugriffsschichten, wie JDBC (Java Database Connectivity), Hibernate, JDO (Java Data Objects) zur Verfügung. Aber nicht nur relationale Datenbanken werden unterstützt, sondern beispielsweise auch NoSQL-Datenbanken und Key/Value-Stores können Problemlos eingesetzt werden.<sup>19</sup>

In Verbindung mit Spring Data wurde eine *MongoDB* zur Speicherung der Ergebnisse eingesetzt. *MongoDB* ist eine Dokument orientierte anpassungsfähige und skalierbare Datenbank. Sie vereint viele nützliche Eigenschaften von Relationalen Datenbanken, wie Sekundärindizes, Auswahlabfragen und Sortierung mit Skalierbarkeit, MapReduce-Aggregationen und raumbezogenen Indizes. Außerdem gibt es bei MongoDB keine festen Schemata, weshalb großen Datenmigrationen normal nicht notwendig sind.<sup>20</sup>

Gewonnene und gespeicherte Daten müssen danach auch noch aufbereitet und visualisiert werden. Webifier setzt dafür auf die Programmiersprache R. R ist eine freie Programmiersprache, entwickelt für statistische Auswertungen und Visualisierungen. Sie zählt zu den prozeduralen Programmiersprachen. Die quelltextoffene Programmiersprache wird ständig weiterentwickelt. Zusätzlich gibt es eine Vielzahl an Packages, welche weitere Funktionalität bereitstellen. Diese sind über ein zentrales Repository abrufbar und so leicht einbindbar in den Quelltext.<sup>21</sup>

---

<sup>16</sup> Vgl. Gutierrez (2016), S. 1

<sup>17</sup> Vgl. Wolff (2011), S. 3

<sup>18</sup> Vgl. Yates u. a. (2006), S. 21

<sup>19</sup> Vgl. Pollack u. a. (2012), S. 3f

<sup>20</sup> Vgl. Chodorow / Dirolf (2010), S. 1f

<sup>21</sup> Vgl. Wollschläger (2014), S. 1ff

Ein wichtiger Bestandteil jedes großen Software-Projektes ist ein gutes Build-Management-Tool. Für webifier wurde *Gradle* als solches gewählt. Ein Build-Prozess besteht grundsätzlich aus zwei Teilschritten. Zum Einen aus dem kompilieren des Codes und zum anderen aus dem verlinkten der benutzen Bibliotheken.<sup>22</sup> Da das manuelle Einbinden von Bibliotheken und kompilieren des Codes bei großen Projekten sehr aufwändig und mühsam sein kann wird hier auf Build-Management-Tools wie *Gradle* zurückgegriffen. Um den Build für den Nutzer möglichst einfach zu gestalten verfolgt Gradle zwei Prinzipien. Das erste Prinzip ist *Convention over Configuration*, was bedeutet, dass soweit es geht ein Standardbuildprozess definiert ist und der Anwender nur die Parameter ändern muss die Projektspezifisch abweichen. Das zweite Prinzip nennt sich Don't Repeat Yourself (DRY). Hierbei geht es darum Redundanzen in der Konfiguration des Buildes zu vermeiden. Diese beiden Prinzipien helfen Gradle, dass meist kurze Build-Skripte ausreichen um komplexe Prozesse abzubilden.<sup>23</sup>

Die Kommunikation zwischen Server und Client erfolgt über Representational State Transfer (REST). Hierbei wird jedes Objekt in REST als Ressource definiert, welche über einen eindeutigen Uniform Ressource Identifier (URI) adressiert werden können. Über die HTTP-Methoden GET, PUT, POST und DELETE können diese Ressourcen geladen, erstellt, geändert oder auch gelöscht werden.<sup>24</sup>

Das Testen von potenziell gefährlichen Webseiten soll natürlich nicht direkt auf dem Server geschehen, da es sonst diesen potenziell gefährden könnte. Deshalb wird hierfür eine Virtualisierung benötigt um die Tests abgekapselt vom Gesamtsystem auszuführen. Dafür wurde Docker als Tool eingesetzt. Docker ist eine Open-Source-Software zur Virtualisierung von Anwendungen. Hierbei wird auf die Container-Technologie gesetzt. Container sind vom Betriebssystem bereitgestellte virtuelle Umgebung zur isolierten Ausführung von Prozessen. Ein Vorteil der Container gegenüber der herkömmlicher virtuelle Maschinen ist der vielfach geringere Ressourcenbedarf.<sup>25</sup>

## 2.3 Technologien und Frameworks der Tests

In diesem Kapitel werden diejenigen Technologien und Frameworks erläutert, die zur Umsetzung der Sicherheitstests verwendet werden.

---

<sup>22</sup> Vgl. Wikipedia (2017)

<sup>23</sup> Vgl. Baumann (2013), S. 6f

<sup>24</sup> Vgl. itwissen.info (2017)

<sup>25</sup> Vgl. Roden (2017)

*Python* ist eine Programmiersprache, die einen schnellen Projektstart ermöglicht und ist auf Integration von verschiedenen Systemen spezialisiert. Die Sprache wird von der Python Software Foundation nach Open Source Standards entwickelt. Die aktuellste Version ist Python 3.6.1, wobei bei der Implementierung der Tests keine unterschiedliche Versionen verwendet werden. Python zählt zu den dynamisch typisierten Programmiersprachen, was bedeutet, dass es wie bei JavaScript erst zur Laufzeit zu einer Typenprüfung kommt. Weiterhin werden Codeblöcke nicht durch Sonderzeichen (wie z.B. geschweifte Klammern in Java) gekennzeichnet, sondern definieren sich an der Einrückungstiefe.<sup>26</sup> Da Python ein bekanntes Open Source Projekt ist, beteiligen sich viele Programmierer an der Entwicklung von Zusatzprogrammen, sogenannten *Packages*. Es gibt mittlerweile bereits über 100.000 Pakete<sup>27</sup>, die alle frei verfügbar sind und über einen einheitlichen Prozess in Projekte eingebunden werden können. Aus diesen Flexibilitätsgründen und wird für die Testlogik in *webifier* überwiegend Python verwendet.

Es gibt verschiedene Möglichkeiten Nutzerverhalten im Browser zu simulieren. *PhantomJS* ist eine davon. Dabei handelt es sich um eine Automatisierungssoftware, die auf der Browser Engine *WebKit* der Firma Apple basiert. *PhantomJS* ist ein Headless Browser, was bedeutet, dass es keine grafische Oberfläche besitzt. Es wird über ein JavaScript gesteuert und über eine ausführbare Datei aufgerufen. Weitere Hauptfeatures sind: Bildschirmaufnahmen, Nutzersimulation und Netzwerküberwachung.<sup>28</sup> Die Technologie wird in *webifier* hauptsächlich verwendet, um selbst erstellten JavaScript in die zu untersuchenden Webseiten einzubetten und erst danach den Code der Webseite auszuführen. In *PhantomJS* gilt es zwischen dem Steuerungsskript und dem in die Webseite eingebetteten Code zu unterscheiden. In ersterem wird die Webseite konfiguriert und die Testlogik ausgeführt. In der Testlogik können wiederum Anweisungen an die Webseite geschickt werden, die mit dem Rest des Steuerungsskripts nichts zu tun haben. Sie haben einen eigenen Geltungsbereich, nämlich den in der Browser Engine.

Um Webseiten mit allen ihren Ressourcen herunterzuladen wurde die freie Software *HTTrack* verwendet. Mit *HTTrack* können Webseiten in einem lokalen Verzeichnis gespeichert werden. Hierfür erzeugt das Programm rekursiv alle notwendigen Verzeichnisse und lädt anschließend alle Ressourcen, wie HTML-, CSS- und JavaScript-Dateien, als auch Bilder und andere Dateien herunter. Außerdem ist es möglich automatisiert

<sup>26</sup> Vgl. Python Software Foundation (2017a)

<sup>27</sup> Vgl. Python Software Foundation (2017b)

<sup>28</sup> Vgl. Hidayat (2017)



alle HTML-Links entsprechend zu modifizieren. Abschließend bietet HTTrack umfassende Konfigurationsoptionen um es für den optimalen Gebrauch anpassen zu können.<sup>29</sup>

Für die Analyse und den Vergleich von Bildern wurde auf die freie JavaScript-Bibliothek Resemble.js zurückgegriffen. Mit Resemble können jegliche Arten von Bildanalyse und Bildvergleich genutzt werden. Ursprünglich wurde es für eine Bibliothek von Phantom JS entwickelt, kann aber inzwischen vielseitig eingesetzt werden. Resemble bietet einige Einstellungsmöglichkeiten um Bilder analysieren und miteinander vergleichen zu können. Als Resultat liefert es bei der Bildanalyse Helligkeits- und Farbwerte des Bildes. Beim Bildvergleich bekommt man den prozentualen Unterschied der beiden Bilder, sowie einige Zusatzinformationen. Außerdem ist es möglich mit Resemble.js ein Differenzbild mit der Hervorhebung der Unterschiede zweier Bilder zu erzeugen.<sup>30</sup>

Zu einer umfassenden Analyse gehört selbstverständlich auch die Analyse des Netzwerktraffics. Dazu wird ein entsprechendes Tool genutzt. Webifier nutzt für diesen Zweck den *Bro Network Security Monitor*. Bro ist ein Unix-basiertes Network Intrusion Detection System (NIDS).<sup>31</sup> Zudem ermöglicht Bro dem Nutzer den Netzwerktraffic zu loggen und mittels eigener Skriptsprache zu filtern.<sup>32</sup> Die Logging-Möglichkeiten werden für die Analyse des Traffics genutzt um mögliche verdächtige Abfragen zu erkennen.

## 2.4 Angriffstypen

Rechner und deren Nutzer werden häufig durch Angriffe aus dem Internet gefährdet. Beispielsweise durch Sicherheitslücken der Browser, durch die dieser zum Absturz gebracht werden kann. Häufig brechen Angreifer in Webserver ein um dort schadhaften Code zu verbreiten oder stellen diesen auf eigenen Webseiten bereit. Besuchen Nutzer solche Webseiten wird der Code des Angreifers vom Browser interpretiert und kann dadurch den Rechner des Nutzers oder auch das gesamte lokale Netzwerk, auch hinter einer Firewall angreifen.<sup>33</sup>

---

<sup>29</sup> Vgl. Roche/ Kauler (2017)

<sup>30</sup> Vgl. Cryer (2017)

<sup>31</sup> Vgl. Ghorbani/ Lu/ Tavallaee (2009), S. 199

<sup>32</sup> Vgl. Bro Network Monitor (2017)

<sup>33</sup> Vgl. Oriyano/ Shimonski (2012), S. 1 ff.

Nach Schätzungen des DIW Wochenberichts entstehen in Deutschland jährlich Schäden in Höhe von insgesamt 3,4 Milliarden Euro.<sup>34</sup> Hochrechnungen zufolge entstehen diese Schäden aus jährlich 14,7 Millionen Fällen von Internetkriminalität. Diese können in die vier Kategorien Schadsoftware, Waren- und Dienstleistungsbetrug, Identitätsdiebstahl und Phishing gruppiert werden.<sup>35</sup>



Abbildung 3: Bedeutung der einzelnen Kategorien von Internetkriminalität<sup>36</sup>

Wie Abbildung 3 zeigt die Bedeutung dieser Kategorien. Unter allen Fällen von Internetkriminalität sind demnach 63 % Delikte mit Schadsoftware, 16% sind Betrüge im Waren- und Dienstleistungsbereich, 14% machen Identitätsdiebstäle aus und 7% kommen von Phishingseiten.

In den folgenden Abschnitten werden nun einige übliche Angriffstypen von Webseiten auf den Nutzer vorgestellt und eine mögliche Überprüfung in webifier dargestellt.

<sup>34</sup> Vgl. Rieckmann/ Kraus (2015), S. 301

<sup>35</sup> Vgl. ebenda, S. 297

<sup>36</sup> Siehe ebenda, S. 297 Abbildung 1

### 2.4.1 Malware

Spyware, Root Kits, Trojaner und viele mehr - alles das ist Malware, welche den Nutzern in unterschiedlichen Weisen kleineren, oder größeren Schaden zuführen. Kurz: Malware ist Software mit bösartiger Wirkung. In diesem Abschnitt werden nun einige Formen von Schadsoftware beschrieben und wie diese in ein System gelangen kann.<sup>37</sup>

Malware ist so vielfältig wie gutartige Anwendungen. Dennoch lässt sie sich auf verschiedene Weisen klassifizieren. Allerdings sind die Wübergänge der einzelnen Klassen fließend. Zum Einen kann Malware im Hinblick auf ihre Verbreitungsmethode und zum Anderen in der Art des Schadens für den ungewollten Anwender unterschieden werden. Alle Klassen vereint jedoch dass Malware im allgemeinen Code enthält, welcher dem Nutzer oder dessen System Schaden zufügt.<sup>38</sup>

Bei der Verbreitungsmethode kann zwischen Viren, Trojanern und Würmern unterschieden werden. Viren sind Programme, welche sich bei der Ausführung selbst kopieren, beispielsweise indem sie ihren Code in andere Programme oder Dokumente des Nutzers einschleusen.<sup>39</sup> Die ersten Viren wurden Anfang der 1980er Jahre in Umlauf gebracht, allerdings spielten Viren sogar schon 1970 in dem Science Fiction Film *The Scarred Man* eine Rolle.<sup>40</sup> Trojaner sind Anwendungen, welche vortäuschen gutartig zu sein, aber Code beinhalten, welcher dem System oder dem User schadet. Trojaner sind seit 1972 bekannt und verbreiten sich üblicher Weise nicht eigenständig.<sup>41</sup> Würmer verbreiten sich üblicherweise von alleine über Netzwerke und infizieren so andere Systeme. Hierfür nutzen sie Schwachstellen in Netzwerkdiensten und schädigt so der Maschine oder dem Anwender.<sup>42</sup> Die ersten Würmer sind wie die ersten Viren in der Science Fiction zu finden. Würmer kommen in dem Roman *The Shockwave Rider* von John Brunner aus dem Jahr 1975 vor. Die ersten realen Würmer waren bereits 1970 im damaligen Arpanet zu finden.<sup>43</sup>

---

<sup>37</sup> Vgl. Kappes (2013), S. 95

<sup>38</sup> Vgl. ebenda, S. 95 f.

<sup>39</sup> Vgl. ebenda, S. 95

<sup>40</sup> Vgl. Aycock (2006), S. 14

<sup>41</sup> Vgl. ebenda, S. 12 f.

<sup>42</sup> Vgl. Kappes (2013), S. 95

<sup>43</sup> Vgl. Aycock (2006), S. 15

Anhand des angerichteten Schadens kann Malware in Spyware, Adware, Malware-Dialer, Zombie-Malware, Backdoors und Root Kits unterteilt werden. Spyware ist Software, welche ohne Wissen des Nutzers Informationen sammelt und weiterleitet. Dadurch könne vertrauliche Daten gestohlen und missbraucht werden.<sup>44</sup> Solche Daten können beispielsweise Benutzernamen und Passwörter, E-Mailadressen, Bankaccounts und Kreditkartennummern oder Softwarelizenzen sein. Mitte der 1990er Jahre war erste Spyware zu finden.<sup>45</sup> Als Adware werden Programme bezeichnet, welche dem Benutzer Werbeanzeigen einblenden.<sup>46</sup> Adware ist ähnlich zu Spyware, da beide Informationen über den Nutzer sammeln. Allerdings ist Adware mehr auf Marketing fokussiert und nutzt die Informationen um dem Nutzer Werbung zu präsentieren.<sup>47</sup> Dialer sind Programme, welche Computern über Modems oder Telefonnetze Zugang zum Internet anbieten. Malware-Dialer nutzen das aus und wählen die Rechner ohne Kenntnis des Nutzers in teure Service-Rufnummern oder Anwahlpunkte im Ausland ein. Allerdings findet man diese Art von Malware nur noch selten, da es inzwischen telefonbasierten Internetzugänge an Bedeutung verlieren. Software, welche Rechner kompromittiert, wird als Zombie-Malware bezeichnet, da dieser so von Angreifern ferngesteuert werden kann.<sup>48</sup> Am häufigsten werden Zombie-Rechner eingesetzt um Spam zu versenden oder mit vielen anderen Denial of Service Angriffe auszuführen.<sup>49</sup> Backdoors sind modifizierte Programme des Systems, über welche Hacker Sicherheitsmechanismen umgehen und sich so unbefugten Zugriff auf den Rechner verschaffen kann. Modifizierte Softwaregruppen, welche zum Ziel haben deren Aktivität oder die eines Angreifers vor Systembenutzern, inklusive Administratoren zu verstecken werden als Root Kits bezeichnet.<sup>50</sup>

Wie Abbildung 4 zeigt nimmt die Verbreitung von Malware in Deutschland weiterhin zu und verliert deshalb nicht an Bedeutung.

---

<sup>44</sup> Vgl. Kappes (2013), S. 95 f.

<sup>45</sup> Vgl. Aycock (2006), S. 16

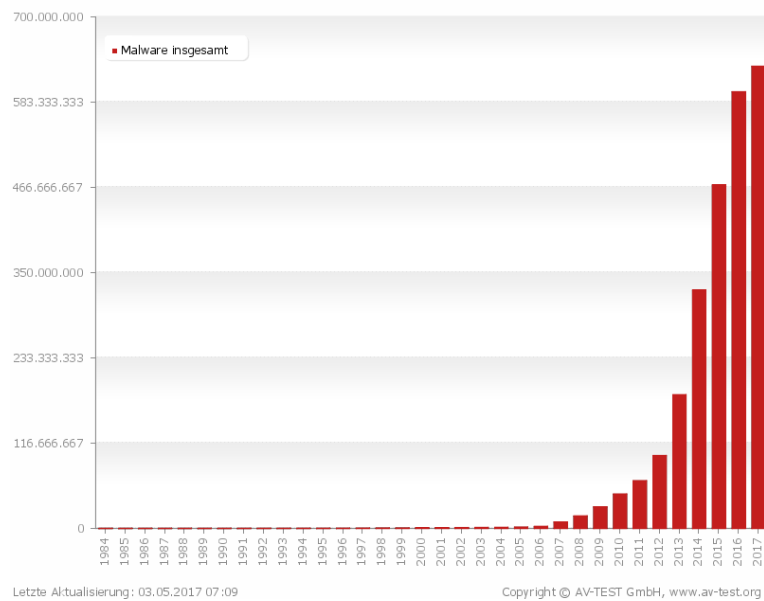
<sup>46</sup> Vgl. Kappes (2013), S. 96

<sup>47</sup> Vgl. Aycock (2006), S. 17

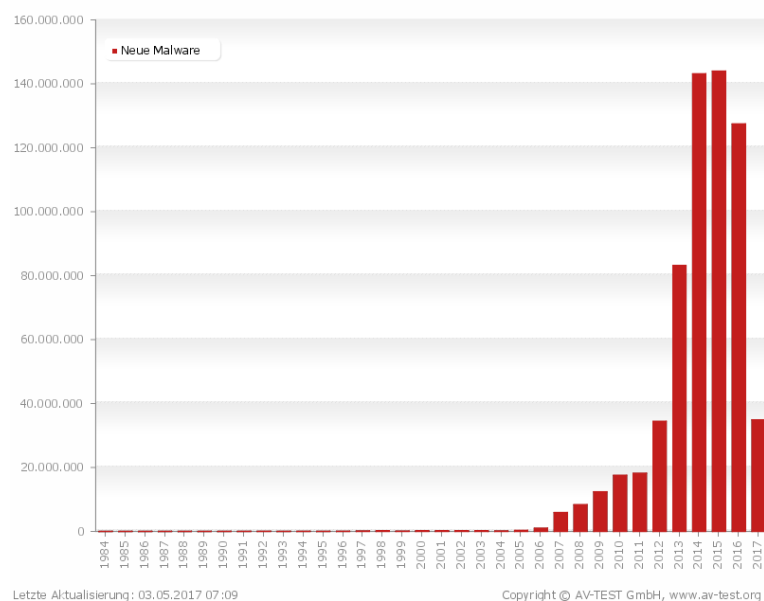
<sup>48</sup> Vgl. Kappes (2013), S. 96

<sup>49</sup> Vgl. Aycock (2006), S. 18

<sup>50</sup> Vgl. Kappes (2013), S. 96

Abbildung 4: Malware Verbreitung in Deutschland<sup>51</sup>

Interessant ist allerdings dass in Abbildung 5 ein deutlicher Rückgang in der Verreitung von neuer Malware zu erkennen ist. Daraus lässt sich schließen, dass deshalb die bereits in Umlauf gebrachte Schadware nach wie vor ausreicht um dem Großteil der Nutzer zu schaden.

Abbildung 5: Verbreitung neuer Malware in Deutschland<sup>52</sup>

<sup>51</sup> Siehe AV-TEST GmbH (2017), Abbildung 1

Die Verbreitung von Malware beginnt größtenteils über Webseiten und E-Mails.<sup>53</sup> Deshalb ist es notwendig mit webifrier Webseiten auf Malware zu prüfen.

### 2.4.2 User Agent Sniffing

Wer sich im WWW bewegt benutzt meist Software, die die Navigationsbefehle des Users in HTTP-Requests umwandeln und an den jeweiligen Webserver schicken. Diese Programme werden *User Agents* genannt. Menschen verwenden Browseranwendungen als *User Agents*, um sich auf einer grafischen Oberfläche durch das Netz zu klicken. Ein weitaus kleinerer, aber umso wichtigerer Teil der Websurfer sind Maschinen, die selbstständig arbeiten: Webcrawler. Dies sind Programme, die nach komplexen Algorithmen arbeiten, indem sie Webseiteninhalte über HTTP herunterladen, analysieren und daraus wieder neue HTTP-Requests generieren.<sup>54</sup> HTTP-Requests beinhalten einen Request-Header, der dem Server Auskunft über den Browser des Clients und den gewünschten Ressourcentyp geben kann.<sup>55</sup> Das wichtigste Header-Feld zur Identifizierung heißt *User Agent*. Es enthält im Normalfall den Browsertyp, die Browserversion und das Betriebssystem des Browsers und wird von einigen Webservern und Webanwendungen benutzt, um je nach Wert des Feldes unterschiedliche Ressourcen zurückzuliefern.<sup>56</sup>

Diese Technik wird *User Agent Sniffing* genannt und ist ein heute unerwünschter Eingriff, der nur selten seine Berechtigung hat. Sie zählt zu den Bad Practices in der Webentwicklung.<sup>57</sup> Das Zurückliefern unterschiedlicher Ressourcen je nach *User Agent* wird in der Fachliteratur hingegen unterschiedlich bewertet. In Gourley/ Totty (2002) (S. 228) wird zunächst ein entscheidendes Problem dieses Verhaltens aufgezeigt. Viele Webseiten passen ihre Inhalte für verschiedene *User Agents* an. Ihr Ziel ist es sicherzustellen, dass ihre Anwendungen auf möglichst allen Geräten laufen. Funktioniert etwas nicht, weil bspw. die Browserversion zu alt ist, dann wird eine Fehlerseite zurückgeliefert. Vollautomatisierte Webcrawler bekommen häufig diese Fehlerseiten zurückgeliefert und arbeiten mit diesen weiter, obwohl sie die Funktionen der Seiten an

<sup>52</sup> Siehe AV-TEST GmbH (2017), Abbildung 2

<sup>53</sup> Vgl. Kappes (2013), S. 97

<sup>54</sup> Vgl. Gourley/ Totty (2002), S. 19

<sup>55</sup> Vgl. Wong (2000), S. 9

<sup>56</sup> Vgl. Gourley/ Totty (2002), S. 259, 528 f.

<sup>57</sup> Vgl. Shepherd (2016)

sich gar nicht anwenden, sondern lediglich deren Quelltext analysieren wollen. Im Gegensatz dazu wird später (S. 402) die Aussage gemacht, dass Webserver durchaus ihre Antworten an den *User Agent* anpassen können und dies nicht so schlimm sei.

Habe der Client einen veralteten Browser, der kein z.B. JavaScript unterstützt, so könne der Webserver in diesem Fall einfach eine Webseite ohne JavaScript zurückliefern. Shepherd (2016) erläutert drei Hauptgründe, warum *User Agent Sniffing* betrieben wird: Bug-Workarounds, Feature-Unterstützung und Browserspezifisches HTML.<sup>58</sup> Da es für all diese Beweggründe bessere Lösungsansätze gibt und *User Agent Sniffing* als Vorstufe zum Browserexploit genutzt werden kann, wird es in dieser Arbeit als Angriff auf den Client angesehen.

### 2.4.3 JavaScript Port & IP Scanning

Um Angriffe über das Netzwerk zu starten muss der Angreifer Kenntnisse über den Netzwerkaufbau und die erreichbaren Services des anzugreifenden Systems haben.<sup>59</sup>

Über die offenen Ports eines Systems kann sich ein potenzieller Angreifer Zugang beschaffen. Jedoch muss zunächst herausgefunden werden, welche Ports erreichbar sind. Hierfür wird eine Technik Namens *Port Scanning* genutzt. Port Scanning ist im Grunde das Abfragen einiger oder auch aller Ports eines Systems. Es gibt heutzutage 65.535 TCP und 65.535 UDP Ports, von denen einige in Systemen offen sind, jedoch die meisten davon geschlossen.<sup>60</sup> UDP und TCP sind zwei verschiedene Internet Protokolle. Sie unterscheiden sich zum einen darin, dass TCP verbindungsorientiert arbeitet, während UDP verbindungslose Kommunikation nutzt.<sup>61</sup> Ein Portscanner nutzt die verschiedenen Eigenschaften der Protokolle aus um festzustellen ob ein Port offen oder geschlossen ist.<sup>62</sup> Die Unterschiede werden hier aber nicht weiter beleuchtet. Einige dieser Ports sind standardisiert und werden von bestimmten Webservices genutzt, wie beispielsweise TCP Port 80, welcher in der Regel von Web Servern eingesetzt wird.

Port Scanning liefert hierbei Informationen welche Ports eines Systems offen für Netzwerkverbindungen sind.

---

<sup>58</sup> Vgl. Shepherd (2016)

<sup>59</sup> Vgl. Tipton/ Krause (2007), S. 937

<sup>60</sup> Vgl. ebenda, S. 937

<sup>61</sup> Vgl. nixCraft (2017)

<sup>62</sup> Vgl. Messier (2016), S. 31

In Listing 3 wird ein Beispielcode für einen, in Java implementierten, simplen Portscanner gezeigt. Dieser prüft alle 65535 TCP Ports eines gegebenen Hosts. Er versucht über jeden dieser Ports eine Socketverbindung aufzubauen (siehe Zeile 19-32), welche er anschließend wieder schließt. Wenn hierbei keine Fehlermeldung vom System geworfen wird weiß der Scanner, dass die Verbindung mit dem getesteten Port aufgebaut wurde und somit dieser Port offen ist. Dieser einfache Portscanner liefert als Ergebnis eine Anzahl an offenen Ports im getesteten System.

```
1 public class Portscanner {
2     public static void main(final String... args) {
3         final ExecutorService es = Executors.newFixedThreadPool(20);
4         final String ip = "127.0.0.1";
5         final int timeout = 200;
6         final List<Future<Boolean>> futures = new ArrayList<>();
7         for (int port = 1; port <= 65535; port++) {
8             futures.add(portIsOpen(es, ip, port, timeout));
9         }
10        es.shutdown();
11        int openPorts = 0;
12        for (final Future<Boolean> f : futures) {
13            if (f.get()) {
14                openPorts++;
15            }
16        }
17        System.out.println("There are " + openPorts + " open ports on host " + ip + " (probed
18                               with a timeout of " + timeout + "ms)");
19    }
20    public static Future<Boolean> portIsOpen(final ExecutorService es, final String ip, final
21        int port, final int timeout) {
22        return es.submit(new Callable<Boolean>() {
23            @Override public Boolean call() {
24                try {
25                    Socket socket = new Socket();
26                    socket.connect(new InetSocketAddress(ip, port), timeout);
27                    socket.close();
28                    return true;
29                } catch (Exception ex) {
30                    return false;
31                }
32            }
33        });
34    }
35 }
```

Listing 3: Beispiel eines simplen Java PortScanners<sup>63</sup>

<sup>63</sup> Zitat StackOverflow (2017)



Es gibt grundsätzlich viele verschiedene Möglichkeiten ein Angriffsziel auf offene Ports zu überprüfen. Die in dem Codebeispiel gezeigte Möglichkeit ist relativ simpel, jedoch effektiv. Jedoch ist es ebenfalls recht einfach einen derartigen Angriff zu blocken, da Schutzprogrammen leicht erkennen können, dass es sich um einen Portscan-Angriff handelt und die entsprechende IP dann für weitere Anfragen blockieren. Um diesen offensichtlichen Angriff zu verschleiern werden oft verschiedene Hosts genutzt. Der Angriff verteilt sich dann auf Anfragen von verschiedenen IP's, was es einem Erkennungsalgorithmus erschwert legitime Anfragen von einem verteilten Portscan zu differenzieren. Je mehr Hosts an einem derartigen Scan beteiligt sind, desto schwieriger wird es den entsprechenden Scan zu erkennen und entsprechende IP's zu blockieren. Zusätzlich spielt noch die Art der Anfrage eine Rolle. Es kann beispielsweise mittels eines Pings ein bestimmter Port angefragt werden. Dies ist aber keine effiziente Methode, da oft Firewalls Pings blockieren.

Im Folgenden wird einer der bekanntesten Scantypen, der SYN-Scan, erläutert. Für das Verständnis eines SYN-Scans muss zunächst erklärt werden, wie in TCP Verbindungen aufgebaut werden.

TCP nutzt für den Verbindungsaufbau den 3-Wege-Handshake. Der Ablauf ist in Abbildung 6 dargestellt. Zuerst sendet der Client einen *SYN* an den Server. Dieser wird dann vom Server empfangen und er sendet einen *ACK* als Antwort und gleichzeitig einen eigenen *SYN* zurück an den Client. Zum Schluss sendet der Client noch einen *ACK* zum Server als Antwort auf dessen *SYN*. Die Verbindung ist danach aufgebaut und es können Daten übermittelt werden.<sup>64</sup>

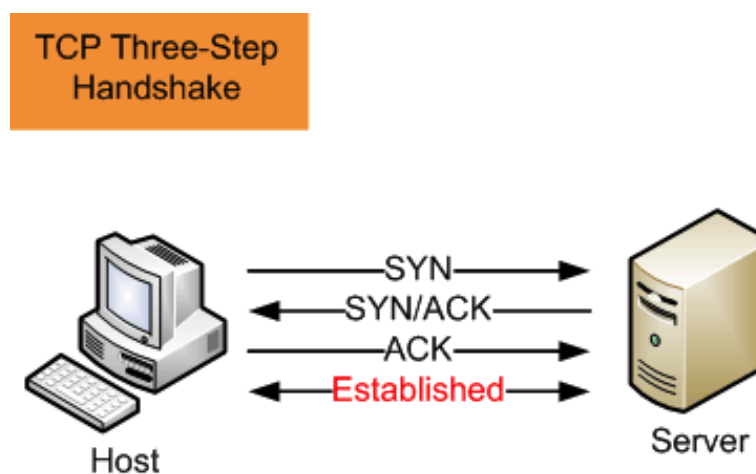


Abbildung 6: 3-Step-Handshake TCP<sup>65</sup>

<sup>64</sup> Vgl. Messier (2016), S. 32

Für einen SYN-Scan versucht nun der Scanner TCP-Verbindungen mit den zu scannenden Ports aufzubauen. Er sendet also SYN-Anfragen an diese. Anhand der Antwort kann der Scanner den Status des Ports erkennen. Antwortet das Angriffsziel mit einem SYN/ACK so ist der Port offen. Ist die Antwort ein Reset-Flag(RST), kann der Port als geschlossen markiert werden. Komplizierter wird es, wenn keine Antwort kommt. Dies kann mehrere Gründe haben. Zum Einen ist es möglich, dass das Angriffsziel keine Verbindung hat. Deshalb sollte vor dem Scan überprüft werden ob der Rechner erreichbar ist. Ein weiterer Grund könnte eine Firewall sein, die Anfragen blockiert. Um dies Herauszufinden wird das SYN-Packet wiederholt gesendet falls keine Antwort erhalten wird. Ist ein Server erreichbar, aber es folgt keine Antwort auf einen SYN kann der Port als gefiltert markiert werden, was aussagt, dass eine Firewall die Verbindungen blockiert.<sup>66</sup>

Um als Webseite die eigenen Nutzer zu scannen muss der Portscan-Code beispielsweise mittels JavaScript in die Seite eingebunden werden. Hier gibt es verschiedene Arten den Code möglichst unauffällig zu platzieren. Eine Möglichkeit wird in Listing 4 gezeigt. Hier wird Portscanning betrieben indem versucht wird von der entsprechenden IP und dem Port eine img-Datei nachzuladen. Je nach Ergebnis wird dann entschieden ob der Port offen oder geschlossen ist. Bei erfolgreichem Laden oder einer Fehlermeldung bedeutet es, dass der Port offen ist, da er reagiert. Läuft der Request in einen Timeout wird der Port als geschlossen angesehen.

Für die Funktion müssen Host-Adresse, der zu scannende Port und ein Timeout-Wert übergeben werden. Zunächst wird nun ein neues Image-Objekt erstellt(Zeile 2). Es wird definiert, dass bei einem Fehler als Rückgabewert oder einem erfolgreichen Laden der Port als offen markiert wird(Zeile 4-9). Bei bestimmten Ports muss noch das Protokoll entsprechend geändert werden. Beispielsweise muss für Port 443 das HTTPS-Protokoll verwendet werden(Zeile 16). Als Standardprotokoll ist HTTP definiert(Zeile 17). Das Protokoll, die Host-Adresse und der Port bilden zusammen den Pfad für das Image-Objekt. Je nach Protokoll muss noch ein eine jpg-Ressource angegeben werden(Zeile 17). Als Name der jpg-Datei wird hier die ID verwendet. Dieser Name ist jedoch irrelevant. Zum Schluss wird noch bei einem Timeout der Port als geschlossen markiert(Zeile 20-24). Mithilfe der Function scan() wird der Portscan gestar-

---

<sup>65</sup> Siehe Aung (2017), Abbildung 1

<sup>66</sup> Vgl. Messier (2016), S. 33

tet. Das Beispiel-Listing bildet nur einen Teil des Portscans ab, zur besseren Übersichtlichkeit wurden initialisierung von Host und Port-Bereich sowie die Ergebnisausgabe nicht dargestellt.

```
1 function scan() {
2     var req = new pingrequest( f.host.value,p,t);
3     req.dorequest();
4 }
5 this.dorequest = function() {
6     var img = new Image();
7
8     img.onerror = function () {
9         if (!img) return;
10        img = undefined;
11        callback( host, port, 'up',id);
12    };
13    img.onload = img.onerror;
14
15    switch(port){
16        case 21: src = 'ftp://' + this.id() + '@' + host + '/'; break;//ftp
17        case 25: src = 'mailto://' + this.getid() + '@' + host ; break;//smtp **
18        case 70: src = 'gopher://' + host + '/'; break;//gopher
19        case 119: src = 'news://' + host + '/'; break;//nntp **
20        case 443: src = 'https://' + host + '/' + this.getid() + '.jpg';
21        default: src = 'http://' + host + ':' + port + '/' + this.getid() + '.jpg';// getid is
                here to prevent cache seekings;
22    }
23    img.src = src;
24    setTimeout(function () {
25        if (!img) return;
26        img = undefined;
27        callback( host, port, 'down',id);
28    }, timeout);
29 };
```

Listing 4: Beispiel eines simplen JavaScript PortScanners

Zusätzlich zum Portscan gibt es noch den IP-Scan. Ein IP-Scan funktioniert grundsätzlich ähnlich zum Portscan. Hier wird jedoch nicht versucht die möglichen Angriffspunkte eines Rechners aufzudecken, sondern das Netzwerk auszuspähen. Da JavaScript clientseitig ausgeführt wird versuchen Angreifer häufig über die bekannten *Heimnetz-IPs*(beispielsweise. 192.168.178.\*) das Netzwerk zu analysieren. Es werden die IP-Bereiche abgesucht nach anderen Rechner, die sich in dem Netzwerk befinden. So kann ein Angreifer Erkenntnisse über das komplette Netzwerk seines Ziels erlangen um so einen erfolgreicher Angriff zu starten.

## 2.4.4 Phishing

Beim Phishing versucht ein Angreifer, in diesem Fall auch Phisher genannt, auf betrügerische Weise vertrauliche oder sensible Anmeldedaten zu bekommen. Um dies zu erreichen fälscht er die elektronische Kommunikation zwischen Opfer und einer vertrauenswürdigen oder öffentlichen Organisation, indem er sich selbst als diese ausgibt. Dies geschieht meist durch E-Mails, welche das Opfer auf eine Webseite locken, welche vermeindlich zur vertrauenswürdigen Organisation gehört, in Wahrheit aber vom Angreifer kontrolliert wird und deshalb Informationen, vorzugsweise Passwörter oder Kreditkartennummern abfängt.<sup>67</sup>

Phishing gibt es seit Anfang der 1990er Jahre, allerdings sind die Zahlen von Phishing-Angriffen in den letzten Jahren drastisch gestiegen. Phishing ist zu einer gefährlichen Kombination aus Social Engineering und technischen Angriffen geworden, welche zum Ziel hat vertrauliche Informationen zu erlangen. Die gewonnenen Daten werden für Betrug, Identitätsdiebstahl und Spionage missbraucht.<sup>68</sup>

Im Folgenden wird ein beispielhafter Phishingangriff auf PayPal geschildert. PayPal ist ein Online-Bezahldienst mit über 18 Millionen Nutzern alleine in Deutschland.<sup>69</sup> Am häufigsten wird PayPal genutzt um Internetkäufe zu bezahlen. Listing 5 zeigt eine E-Mail, mit der ein PayPal-Nutzer auf eine Phishing-Seite gelockt werden soll.<sup>70</sup>

Sehr geehrter PayPal-Kunde, sehr geehrte PayPal-Kundin,

wir haben gerade einen oder mehrere Loginversuche von einer fremden IP-Adresse auf Ihr PayPal-Konto festgestellt.

Wenn Sie in der letzten Zeit unterwegs auf Ihren Account zugegriffen haben, könnten die ungewöhnlichen Loginversuche von Ihnen stammen. Auch wenn die Loginversuche nicht von Ihnen stammen, besuchen Sie PayPal bitte sobald wie möglich um Ihre Identität zu verifizieren:

[https://www.paypal.com/signin?country.x=DE&locale.x=de\\_DE](https://www.paypal.com/signin?country.x=DE&locale.x=de_DE)

Die Bestätigung Ihrer Identität ist eine Sicherheitsmaßnahme, mit der sichergestellt wird, dass Sie die einzige Person sind, die Zugriff auf Ihr Konto hat.

Vielen Dank für Ihre Unterstützung um gemeinsam Ihr Konto zu schützen.

Mit freundlichen Grüßen,  
PayPal

-----

<sup>67</sup> Vgl. Jakobsson/ Myers (2006), S. 1

<sup>68</sup> Vgl. ebenda, S. 1 f.

<sup>69</sup> Vgl. PayPal (2017)

<sup>70</sup> Vgl. Jakobsson/ Myers (2006), S. 10

SCHÜTZEN SIE IHR PASSWORT

Geben Sie ihr Passwort niemals an Dritte weiter und nutzen Sie es ausschließlich um sich auf <https://www.paypal.com/> anzumelden. Schützen Sie sich vor Betrug, indem Sie einen neuen Browser öffnen und jedes mal die PayPal Url eintippen um sich anzumelden.

-----

Bitte antworten Sie nicht auf diese E-Mail. Nachrichten, die an diese Adresse gesendet werden können nicht beantwortet werden. Wenn Sie Hilfe benötigen melden Sie sich in Ihrem PayPal-Konto an und klicken Sie auf den Hilfe-Link im Menü.

PayPal E-Mail ID PP321

### Listing 5: Phishing Lockmail<sup>71</sup>

Die in Listing 5 dargestellte E-Mail täuscht dem Kontoinhaber vor, dass eine Fremde Person auf das Konto zugegriffen hat und animiert ihn so dazu dem vermeintlich sichern Link zu folgen um seine Identität zu verifizieren um seinen Account zu schützen. Nebenbei sei noch erwähnt, dass der Link in der E-Mail natürlich nicht auf die originale PayPal-Webseite verweist, sondern auf die Phishing-Seite des Angreifers. Der Hinweis „Schützen Sie Ihr Passwort“ verleiht der E-Mail noch ein authentischeres Aussehen und würde der Nutzer dem Rat folgen wäre dieser Phishing-Angriff wirkungslos. Viele Nutzer nehmen diesen Rat auch wahr, nutzen aber trotzdem den bereitgestellten Link aus der E-Mail, da diese ja offensichtlich von PayPal stammt und deshalb vertrauenswürdig ist.<sup>72</sup>

Üblicher Weise wird auch die Absenderadresse der E-Mail gefälscht und eine originaladresse von PayPal, beispielsweise *service@paypal.com* verwendet. Wenn der Empfänger der E-Mail nun den Link aus selbiger öffnet wird er auf die in Abbildung 7 dargestellte Webseite geleitet, welche ihn zur eingabe seiner Anmeldedaten auffordert.<sup>73</sup>

<sup>71</sup> Zitat Jakobsson/ Myers (2006), S. 11 Abbildung 1.4, Übersetzung Samuel Philipp

<sup>72</sup> Vgl. ebenda, S. 10

<sup>73</sup> Vgl. ebenda, S. 10

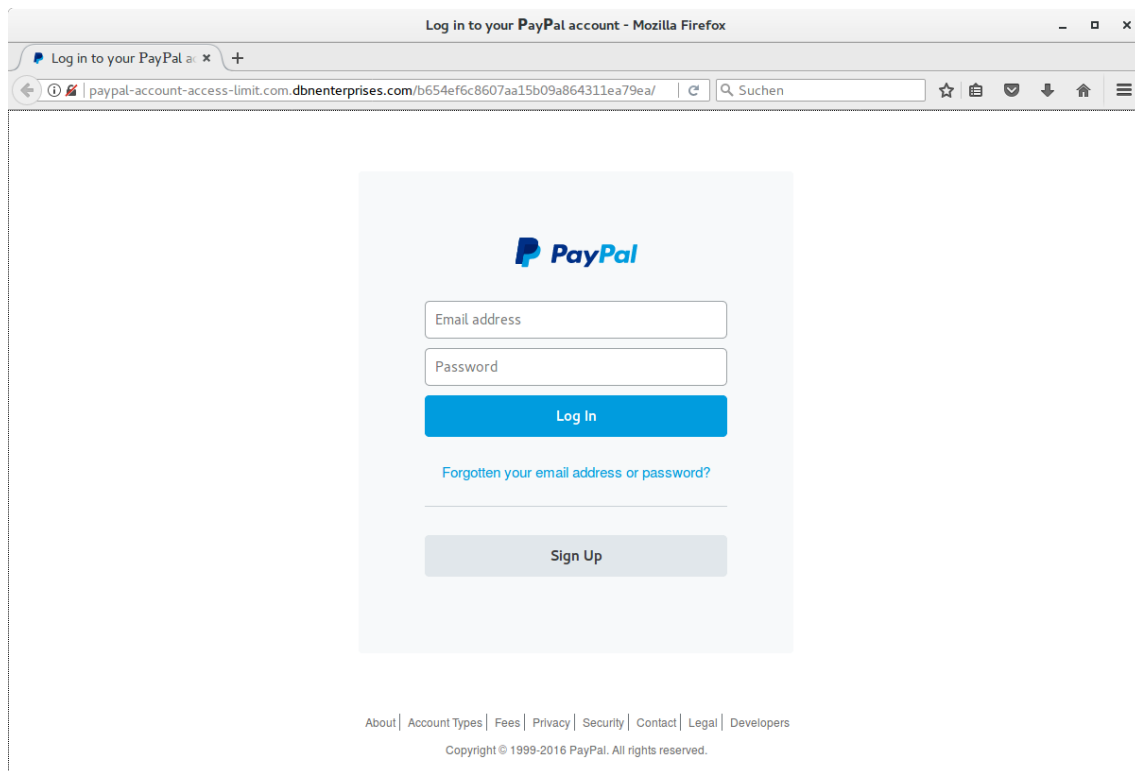


Abbildung 7: PayPal Phishing Webseite

Das Aussehen der Phishing-Webseite ist dem des Originals (Abbildung 8) sehr ähnlich. Wenn das Opfer nun seine Benutzernamen und sein Passwort eingegeben hat, ist das erste Ziel des Angreifers bereits erreicht, denn er hat gültige Zugangsdaten zu einem PayPal-Account erhalten. Um aber noch mehr Daten zu bekommen und dem Opfer den Angriff weiterhin zu verschleiern, wird der Nutzer in vielen Fällen auf einer nachfolgenden Seite gebeten, auch noch seine Anschrift und Kreditkartendaten zu bestätigen, indem er diese auch noch eingeben muss. Danach wird der Nutzer wieder „abgemeldet“ und anschließend auf die originale PayPal-Webseite (Abbildung 8) weitergeleitet. Damit ist der Phishing-Angriff abgeschlossen und der Angreifer wird keine Zeit verlieren, die Daten zu missbrauchen.<sup>74</sup>

<sup>74</sup> Vgl. Jakobsson/ Myers (2006), S. 10 ff.



Abbildung 8: PayPal Original Webseite

Das Vorgehen im vorausgegangenen Beispiel ist sehr typisch für Phishing-Angriffe und kann deshalb auf sehr viele andere Seiten übertragen werden.

## 3 Konzept

In diesem Kapitel werden das Gesamtkonzept und die Konzepte der einzelnen Tests vorgestellt. Das Gesamtkonzept umfasst die Einzelnen Komponenten von *webifier* und deren Zusammenspiel. Im Folgenden wird nun das Gesamtkonzept beschrieben.

### 3.1 Gesamtkonzept

*webifier* ist in viele Unterkomponenten aufgeteilt. Dies hat den Zweck, bestehende Programme und zukünftige Erweiterungen möglichst getrennt voneinander entwickeln zu können und einzelne Features ab- oder zuzuschalten. In diesem Abschnitt wird zuerst die Entwurfsarchitektur der Gesamtanwendung erklärt. Dabei wird hauptsächlich auf den Zweck der Komponenten für die Anwendung und auf deren Zusammenspiel eingegangen.



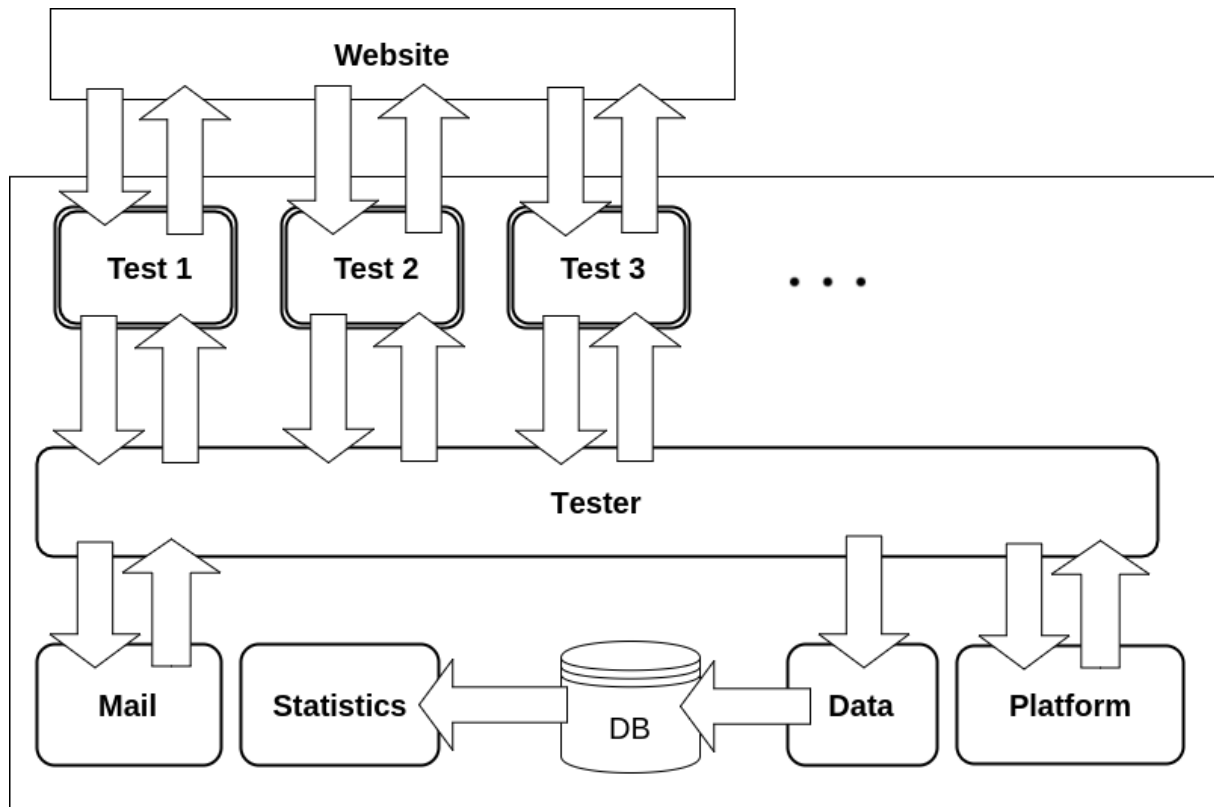


Abbildung 9: Konzeptarchitektur der Gesamtanwendung

Grundsätzlich wird webifier benutzt, um Webseiten auf verschiedene Sicherheitsaspekte zu untersuchen. Abbildung 9 zeigt den groben Plan der Anwendung. Es zeigt den konzeptionellen Zusammenhang zwischen den Komponenten, wobei die Datenbank und der Mail Server nicht als eigene Komponenten entwickelt wurden und deshalb keine eigenen Unterabschnitte erhalten. Die Aufnahme der URLs erfolgt über eine Webseite, die von der Plattform an den Browser des Nutzers ausgeliefert wird. Die Plattform leitet die Uniform Ressource Locator (URL) aus der Nutzeranfrage an dann an den Tester weiter. Der Tester ist die Komponente, die die URL auf Korrektheit überprüft und ggf. Weiterleitungen auflöst. Entsteht dabei ein Fehler, so wird dieser an die Plattform propagiert und schließlich im Browser angezeigt. Wurde die Auflösung der URL erfolgreich ausgeführt, werden die Sicherheitstests gestartet.

Eine weitere Möglichkeit, webifier zu verwenden, ist über Mailverkehr. Hat der Nutzer eine Spam-verdächtige E-Mail in seinem Postfach, so kann er diese an ein bestimmtes Postfach von webifier weiterleiten. Dort werden Links aus der Mail zusammengetragen und an den Tester weitergeleitet.

Im Tester wird die Ausführung der einzelnen Tests gestartet, überwacht und koordiniert. Wichtig dabei ist, dass alle Tests abgeschottet vom Rest des Systems laufen, da auf ihnen schadhafte Webseiten aufgerufen werden. Es können beliebig viele Tests im Tester hinzugefügt werden, indem sie in der Konfiguration angegeben werden. Jeder dieser Tests gibt ein eigenes Ergebnis, sowie weitere Informationen zurück. Diese Ergebnisse werden dann miteinander verrechnet und es wird eine Gesamteinschätzung zusammen mit allen relevanten Informationen der einzelnen Tests ausgegeben.

Alle Ergebnisse der Tests werden anhand von Schwellwerten und Gewichtungen ausgerechnet, wobei die Anwendung zwischen vier Ergebnistypen unterscheidet: *CLEAN*, *SUSPICIOUS*, *MALICIOUS* und *UNDEFINED*.

**unbedenklich**

CLEAN

Die Webseite verhält sich harmlos und ist ungefährlich.



Abbildung 10: Icon für den Testergebnistyp *CLEAN*

**verdächtig**

SUSPICIOUS

Die Webseite verhält sich bedenklich und ist suspekt. Diese Seite sollte gemieden werden.



Abbildung 11: Icon für den Testergebnistyp *SUSPICIOUS*

**bedrohlich**

MALICIOUS

Die Webseite verhält sich kritisch und ist gefährlich. Sie sollte unter keinen Umständen aufgerufen werden.



Abbildung 12: Icon für den Testergebnistyp *MALICIOUS*

**unbekannt**

UNDEFINED

Der Test konnte aus technischen oder zeitlichen Gründen nicht vollendet werden. Es kann keine Aussage über die Bös- oder Gutartigkeit der Seite gemacht werden.



Abbildung 13: Icon für den Testergebnistyp *UNDEFINED*

Das berechnete Gesamtergebnis des Testers wird an die webifier Platform bzw. an webifier Mail weitergeleitet. Schließlich wird das Ergebnis dem Nutzer über den Browser bzw. über eine Antwortmail präsentiert. Zusätzlich werden alle Ergebnisse des Testers an webifier Data geschickt und dort in einer Datenbank persistiert. Diese Daten werden von webifier Statistics für Analysen und Auswertungen genutzt

### 3.1.1 webifier Tests

Webifier Tests ist der Oberbegriff für sämtliche von webifier durchgeführten Tests bei der Analyse einer Webseite. Wie bei der gesamten Anwendung wird auch bei den Tests viel Wert auf Modularität gelegt. Jeder Test bildet ein eigenständiges Bauteil, welches nach belieben integriert oder entfernt werden kann ohne Effekte auf die Lauffähigkeit der Gesamtanwendung.

Da webifier auf die Analyse von maliziösen Seiten ausgelegt ist gibt es bei den Tests einige Punkte zu beachten um das System vor Viren und Schadcode zu schützen. Jeder Test wird in einer vom Gesamtsystem abgekapselten Laufzeitumgebung ausgeführt. Aus einem Test heraus darf nicht auf das System zugegriffen werden, da die Tests gegebenenfalls mit Schadcode befallen werden können durch das Erforschen von maliziösen Seiten. Es soll vermieden werden, dass sich Schadcode oder Viren von den Tests auf den Server verbreiten. Nach Durchlauf und Übermittlung des Ergebnisses löscht der Test sich selbst und alle Laufzeitdaten. Als Ergebnis werden keinerlei Dateien versendet, es beschränkt sich auf eine Weitergabe des Ergebnisses in Form einer Zeichenkette. Damit soll vermieden werden, dass sich eventuell mit Viren befallene Dateien weiter auf dem System ausbreiten können.

Ein Test liefert sein Ergebnis an den Tester, welcher dies dann im folgenden weiterverarbeitet. Das Starten und Organisieren der Tests wird von webifier Tester durchgeführt. Den Aufbau und die Funktionsweise des Testers wird im nächsten Abschnitt beschrieben. Zum Starten eines Tests werden zwei Startparameter mitgegeben. Zum Einen wird die ID zur Identifikation mitgegeben und zum anderen die vollständig aufgelöste URL der zu testenden Seite.

Webifier stellt 9 verschiedene Tests um eine Webseite zu überprüfen. Das Konzept der einzelnen Test wird in jeweils eigenständigen Kapiteln erläutert. Hier folgt noch ein Überblick über die einzelnen Tests.

| Test                                | Beschreibung   |
|-------------------------------------|--|
| Virensan der Webseite               | Testet die Dateien einer Seite auf Viren                           |
| Vergleich in verschiedenen Browsern | Test ob sich die Seite bei verschiedenen Browsern anders verhält   |
| Überprüfung der Port-Nutzung        | Überprüft ob die Seite Portscanning betreibt                       |
| Überprüfung der IP-Nutzung          | Überprüft ob die Seite IPScanning betreibt                         |
| Prüfung aller verlinkten Seiten     | Testet die Links auf der Webseite gegen die Datenbank von webifier |
| Google Safe Browsing                | Nutzt die Google-API um die Webseite von Google testen zu lassen   |
| Überprüfung des SSL-Zertifikats     | Überprüft das SSL-Zertifikat der Webseite                          |
| Erkennung von Phishing              | Testet ob es sich um eine Phishingseite handelt                    |
| Screenshot der Seite                | Gibt dem Nutzer einen Screenshot der Webseite                      |

Tabelle 1: Beschreibung der einzelnen Tests

### 3.1.2 webifier Tester

Der webifier Tester verwaltet alle Tests, führt diese aus und berechnet aus den einzelnen Ergebnissen der Tests ein Gesamtergebnis. Alle auszuführenden Tests werden in einer Konfigurationsdatei angegeben und können deshalb dynamisch angepasst werden. Da jeder Test in einem eigenen Prozess läuft wird beim Starten des Testers die Konfigurationsdatei geladen. Anschließend werden die einzelnen Tests ausgeführt und auf ein Ergebnis gewartet. Liegt von allen Tests das Ergebnis vor wird ein Gesamtergebnis berechnet. Die Berechnung dieses Ergebnisses wird im Folgenden genauer erklärt.

Das Ergebnis kann entweder unbedenklich (*CLEAN*), verdächtig (*SUSPICIOUS*), bedrohlich (*MALICIOUS*) oder unbekannt (*UNDEFINED*) sein. Für die Berechnung des Endergebnisses erhält jeder Test wie in Tabelle 2 dargestellt eine Gewichtung, da einige Tests mehr über die Vertrauenswürdigkeit oder Gefahr einer Webseite aussagen als andere. Am meisten fallen der *Virensan der Webseite* und die *Erkennung von Phishing* ins Gewicht, da dieses die ausschlaggebendsten Tests sind. Am wenigsten gewichtet sind der *Vergleich in verschiedenen Browsern*, weil dies nur ein Indiz ist, da es auch viele Webseiten, wie die von YouTube, Nachrichtensendern, Blogs oder Ähnlichem gibt, welche immer dynamischen Inhalt bereitstellen und die *Prüfung der verlinkten Seiten*, da dies

immer vom Datenbestand abhängt. Der *Screenshot der Seite* fällt nicht ins Gewicht, da dies kein Test im eigentlichen Sinne ist, sondern nur eine zusätzliche Information für den Nutzer darstellt. In Abschnitt 3.2 wird die Wahl der Gewichtungen für die einzelnen Tests noch ausführlicher erläutert.

| Test                                | Gewichtung | Prozentuale Gewichtung |
|-------------------------------------|------------|------------------------|
| Virensan der Webseite               | 5          | ~0,208                 |
| Vergleich in verschiedenen Browsern | 1          | ~0,042                 |
| Überprüfung der Port-Nutzung        | 3          | 0,125                  |
| Überprüfung der IP-Nutzung          | 3          | 0,125                  |
| Prüfung aller verlinkten Seiten     | 1          | ~0,042                 |
| Google Safe Browsing                | 3          | 0,125                  |
| Überprüfung des SSL-Zertifikats     | 3          | 0,125                  |
| Erkennung von Phishing              | 5          | ~0,208                 |
| Screenshot der Seite                | 0          | 0                      |

Tabelle 2: Gewichtungen der einzelnen Tests

Die prozentuale Gewichtung ergibt sich aus  $\frac{\text{Testgewichtung}}{\text{Summe der Gewichungen aller Tests}}$ .

Ein weiterer wichtiger Punkt, der für die Berechnung des Gesamtergebnisses festgelegt wurde ist, dass mindestens 50% aller Tests (berechnet anhand der prozentualen Gewichtung) ein bekanntes Ergebnis, also *CLEAN*, *SUSPICIOUS* oder *MALICIOUS* haben müssen. Ist der Anteil bekannter Ergebnisse kleiner lässt sich kein zuverlässiges Ergebnis berechnen, da dieses sonst von zu wenigen ausschlaggebenden Faktoren abhängen würde.

Liefern also mehr als die Hälfte der Tests ein bekanntes Ergebnis, so kann daraus nun das Gesamtergebnis berechnet werden. Hierfür wird für jedes Testergebnis ein Wert zwischen 0 und 1 berechnet, welcher anschließend mit der prozentualen Gewichtung des Tests multipliziert wird. Die Werte der einzelnen Testergebnisse ergeben sich wie in Tabelle 3 dargestellt. Ist das Testergebnis *CLEAN* oder *UNDEFINED* ist der Ergebniswert 0 und geht so nicht weiter in die Wertung ein. Ist das Ergebnis *MALICIOUS* wird der Wert 1. Dadurch fällt das Gewicht dieses Tests voll in die Wertung. Ist das Ergebnis *SUSPICIOUS* so wird die prozentuale Gewichtung des Tests als Ergebniswert gewählt. So fließt dieser Test mit dem Quadrat der Gewichtung in das Gesamtergebnis ein.

| Testergebnis      | Ergebniswert                     |
|-------------------|----------------------------------|
| <i>CLEAN</i>      | 0                                |
| <i>SUSPICIOUS</i> | Prozentuale Gewichtung des Tests |
| <i>MALICIOUS</i>  | 1                                |
| <i>UNDEFINED</i>  | 0                                |

Tabelle 3: Zuordnung Testergebnis zu Ergebniswert

Anschließend werden die Werte aller Tests zu einem Endergebnis aufsummiert. Daraus ergibt sich im Gesamtergebnis ein Minimalwert von 0 und ein Maximalwert von 1. Dieser Wertebereich wird nun wie folgt auf die drei Ergebnisse *CLEAN*, *SUSPICIOUS* und *MALICIOUS* verteilt. Die Tests mit der größten Gewichtung sollen hierbei ausschlaggebend sein. Daraus ergibt sich die prozentuale Gewichtung des Tests mit der größten Gewichtung als Minimalwert für *MALICIOUS* und das Quadrat der prozentualen Gewichtung des Tests mit der größten Gewichtung als Minimalwert für *SUSPICIOUS*.

Daraus lässt sich für die Werte der Tests aus Tabelle 2 die folgende Werteverteilung der Gesamtergebniswerte ableiten:

$$0 \leq \textit{CLEAN} < 0,043402\bar{7} \leq \textit{SUSPICIOUS} < 0,208\bar{3} \leq \textit{MALICIOUS} \leq 1$$

Zusätzlich zu dem berechneten Gesamtergebnis stellt der Tester auch alle Ergebnisse der Einzeltests und deren spezifischen Testinformationen bereit. Außerdem werden alle Ergebnisse zu Persistierung an das Modul webifier Data gesendet, welches in Abschnitt 3.1.5 genauer erläutert wird.

### 3.1.3 webifier Plattform

webifier Plattform ist eine Webanwendung, die auf den Tester aufsetzt und eine UI für diesen zu Verfügung stellt. Außerdem bereitet sie die Ergebnisse des Testers grafisch für den Benutzer auf.

Der Nutzer hat die Möglichkeit auf der ersten Seite von webifier Plattform eine Url einzugeben, welche getestet werden soll. Da die Kapazität jedes Systems beschränkt ist verwaltet die Plattform alle Anfragen zur Webseitenüberprüfung in einer Warteschlange. In einer Konfigurationsdatei kan angegeben werden, wie viele Tests parallel ausgeführt werden sollen. So wird die Warteschlange nach und nach abgearbeit und anschließend werden die Ergebnisse des Testers für den Benutzer visuell aufbereitet. Das bereitgestellte Ergebnis umfasst zum Einen das Gesamtergebnis, welches vom Tester berechnet wurde und zu Anderen sowohl die Ergebnisse der einzelnen Tests, als auch die Zusätzlichen Informationen, welche von diesen bereitgestellt wurden.

So erhält der Nutzer einen umfassenden Bericht über die Vertrauenswürdigkeit oder die ausgehende Gefahr der überprüften Webseite.

### 3.1.4 webifier Mail

Diese Komponente ist ein Zusatzfeature, mit dem verdächtige Mails über den webifier Tester überprüft werden können. Nutzer müssen dazu lediglich ihre empfangene Mail an für *webifier Mail* bereitgestelltes Postfach weiterleiten. Eine Beispielhafte E-Mail wird im Screenshot in Abbildung 14 dargestellt. In *webifier Mail* werden dann bis zu fünf in der Mail enthaltene Links an den Tester weitergeleitet. Ist die Prüfung vollendet wird eine Antwortmail mit den Ergebnissen an den Nutzer zurückgeschickt.



Abbildung 14: Screenshot einer Spam-Mail



### 3.1.5 webifier Data

webifier Data ist die Persistenzkomponente von webifier. webifier Tester nutzt webifier Data um alle Testergebnisse an einem zentralen Ort abzulegen, egal wo dieser ausgeführt wird.

Ein Testergebnis, welches in dem Datamodul gespeichert wird enthält einmal die eingegebene Url und die getestete Url, das Gesamtergebnis, sowohl den Typ (*CLEAN*, *SUSPICIOUS*, *MALICIOUS* oder *UNDEFINED*), als auch den Ergebniswert. Außerdem wird die Testlaufzeit gespeichert. Zusätzlich werden noch weitere Informationen zu den einzelnen Tests gespeichert. Dazu zählen der Name, die Konfigurationsparameter, wie beispielsweise die Gewichtung, das Resultat und die Detailinformationen zu diesem.

Die Komponente stellt auch eine Schnittstelle zum Abfragen der bereits gespeicherten Ergebnisse bereit. Diese wird beispielsweise vom Test *Prüfung aller verlinkten Seiten* verwendet. Die Funktionsweise dieses Tests wird in Abschnitt 3.2.5 erklärt.

Alle Ergebnisse werden in einer Datenbank abgelegt. Da die zusätzlichen Informationen der einzelnen Tests teilweise sehr unterschiedlich sind, kommt hierfür keine relationale Datenbank in Frage. webifier Data nutzt deshalb zur Speicherung aller Daten die Dokument basierte Datenbank MongoDB. Alle weiteren Informationen hierzu folgen im Umsetzungsteil dieser Arbeit.

### 3.1.6 webifier Statistics

Webifier Statistics ist die Statistikoberfläche von webifier. Hier werden alle Daten der analysierten Webseiten aufbereitet und in visueller Form dargestellt. Die Daten stammen aus den Ergebnissen aller Tests, welche von webifier Data abgespeichert wurden.

Webifier Statistics liefert dem Nutzer eine Vielzahl an verschiedenen Graphen, welche bestimmte Teilaspekte beleuchten. Diese enthalten zum Einen die Gesamtauswertungen, welche sich mit der allgemeinen Datenauswertung jedes Gesamttests beschäftigen. Zum Anderen gibt es noch die Einzelauswertungen der Tests, die testspezifische Ergebnisse auswerten.

Alle Auswertungen werden dem Nutzer über eine Weboberfläche zugänglich gemacht. Als Einstieg gibt es ein *Dashboard* mit einigen Zahlen und Fakten zu den Aktivitäten auf webifier. Auf die einzelnen Auswertungen wird in der Auswertung genauer eingegangen.

## 3.2 Testarten

In diesem Abschnitt werden nun die einzelnen Tests vorgestellt, mit welchen die zu überprüfende Webseite analysiert wird. Wie bereits erwähnt werden alle dieser Tests vom Tester verwaltet und ausgeführt.

### 3.2.1 Virensan der Webseite

Der Virensan der Webseite führt nutzt verschiedene Virensanner um die Webseite auf Malware zu überprüfen. Um dies zu realisieren wird zunächst die Webseite inklusive aller enthaltenen Dateien und Links heruntergeladen und gespeichert. Anschließend werden die Virensanner gestartet, welche die heruntergeladenen Dateien überprüfen. Abschließend werden alle Ergebnisse der einzelnen Scans zusammengeführt und daraus ein Endergebnis berechnet.

Für das Endergebnis werden zunächst alle gescannten Dateien klassifiziert. Wird eine Datei von keinem der Virensanner als Malware eingestuft wird diese als *CLEAN* gekennzeichnet. Klassifiziert nur ein Virensanner die Datei als Malware, wird diese also *SUSPICIOUS* eingestuft. Halten mehr als ein Virensanner eine Datei für Malware ist diese *MALICIOUS*. Sind alle Dateien als *CLEAN* eingestuft, so ist auch das Endergebnis dieses Tests *CLEAN*. Sollten ein oder mehrer Dateien *SUSPICIOUS* sein wird auch das Endergebnis *SUSPICIOUS*. Das selbe gilt danach für *MALICIOUS*. Sobald eine Datei *MALICIOUS* ist, ist das Endergebnis ebenfalls *MALICIOUS*.

Zusätzlich zum Endergebnis wird noch die gesamte Liste der gescannten Dateien inklusive der jeweiligen Klassifizierung bereitgestellt und vom Tester weitergegeben.

### 3.2.2 Vergleich in verschiedenen Browsern

In Unterabschnitt 2.4.2 wurde bereits das Thema *User Agent Sniffing* behandelt. Dieser Test soll genau dieses Problem aufgreifen. Es sollen Hypertext Transfer Protocol (HTTP)-Anfragen von verschiedenen Browsertypen mit entsprechenden User Agents an den Webserver der Zielseite geschickt werden. Danach werden die Antworten untereinander auf Unterschiede untersucht. Dabei entstehen Kennwerte für die Übereinstimmung der Antworten, die miteinander verrechnet werden müssen. Dieser berechnete Wert für die Durchschnittsübereinstimmung muss schließlich anhand von passenden Schwellwerten die richtige Ergebnisklasse eingeteilt werden.

### 3.2.3 Überprüfung der Port-Nutzung

Der Test auf Port Scanning analysiert die Nutzung der Ports einer Webseite. Hierfür wird die Webseite automatisch vom Test geöffnet und dessen JavaScript ausgeführt. Parallel dazu muss die Netzwerkaktivität überwacht werden. Es werden alle eingehenden Anfragen auf das Testsystem zunächst geloggt. Da das Testsystem abgekapselt vom restlichen System ist, ist es irrelevant von welcher IP-Adresse die Anfragen kommen. Alle Anfragen lassen sich auf die aufgerufene Seite zurückführen, da restliche Netzwerkaktivität abgeschaltet ist. Dies ist wichtig, da es durchaus möglich ist, dass die Webseite nicht selbst einen Portscan-Angriff startet sondern beispielsweise über einen Drittrechner oder ein Botnetz gescannt wird. Zudem könnten die Ports auch lokal auf dem Client über JavaScript gescannt werden. Deshalb werden lediglich die angefragten Ports im Log gespeichert.

Nach erfolgreichem Durchschauen der Webseite beginnt die Analyse. Hier müssen alle Portanfragen klassifiziert werden. Es gibt eine Reihe von legitimen Portanfragen, welche beispielsweise Port 80 für HTTP oder Port 443 für SSL sind. Diese Anfragen werden dann als harmlos markiert und somit ignoriert. Alle Anfragen, welche sich auf unspezifizierte Ports beziehen, werden als verdächtig markiert. Je nach Anzahl der verdächtigen Anfragen wird dann entschieden, ob die Seite als bedrohlich, verdächtig oder sauber klassifiziert wird. Dieses Ergebnis wird dann mitsamt der gefundenen verdächtigen Portanfragen zurückgegeben.

### 3.2.4 Überprüfung der IP-Nutzung

Der Test auf IP Scanning beschäftigt sich mit der Analyse der IP-Anfragen, welche durch eine Webseite ausgelöst werden. Wie auch bereits bei Portscanning beschrieben wird die Webseite automatisch geöffnet und dessen JavaScript ausgeführt. Die Netzwerküberwachung hat hier jedoch einen anderen Fokus. Es werden die IPs der gesendeten Anfragen geloggt. Beim IP Scanning wird grundsätzlich versucht über die bekannten Heimnetz-IP-Netze weitere im Netzwerk angeschlossene Geräte zu erkennen um beispielsweise Viren auf dem gesamten Netzwerk zu verbreiten. Diese Angriffe werden über JavaScript auf dem Clienten gestartet. Deshalb werden die vom Clienten gesendeten Anfragen protokolliert. Hiervon werden lediglich die IPs gespeichert.

Nach dem Speichern aller IPs werden diese klassifiziert. Der Test vergleicht alle Anfragen mit den bekannten Heimnetz-IPs (beispielsweise 192.168.178.\*). Anfragen, welche sich nicht auf diese Adressen zurückführen lassen werden herausgefiltert, da diese irrelevant für den Test sind. Anhand der Anzahl der verdächtigen Adressen wird im Abschluss wieder die Seite klassifiziert und das Ergebnis mitsamt den Adressen zurückgeliefert an den Tester.

### 3.2.5 Prüfung aller verlinkten Seiten

Dieser Test sucht innerhalb des Quellcodes der Anwendung nach Hyperlinks und referenzierten Ressourcen. Für alle gefundenen Hosts wird dann geprüft, ob bereits Einträge dazu in der Datenbank unter webifier Data vorhanden sind. Gibt es einen Eintrag, der *SUSPICIOUS* ist, dann ist das ganze Endergebnis *SUSPICIOUS*. Falls jedoch mehr als 34% der Hosts *UNDEFINED* sind, dann ist auch das Gesamtergebnis *UNDEFINED*. Sobald ein Eintrag *MALICIOUS* ist, so ist auch das Gesamtergebnis *MALICIOUS*. Ansonsten liefert der Test *CLEAN* zurück.

### 3.2.6 Google Safe Browsing

Safe Browsing is a Google service that lets client applications check URLs against Google's constantly updated lists of unsafe web resources.<sup>75</sup>

---

<sup>75</sup> Vgl. Google Developers (2017)

Dieser Cloud-Service erlaubt es der Anwendung also eine weitaus größere Datenbank als ihre eigene anzuzapfen.

Der Service unterscheidet zwischen vier Bedrohungsstypen: Malwareseiten, Sozialmanipulationsseiten, ungewollte Software und potenziell schadhafte Webanwendungen. Weitere Vorteile sind die Autorität des Unternehmens und die einfach zu bewerkstellende Integration des Services. Leider können die Einträge aus der Datenbank des Services nicht exportiert werden, sondern können nur gegen Listen von Links checked werden. In diesem Test werden wie bei der Prüfung der verlinkten Seiten alle Links zu Webseiten und Ressourcen zusammengetragen. Diese werden dann aber nicht an webifier Data, sondern an den Google Safe Browsing Dienst weitergeleitet. Sobald ein Treffer aus dem Service zurückgemeldet wird, ist das Ergebnis *SUSPICIOUS*. Falls sich mehr als 40% der Links in der Datenbank befinden, so wird die Webseite als *MALICIOUS* eingestuft. Ist die zu testende Webseite selbst ein Match, so ist das Ergebnis ebenfalls *MALICIOUS*. Wird kein Eintrag gefunden, dann zählt die Seite als *CLEAN*.

### 3.2.7 Überprüfung des SSL-Zertifikats

Die Überprüfung des SSL-Zertifikats der Webseite sucht nach einem vorhandenen Zertifikat und validiert dieses, sofern die Webseite eines nutzt. Hierfür liest es die dafür notwendigen Informationen des Zertifikats aus und berechnet anschließend ein Testergebnis.

Stellt die Webseite kein Zertifikat zur Verfügung so ist das Testergebnis *SUSPICIOUS*, da es in Zeiten von Let's Encrypt<sup>76</sup> jedem möglich ist ein SSL-Zertifikat kostenlos zu erwerben und so die Sicherheit der eigenen Webseite zu erhöhen. Nutzt die Webseite ein valides Zertifikat ist das Ergebnis *CLEAN*. Weist das Zertifikat Fehler auf ist das Ergebnis *MALICIOUS*. Solche Fehler können beispielsweise sein, dass das Zertifikat abgelaufen ist, dass es selbst signiert wurde oder dass es für den falschen Host genutzt wird.

---

<sup>76</sup> <https://letsencrypt.org/>

### 3.2.8 Erkennung von Phishing

webifier enthält auch einen sehr einfachen Test zur Erkennung von Phishing. Dieser sucht zuerst nach den Schlagwörtern der gegebenen Webseite. Hierfür zählt er die Häufigkeit aller vorkommenden Wörter die mehr als drei Buchstaben haben. Wörter in Bildbeschreibungen und Überschriften werden doppelt gewichtet, Wörter im Titel der Webseite werden fünffach gewichtet. Haben mehrere Wörter die gleiche Gewichtung, so fällt die Länge der Wörter auch noch ins Gewicht und längere Wörter werden bevorzugt. Die vier Wörter, die im Ranking am höchsten stehen werden anschließend als Schlüsselwörter gewählt.

Anschließend werden mit Hilfe öffentlicher Suchmaschinen mögliche Duplikate der Webseite gesucht. Für diese Suche werden die ausgewählten Schlagwörter verwendet. Nun werden die Ergebnisse aller Suchmaschinen zusammengeführt und ebenfalls gewichtet. Je mehr Suchmaschinen einen Link gefunden haben, desto höher steigt dieser Link im Ranking. Als nächstes muss diese Liste der möglichen Duplikate nach Originalen, welcher der gegebenen Webseite entsprechen gefiltert werden, da es sehr wahrscheinlich ist, dass diese ebenfalls in der Liste der Links enthalten ist. Als letztes wird die Liste noch auf maximal zehn Einträge gekürzt.

Nun werden alle gefundenen Links mit der gegebenen Webseite verglichen und für jeden gefundenen Link ein Ergebnis berechnet. Der Vergleich erfolgt auf drei Ebenen: es werden der Inhalt und der Quelltext der beiden Webseiten, aber auch das Aussehen verglichen. Jeder dieser Vergleiche gibt die prozentuale Übereinstimmung der zu vergleichenden Webseiten zurück. Anschließend werden diese drei Ergebnisse zu einem Gesamtergebnis verrechnet. In diese Rechnung fließt das Ergebnis des Screenshotsvergleichs mit doppeltem Gewicht ein, da dieser Vergleich am aussagekräftigsten ist.

Aufgrund der Komplexität wird die genaue Berechnung des Ergebnisses in Abschnitt 4.2.8 erklärt und hier nun vereinfacht dargestellt. Stimmen die beiden Webseiten zu 80% überein, so ist das Ergebnis des Links *SUSPICIOUS*, stimmen die beiden Webseiten zu 90% überein, so ist das Ergebnis *MALICIOUS*. Das Endergebnis des Tests wird abschließend wie folgt berechnet: wurde mindestens ein verdächtiger Link gefunden, so ist das Gesamtergebnis *SUSPICIOUS*, wurde mindestens ein bedrohlicher Link gefunden, so ist das Ergebnis *MALICIOUS*, andernfalls *CLEAN*. Zusätzlich wer-

den noch die gefundenen Schlagwörter, die gefundenen Phishingseiten, deren Vergleichswerte und ein Überlagerungsscreenshot mit der Originalseite für den Tester bereitgestellt.

### **3.2.9 Screenshot der Seite**

Der Screenshot-Test ist kein Test im eigentlichen Sinne. Er liefert keine Aussage über die Bedrohlichkeit einer Webseite. Deshalb liefert er immer als Ergebnis sauber und bleibt ungewichtet in der Gewichtung im Tester. Trotzdem wurde er implementiert um den Nutzern einen Blick auf die Seite zu geben, welche sie von webifier haben scannen lassen. Dies kann besonders interessant sein, da viele Nutzer auch daran interessiert sind wie die Seiten denn aussehen und was dort an Text oder Bilder zu sehen ist. Jedoch sollte keiner der Nutzer, auf eine als bedrohlich markierte Webseite, mit seinem Webbrowser zugreifen. Deshalb wird hier die Möglichkeit gegeben sich gefahrlos einmal die Webseite anzuschauen.

## 4 Umsetzung

In diesem Kapitel wird nun aufbauend auf dem im vorherigen Kapitel beschriebenen Konzept die Umsetzung von *webifier* beschrieben. Zunächst folgt nun die Erläuterung der Gesamtumsetzung, gefolgt von der Umsetzung der Teilanwendungen. Abschließend wird die Implementierung der einzelnen Tests vorgestellt.

### 4.1 Gesamtanwendung

Dieser Abschnitt beschreibt wie die Anwendung implementiert ist und welche Technologien wie eingesetzt wurden. Zunächst findet ein grober Umriss über die Gesamtarchitektur statt, der vor allem die Zusammenhänge und Abhängigkeiten zwischen den einzelnen Komponenten darlegt. In den nachfolgenden Unterabschnitten wird die Umsetzung der einzelnen getrennten Komponenten für sich erklärt.

Abbildung 15 zeigt alle umgesetzten Komponenten mit ihrer jeweils wichtigsten Technologie. Zusätzlich sind zwischen zusammenspielenden Komponenten die jeweils verwendeten Kommunikationstechnologien (hauptsächlich Protokolle) angegeben.

Der Standard Use Case beginnt, wenn der Nutzer eine verdächtige URL findet und diese prüfen möchte. Dazu öffnet er mit seinem Browser die Adresse von *webifier*: `www.webifier.de`. Daraufhin schickt der Browser eine HTTP-GET Anforderung an *webifier Plattform*, den Webserver der Anwendung. Dort wird die Anfrage angenommen und durch das Spring Web Framework verarbeitet. Dabei wird ein neuer *Tester*-Prozess mit der zu prüfenden URL über Command Line Interface (CLI) gestartet. Abbildung 15 zeigt dies im rechten unteren Bereich.

Ein weiterer Use Case ist die Prüfung von verdächtigen E-Mails. Empfängt der Nutzer eine Spam-verdächtige E-Mail, so kann er diese auch über *webifier* prüfen lassen. Dazu leitet er die E-Mail über sein E-Mail-Programm an die Adresse weiter. Der Client schickt



die E-Mail dann über Simple Mail Transfer Protocol (SMTP) zum Mail-Server des Nutzers, der diese dann wieder über SMTP an den Mail-Server von *webifier* übermittelt. Sobald eine Mail an diesem Server ankommt, wird diese über Internet Mail Access Protocol (IMAP) an das E-Mail-Verwaltungsmodul *webifier Mail* weitergegeben. Diese Komponente durchsucht die E-Mail nach Links und reicht bis zu fünf URLs zur Prüfung an *webifier Tester* weiter. Dazu wird wie bei der *webifier Plattform* pro URL ein Prüfdurchlauf beim Tester über CLI angefordert. Dieser Vorgang ist in Abbildung 15 auf der linken Seite zusehen.

Der *webifier Tester* ist ein eigenständiges Java-Programm, das für die Durchführung der Sicherheitstests zuständig ist. Es ist deshalb in der Grafik zwischen den Tests und dem Rest des System abgebildet. Der Aufruf des Testers erfolgt über CLI, wobei er als Parameter die zu überprüfende URL erwartet. Im ersten Schritt wird die URL validiert und ggf. über etwaige HTTP-Weiterleitungen aufgelöst. Entsteht dabei ein Fehler, so gibt der Tester ihn an seinen Aufrufer zurück und beendet sich.

War die Vorprüfung erfolgreich, so werden alle ihm bekannten Tests über CLI mit der URL als Parameter gestartet. Alle Tests laufen in einem *Docker*-Container, um die Systemsicherheit zu gewährleisten. Sobald ein Test fertig ist gibt er sein Ergebnis und die Zusatzinformationen über die Standardausgabe aus. Der Tester nimmt dies zur Kenntnis und propagiert dieses Teilergebnis an den Aufrufer weiter. Sobald alle Tests ihr Ergebnis über zurückgemeldet haben, berechnet der *Tester* das Endergebnis und liefert dieses an seinen Aufrufer zurück.

Ist der Aufrufer die *webifier Plattform*, so wird das Ergebnis jedes fertigen Einzeltests per direkt an den Browser des Nutzers geschickt. So werden nach und nach alle Ergebnisse angezeigt, bis schließlich das Gesamtergebnis präsentiert wird. Liefert der Tester einen Fehler, so wird dieser ebenfalls an den Nutzer weitergeleitet. Wenn der Tester durch *webifier Mail* gestartet wurde, dann wartet das Modul, bis die Ergebnisse aller URLs feststehen und sendet dann eine E-Mail mit allen Ergebnissen an den Nutzer zurück. Dies geschieht wie entgegengesetzt zum Weg der Anfrage über IMAP und SMTP.

Zusätzlich zur Standardausgabe der Ergebnisse an den jeweiligen Aufrufer, schreibt der *webifier Tester* seine Ergebnisse in die zentrale Persistenzkomponente *webifier Data*. Diese lässt sich über eine REST-Schnittstelle ansprechen. Über diese Schnittstelle speist der Tester seine Ergebnisse in die *mongoDB*-Datenbank.

Das Modul *webifier Statistics* ist zuständig für alle Auswertungen, die auf den Ergebnissen der Anwendung gemacht werden. Diese Analysen werden mit *R* spezifiziert. Die Komponente greift direkt auf die Datenbank zu und führt regelmäßig Analyseprozesse durch. Dabei entstehen HTML-Berichte, die von Interessenten unter `statistics.webifier.de` abgerufen werden können. Dieser Use Case wird in Abbildung 15 in der Mitte gezeigt.

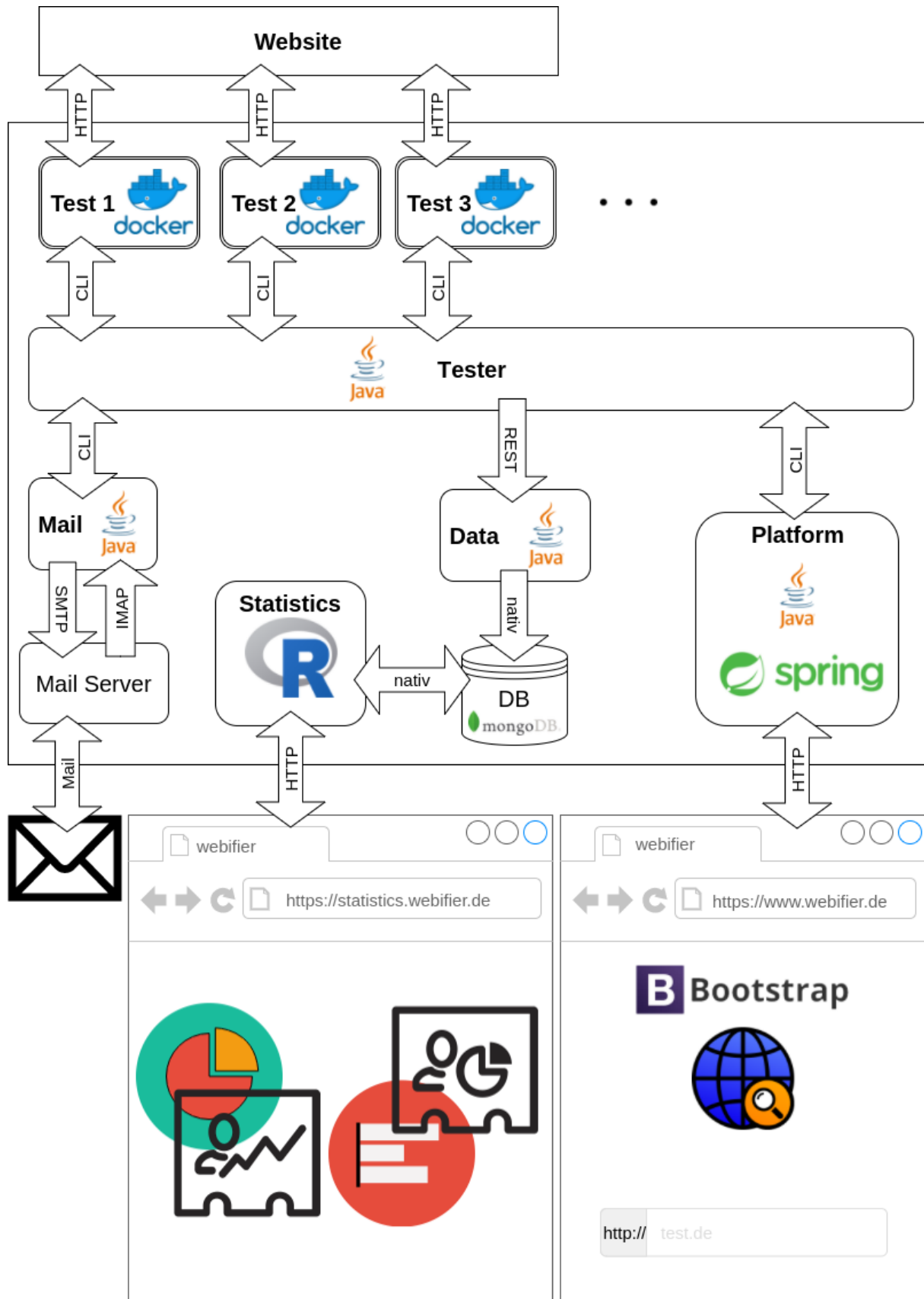


Abbildung 15: Umgesetzte Architektur der Gesamtanwendung

### 4.1.1 webifier Tests

In diesem Abschnitt wird der allgemeine Aufbau, welcher für alle Tests von *webifier* gilt, erläutert.

Um die Tests vom Gesamtsystem abzukapseln wird auf Docker gesetzt. Hierbei wird für jeden Test ein eigenes Image geschrieben. Die Tests werden vom Tester dann gestartet. So wird jeder Test in einem eigenen Container ausgeführt. So ist sichergestellt, dass die Tests unabhängig von äußeren Faktoren sind und sich gegenseitig oder das Gesamtsystem nicht beeinflussen.

Die Technologien der einzelnen Tests sind abhängig vom jeweiligen Test und werden deshalb in den jeweiligen Kapiteln erläutert. Die Ergebnisübermittlung der Tests an den Tester wird mittels JavaScript Object Notation (JSON)-Strings realisiert. Wie in Listing 6 zu sehen besteht das JSON aus dem Testergebnis und einer ResultInfo. Die ResultInfo variiert von Test zu Test. Hier können für jeden Test weitergehende Informationen übermittelt werden. Für den Test auf Portscanning wird beispielsweise eine Liste von verdächtigen Portanfragen übermittelt.

```
1  {
2      "result": "clean" | "suspicious" | "malicious" | "undefined",
3      "info": {
4          ...
5      }
6  }
```

Listing 6: Result JSON

- Beschreiben der Startparameter URL und ID

### 4.1.2 webifier Tester

Der *webifier Tester* wurde als Anwendung für das CLI in Java implementiert. Der Tester kann mit Hilfer verschiedener Parameter in seinem Verhalten gesteuert werden. Die Option `-h` gibt beispielsweise die in Listing 7 dargestellte Hilfe aus.

```
1  usage: java -jar webifier-tester.jar
2  -h,--help          Print this help screen.
3  -i,--id <ID>       Set the id for this test
4  -o,--output <FORMAT> Set the format of the output. Valid formats are
5                      JSON and XML.
6  -u,--url <URL>     The url that should be tested.
```

Listing 7: *webifier Tester* - Hilfe

Die einzig erforderliche Option ist `-u` mit welcher die zu überprüfende Url angegeben wird. Mit der Option `-i` kann dem Test eine Id gegeben werden. Wird keine Id angegeben generiert der Tester eigenständig eine Id für den gestarteten Test. Mit der Option `-o` kann ein Ausgabeformat spezifiziert werden. Dies ist vorallem für die automatisierte Testausführung, beispielsweise mit *webifier Plattform* relevant. Mögliche Ausgabeformate sind JSON und Extensible Markup Language (XML). Ist ein Ausgabeformat angegeben werden alle Events (Start und Ende der Tests) im jeweiligen Format ausgegeben. Wird kein Format spezifiziert so werden die Ergebnisse wie in Listing 8 dargestellt ausgegeben.

```
1 $ java -jar webifier-tester.jar -u securitysquad.de
2 Resolver started for url securitysquad.de
3 Resolver finished! Result:
4 The resolved url is 'https://www.securitysquad.de/' and it is reachable.
5 Start Tester for url https://www.securitysquad.de/
6 Test 'VirusScan' started!
7 Test 'PhishingDetector' started!
8 Test 'CertificateChecker' started!
9 Test 'Screenshot' started!
10 Test 'IpScan' started!
11 Test 'GoogleSafeBrowsing' started!
12 Test 'LinkChecker' started!
13 Test 'PortScan' started!
14 Test 'HeaderInspection' started!
15 Test 'CertificateChecker' finished! Result:
16 The given url is clean!
17 Test 'HeaderInspection' finished! Result:
18 The given url is clean!
19 Test 'Screenshot' finished! Result:
20 The given url is clean!
21 Test 'GoogleSafeBrowsing' finished! Result:
22 The given url is clean!
23 Test 'LinkChecker' finished! Result:
24 The test result is undefined. Maybe the test returned an error!
25 Test 'IpScan' finished! Result:
26 The given url is clean!
27 Test 'PortScan' finished! Result:
28 The given url is clean!
29 Test 'PhishingDetector' finished! Result:
30 The given url is clean!
31 Test 'VirusScan' finished! Result:
32 The given url is clean!
33 Tester finished for url https://www.securitysquad.de/
34 The url is clean!
```

Listing 8: *webifier Tester* - Standardausgabe

Wie im Konzept bereits erwähnt verwaltet der Tester alle auszuführenden Tests. Um die Tests dynamisch anpassen zu können werden alle notwendigen Parameter in einer Konfigurationsdatei gespeichert. Listing 9 zeigt einen Ausschnitt dieser Datei.

Jeder Test hat einen eindeutigen Namen, seine Gewichtung, ein Befehl zum Ausführen und zum Beenden des Tests, sowie dafür vorgesehene Timeoutzeiten in Sekunden. Außerdem hat jeder Test einen Parameter, welcher die Java-Klasse für das Testergebnis angibt und einen Parameter mit dem der Test aktiviert oder deaktiviert werden kann. Bei der Ausführung und beim Beenden der Tests werden die Platzhalter #ID und #URL durch die generierten, bzw. vom Nutzer angegebenen Daten ersetzt.

```
1 {
2   "resolver": {
3     "name": "resolver",
4     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-resolver",
5     "startup_timeout_seconds": 60,
6     "shutdown": "docker stop #ID",
7     "shutdown_timeout_seconds": 30
8   },
9   "tests": [
10    {
11      "name": "VirusScan",
12      "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-virusscan",
13      "startup_timeout_seconds": 600,
14      "shutdown": "docker stop #ID",
15      "shutdown_timeout_seconds": 30,
16      "result_class": "de.securitysquad.webifier.output.result.virusscan.
17        TestVirusScanResultInfo",
18      "weight": 5,
19      "enabled": true
20    }
21    ...
22  ],
23  "preferences": {
24    "push_result_data": true
25  }
26 }
```

Listing 9: *webifier Tester* - Ausschnitt Konfigurationsdatei<sup>77</sup>

Am Ende der Datei lässt sich noch die Einstellung festlegen, ob das Endergebnis an *webifier Data* gesendet werden soll oder nicht. Am Anfang der Datei lässt sich der so genannte *Resolver* konfigurieren. Dieser prüft vor allen anderen Tests ob die angeforderte Seite überhaupt erreichbar ist und löst wenn nötig Weiterleitungen der Url auf und gibt das Ergebnis an den Tester zurück.

<sup>77</sup> Der vollständige Inhalt der Konfigurationsdatei befindet sich in Anhang B.

Ist die angegebene Url erreichbar wird die vom *Resolver* aufgelöste Url verwendet und alle anderen Tests damit gestartet. Nun wartet der Tester bis alle Ergebnisse der Tests verfügbar sind oder die Angegebenen Timeouts erreicht sind. Im Falle eines Timeouts erhält der Test das Ergebnis *UNDEFINED*. Abschließend wird das Gesamtergebnis für die angegebene Url wie bereits in Abschnitt 3.1.2 beschrieben berechnet. Listing 10 zeigt einen Ausschnitt der Implementierung der Ergebnisberechnung.

```

1 private WebifierOverallTestResult calculateOverallResult() {
2     ...
3     if (undefinedPercentage > MAX_UNDEFINED_TEST_PERCENTAGE) {
4         return new WebifierOverallTestResult(WebifierResultType.UNDEFINED);
5     }
6     double result = 0;
7     for (WebifierTest<TestResult> test : tests) {
8         double testWeight = (double) test.getData().getWeight() / (double) weightSum;
9         result += getTestResultValue(test.getResult().getResultType(), testWeight) *
                testWeight;
10    }
11    if (result >= maliciousMin) {
12        return new WebifierOverallTestResult(WebifierResultType.MALICIOUS, result);
13    }
14    if (result >= suspiciousMin) {
15        return new WebifierOverallTestResult(WebifierResultType.SUSPICIOUS, result);
16    }
17    return new WebifierOverallTestResult(WebifierResultType.CLEAN, result);
18 }

```

Listing 10: *webifier Tester* - Ausschnitt Ergebnisberechnung<sup>78</sup>

Nachdem alle Tests ausgeführt wurden und das Gesamtergebnis zusammengefasst wurde wird dieses über die von *webifier Data* bereitgestellte Schnittstelle dort gespeichert. Die Kommunikation mit *webifier Data* läuft ebenfalls über das JSON-Format. Genaueres hierzu folgt in Abschnitt 4.1.5.

### 4.1.3 webifier Plattform

In diesem Abschnitt wird nun die Umsetzung von *webifier Plattform* beschrieben. Diese Komponente wurde mit Java umgesetzt und basiert auf dem Spring-Framework. Zusätzlich kamen im Frontend die Technologien HTML, CSS und JavaScript, sowie die Bibliotheken Bootstrap und jQuery zu Einsatz. Zunächst wird nun das Backend beschrieben, danach wird die Oberfläche der Plattform vorgestellt.

<sup>78</sup> Der vollständige Inhalt der Ergebnisberechnung befindet sich in Anhang C.

*webifier Plattform* ist eine Webanwendung und bietet eine benutzerfreundliche Oberfläche zur Bedienung von *webifier Tester*. Um die *Plattform* für die optimale Nutzung des Testers zu konfigurieren gibt es die in Listing 11 dargestellte Datei, mit der alle notwendigen Parameter angepasst werden können.

```
1 {  
2   "tester": {  
3     "command": "java -jar webifier-tester.jar -u #URL -i #ID -o JSON",  
4     "timeout": 15,  
5     "parallel": 1  
6   }  
7 }
```

Listing 11: *webifier Plattform* - Konfigurationsdatei

Zunächst muss in der Konfigurationsdatei der Befehl zur Ausführung des Testers angegeben werden. Standardmäßig sollte der Tester im selben Verzeichnis liegen wie die Plattform. Ist dies nicht der Fall, muss der Pfad der Datei entsprechend geändert werden. Außerdem kann ein Timeout für den Tester angegeben werden. Standardmäßig liegt dieses bei 15 Minuten. Der wahrscheinlich wichtigste Parameter zur optimale Nutzung der vorhandenen Ressourcen ist der letzte Parameter. Mit diesem kann angegeben werden wie viele Tests parallel ausgeführt werden sollen. Per default werden alle Tests sequentiell ausgeführt. Wird die Plattform auf einem leistungsstarken System betrieben kann die *anzahl* entsprechend der vorhandenen Ressourcen angepasst werden.

Die Plattform stellt außerdem eine Möglichkeit zur Massenüberprüfung von Webseiten zur Verfügung. Hierfür kann eine Liste von Urls in form eines Texts oder einer Datei angegeben werden. In diesem Modus ist es allerdings nicht möglich alle Ergebnisse direkt zu sehen, da der Vorgang je nach größe der Liste mehrere Tage oder Wochen dauern kann. Deshalb ist dieser Modus eher für langfristige Analysen geeignet.

Im Folgenden wird nun noch einmal der Ablauf einer Überprüfung beschrieben und die Oberfläche von *webifier Plattform* dargestellt. Besucht der Nutzer die Webseite der Plattform sieht er zunächst die in Abbildung 16 gezeigte Startseite. Hier kann der User nun eine beliebige Url in das Eingabefeld tippen und anschließend die Überprüfung starten. Außerdem bietet die Startseite Links zur bereits beschriebenen Batchverarbeitung und zu *webifier Statistics*. Dieses Modul wird in Abschnitt 4.1.6 ausführlich dargestellt wird.



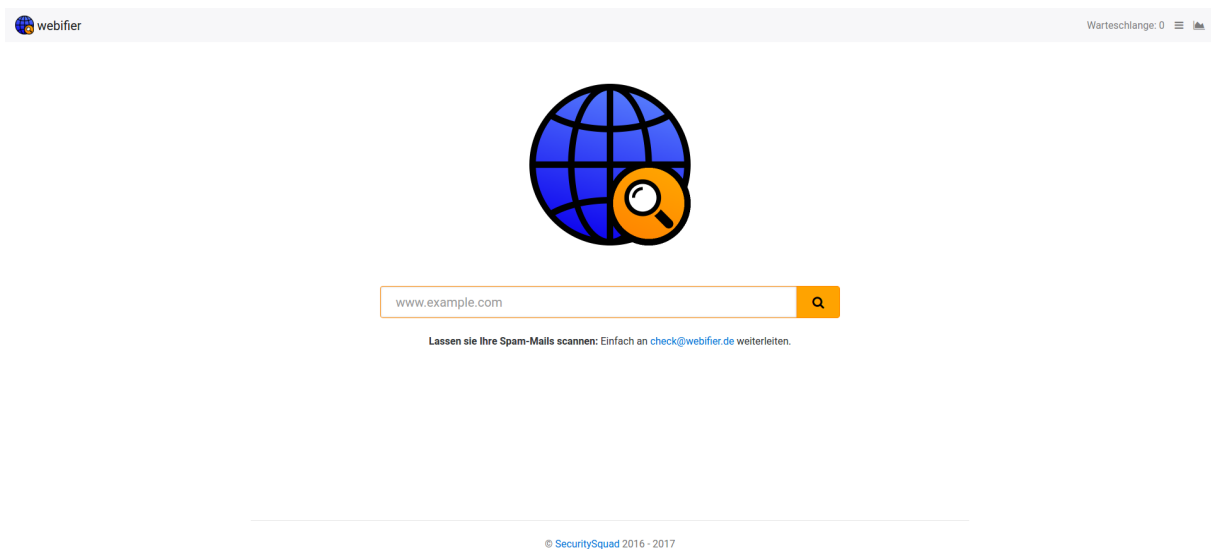


Abbildung 16: *webifier* Plattform - Startseite<sup>79</sup>

Nachdem die Überprüfung einer Webseite gestartet wurde wird der Nutzer auf die in Abbildung 17 abgebildete Seite geleitet, welche sich nach und nach mit den Ergebnissen der einzelnen Tests füllt, sobald diese vorliegen. Sind alle Tests beendet, wird auch das Endresultat angezeigt. Die Ergebnisseite bietet zunächst einen kompakten Überblick über alle ausgeführten Tests und deren Ergebnisse.

<sup>79</sup> Die Abbildung befindet sich in besserer Qualität in Anhang D.

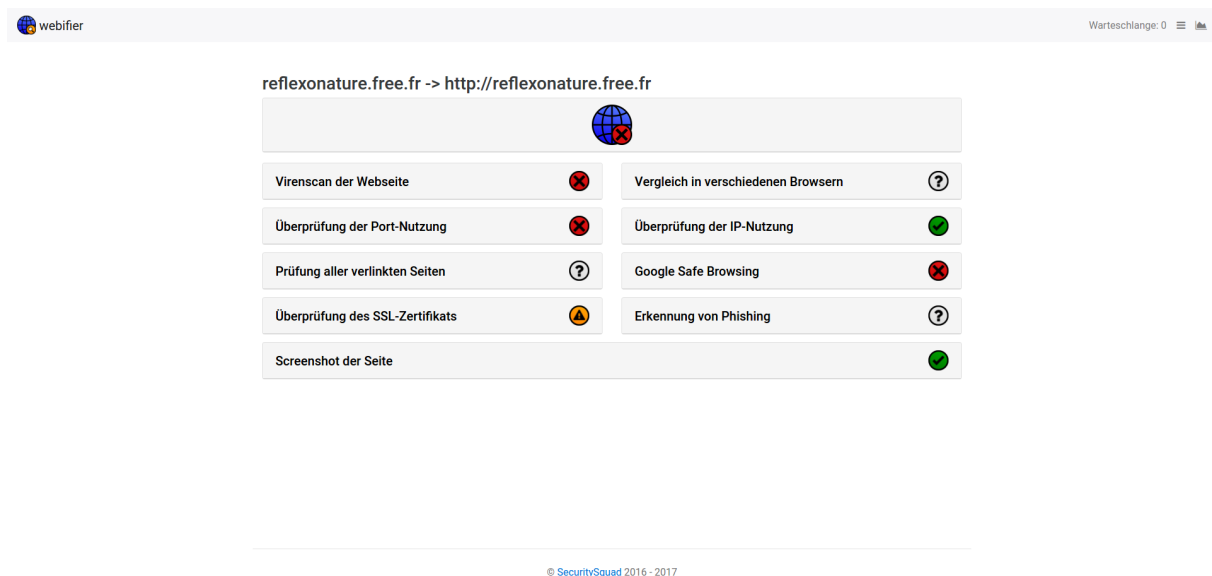


Abbildung 17: *webifier Plattform* - Ergebnisseite<sup>80</sup>

Möchte der Nutzer noch genauere Informationen zu den Ergebnissen eines Tests, so lassen sich alle Testfelder mit einem Klick darauf ausklappen. Im Folgenden werden nun die Detailansichten der einzelnen Testergebnisse gezeigt und erläutert.

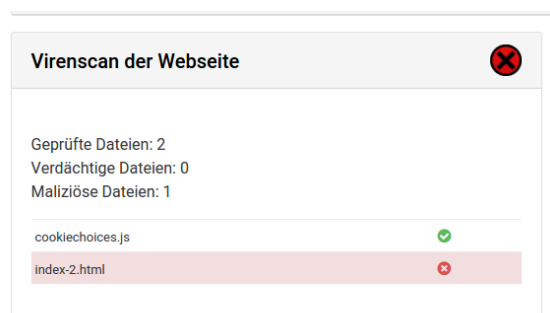


Abbildung 18: *webifier Plattform* - Virensan der Webseite

<sup>80</sup> Diese und weitere Ergebnisseiten befinden sich in besserer Qualität in Anhang D.

Die Detailansicht des Virenskans, welche in Abbildung 18 dargestellt ist, zeigt einmal die Anzahl aller gescannten Dateien, sowie die Anzahlen der gefundenen verdächtigen oder maliziösen Dateien. Außerdem erhält die Ansicht eine genaue Auflistung aller Dateien mit entsprechendem Ergebnis. So lässt sich genau feststellen, welche Dateien welche Bedrohung darstellen.

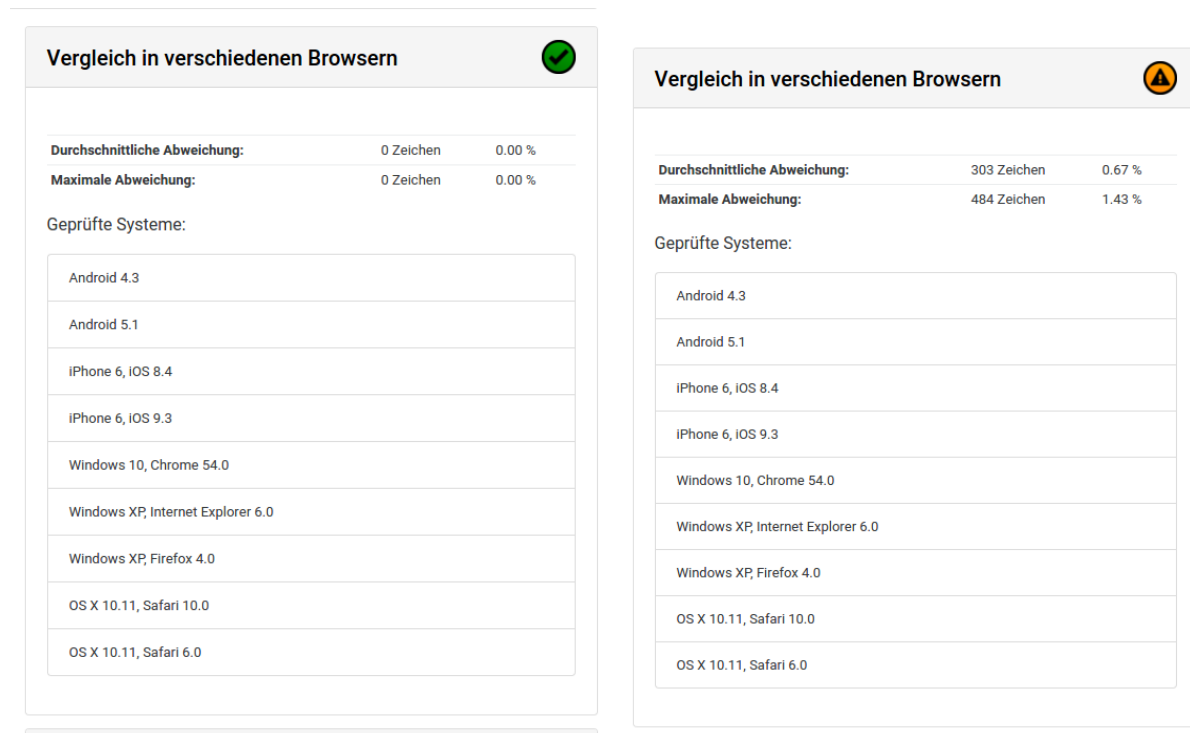


Abbildung 19: *webifier Plattform* - Vergleich in verschiedenen Browsern

Der Vergleich in verschiedenen Browsern zeigt die maximale und die durchschnittliche Abweichung, sowohl als absoluten, als auch als prozentualen Wert. Zusätzlich erhält der Nutzer eine Übersicht über alle Systeme, welche getestet und miteinander verglichen wurden. Zwei Beispielergebnisse hierfür sind in Abbildung 19 zu sehen.

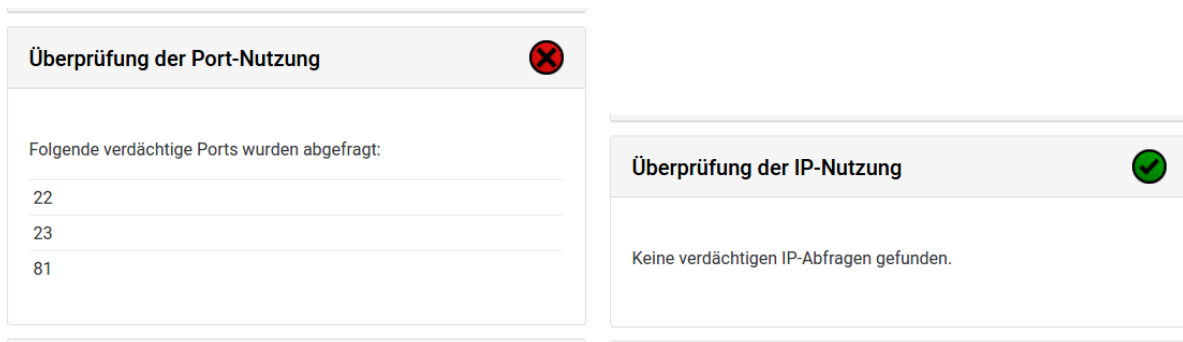
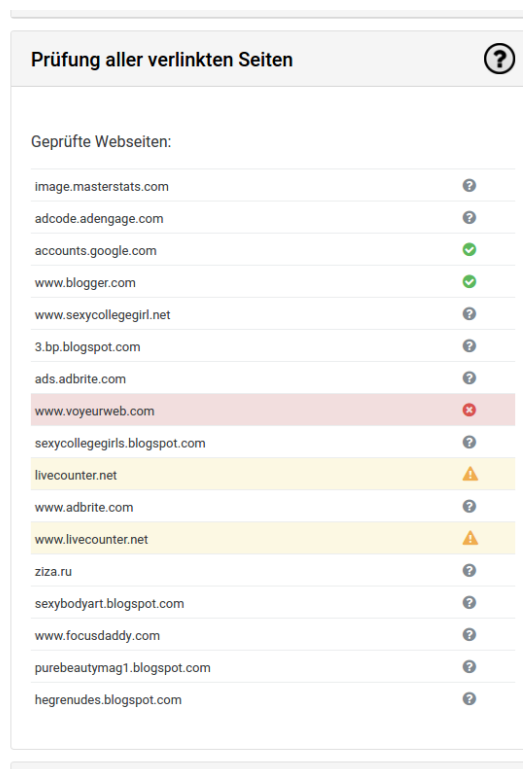
Abbildung 20: *webifier Plattform* - Überprüfung der IP- und Port-Nutzung

Abbildung 20 zeigt die Ergebnisse der Überprüfung der Port-Nutzung, welche im Falle eines verdächtigen oder bedrohlichen Ergebnisses eine Liste mit allen gefundenen Ports enthält. Das Ergebnis der Überprüfung der IP-Nutzung ist gleich aufgebaut und ebenfalls in Abbildung 20 abgebildet. Es stellt bei entsprechenden Funden eine Liste aller IP-Adressen bereit.

Abbildung 21: *webifier Plattform* - Prüfung aller verlinkten Seiten

Das Ergebnis der Prüfung aller verlinkten Seiten enthält eine einfache Liste mit allen gefundenen Links und dem entsprechenden Ergebnis aus *webifier Data*. Abbildung 21 zeigt eine solche Liste.

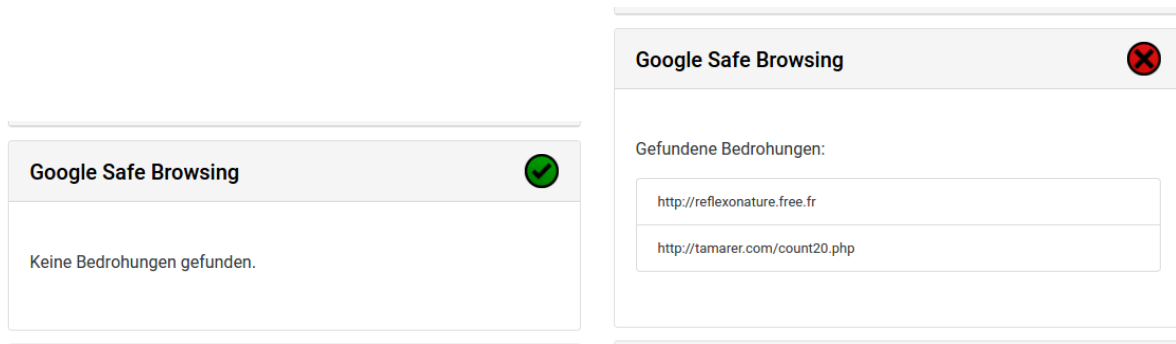


Abbildung 22: *webifier Plattform* - Google Safe Browsing

Die Detailansicht des Google Safe Browsing Ergebnisses ist ähnlich dem der Prüfung aller verlinkten Seiten. Allerdings listet diese nur alle gefundenen Bedrohungen auf und nicht alle geprüften Links. Abbildung 22 enthält zwei Beispielresultate.

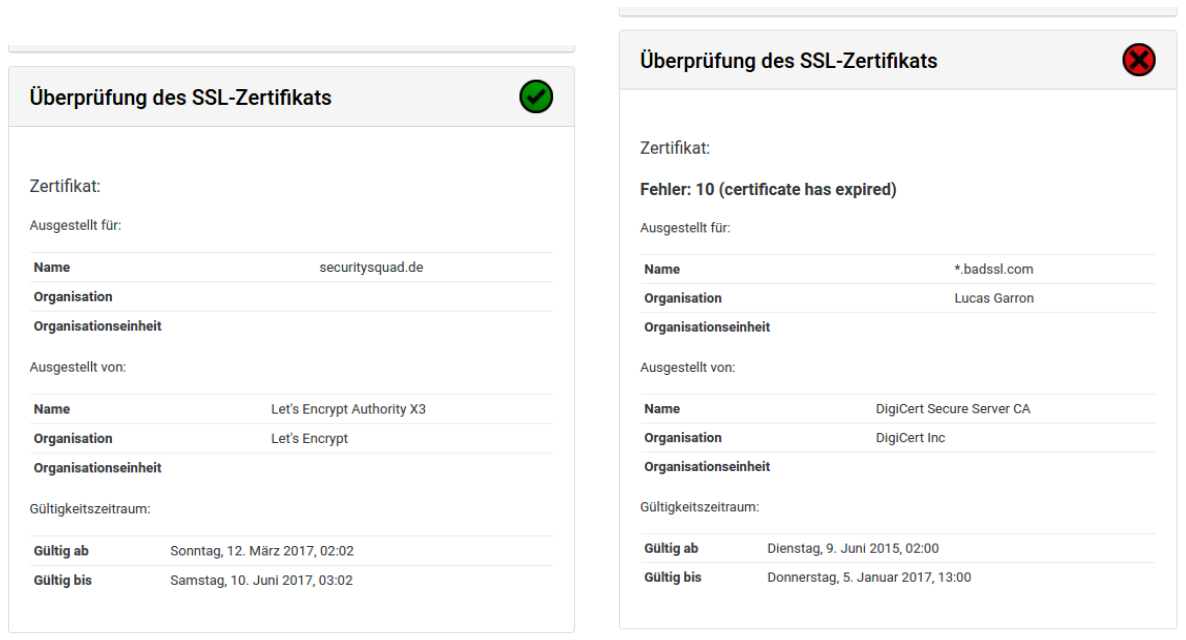


Abbildung 23: *webifier Plattform* - Überprüfung des SSL-Zertifikats

Das Ergebnis der Überprüfung des SSL-Zertifikates enthält alle Informationen des Zertifikats, sofern die Webseite eines nutzt. Wie in Abbildung 23 dargestellt, zeigt die Detailansicht einmal für wen das Zertifikat ausgestellt wurde, aber auch wer es ausgestellt hat. Außerdem wird der Gültigkeitszeitraum des Zertifikats angezeigt und im Fehlerfall der gefundene Fehler.

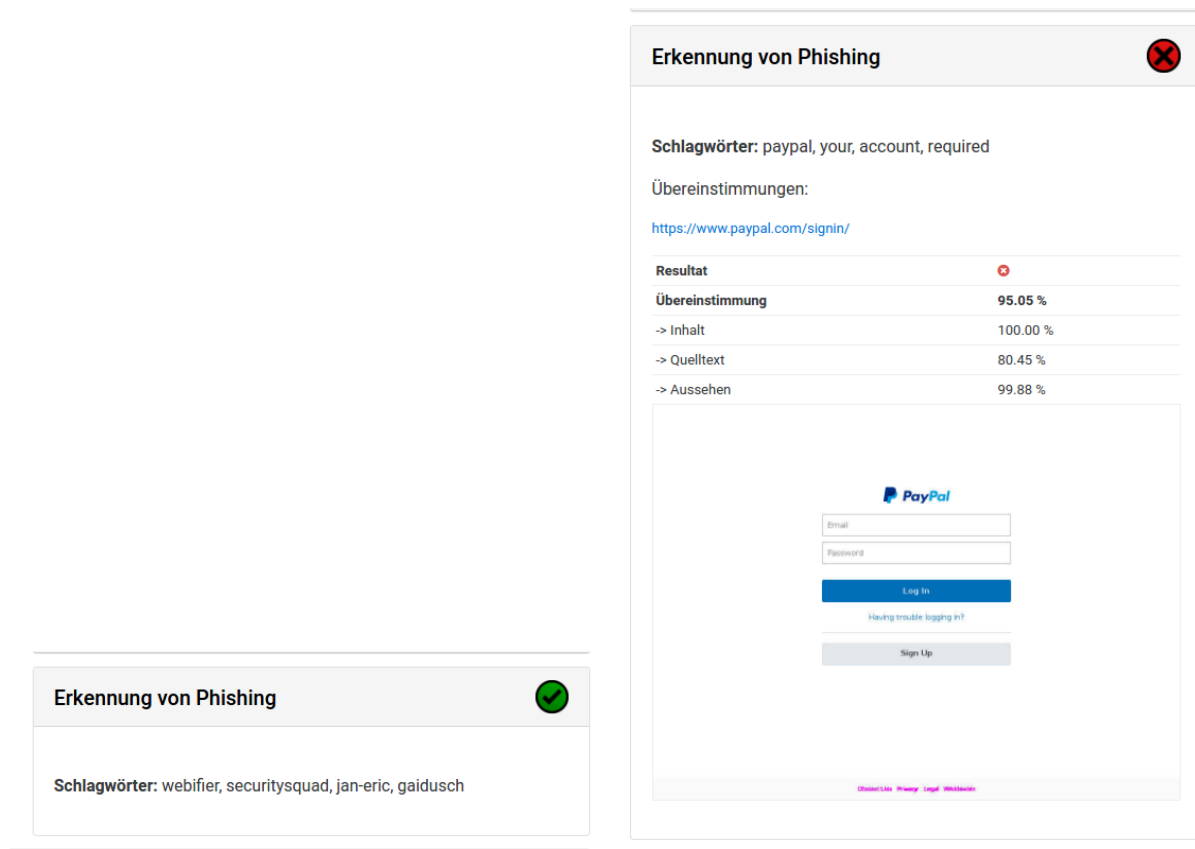


Abbildung 24: *webifier Plattform* - Erkennung von Phishing

Die Erkennung von Phishing stellt ebenfalls einige Informationen zur Verfügung, wie Abbildung 24 zeigt. Es werden in jedem Fall die gefundenen Schlagwörter der Webseite angezeigt. Ist das Ergebnis verdächtig oder bedrohlich, so wird die vermeindliche Originalseite verlinkt, die Werte der prozentualen Übereinstimmungen insgesamt und von Inhalt, Quelltext und Aussehen separat aufgelistet und ein Bild der beiden überlagerten Seiten gezeigt. Alle Inhalte, welche sich unterscheiden werden rosa dargestellt. Im Beispiel aus der Abbildung unterscheidet sich demnach nur der Text der Fußzeile.

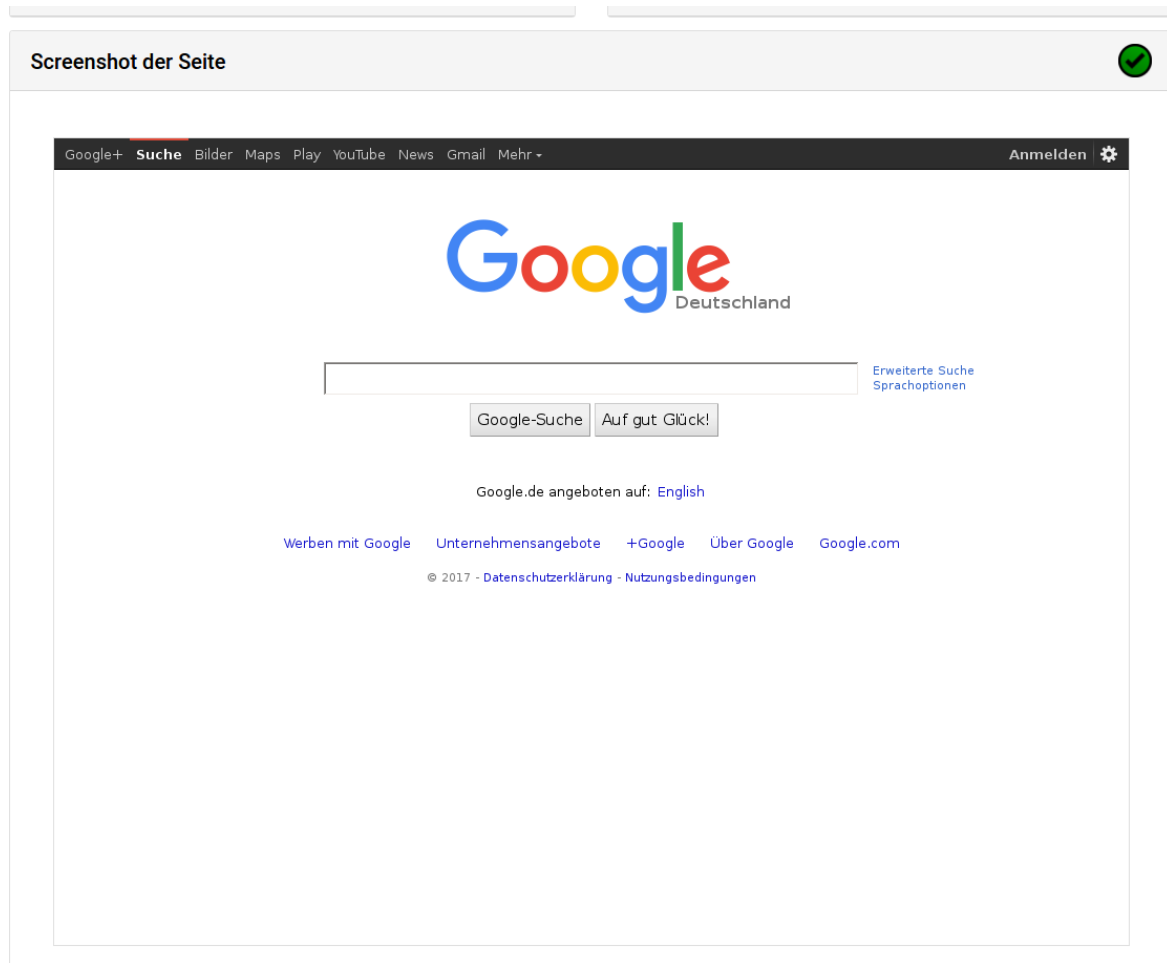


Abbildung 25: *webifier Plattform* - Screenshot der Seite

Der Screenshot der Seite wird beim Ausklappen des Panels einfach angezeigt, so wie es in Abbildung 25 dargestellt. Dies gibt dem Nutzer die Möglichkeit, sich die Webseite anzusehen, ohne sie selbst zu besuchen.

Wie nun gezeigt, bietet die Plattform sehr viele interessante Zusatzinformationen zu den Testergebnissen, mit denen das Gesamtergebnis noch genauer erklärt wird. Außerdem bekommt der Nutzer so genügend Informationen, um die Plausibilität der Testergebnisse selbst noch einmal zu überprüfen.

#### 4.1.4 webifier Mail

Von der Architektur her ähnlich wie *webifier Plattform* ist *webifier Mail* ebenfalls eine Java-Anwendung, die auch auf dem Spring-Framework basiert. Sie ist vergleichbar mit einem E-Mail-Client, welcher via IMAP seine Mails abfragt. Die eingehenden Nachrichten werden auf Links hinuntersucht, von denen anschließend bis zu fünf vom Tester geprüft werden. Sobald alle Links der E-Mail gefunden wurden und diese der Warteschlange hinzugefügt wurden sendet *webifier Mail* eine erste Antwort mit dem Betreff `EINGEREIHT: Nachrichtenbetreff` zurück, dass der Nutzer ein Feedback bekommt, dass die Verarbeitung seiner Nachricht beginnt. Eine Solche Nachricht ist in Abbildung 26 dargestellt. In dieser sind einmal alle Links enthalten, sowie deren Position in der Warteschlange. So hat der Nutzer einen Anhaltspunkt wie lange die Überprüfung dauern kann.

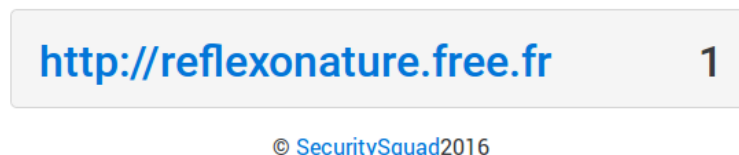


Abbildung 26: *webifier Mail* - Screenshot der Startmail

Sobald alle Links der Nachricht vom Tester analysiert und dessen Ergebnisse in *webifier Mail* angekommen sind wird eine Ergebnismail generiert und an den Nutzer zurückgeschickt. Eine solche ist in Abbildung 27 zu sehen. Das Gesamtergebnis sowie alle Teilergebnisse werden farblich entsprechend des Ergebnisses gekennzeichnet um dem Nutzer dadurch die Vertrauenswürdigkeit der Webseite deutlich zu machen. Häufig werden externe Ressourcen wie Bilder nicht von den E-Mailprogrammen nachgeladen, wesshalb dadurch auch eine Kennzeichnung der Ergebnisse gewährleistet ist. Außerdem wird in dem Betreff der Antwortmail das Endergebnis aller Links der original E-Mail als Prefix verwendet um dem Nutzer vor dem eigentlichen Lesen der Auswertung schonmal eine Rückmeldung geben Zu können. Im Beispiel von Abbildung 27 würde der Betreff `BEDROHLICH: Nachrichtenbetreff` sein.





Abbildung 27: *webifier Mail* - Screenshot der Ergebnismail

Die Konfiguration von *webifier Mail* ist identisch mit der der Plattform, weshalb diese hier nicht noch einmal erklärt wird.

#### 4.1.5 webifier Data

*webifier Data* stellt eine Schnittstelle zur globalen Datenspeicherung der Testresultate zur Verfügung. Die Komponente setzt ebenfalls auf dem Spring-Framework auf und wurde deshalb in Java implementiert. Außerdem wird die Spring-Bibliothek Spring-Data eingesetzt um Java-Objekte auf Dokumente zu übertragen und in einer MongoDB zu persistieren.

Das Modul bietet eine REST-API zur externen Nutzung, beispielsweise im *webifier Tester* oder in der Prüfung aller verlinkten Seiten. Die API stellt die folgenden Aktionen bereit: `/push`, `/check` und `/count`. Mit der Methode `/push` können Testergebnisse in *webifier Data* abgelegt werden. Die entsprechende Resource, welche im Inhalt des POST-Requests gesendet werden muss ist in Listing 12 dargestellt.

```

1  {
2      "id" : "33e76954-dc94-48a9-a816-99ddeb647887",
3      "enteredUrl" : "securitysquad.de",
4      "testedUrl" : "https://www.securitysquad.de/",
5      "result" : {
6          "resultType" : "CLEAN",
7          "resultValue" : 0
8      }
9      "duration" : 46220,
10     "testResults" : [
11         {
12             "testId" : "33e76954-dc94-48a9-a816-99ddeb647887_07bb1b72-fd9f-4286-ade5-2
13                 d71cf47580a",
14             "testData" : {
15                 "name" : "VirusScan",
16                 "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-
17                     virusscan",
18                 "shutdown" : "docker stop #ID",
19                 "enabled" : true,
20                 "weight" : 5,
21                 "startupTimeoutInSeconds" : 600,
22                 "shutdownTimeoutInSeconds" : 30,
23             },
24             "result" : {
25                 "result" : "CLEAN",
26                 "resultInfo" : {
27                     ...
28                 }
29             },
30             "duration" : 31191
31         },
32         ...
33     ]
34 }

```

Listing 12: *webifier Data* - Ausschnitt Inhalt push-Request

Die Methode `/check` bietet die Möglichkeit *webifier Data* nach einer oder mehreren Urls zu durchsuchen. Hierfür muss im Inhalt des POST-Requests die in Listing 13 gezeigte Ressource gesendet werden. Diese enthält eine Liste der zu überprüfenden Links.

```

1  {
2      "urls": [
3          "https://fonts.googleapis.com/css?family=Montserrat:400,700",

```

```

4      "https://fonts.googleapis.com/css?family=Lato:400,700,400italic,700italic",
5      "https://www.google.com/recaptcha/api.js",
6      "https://cdnjs.cloudflare.com/ajax/libs/jquery-easing/1.3/jquery.easing.min.js",
7      "https://www.gstatic.com/recaptcha/api2/r20170503135251/recaptcha__de.js",
8      "https://fonts.gstatic.com/s/lato/v13/v0SdcGFAl2aezM9Vq_aFTQ.ttf",
9      "https://fonts.gstatic.com/s/lato/v13/DvlFBScYlr-FMtZSYIYoYw.ttf",
10     "https://www.gstatic.com/recaptcha/api2/r20170503135251/fallback__ltr.css",
11     "https://fonts.googleapis.com/css?family=Roboto:400,500",
12     "https://www.gstatic.com/recaptcha/api2/logo_48.png",
13     "https://github.com/SecuritySquad",
14     "https://www.webifier.de/",
15     "https://github.com/SecuritySquad/webifier-platform",
16     "http://djbrown.de/",
17     "https://www.facebook.com/danjoebro",
18     "https://github.com/djbrown",
19     "https://plus.google.com/114056971447496286521",
20     "https://github.com/jockelmore",
21     "https://samuel-philipp.de/",
22     "https://github.com/samuel-p",
23     "https://plus.google.com/u/0/+SamuelPd"
24 ]
25 }

```

Listing 13: *webifier Data* - Inhalt check-Request

Wie Listing 14 zeigt gruppiert *webifier Data* zunächst alle Links anhand deren Host-Teil und entfernt alle ungültigen Einträge. Anschließend werden alle Resultate für jeden Eintrag der Liste geladen und daraus ein Gesamtergebnis pro Host berechnet. Dieses wird schließlich mit dem Host als schlüssel zur Liste *hostResults* hinzugefügt. Abschließend wird die erzeugte Liste als Ergebnis zurückgegeben und als Antwort zurückgesendet. Diese Antwort ist in Listing 15 abgebildet.

```

1  @Override
2  public WebifierCheckTestResultsResponse checkTestResultsRequest (
3      WebifierCheckTestResultsRequest request) {
4      List<String> hosts = request.getUrls().stream().map(url -> {
5          try {
6              return filterHost (url);
7          } catch (MalformedURLException e) {
8              return null;
9          }
10     }).distinct().filter(StringUtils::isNotEmpty).collect(toList());
11     Map<String, WebifierTestResultDetails> hostResults = new HashMap<>();
12     hosts.forEach(host -> {
13         List<WebifierTestResultData> data = dataPersistenceService.getTestResultDataByHost (
14             host)
15             .stream().filter(d -> d.getOverallResultType() != WebifierTestResult.UNDEFINED)
16             .collect(toList());
17         if (data.isEmpty()) {
18             hostResults.put(host, WebifierTestResultDetails.UNDEFINED);
19         } else {

```

```

17         double resultValue = data.stream().mapToDouble(this::mapDataResultToIndex).average
18             ().orElse(1);
19         hostResults.put(host, mapResultValueToResult(resultValue));
20     }
21     });
22     if (hostResults.isEmpty()) {
23         return new WebifierCheckTestResultsResponse(false);
24     }
25     return new WebifierCheckTestResultsResponse(true, hostResults);

```

Listing 14: *webifier Data* - Verarbeitung check Methode

```

1  {
2      "success": true,
3      "hosts": {
4          "www.facebook.com": "CLEAN",
5          "fonts.googleapis.com": "UNDEFINED",
6          "plus.google.com": "CLEAN",
7          "fonts.gstatic.com": "UNDEFINED",
8          "cdnjs.cloudflare.com": "UNDEFINED",
9          "github.com": "CLEAN",
10         "djbrown.de": "UNDEFINED",
11         "samuel-philipp.de": "UNDEFINED",
12         "www.google.com": "SUSPICIOUS",
13         "www.gstatic.com": "UNDEFINED",
14         "www.webifier.de": "CLEAN"
15     }
16 }

```

Listing 15: *webifier Data* - Inhalt check-Response

Die Methode `/count` liefert einfach die aktuelle Gesamtzahl der in *webifier Data* vorhandenen Testergebnisse.

Die Testergebnisse werden in der MongoDB auf zwei Dokumente verteilt. Das erste Dokument (`webifierTestResultData`) enthält alle Gesamtdaten des Resultats, das zweite (`webifierSingleTestResultData`) umfasst alle Einzelergebnisse des Resultats. Deshalb werden pro Testergebnis ein Dokument in `webifierTestResultData` und acht in `webifierSingleTestResultData` gespeichert.

Zur Persistierung werden alle Informationen der `/push` Methode (Listing 12) auf die entsprechenden Dokumente gemappt. Listing 16 zeigt die transformierten Daten aus `webifierTestResultData`. Anhang E enthält die transformierten Dokumente aller Einzelergebnisse in `webifierSingleTestResultData`.

```

1 {
2   "_id" : "d62e8c94-21c9-4d69-9337-186b38f73cce",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierTestResultData",
4   "testerId" : "56821bd8-f652-4b59-b7ec-12faa7cf6cae",
5   "enteredUrl" : "ukauctionline.co.uk",
6   "testedUrl" : "http://ukauctionline.co.uk",
7   "host" : "ukauctionline.co.uk",
8   "overallResultType" : "SUSPICIOUS",
9   "overallResultValue" : 0.057291666666666664,
10  "durationInMillis" : NumberLong(227639),
11  "datetime" : ISODate("2017-03-28T22:32:53.834Z"),
12  "testResults" : [
13    DBRef("webifierSingleTestResultData", "5ecb25e6-457b-41ab-9241-a60f8a484131"),
14    DBRef("webifierSingleTestResultData", "abb70569-64d3-47ae-b3e3-ff8086c8f937"),
15    DBRef("webifierSingleTestResultData", "56dbbcb3-9628-4dd4-bb20-692bd86c5410"),
16    DBRef("webifierSingleTestResultData", "64ae09d4-a79e-4194-b77c-3a7891342567"),
17    DBRef("webifierSingleTestResultData", "05ba8488-4b69-40f9-8a9d-661c09daab65"),
18    DBRef("webifierSingleTestResultData", "e0ce8955-feb7-4f7a-adfb-b7b8116a9d28"),
19    DBRef("webifierSingleTestResultData", "9affd5e9-37eb-4d83-b471-ceab0c19155c"),
20    DBRef("webifierSingleTestResultData", "c2b02bf9-2073-451b-a566-295fe1801e51")
21  ]
22 }

```

Listing 16: *webifier Data* - Beispieldokument *webifierTestResultData*

### 4.1.6 webifier Statistics

*webifier Statistics* wird in R implementiert. Hierzu werden Flexdashboards<sup>81</sup> verwendet. Zur Generierung der Grafiken wurde auf verschiedene Librarys, wie beispielsweise Plot.ly, zurückgegriffen um den Entwicklungsaufwand für die Visualisierungen zu minimieren. Die Anordnung der Grafiken wird über ein bestimmtes Layout definiert. Jede Grafik wird prinzipiell in 3 Schritten erstellt:

1. Daten aus der MongoDB laden
2. Daten in die benötigte Form transformieren
3. Entsprechende API ansteuern für Generierung der Grafik

```

1  ### Durchschnittliche Analysezeit
2
3  ```{r}
4  result <- dbGetQueryForKeys(mgl, 'webifierTestResultData', "{}", "{durationInMillis:1}", skip
5    =0, limit=Inf)
6  mean.dur <- mean(result$durationInMillis)/1000

```

<sup>81</sup> <http://rmarkdown.rstudio.com/flexdashboard/index.html>

```
6 mean.dur <- round(mean.dur)
7 tp <- seconds_to_period(mean.dur)
8 valueBox(paste(minute(tp), 'min ', second(tp), 's', sep=""), icon="fa-hourglass-half", color="
  grey")
9 '''
```

Listing 17: *webifier Statistics* - Beispiel R-Grafik

Im Codebeispiel 17 ist der Codeablauf für eine Valuebox zu sehen. Dieses Beispiel wurde ausgewählt um den Erstellungsprozess für die Grafiken zu erklären. Dies lässt sich auf alle anderen Grafiken übertragen.

Die Überschriften der Grafiken werden mit ### markiert. Der R-Code befindet sich in Chunks, diese werden speziell markiert um dem Compiler kenntlich zu machen welches der R-Code ist.

Im Beispiel werden zunächst benötigten Daten aus der MongoDB geladen. Da hier eine Valuebox für die Anzeige der durchschnittlichen Analysezeit generiert wird werden nur die Analysezeiten(*durationInMillis*) benötigt. Diese werden anschließend gemittelt und von Millisekunden in Minuten/Sekunden transformiert. Zur Erstellung der Valuebox muss nun nurnoch der Text, die Farbe und ein passendes Icon ausgewählt werden. Die Generierung und Platzierung übernimmt Flexdashboard. Als Ausgabe wird eine HTML-Datei generiert, welche dann in den Webserver eingebunden wird um sie für die Nutzer zugänglich zu machen.

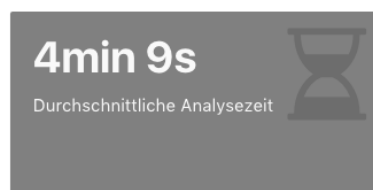


Abbildung 28: Generierte Valuebox

In Abbildung 28 ist die fertig generierte Valuebox mit Überschrift, Text und Icon in passender Farbe dargestellt.

Für stets aktuelle Grafiken wird das R-Skript für die Statistiken mehrfach täglich neu gebaut um die aktuellen Daten mit einzubeziehen. Von einer *On the fly*-Generierung der Grafiken wurde abgesehen, da dies für den Server zu rechenintensiv wäre.

## 4.2 Tests

In diesem Abschnitt werden nun die Implementierungen aller Einzeltests erklärt. Bei allen kam zur Isolierung die Technologier Docker zum Einsatz.

### 4.2.1 Virensan der Webseite

Der Virensan der Webseite wurde in Python umgesetzt und lädt zunächst alle Ressourcen der Webseite in ein lokales Verzeichnis herunter. Hierbei kommt die Software HTTrack zum Einsatz. HTTrack wird für diesen Zweck speziell konfiguriert, um die Antwortzeiten so gering wie möglich zu halten. Zunächst wird der Downloadmodus so gewählt, dass einfach nur alle Dateien gespeichert werden, ohne sie für die dauerhafte Nutzung bereitzuhalten. Des weiteren darf keine Datei im Inhalt verändert werden, da so das Ergebnis verfälscht werden könnte. Um Zeit zu sparen wird zusätzlich auf die Protokollierung verzichtet. Außerdem sollen die `robots.txt` Dateien ignoriert werden, da damit Malware auch vor Suchmaschinen versteckt werden kann. Schließlich wird HTTrack eine maximale Laufzeit zum Download aller Dateien von drei Minuten gewährt.

Zum Scannen aller gespeicherten Dateien kommen drei Virens Scanner zum Einsatz. Es werden die Antivirenprogramme ClamAV<sup>82</sup>, AVG<sup>83</sup> und Comodo Antivirus<sup>84</sup> verwendet. Um auch hier die Antwortzeiten gering zu halten werden die Virens Scanner parallel ausgeführt.

Abschließend werden alle Ergebnisse der Antivirenprogramme ausgewertet und daraus das Gesamtergebnis gebildet. Dieses wird inklusive der Liste aller bewerteten Dateien weitergeleitet.

---

<sup>82</sup> <http://www.clamav.net/>

<sup>83</sup> <http://www.avg.com/de-de/homepage>

<sup>84</sup> <https://antivirus.comodo.com/>

## 4.2.2 Vergleich in verschiedenen Browsern

Dieser Test wird durch zwei Schritte ermöglicht. Zunächst werden für eine Liste von Browserkonfigurationen deren HTTP-Header ermittelt, danach werden Requests mit diesen Headern an die zu testende URL geschickt und die Antworten untereinander verglichen. Damit nicht bei jedem Testdurchlauf alle Header neu eingeholt werden müssen, werden sie in einer Datei abgespeichert.

Zur Umsetzung dieses Tests wurde zunächst das Python-Skript `collect.py` geschrieben, welches über einen Cloud-Service die Werte für das HTTP-Headerfeld *User Agent* abspeichert. Zuerst werden alle gewünschten Browserkonfigurationen über eine JSON-Datei eingelesen (s. Zeile 31). Diese Datei enthält eine Liste von Browserobjekten, die die Konfigurationen im Attribut `Ein Element` dieser Liste ist beispielhaft in Listing 18 zu sehen, wobei das "headers"-Attribut vor Programmablauf noch nicht gesetzt ist.

```
1 [{"configuration": {
2     "platform": "Windows XP",
3     "browserName": "internet explorer",
4     "version": "6.0",
5     "name": "Windows XP, Internet Explorer 6.0"
6 }},
7 "headers": {
8     "Host": "www.reliply.org",
9     "Request Method": "GET",
10    "User Agent": "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; .NET CLR 1.1.432)",
11    "Accept Encoding": "gzip, deflate",
12    "Accept Language": "en-us",
13    "HTTP Connection": "keep-alive"
14 } } ]
```

Listing 18: Beispiel für Browserkonfiguration und gespeicherte Header

Danach werden für jeder dieser Konfigurationen die entsprechenden Header ermittelt. Dazu wird eine Verbindung mittels Secure Shell (SSH) zum Service *Sauce Labs* aufgebaut. Dabei wird auch ein ferngesteuerter Browser mit der Konfiguration des lokalen Browserobjekts initialisiert. Das Browserobjekt ist wie ein Headless Browser über das Modul *WebDriver* steuerbar. Im nächsten Schritt wird der Browser dann angewiesen auf die Webseite `http://www.reliply.org/tools/requestheaders.php` zu gehen. Auf dieser Seite werden alle HTTP-Headerfelder und deren Werte in einer Tabelle aufgelistet. Ein Beispiel dafür ist in Abbildung 29 zu sehen. Diese Tabelle beinhaltet in der ersten Spalte den Headernamen und in der zweiten den Wert. Diese beiden Daten werden in jeder Zeile der Tabelle Der Brower greift auf diese beiden Daten



in jeder Zeile der Tabelle zu und speichert diese im Browserobjekt. Es werden alle Header bis auf den Host-Header abgespeichert, da dieser lediglich der Domain des Zielserver entspricht. Nachdem alle Header zu den Browserkonfigurationen heruntergeladen wurden, wird die Liste der Browserobjekte in die Datei browser.json abgespeichert.

| HTTP Request Headers Sent From Your Browser, Plus Explanation |  |  |
|---|--|--|
| Header Name   | Your Header Value  | Explanation  |
| Accept Language   | en-US,en;q=0.5   | This is the preferred language of the browser. Some websites look at this header and change the language of the page according to the accept language. In many browsers you can change the HTTP Accept Language if you look deep enough in the configuration/settings/options. Look here for <a href="#">How To Change My HTTP Accept Language</a>   |
| Accept Encoding   | gzip, deflate  | This tells the webserver if your browser is able to accept compressed webpages. This can make a big difference in download time and bandwidth usage. For compressed webpages to be used, the browser must accept them <i>and</i> the webserver must send them. Look here for an easy way to <a href="#">Enable Gzip Compression on your Php Webpages</a> (Saves bandwidth and reduces download times). |
| User Agent  | Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:53.0) Gecko/20100101 Firefox/53.0 | This identifies the make and model of your browser. Some websites use this information to indirectly know the capabilities of your browser and change (or tailor) the webpage for your best browsing experience. Some browsers actually allow you to modify the User Agent. For example the Opera browser Allows you to change its identity to Microsoft Internet Explorer or Mozilla.                 |
| HTTP Connection   | <i>Your browser did not send this request header</i>                         | Defines the type of connection your browser wants to establish with the webserver.   |
| Host  | www.reliply.org  | This is the host or domain name that the browser is attempting to connect to.  |
| Request Method  | GET  | This can be GET, POST, PUT or HEAD. GET is the "normal" method for viewing a webpage. POST can be used to send form data and the HEAD request method is for retrieving only the headers of a page instead of downloading the entire page.  |

Abbildung 29: Tabelle der HTTP-Header auf reliply.org

Bei der Testausführung wird dann das Python-Skript check.py aufgerufen. Darin wird zunächst die Liste der Browserobjekte aus der Datei geladen. Danach wird über das Paket *requests* für jeden Browser ein HTTP-Request an die zu testende Seite geschickt. Die Antwort wird wiederum im Browserlement gespeichert, sodass es für die folgenden Vergleiche zur Verfügung steht.

Es werden alle Browser jeweils genau einmal mit allen anderen Browsern verglichen. Das bedeutet, dass das erste Element mit allen weiteren  $n - 1$  Elementen verglichen werden muss. Das zweite mit den letzten  $n - 2$  usw. Das Vorletzte muss dann nur noch mit dem letzten Element verglichen werden, was vom informatischen Aufwand her abschätzbar ist durch den Aufwand der Gaußschen Summenformel ( $\frac{n^2+n}{2}$ ). Dieser Algorithmus hat also eine Komplexität von  $\mathcal{O}(n^2)$  und kann damit zum Engpass werden, falls es zu viele Browser geben sollte.

Ursprünglich basierte die Berechnung der Abweichungen auf Prozentwerten. Während der Implementierung ist aufgefallen, dass die prozentualen Abweichungen der Antworten keine zuverlässige Auskunft über einen möglichen Angriff auf den Nutzer geben können. Aus diesem Grund wird die Ergebniskalkulation nun auf absoluten Abweichungswerten ausgeführt. Jede Browserantwort wird jeweils einmal mit jeder anderen verglichen und der niedrigste absolute Übereinstimmungswert als `worst\_diff` gespeichert. Es wird der schlechteste Übereinstimmungswert für die Ergebnisberechnung gewählt, da er bei einer Durchschnitts- oder Medianberechnung statistisch untergehen würde und somit gezielte Angriffe gegen eine Browsergruppe nicht registriert werden würden. Zur Berechnung dieses Wertes werden zunächst die übereinstimmenden Blöcke der beiden Antworten durch das *statistics*-Paket von Python ermittelt. Durch diese wird die gesamte Übereinstimmungslänge berechnet, die dann von der Gesamtlänge der größeren Antwort abgezogen wird. Zur Ergebnisberechnung wird schließlich überprüft, ob der schlechteste Wert unter einer bestimmten Grenze liegt. Liegt der Wert über 500, dann ist das Ergebnis *MALICIOUS*. Liegt der Wert zwischen 500 und 50, so ist das Ergebnis *SUSPICIOUS*, ansonsten ist es *CLEAN*.

### 4.2.3 Überprüfung der Port-Nutzung

Bei diesem Test wird überprüft ob die Seite versucht einen Portscan auf dem Computer des Anwenders zu betreiben. Hierfür werden 3 Techniken eingesetzt. Die wichtigsten Aufgaben werden von PhantomJS und Bro erledigt. Bro ist ein Netzwerkmonitoring-tool und wird hier genutzt um den Traffic welcher zwischen Webseite und Client entsteht zu protokollieren und in einer Logdatei abzuspeichern. PhantomJS ist ein *headless Browser*, welcher genutzt wird um die Webseite aufzurufen und dessen Javascript auszuführen. Das ganze funktioniert hier ohne grafische Oberfläche.

Der Ablauf des Tests sieht wie folgt aus: Zunächst wird Bro initialisiert und es werden Filter angelegt um lediglich die Ports, der eingehenden Anfragen, mitzuloggen und in der Logdatei abzuspeichern. Ist Bro vollständig initialisiert und einsatzbereit startet PhantomJS mit dem Aufrufen der Seite und Ausführen des JavaScript-Codes. Währenddessen speichert Bro alle Netzwerkaktivitäten. Sobald der Durchlauf von PhantomJS abgeschlossen ist wird mittels Python die Validierung des Ergebnisses gestartet. Hier werden die angefragten Ports aus der Logdatei geladen und klassifiziert. Die Ports 80 und 443 werden verworfen, da diese die HTTP und SSL Ports sind und somit als harmlos klassifiziert werden können. Die weiteren Ports werden in einer Liste

an riskanten Ports gespeichert. Die Anzahl an Ports in dieser Liste bestimmt nun das Ergebnis des Testes. Wurden keine verdächtigen Portanfragen gefunden wird das Ergebnis *unbedenklich* übermittelt. Bei 1 oder 2 Ports in der Liste gibt der Test *verdächtig* als Ergebnis zurück. Sollte die Anzahl größer gleich 3 sein wird die Seite von diesem Test als *bedrohlich* eingestuft. Zusätzlich zum Ergebnis wird die Liste der riskanten Ports in der Ergebnisinformation weitergeleitet.

#### 4.2.4 Überprüfung der IP-Nutzung

Der Test auf verdächtige IP-Anfragen ist bis auf 2 Änderungen identisch zu vorherigem Test auf Portscanning. Deshalb werden in diesem Kapitel nur die Unterschiede beleuchtet.

Der erste Unterschied liegt in der Initialisierung von Bro. Hier werden Filter angewendet um die IPs, der ausgehenden Anfragen, zu loggen. Hier müssen die ausgehenden Anfragen betrachtet werden, da bei dieser Art von Angriff versucht wird mittels clientseitig ausgeführtem JavaScript das Netzwerk des Anwenders auszuspähen. Den Aufruf der Seite übernimmt auch hier PhantomJS. Bei der darauf folgenden Validierung werden die IPs auf bekannte Heimnetzadressbereiche wie beispielsweise 192.168.178.\* oder 192.168.2.\* gemappt. Auch hier werden verdächtige IPs in einer Liste gespeichert. Die Anzahl der Elemente in dieser Liste bestimmt das Ergebnis des Testes. Hierbei sind die Schwellwerte identisch mit denen des Portscanning-Tests, also bei 0 Abfragen wird *sauber* zurückgegeben, bei 1-2 wird *verdächtig* zurückgegeben und bei >3 wird die Seite als *bedrohlich* eingestuft. Zusätzlich zum Ergebnis wird die Liste der riskanten IPs in der Ergebnisinformation weitergeleitet.

#### 4.2.5 Prüfung aller verlinkten Seiten

Einstiegspunkt ist das Python-Skript `check.py`. Ihm wird die zu testende Webseitenurl mitgeliefert. Daraufhin wird ein Unterprozess gestartet, in dem *PhantomJS*(s. Abschnitt 2.3) mit dem Steuerungsskript `check.js` ausgeführt wird. Dieses bewerkstelligt das Sammeln der Links, indem zum einen ein Handler auf das *PhantomJS*-spezifische *ResourceRequested*-Event geschaltet werden. Zum anderen, wird ein JavaScript Code in

die Webseite eingebettet, der alle Anchor-Elemente (`<a></a>`) findet. Das JavaScript Programm gibt schussendlich alle gefundenen Links auf der Konsole aus, wobei zusätzlich auch die Adresse der zu testenden Webseite mit ausgegeben wird.

Zurück im Python-Skript werden die Links über einen POST-Request an die REST-API von *webifier Data* gesendet. Dort werden aus allen Web-Links die Hostnamen extrahiert und jeweils in der Datenbank geschaut, ob ein Eintrag dazu vorhanden ist. Das Antwortobjekt beinhaltet im Attribut `hosts` die Liste der überprüften Hostnamen und deren Wert in der Datenbank. Über diese Liste wird iteriert und die Häufigkeiten der vier Ergebnistypen hochgezählt. Dabei werden auch jeweils der Hostname und der Datenbankwert in ein Informationsobjekt gespeichert. Die Ergebnisberechnung findet analog zu Unterabschnitt 3.2.5 mit mehreren statt. Dieses Ergebnis wird zusammen mit einer Liste der Informationsobjekte an den Tester zurückgeliefert.

#### 4.2.6 Google Safe Browsing

Wie bei der Prüfung aller verlinkten Seiten hat der Google Safe Browsing Test ein Python-Skript für die Logik und ein PhantomJS-Skript für das Herausfiltern der Links. Da beide Tests das gleiche PhantomJS-Skript verwenden, wird an dieser Stelle auf den ersten Absatz des letzten Unterabschnitts verwiesen.

Anders hingegen, als beim vorherigen Test, werden die Links nicht an *webifier Data*, sondern an den externen Cloud-Service von Google geschickt. Zuvor muss jedoch die Anfrage konfiguriert werden. Dabei werden unter anderem die gewünschten Ergebnistypen (hier: alle vier, s. Unterabschnitt 3.2.6), bedrohte Systeme (hier: `ANY_PLATFORM`) und letztendlich die URLs angegeben werden. Findet der Service zu den Links keine Einträge in seiner Datenbank, so gibt er keine liste von matches aus. In diesem Fall ist das Ergebnis *CLEAN*. Der Rest der Ergebnisberechnung findet wie im Konzept beschrieben statt. Die Ausgabe des Tests beinhaltet neben dem Ergebnis die Liste der matches.

### 4.2.7 Überprüfung des SSL-Zertifikats

Der Test zur Überprüfung des SSL-Zertifikates wurde mit Python umgesetzt. Technisch wurde zur Überprüfung auf die CLI-Anwendung *openssl* gesetzt. Mit dieser wurden alle verfügbaren Informationen des Zertifikats der Webseite ausgelesen und zwischengespeichert.

Anschließend wurden alle Daten zusammengetragen und daraus das entsprechende Endergebnis erzeugt. Nutzt die Webseite kein Zertifikat, so ist das Ergebnis *SUSPLCIOUS*. Hat die Webseite ein gültiges Zertifikat, dann ist das Ergebnis des Tests *CLEAN*. Nutzt die Webseite ein fehlerhaftes Zertifikat, so wird das Ergebnis *MALICIOUS* zurückgegeben. Tritt während dem Test ein unbekannter Fehler auf, so endet der Test mit dem Ergebnis *UNDEFINED*. Zusätzlich zum Ergebnis werden alle ausgelesenen Informationen des Zertifikats an den Tester zurückgegeben.

### 4.2.8 Erkennung von Phishing

Die Erkennung von Phishing kann sehr komplex und umfassend werden. Deshalb wurde der Phishing Detector für diese Arbeit sehr einfach gehalten und beschränkt sich auf eine Grundlagenuntersuchung. Der Test umfasst sieben Teilschritte, welche nun genauer erläutert werden.

Im ersten Schritt wird die gegebene Webseite mit Hilfe von PhantomJS aufgerufen und ausgewertet. Diese Auswertung umfasst die gefundenen Schlüsselwörter der Seite, welche im nächsten Schritt zum Finden möglicher Originale benötigt werden. Außerdem wird ein Screenshot von der Webseite gespeichert, der Quelltext ausgelesen und auf die zum Inhalt gehörenden Wörter reduziert. Zusätzlich wird untersucht, ob die Seite Passworteingabefelder enthält. All diese Informationen gibt PhantomJS an das Python-Skript zurück.

Wenn mindestens ein Schlüsselwort gefunden wurde folgt nun der nächste Schritt. Wurde kein Schlüsselwort gefunden, so endet der Test mit dem Ergebnis *UNDEFINED*. Der zweite Schritt ist das Finden von Links für mögliche Originale der zu überprüfenden Seite. Hierfür wird einmal das erste Schlüsselwort verwendet um eine mögliche Domain daraus zu bilden. Für die Domainbildung werden die 5 meist genutzten Domainendungen<sup>85</sup> (.com, .ru, .net, .org und .de) eingesetzt. Außerdem werden alle

<sup>85</sup> [https://w3techs.com/technologies/overview/top\\_level\\_domain/all](https://w3techs.com/technologies/overview/top_level_domain/all)

gefundenen Schlagwörter genutzt um in verschiedene Suchmaschinen danach zu suchen und so Links zu möglichen Dupplikaten zu finden. Zur Suche werden die drei Suchmaschinen DuckDuckGo<sup>86</sup>, Ixquick<sup>87</sup> und Bing<sup>88</sup> verwendet. Von jeder Suchmaschine werden die ersten zehn Einträge weiterverarbeitet und in einer Gesamtliste mit den erzeugten Domains zusammengeführt. Links die in mehreren Suchmaschinen aufgelistet werden steigen im Ranking auf. Am Ende dieses Schrittes wird die Liste auf 25 Einträge reduziert.

Im nächsten Schritt müssen nun die Originalseiten herausgefiltert und aus der Liste der möglichen Originale entfernt werden, da viele Webseiten, beispielsweise die von Google unter vielen unterschiedlichen Domains erreichbar ist. Um dies zu erreichen wurden mehrere Methoden angewandt. Als erstes werden alle Links aufgelöst, um tatsächlich den Link der zu vergleichenden Seite zu nutzen. Ist der Link nicht erreichbar, oder führt er ins Nichts, so wird er aussortiert. Anschließend wird die Url auf die registrierte Top-Level Domain (TLD) reduziert und mit der gegebenen Seite verglichen. Beispielsweise wird `https://abc.example.com/xyz` auf `example.com` minimiert. Sind beide gleich, so wird der Link verworfen. So werden Unterseiten und Subdomains der gegebenen Seite aussortiert. Als nächstes wird die IP-Adresse der Domain aufgelöst und mit der zu prüfenden Seite verglichen. Sind diese identisch wird auch dieser Link aus der Liste entfernt. Als letztes werden die Zertifikatsinformationen der Links ausgelesen, sofern ein SSL-Zertifikat verfügbar ist. Anschließend werden die *Subject*-Informationen des Zertifikates mit denen der gegebenen Seite verglichen. Sind diese gleich, so scheidet auch dieser Link aus. Nun wird die Liste der übrig gebliebenen Links noch auf zehn Links reduziert, welche nun im nächsten Schritt aufgerufen werden.

Der vierte Schritt ist das abrufen der Links der möglichen Originalwebseiten. Wie im ersten Schritt für die gegebene Seite werden nun für alle Links der Quelltextausgelesen, daraus der Inhalt extrahiert und ein Screenshot angefertigt. Alle hierbei gewonnenen Informationen werden im folgenden Schritt für den Vergleich benötigt.

Nun folgt der eigentliche Schritt des Vergleichens der möglichen Originale mit der zu validierenden Webseite. Der Vergleich erfolgt in den drei Bereichen Quelltext, Inhalt und Aussehen. Zum Quelltext- und Inhaltsvergleich werden Funktionen (*diff-lib.SequenceMatcher(...).quick\_ratio()*) der Python-Standardbibliothek verwendet. Für

---

<sup>86</sup> <https://duckduckgo.com/>

<sup>87</sup> <https://www.ixquick.com/>

<sup>88</sup> <https://www.bing.com/>

den Vergleich der Screenshots wurde auf die freie Bibliothek Resemble.js zurückgegriffen. Diese liefert für den Vergleich von zwei Bildern die prozentuale Übereinstimmung und erzeugt einen Differenzbild. All diese Daten wurden den Links zugeordnet und im nächsten Schritt zur Ergebnisberechnung verwendet.

Im vorletzten Schritt werden nun alle Links klassifiziert. Wie Listing 19 werden zunächst werden die Schwellwerte der einzelnen Ergebnisklassen festgelegt. Diese sind davon abhängig, ob Passworteingabefelder auf der Seite vorhanden sind und ob die Seite ein SSL-Zertifikat nutzt. Besitzt die Seite mindestens ein Passwortfeld und nutzt kein Zertifikat, so werden die Schwellwerte um 0.05 reduziert. Anschließend wird die Durchschnittsübereinstimmung der beiden Seiten berechnet. In diese Rechnung fließt das Ergebnis des Screenshotvergleichs mit doppeltem Gewicht ein, da dieses am Aussagekräftigsten ist. Letztendlich werden die Links mit Hilfe der Schwellwerte und der berechneten Werte der Übereinstimmung klassifiziert. Hierfür gibt es einmal einen Mindestschwellwert für die Durchschnittsübereinstimmung, ab dem ein Ergebnis greift und Schwellwerte für die einzelnen Bereiche, in denen verglichen wurde. Die Bedingungen in den Zeilen 20 und 25 in Listing 19 zeigen die Fälle, in denen ein Link *SUSPICIOUS* oder *MALICIOUS* ist.

```
1 def calculate_result(website, response):
2     ratio_subtract = 0
3     if website['password_field'] and not website['certificate']:
4         ratio_subtract = 0.05
5
6     suspicious_min = 0.82 - ratio_subtract
7     suspicious_min_with_other = 0.77 - ratio_subtract
8     suspicious_screenshot_min = 0.91 - ratio_subtract
9     suspicious_html_min = 0.93 - ratio_subtract
10    suspicious_content_min = 0.95 - ratio_subtract
11
12    malicious_min = 0.9 - ratio_subtract
13    malicious_min_with_other = 0.85 - ratio_subtract
14    malicious_screenshot_min = 0.96 - ratio_subtract
15    malicious_html_min = 0.97 - ratio_subtract
16    malicious_content_min = 0.98 - ratio_subtract
17
18    ratio = (response["screenshot_ratio"] * 2 + response["html_ratio"] + response["
19        content_ratio"]) / 4
20    result = "CLEAN"
21    if ratio > suspicious_min or (ratio > suspicious_min_with_other
22        and (response["screenshot_ratio"] > suspicious_screenshot_min
23            or response["html_ratio"] > suspicious_html_min
24            or response["content_ratio"] > suspicious_content_min)):
25        result = "SUSPICIOUS"
26    if ratio > malicious_min or (ratio > malicious_min_with_other
27        and (response["screenshot_ratio"] > malicious_screenshot_min
28            or response["html_ratio"] > malicious_html_min
```

```
28         or response["content_ratio"] > malicious_content_min)):
29     result = "MALICIOUS"
30     return ratio, result
```

Listing 19: Ergebnisberechnung der Erkennung von Phishing

Der letzte Schritt berechnet abschließend das Endergebnis des Tests. Ist mindestens ein Link *SUSPICIOUS* oder *MALICIOUS*, so ist auch das Endergebnis entsprechend *SUSPICIOUS* oder *MALICIOUS*. Andernfalls ist das Gesamtergebnis des Tests *CLEAN*.

#### 4.2.9 Screenshot der Seite

Der Screenshot der Seite erfolgt über eine von PhantomJS gelieferte Methode um den Seiteninhalt aufzunehmen und als Bilddatei abzuspeichern. PhantomJS wird hierbei genutzt da der Test in einem Docker ohne grafische Benutzeroberfläche läuft und deshalb ein headless Browser nötig ist um die Seite aufzurufen. Nachdem die Seite in einer Bilddatei abgespeichert ist, wird diese als base64-encoded String weitergegeben. Der Test liefert immer das Ergebnis *sauber*, welches aber für den Tester irrelevant ist, da der Screenshot-Test keine Gewichtung im Tester hat. In der Ergebnisinformation wird der base64 encodierte String weitergegeben, welcher dann von der Plattform interpretiert und als Bild für den Nutzer dargestellt wird.



## 5 Analyse

In diesem Kapitel wird der Verlauf der Analyse und deren Ergebnisse dargestellt. Für die Analyse wurden mehrere freie Domainlisten aus dem Internet verwendet. Alle verwendeten Listen lassen sich in drei Kategorien gruppieren.

Die erste Kategorie ist „Top Webseiten“. Diese enthält Listen, welche die meistgenutzten Webseiten, Internetdienste und Plattformen auflisten. Da das Ziel der Analyse aber das Finden von möglichst vielen bedrohlichen Seiten war wurde aus dieser Kategorie nur eine Domainliste eingesetzt. Diese stammt von Pofex<sup>89</sup>. Diese Kategorie umfasste 1.000 Domains. Die zweite Kategorie umfasst frei bereitgestellte Blacklists, welche beispielsweise in Firewalls oder Netzwerkfaltern genutzt werden. Hier wurden Blacklists von den drei Anbietern SquidGuard<sup>90</sup>, Shalla Secure Services<sup>91</sup> und URL.Blacklist.com<sup>92</sup> verwendet. Diese Kategorie umfasste knapp sieben Millionen Domains. Die letzte Kategorie enthält als gefährlich gekennzeichnete Webseiten („Dangerous Lists“). Diese Listen stammen von OpenPhish<sup>93</sup>, Malware Domain List<sup>94</sup> und DNS-BH – Malware Domain Blocklist<sup>95</sup>. Diese Kategorie umfasst mehrere Tausend Domains.

Die Listen der Kategorien „Top Webseiten“ und „Dangerous Lists“ wurden vollständig von webifier überprüft. Die Listen aus der Kategorie „Blacklists“ wurden zufällig ausgewählt und stellen aufgrund des Umfangs den größten Anteil der geprüften Domains dar.

Nachdem die Domains von webifier überprüft und die Ergebnisse in der Datenbank abgelegt worden sind werden sie von webifier-statistics für die Auswertung genutzt. Hier wird eine interaktive Weboberfläche erzeugt. Abbildung 30 zeigt den Einstiegs-

---

<sup>89</sup> <http://www.pofex.com/>

<sup>90</sup> <http://www.squidguard.org/>

<sup>91</sup> <http://www.shallalist.de/>

<sup>92</sup> <http://www.urlblacklist.com/>

<sup>93</sup> <https://openphish.com/>

<sup>94</sup> <http://www.malwaredomainlist.com/>

<sup>95</sup> <http://www.malwaredomains.com/>

punkt auf die Webseite. Hier sind Zahlen über die Anzahl der getesteten Seiten pro Tag und insgesamt aufgelistet. Diese werden anhand des Ergebnisses gruppiert. Zusätzlich findet sich hier noch die durchschnittliche Analysezeit für eine Webseite. Darunter ist eine Beschreibung was webifier-statistics ist.

Die Grafik visualisiert die Aktivität von webifier über den gesamten, von webifier-data, gespeicherten Zeitraum. Die Schwankungen in der Analyseaktivität sind zum einen auf 2 Serverausfälle zurückzuführen. Des Weiteren gibt es Abweichungen, da die Analysezeit abhängig von der Größe und Komplexität der Webseite ist. So wurden an Tagen mit weniger Aktivität komplexere Webseiten getestet.

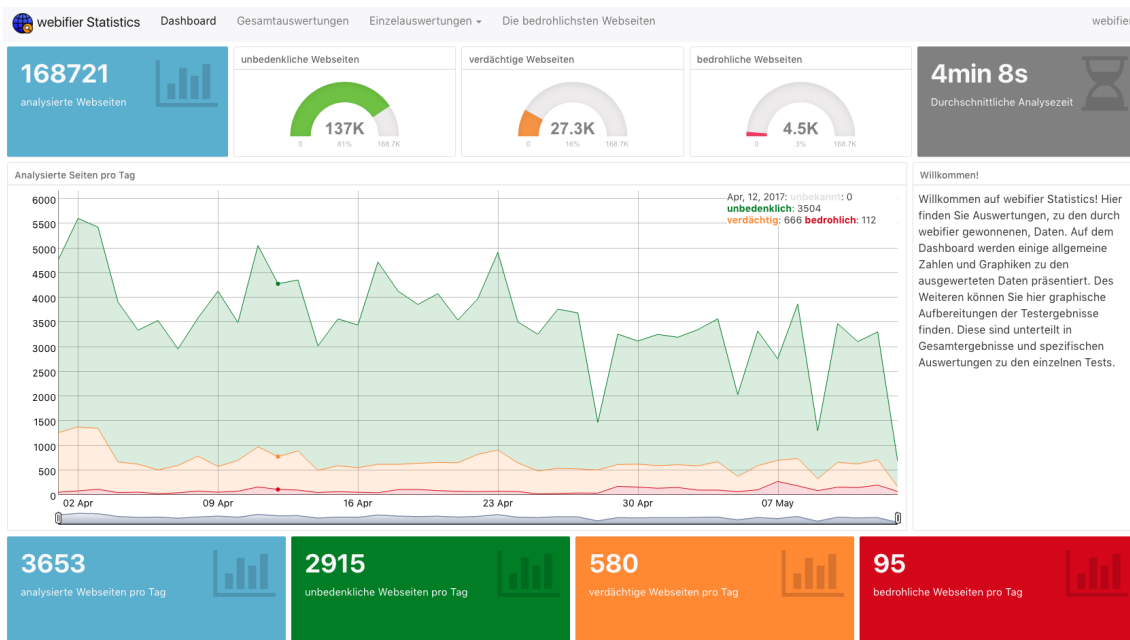


Abbildung 30: Webifier Statistics Dashboard<sup>96</sup>

An der Oberkante ist eine Navigationsleiste zu finden. Diese wird auf allen Teilseiten angezeigt. Der Nutzer kann hier zwischen den einzelnen Seiten wechseln, hinter dem Reiter *Einzelauswertungen* befindet sich ein Dropdown-Menü zur Auswahl des jeweiligen Testes für den der Nutzer die Statistiken betrachten möchte. In den nachfolgenden Kapiteln wird auf diese Auswertungen genauer eingegangen.

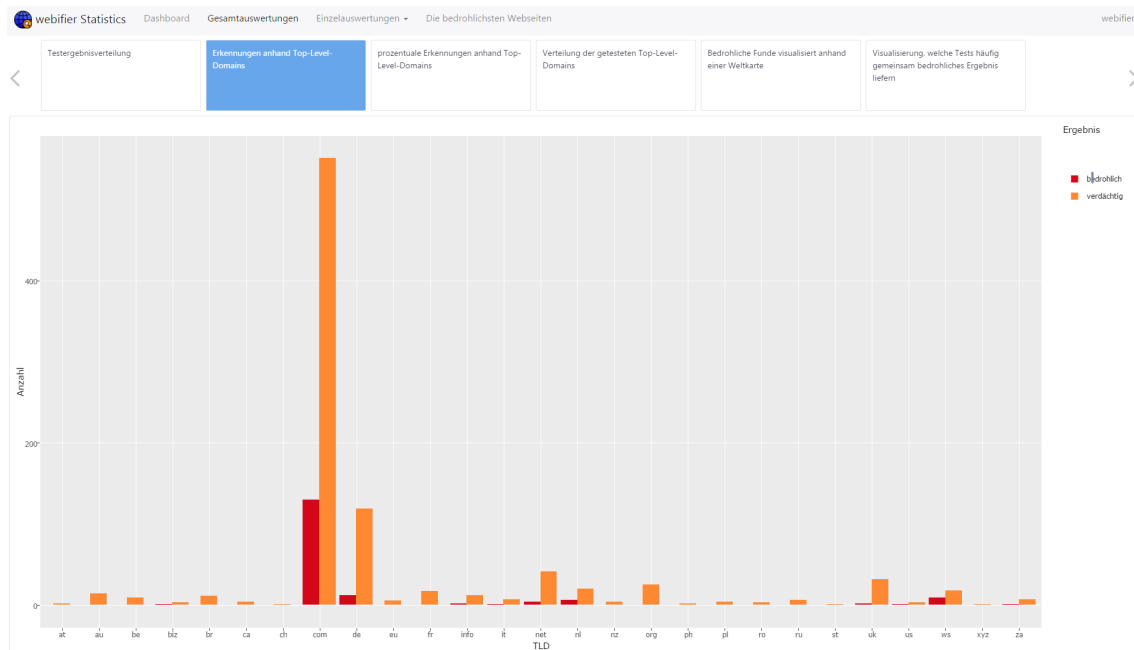
<sup>96</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

## 5.1 Gesamtauswertungen

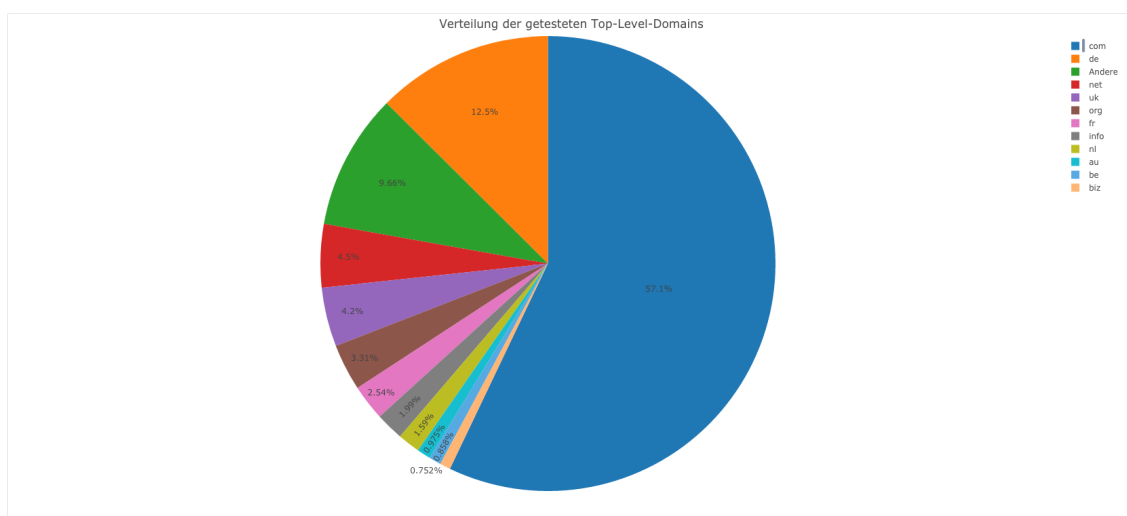
In den Gesamtauswertungen finden sich alle Statistiken, die sich auf die Gesamttests der Webseite beziehen. Ein Gesamttest ist der komplette Durchlauf aller Tests einer Webseite mit aggregiertem Ergebnis. In den folgenden Abschnitten werden die Grafiken beschrieben, die dort zu finden sind.

Der Nutzer hat hier wieder die Möglichkeit über eine zweite Navigationsleiste zwischen den einzelnen Statistiken zu wechseln (siehe Abbildung 31 oben). In den folgenden Abbildungen wird dies zugunsten der Übersichtlichkeit nicht angezeigt.

Die erste Statistik (Abbildung 31) visualisiert die Anzahl der bedrohlichen und verdächtigen Funde anhand der TLD. Zur besseren Übersichtlichkeit werden hier nur Seiten mit bedrohlichem oder verdächtigem Ergebnis gezeigt. Diese werden dann anhand ihrer TLD aggregiert und als Balkendiagramm dargestellt. Es werden nur TLD, welche mindestens 50 Einträge in der Datenbank haben. Der rote Balken steht für die Anzahl an bedrohlichen Ergebnissen der jeweiligen TLD und der gelbe Balken für die verdächtigen Funde. Es fällt auf, dass die TLD .com und .de die größten Anteile an beiden Balken haben. Dies ist darauf zurückzuführen, dass diese im WWW am weitesten verbreitet sind. Da hier nicht die prozentuale Verteilung der Ergebnisse auf eine TLD zu erkennen ist, kann nur Aussage über die Anzahl der Funde in der Datenbank Aussage getroffen werden. Die allgemeine Bedrohlichkeit einer TLD kann hieraus nicht abgeleitet werden.

Abbildung 31: Erkennungen anhand Top-Level-Domains<sup>97</sup>

Das, in Abbildung 32 dargestellte, Tortendiagramm zeigt die Verteilung der getesteten Ergebnisse anhand der TLD auf. Auch hier bilden die Top-Level Domains .com (blau - 57.1%) und .de (orange - 12.5%) den größten Anteil. Dies bestätigt den Inhalt der vorherigen Grafik, dass diese TLD, auch für maliziöse Zwecke, sehr verbreitet sind. Sie werden gefolgt von ebenfalls bekannten Adressen wie .net oder .uk.

Abbildung 32: Verteilung der getesteten Top-Level-Domains<sup>98</sup>

<sup>97</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

Die Abbildung 33 befasst sich ebenfalls mit den Top-Level Domains. Hier wird die prozentuale Verteilung der Ergebnisse unbedenklich(grün),verdächtig(gelb) und bedrohlich(rot) aufgezeigt. Hier fällt auf, dass die bedrohlichsten TLDs eher unbekannte sind, wie beispielsweise .pictet oder .sa. Diese beiden kommen auf jeweils 100% bedrohliche Ergebnisse. Das bedeutet, dass jede Seite mit dieser TLD von webifier als bedrohlich erkannt wurde. Die .com-Domain liegt hier im Mittelfeld mit knappen 80% sauberen Webseiten. Weiterhin ist zu sehen, dass die deutsche TLD(.de) in der Liste nicht auftaucht. Dies ist darauf zurückzuführen, dass für diese Auswertung alle Top-Level-Domains mit 100% sauberen Webseiten vernachlässigt wurden. Wie aus Abbildung 31 bekannt, wurden aber auch hier bedrohliche Seiten gefunden. Diese sind prozentual gesehen jedoch ein so geringer Anteil, dass sie durch Rundung vernachlässigt wurden.

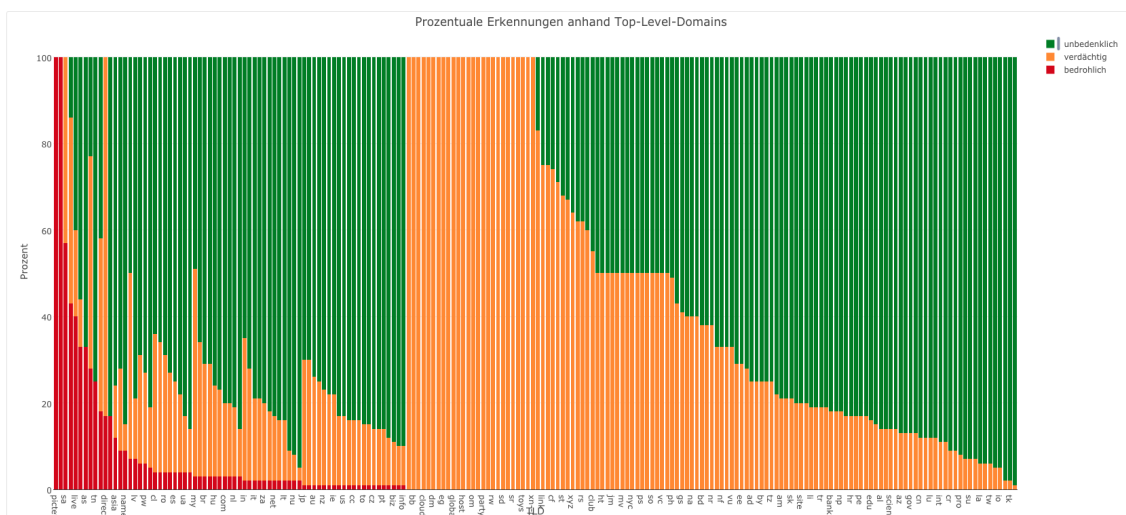


Abbildung 33: prozentuale Erkennungen anhand Top-Level-Domains<sup>99</sup>

Aus den 3 Grafiken zur Verteilung bezogen auf die Top-Level Domains kann erkannt werden, dass bestimmte, eher unbekannte, Top-Level Domains öfter für bedrohliche Zwecke verwendet werden. Jedoch werden auch die bekannten (bspw. .com und .de) Adressen genutzt. Dies lässt sich erklären, dass viele Nutzer im WWW bei Webseiten, die auf .de oder .com enden unvorsichtig werden und sich sicher fühlen, da ihnen die TLD bekannt ist.

<sup>98</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

<sup>99</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

In Abbildung 34 wird eine Weltkarte dargestellt, welche sich anhand des Risikofaktors der Länder verfärbt. Der Risikofaktor berechnet sich aus dem durchschnittlichen Ergebniswert aller Ergebnisse des Landes und dies wird mit 100 multipliziert um den prozentualen Wert zu erhalten. Hierbei werden die TLD den Ländern zugeordnet. Domains wie .com, welche nicht auf ein bestimmtes Land zurückgeführt werden können, werden in dieser Auswertung vernachlässigt. Länder, für die es keine Zuordnung gab bleiben unausgefüllt.

Es ist zu schnell zu Erkennen, dass die Top-Level Domains für Saudi Arabien(.sa) und Libyen(.ly) den größten Risikofaktor haben. Ein Problem bei dieser Darstellung ist, dass nur die Top-Level Domain der Internetadresse entscheidend ist für die Zuordnung. Für eine genauere Zuordnung müsste man die Lokation anhand der IP-Adresse herausfinden und diese dann den Ländern zuordnen. Denn eine .sa-Domain bedeutet nicht zwangsläufig, dass der Betreiber der Webseite auch in dem jeweiligen Land ansässig ist. Oft wird die Webseite von einem anderen Standort aus betrieben. Einige Betreiber nutzen eine bestimmte Top-Level Domain auch wegen dem Namen. Des Weiteren könnten mit einer Adressierung über IP-Adresse auch jene TLD mit einbezogen werden, welche keinem Land zugeordnet sind.

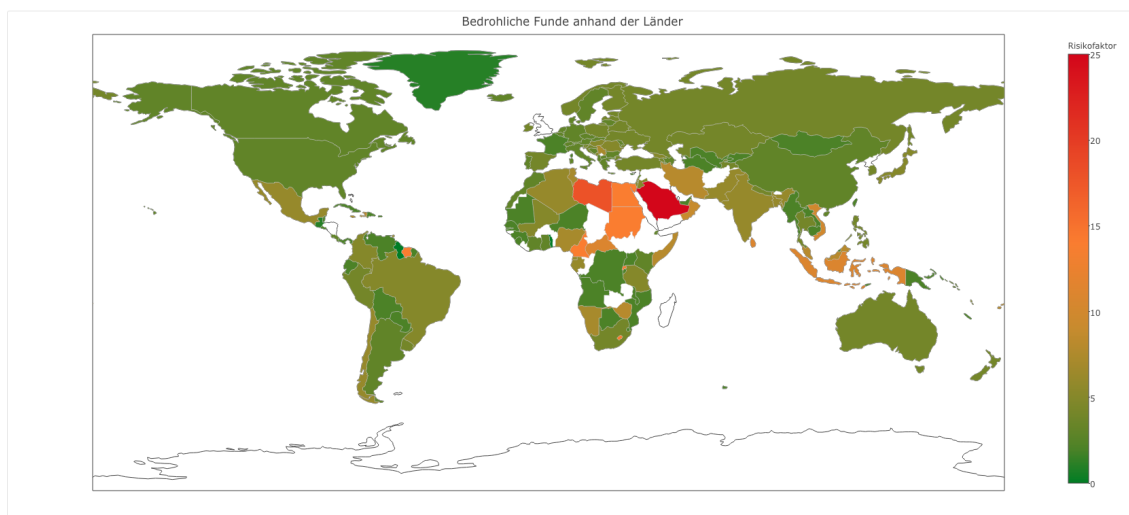


Abbildung 34: Bedrohliche Funde visualisiert anhand einer Weltkarte<sup>100</sup>

Im folgenden Balkendiagramm (siehe Abbildung 35) wird die Verteilung der Ergebnisse der einzelnen Tests dargestellt. Für jeden Test werden bis zu 4 Balken für die jeweiligen Ergebnistypen dargestellt.

<sup>100</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

Hier fällt auf, dass der IP-Scan-Test und der Virusscan-Test nahezu nur saubere Ergebnisse haben. Desweiteren ist zu Erkennen, dass der CertificateChecker sehr viele verdächtige Seiten gefunden hat. Was dieser Wert aussagt wird im späteren Kapitel über den Test genauer beleuchtet. Eine weitere Beobachtung gibt es bei dem LinkChecker-Test. Dieser hat viele unbekannte Ergebnisse. Dies ist darauf zurückzuführen, dass gerade zu Anfang der Analyse der Datenbestand sehr gering war, so war die Wahrscheinlichkeit, dass eine verlinkte Seite bereits in unseren Daten vorhanden ist sehr gering, somit hat der LinkChecker oft keine Aussage über die Bedrohlichkeit der Seite treffen können.

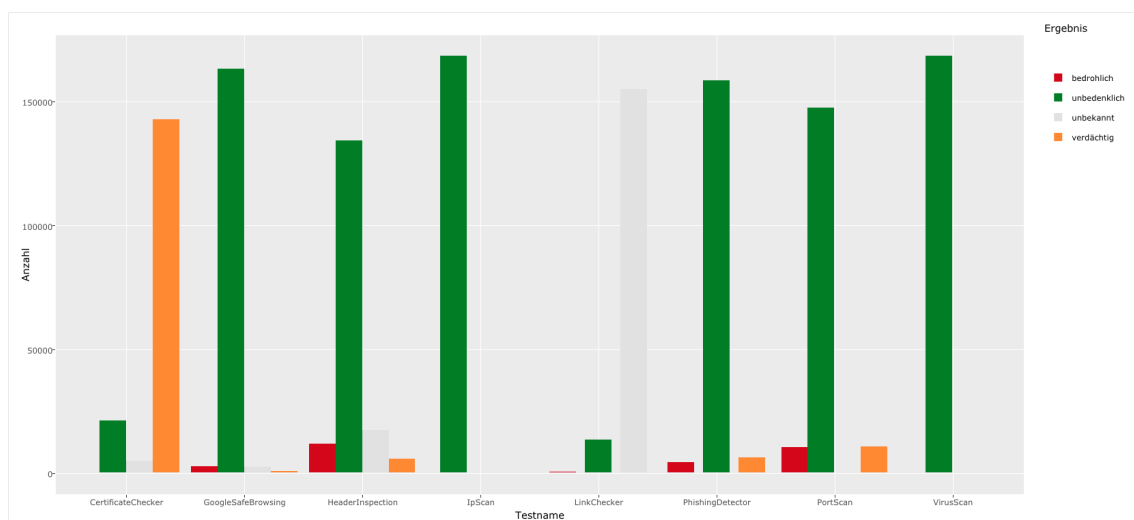
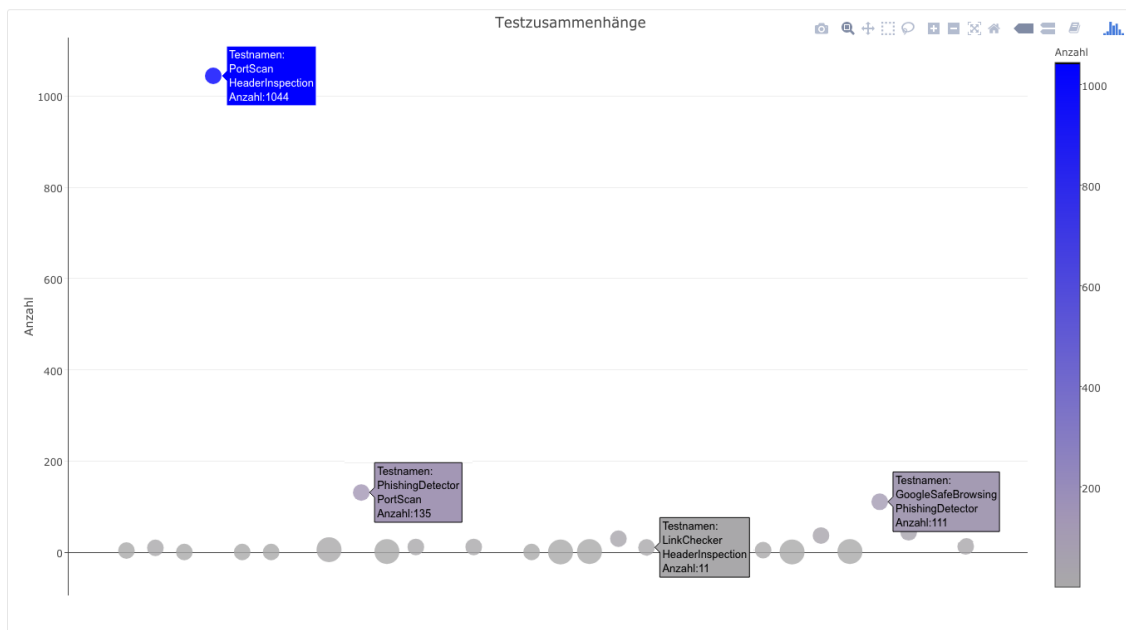


Abbildung 35: Testergebnisverteilung<sup>101</sup>

Die nächste Grafik (siehe Abbildung 36) visualisiert die Zusammenhänge der Tests. Es wird aufgezeigt, welche Tests häufig gemeinsam mit anderen Tests das Ergebnis bedrohlich zur gleichen Webseite liefern. Die Größe der einzelnen Punkte wird über die Anzahl der involvierten Tests bestimmt. Die Anzahl sagt aus wie oft diese Tests die gleiche Seite als bedrohlich klassifiziert haben.

Es ist festzustellen, dass am häufigsten die Tests HeaderInspection und PortScan in der Analyse übereinstimmen. Jedoch lässt eine Anzahl von 1.044 Übereinstimmungen bei knapp 170.000 getesteten und ungefähr 4.500 bedrohlichen Seiten nicht auf einen direkten Zusammenhang schließen. Die weiteren Tests schlagen wenig gemeinsam aus, woraus sich kein Zusammenhang feststellen lässt.

<sup>101</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

Abbildung 36: Visualisierung der Testzusammenhänge<sup>102</sup>

Die Tabelle in Abbildung 37 gehört nichtmehr direkt zu den Gesamtauswertungen. Sie ist unter einem eigenen Reiter angesiedelt und dient dazu dem Nutzer zu zeigen, welche Seiten den größten Ergebniswert hatten.

Show 10 entries

Search:

| Rang | host                         |
|------|------------------------------|
| 1    | reflexonature.free.fr        |
| 2    | peferctlindy.blogspot.de     |
| 3    | actiumresources.com          |
| 4    | eroticletter.com             |
| 5    | productivity-engineering.com |
| 6    | uofrock.com                  |
| 7    | wiedemann.com                |
| 8    | www.datilidoit.com           |
| 9    | www.shoe.org                 |
| 10   | www.michaelconley.com        |

Showing 1 to 10 of 10 entries

Previous 1 Next

Abbildung 37: Top 10: Die bedrohlichsten Webseiten<sup>103</sup>

<sup>102</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.



## 5.2 Einzelauswertungen

In den folgenden Abschnitten werden die Ergebnisse der einzelnen Tests beschrieben und diese bewertet. Abbildung 38 zeigt ein Beispiel für eine Seite der Einzelauswertungen eines Testes. Diese Seite ist für die meisten Tests identisch. Deshalb werden im folgenden nur die Kuchendiagramme gezeigt. Für die Überprüfung der Port-Nutzung und dem Virensan der Webseite sind weitere Kennzahlen vorhanden, welche im jeweiligen Kapitel erläutert werden.

In den Einzelauswertungen sind Kennzahlen zur Anzahl der getesteten Seiten und zur durchschnittlichen Analysezeit. Darunter befindet sich ein Kuchendiagramm, welches die prozentuale Verteilung der Testergebnisse visualisiert.

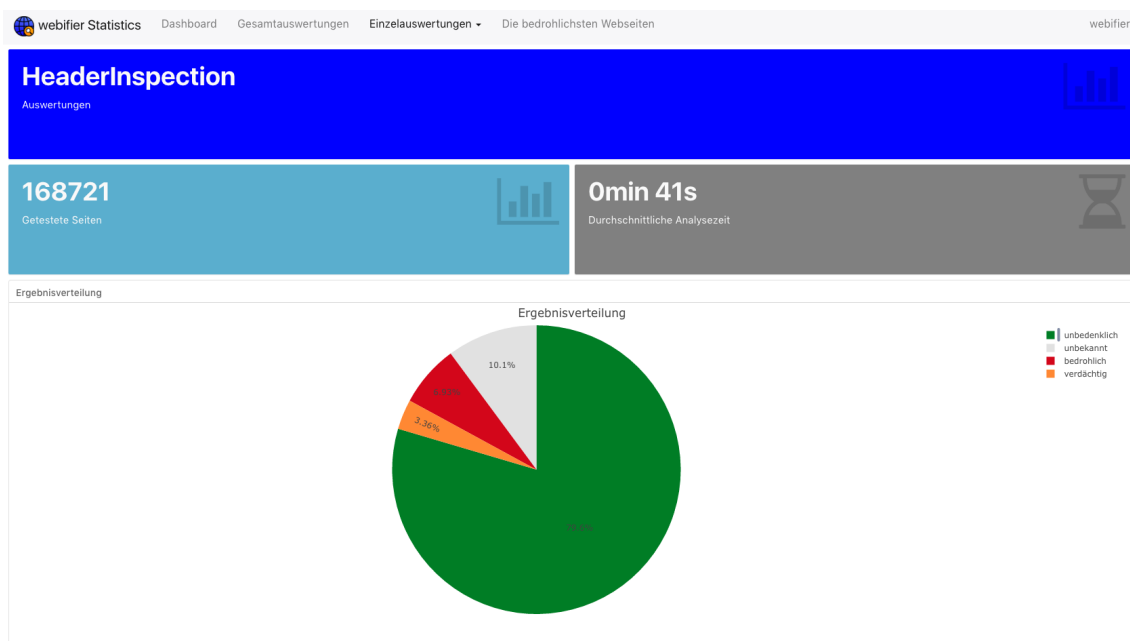


Abbildung 38: Einzelauswertung: Vergleich in verschiedenen Browsern<sup>104</sup>

<sup>103</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

<sup>104</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

### 5.2.1 Virensan der Webseite

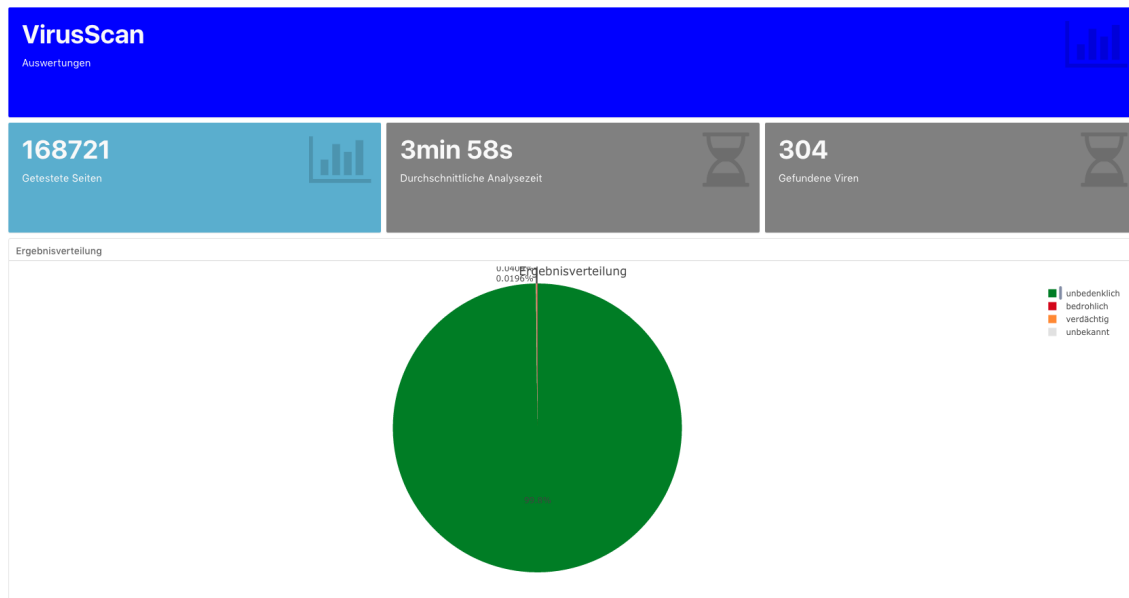


Abbildung 39: Einzelauswertung: Virensan der Webseite<sup>105</sup>

Wie das Kreisdiagramm in Abbildung 39 zeigt enthielten nur etwa 0,2% der überprüften Domains Malware. Dennoch wurden insgesamt über 300 Schadprogramme entdeckt, weshalb die Verbreitung von Viren, Würmern, Trojanern und anderer Malware nach wie vor auch über Webseiten im Internet geschieht. Aus diesem Grund sollte besonders bei zwilichtigen Webangeboten Vorsicht geboten sein.

<sup>105</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

### 5.2.2 Vergleich in verschiedenen Browsern

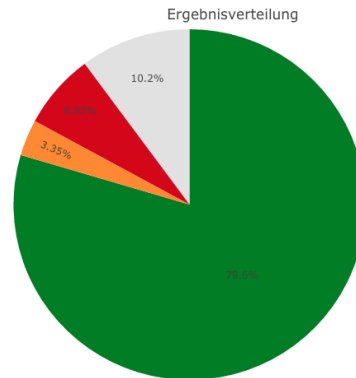


Abbildung 40: Vergleich in verschiedenen Browsern - Testergebnisverteilung

Im *Vergleich in verschiedenen Browsern* wurde die größte Quote an böartigen Webseiten erreicht. Laut Abbildung 40 wurden durch diesen Test 6,93% der Webseiten als *MALICIOUS* eingestuft. Das Kriterium für dieses Ergebnis ist eine maximale Abweichung von 500 Zeichen zwischen den erhaltenen Antworten der diversen Browser. In diese Größenordnung ist zu groß für Zeitangaben oder andere Unterschiede, die durch den Zeitpunkt des Aufrufs entstehen können. Leider kann durch die bloße Angabe der Zeichenunterschiedslänge keine qualitative Aussage über den Hintergrund dieser Verschiedenheit gemacht werden. Besser wäre es, wenn man die unterschiedlichsten Antworten auf JavaScript und Links zu anderen Webressourcen untersuchen würde.

### 5.2.3 Überprüfung der Port-Nutzung

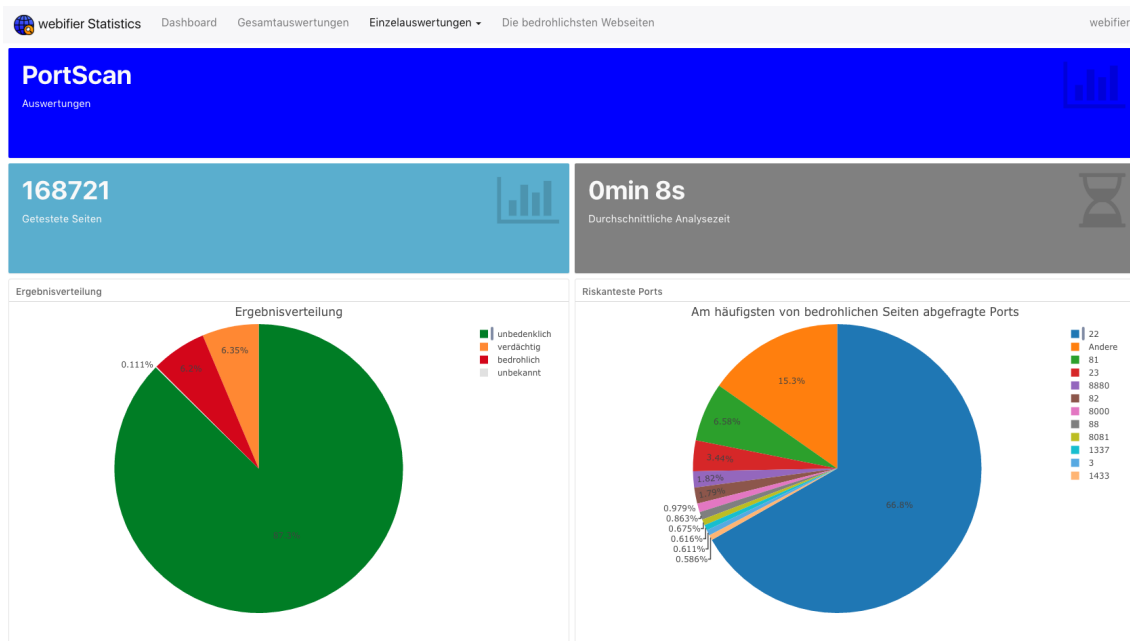


Abbildung 41: Einzelauswertung: Überprüfung der Port-Nutzung<sup>106</sup>

Abbildung 41 zeigt die Ergebnisse des Testes zur Überprüfung der Portnutzung. Hier ist festzustellen, dass Portscanning ein eher wenig genutztes Mittel ist, da 87.3% der Webseiten unbedenklich sind. Zudem läuft der Test mit gerade einmal 0.111% unbekannten Ergebnissen sehr stabil, denn das Ergebnis unbekannt kann hier nur entstehen indem der Test abstürzt.

Neben der Ergebnisverteilung gibt es hier noch ein weiteres Kuchendiagramm zur Visualisierung der abgefragten Ports. Hier werden alle Ports anhand der Häufigkeit ihrer Abfrage aufgelistet. Hier ist erkennbar, dass Port 22, mit 66.8%, der am häufigsten abgefragte Port ist. Da dies der für SSH standardisierte Port ist, kann dies ein Indiz sein, dass Webseiten versuchen sich über SSH Zugriff verschaffen wollen.

<sup>106</sup> Die Abbildung befindet sich in besserer Qualität in Anhang F.

### 5.2.4 Überprüfung der IP-Nutzung

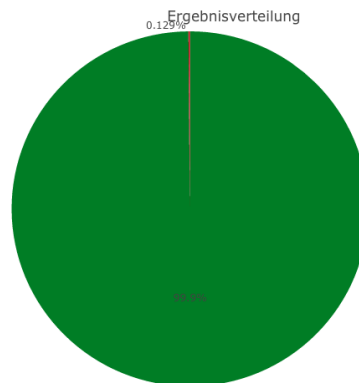


Abbildung 42: Überprüfung der IP-Nutzung - Testergebnisverteilung

Der Test zur Überprüfung der IP-Nutzung hat keinerlei bedrohliche oder verdächtige Webseiten erkannt (siehe Abbildung 42). Wie auch der Test zur Überprüfung der Port-Nutzung läuft dieser Test sehr stabil mit 0.129% unbekannten Ergebnissen. Da beide Tests fast identisch implementiert sind wird ein Fehler in der Umsetzung ausgeschlossen.

### 5.2.5 Prüfung aller verlinkten Seiten

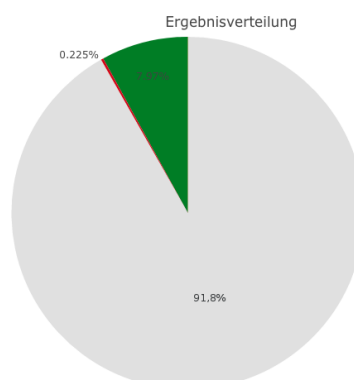


Abbildung 43: Prüfung aller verlinkten Seiten - Testergebnisverteilung

Während der Analysephase wurden vom Test *Prüfung aller verlinkten Seiten* ganze 91,8% der Seiten als *UNDEFINED* klassifiziert (Abbildung 43). Dieses Ergebnis ist nicht allzu überraschend, da in diesem Test gegen die systeminterne Datenbank getestet

wird. Diese war zu Beginn der Analysephase noch komplett leer, sodass **alle** Tests *UNDEFINED* ergaben. Im Laufe der Analyse wurde die Datenbank voller, was man an den 7,97% sauberen und immerhin 0,225% maliziösen Einträgen erkennen kann.

### 5.2.6 Google Safe Browsing

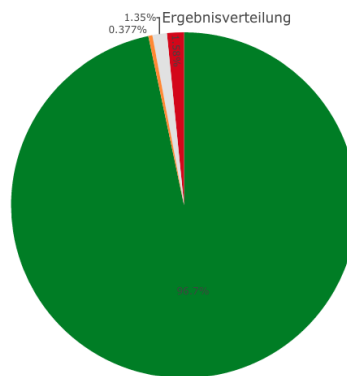


Abbildung 44: Google Safe Browsing - Testergebnisverteilung

Wie in Abbildung 44 zu sehen ist, wurden lediglich 1,58% der getesteten Seiten als *MALICIOUS* und ganze 96,7% als *CLEAN* eingestuft. Leider kann nicht davon ausgegangen werden, dass alle Seiten im *CLEAN*-Bereich tatsächlich unbedenklich sind. Denn es ist sehr wahrscheinlich, dass Google viele der getesteten Seiten noch nicht in ihrer Datenbank abgespeichert hat. Besser wäre es, wenn der Google Safe Browsing Service eine Art *CLEAN*-Ergebnis für bekannte, nichtmaliziöse Webseiten ausgeben würde, statt einfach kein Ergebnis zurückzuliefern.

### 5.2.7 Überprüfung des SSL-Zertifikats

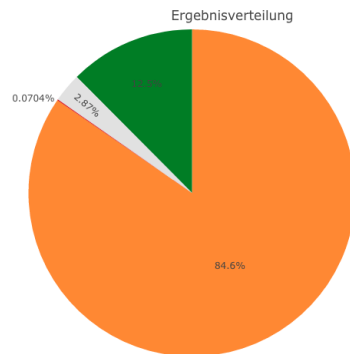


Abbildung 45: Überprüfung des SSL-Zertifikats - Testergebnisverteilung

Sehr positiv an diesem Test ist wohl, dass kaum fehlerhafte Zertifikate gefunden wurden, wie Abbildung 45 zeigt. Beunruhigend ist hingegen das fast 84% aller getesteten Webseiten überhaupt kein Zertifikat verwenden. Zumal der Erwerb eines Zertifikates auch finanziell keine Probleme mehr darstellt, da es wie an anderer Stelle bereits erwähnt Vergabestellen gibt, welche kostenlose Zertifikate ausstellen.

### 5.2.8 Erkennung von Phishing

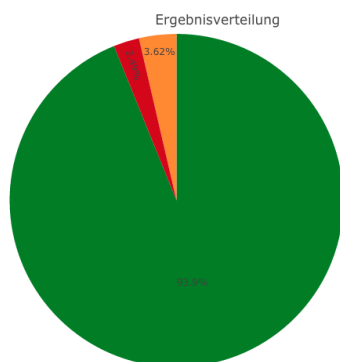


Abbildung 46: Erkennung von Phishing - Testergebnisverteilung

Etwa 6% der überprüften Seiten wurden als Phishingseiten eingestuft. Absolut betrachtet wurden ca. 10.000 Phishingseiten von webifier gefunden. Allerdings sollten diese Werte aufgrund der Einfachheit des mit Vorsicht betrachtet werden, da deshalb

auch einige Webseiten fehlerhaft klassifiziert wurden. Wichtig zu beachten ist außerdem das viele Phishingseiten nur wenige Minuten oder Stunden existieren, weshalb der Erkennungszeitraum auch entsprechend gering ist.

## 5.3 Bewertung der Ergebnisse

Zur Bewertung der Ergebnisse müssen alle Tests betrachtet werden. Verwunderlich ist hierbei, die geringe Anzahl an gefundener Viren. In den Grundlagen wurde analysiert, dass die Verbreitung von Malware in Deutschland stark zugenommen hat (siehe Abbildung 4), jedoch die Ergebnisse von webifier auf eine geringe Malware-Verbreitung schließen lassen. Dies kann beweisen, dass Malware sich nicht durch maliziöse Webseiten verbreitet. Ein weiterer Grund für dieses Ergebnis ist, dass eventuelle versteckte Downloads auf den entsprechenden Seiten nicht mit analysiert wurden.

Des Weiteren ist überraschend, dass knapp 84% der getesteten Webseiten kein Zertifikat nutzen, ob wohl dies heutzutage ohne finanziellen Aufwand erhältlich ist.

In der Verteilung der getesteten Top-Level Domains lassen sich Unterschiede bezüglich der weltweiten Anzahl an genutzten Top-Level Domains<sup>107</sup> erkennen. Der hauptsächliche Unterschied liegt in der Nutzung der russischen Domain. Diese liegt weltweit auf Platz 2 der am häufigsten genutzten Domainendungen. In den von webifier getesteten Seiten taucht diese Endung allerdings sehr selten auf. Die .com-Domain ist in der Analyse von webifier ähnlich stark vertreten wie in der weltweiten Verteilung.

Zur Bewertung der Ergebnisse müsste noch die Güte der genutzten Blacklists überprüft werden. Da dies nicht einfach ist würde eine größer angelegte Suche mit weiteren Webseiten die Ergebnisse aussagekräftiger machen. Im hier beschränkten Zeitraum von 1 Monat zur Analyse konnten bisher nur knapp 170.000 Webseiten analysiert werden. Dies stellt nur einen kleinen Bruchteil der im World Wide Web verfügbaren Webseiten dar. Eine länger angelegte Suche würde die Ergebnisse aussagekräftiger machen.

Bei Betrachtung der Tests zur Überprüfung der Port- & IP-Nutzung ist festzustellen, dass die Verbreitung dieser Angriffstypen eher gering ausfällt. Dies ist nicht verwunderlich, da diese Art von Angriffstypen meist für gezielte Angriffe genutzt werden. Der Angreifer hat meist ein bestimmtes Ziel im Auge und versucht mittels IP- &

<sup>107</sup> [https://w3techs.com/technologies/overview/top\\_level\\_domain/all](https://w3techs.com/technologies/overview/top_level_domain/all)



Port-Scanning Schwachstellen im System des Opfers zu finden, über welche dann ein Hacker-Angriff gestartet werden kann. Die Nutzung dieser Scans auf Webseiten um sämtliche Nutzer auszuspähen ist daher wie erwartet gering.

Zur Bewertung der Ergebnisse lässt sich abschließend sagen, dass die Anzahl an gefundenen Bedrohungen geringer als erwartet ist, dies sich aber auch auf die begrenzte Analysezeit zurückführen lässt. Im folgenden Kapitel wird auf die Zukunft von webifier eingegangen, wie die Plattform erweitert und verbessert werden kann um weitere Auswertungen zu starten.

## 6 Ausblick

Mit webifier wurde eine modulare und leicht zu erweiternde Basis zur automatisierten Überprüfung von Webseiten geschaffen. Der Umfang von webifier kann in Zukunft durch zusätzliche Projekte erweitert werden.

Der Umfang von webifier kann sowohl in Form von neuen Tests zur Überprüfung weiterer clientseitigen Angriffsszenarien, als auch in Form neuer Teilanwendungen von webifier, welche neue Funktionen für die Endnutzer bereitstellen, erweitert werden. So kann die Bedrohlichkeit von Webseiten noch besser klassifiziert werden und dem Nutzer können noch genauere Informationen zur Verfügung gestellt werden.

Ein solches Modul könnte beispielsweise ein Browserplugin sein, welches automatisch vor dem Abrufen einer Webseite webifier Data nach Einträgen zu dieser Seite durchsucht und den Nutzer bei Bedrohungen entsprechend warnt. Eine andere Möglichkeit wäre es die Analyse direkt über den Browser anzustoßen, was allerdings gravierende Auswirkungen auf das Nutzerverhalten hätte, da die Überprüfung einer Webseite etwa vier Minuten in Anspruch nimmt.

Des Weiteren können bereits implementierte Test noch erweitert und so das Gesamtergebnis der Überprüfungen optimiert werden, da viele Tests noch sehr oberflächlich sind und alleine Potential für mehrere Arbeiten liefern.

Außerdem kann durch weitere Überprüfungen von Webseiten der Datenbestand erweitert werden, was neue Auswertungsmöglichkeiten für webifier Statistics schafft. Mit größeren Datenmengen können zuverlässiger statistische Zusammenhänge analysiert werden. Durch weitere Diagramme können neue Erkenntnisse gewonnen werden.

webifier Plattform und webifier Mail könnten noch über einen gemeinsamen Scheduler abstrahiert werden. Dieser könnte auch eine horizontale Skalierbarkeit der Überprüfung ermöglichen, welche momentan noch nicht implementiert ist.

Des weiteren bietet das Darknet ein großes Potential um weitere Analysen zu starten. Hierfür müsste ein neues Modul entwickelt werden welches die Webseiten abrufen, da diese nicht über eine normale Verbindung erreicht werden können.

Dank seiner Modularität bietet webifier optimale Möglichkeiten für eine Weiterentwicklung.

## 7 Fazit

Die Motivation dieser Arbeit war es den Nutzern des Internets eine Möglichkeit zu bieten Webseiten auf ihre Vertrauenswürdigkeit, bzw. ihre Bedrohlichkeit hin überprüfen zu lassen. Hierfür sollten die Aktivitäten von Webseiten anhand verschiedener Tests analysiert und ausgewertet werden. Mit webifier wurde ein solches System geschaffen.

Bereits in der Konzeption wurden Modularität und einfache Erweiterbarkeit des Systems als Ziele verfolgt. So wurde webifier in seine Teilanwendungen gegliedert, welche über einheitliche Schnittstellen miteinander kommunizieren. Dem Nutzer wurde die Nutzung über webifier Plattform und webifier Mail ermöglicht.

Aus diesem Konzept folgte die Entwicklung der einzelnen Teilanwendungen. Diese wurden mit weit verbreiteten Technologien und Frameworks umgesetzt. Außerdem war es notwendig die Tests in einer isolierten Umgebung auszuführen, um das eigene System vor Angriffen und möglichem Schaden zu schützen.

Darauf folgend wurde eine großflächige Analyse von Webseiten durchgeführt, deren Ergebnisse in webifier Data gespeichert wurden. Anschließend nutzte webifier Statistics diese Daten um Auswertungen zu erstellen und diese für den Nutzer zu visualisieren. Bei diesen Diagrammen wurden die gesammelten Daten genau betrachtet. In der Folge wurden die gewonnenen Erkenntnisse untersucht und bewertet.

Abschließend lässt sich festhalten, dass webifier ein großes Potential und viele Möglichkeiten zur Weiterentwicklung und Verwirklichung neuer Ideen geschaffen hat.

## Literaturverzeichnis

Aung, (2017):

TCP Three-Step Handshake, <http://yehtetaung-internetworking.blogspot.de/2015/01/tcp-three-step-handshake.html>, Einsichtnahme: 13.05.2017

Aycock, (2006):

Computer Viruses and Malware, 1. Auflage, Springer US

Baumann, (2013):

Gradle - ein kompakter Einstieg in das Build-Management-System, 1. Auflage, dpunkt.verlag

Bro Network Monitor, (2017):

Introduction, <https://www.bro.org/sphinx/intro/index.html>, Einsichtnahme: 28.04.2017

Chodorow, Dirolf /, (2010):

MongoDB: The Definitive Guide, 1. Auflage, O'Reilly Media

Cosmina, (2016):

Pivotal Certified Professional Spring Developer Exam: A Study Guide, 3. Auflage, Apress

Cryer, (2017):

Resemble.js : Image analysis and comparison, <http://huddle.github.io/Resemble.js/>, Einsichtnahme: 23.04.2017

Flanagan, (2010):

jQuery Pocket Reference: Read Less, Learn More, 1. Auflage, O'Reilly Media

Ghorbani, Lu /, Tavallaee /, (2009):

Network Intrusion Detection and Prevention: Concepts and Techniques, 1. Auflage, Springer Verlag

Google Developers, (2017):

Google Safe Browsing | Google Developers, <https://developers.google.com/safe-browsing/>, Einsichtnahme: 15.05.2017

Gosling, u. a. (2014):

The Java Language Specification - Java SE 8 Edition, 5. Auflage, Addison-Wesley

Gourley, Totty /, (2002):

HTTP: The Definitive Guide, 1. Auflage, O'Reilly Media

Gutierrez, (2016):

Pro Spring Boot, 1. Auflage, Apress

Hidayat, (2017):

PhantomJS | PhantomJS, <http://phantomjs.org>, Einsichtnahme: 15.05.2017

itwissen.info, (2017):

REST (representational state transfer), <http://www.itwissen.info/REST-representational-state-transfer.html>, Einsichtnahme: 22.04.2017

Jackson, (2007):

Web Technologies: A Computer Science Perspective, 1. Auflage, Pearson/Prentice Hall

Jakobsson, Myers /, (2006):

Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft, 1. Auflage, Wiley

Johns, (2017):

Martin Johns, [www.martinjohns.com](http://www.martinjohns.com), Einsichtnahme: 24.04.2017

Kappes, (2013):

Netzwerk- und Datensicherheit: Eine praktische Einführung, 2. Auflage, Springer Vieweg

Messier, (2016):

Penetration Testing Basics: A Quick-Start Guide to Breaking into Systems, 1. Auflage, Springer Science+Business Media New York

nixCraft, (2017):

What is the difference between UDP and TCP internet protocols?, <https://www.cyberciti.biz/faq/key-differences-between-tcp-and-udp-protocols/>, Einsichtnahme: 11.05.2017

Oriyano, Shimonski /, (2012):

Client-Side Attacks and Defense, 1. Auflage, Elsevier Science

PayPal, (2017):

PayPal - Über uns - PayPal, <https://www.paypal.com/de/webapps/mpp/about>, Einsichtnahme: 10.05.2017

Pollack, u. a. (2012):

Spring Data: Modern Data Access for Enterprise Java, 1. Auflage, O'Reilly Media

Python Software Foundation, (2017a):

PhantomJS - Wikipedia, <https://www.python.org/>, Einsichtnahme: 21.04.2017

– (2017b):

PyPI - the Python Package Index: Python Package Index, <https://pypi.python.org/pypi>, Einsichtnahme: 15.05.2017

Rieckmann, Kraus /, (2015):

Tatort Internet: Kriminalität verursacht Bürgern Schäden in Milliardenhöhe, in: DIW Wochenbericht Nr. 12.2015, S. 295–302

Roche, Kauler /, (2017):

HTTrack Website Copier - Free software offline browser, <http://www.httrack.com>, Einsichtnahme: 23.04.2017

Roden, (2017):

Anwendungen mit Docker transportabel machen, <https://www.heise.de/developer/artikel/Anwendungen-mit-Docker-transportabel-machen-2127220.html>, Einsichtnahme: 22.04.2017

Shepherd, (2016):

Browser detection using the user agent - HTTP | MDN, [https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser\\_detection\\_using\\_the\\_user\\_agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent), Einsichtnahme: 10.05.2017

Spurlock, (2013):

Bootstrap

StackOverflow, (2017):

Fastest way to scan ports with Java, <http://stackoverflow.com/questions/11547082/fastest-way-to-scan-ports-with-java>, Einsichtnahme: 11.05.2017

AV-TEST GmbH, (2017):

Malware - AV-TEST, <https://www.av-test.org/de/statistiken/malware/>, Einsichtnahme: 11.05.2017

The jQuery Foundation, (2017):

jQuery, <https://jquery.com/>, Einsichtnahme: 14.05.2017

Tipton, Krause /, (2007):

Information Security Management Handbook, 6. Auflage, Auerbach Publications

Wikipedia, (2017):

Erstellungsprozess, <https://de.wikipedia.org/wiki/Erstellungsprozess>, Einsichtnahme: 22.04.2017

Wolff, (2011):

Spring 3 – Framework für die Java Entwicklung, 3. Auflage, dpunkt.verlag

Wollschläger, (2014):

Grundlagen der Datenanalyse mit R: Eine anwendungsorientierte Einführung, 3. Auflage, Springer Verlag

Wong, (2000):

HTTP Pocket Reference: Hypertext Transfer Protocol, 1. Auflage, O'Reilly Media



World Wide Web Consortium (W3C), (2014):

HTML5, <https://www.w3.org/TR/2014/REC-html5-20141028/single-page.html>, Einsichtnahme: 24.04.2017

Yates, u. a. (2006):

Expert Spring MVC and Web Flow, 1. Auflage, Apress

# Anhang

## TEIL A: Autoren der einzelnen Kapitel

Auf den folgenden Seiten werden die Kapitel in den Farben der Autoren markiert. Dabei steht die Farbe blau für **Daniel Brown**, grün für **Jan-Eric Gaidusch** und gelb für **Samuel Philipp**.

---

### Abstract

#### 1 Einleitung

##### 1.1 Einführung

##### 1.2 Hintergrund

##### 1.3 Aufgabenstellung

##### 1.4 Team

##### 1.5 webifier

#### 2 Grundlagen

##### 2.1 Frontend Technologien und Framework

##### 2.2 Backend Technologien und Frameworks

###### - Java

###### - Spring

###### - MongoDB

###### - Gradle

- Rest

- Docker

- R

## 2.3 Technologien und Frameworks der Tests

- Python

- PhantomJS

- Bro

- HTtrack

- Resemble.js

## 2.4 Angriffstypen

### 2.4.1 Malware

### 2.4.2 Request Header Investigation

### 2.4.3 JavaScript Port & IP Scanning

### 2.4.4 Phishing

## 3 Konzept

### 3.1 Gesamtkonzept

#### 3.1.1 webifier Tests

#### 3.1.2 webifier Tester

#### 3.1.3 webifier Platform

#### 3.1.4 webifier Mail

#### 3.1.5 webifier Data

#### 3.1.6 webifier Statistics

### 3.2 Testarten

#### 3.2.1 Virenscan

### 3.2.2 Vergleich in verschiedenen Browsern

### 3.2.3 Test auf Port Scanning

### 3.2.4 Test auf IP Scanning

### 3.2.5 Link Checker

### 3.2.6 Google Safe Browsing

### 3.2.7 Überprüfung des Zertifikats

### 3.2.8 Erkennung von Phishing

### 3.2.9 Screenshot

## 4 Umsetzung

### 4.1 Gesamtanwendung

#### 4.1.1 webifier Tests

#### 4.1.2 webifier Tester

#### 4.1.3 webifier Platform

#### 4.1.4 webifier Mail

#### 4.1.5 webifier Data

#### 4.1.6 webifier Statistics

### 4.2 Tests

#### 4.2.1 Virensan

#### 4.2.2 Vergleich in verschiedenen Browsern

#### 4.2.3 Test auf Port Scanning

#### 4.2.4 Test auf IP Scanning

#### 4.2.5 Link Checker

#### 4.2.6 Google Safe Browsing

#### 4.2.7 Überprüfung des Zertifikats

#### 4.2.8 Erkennung von Phishing

#### 4.2.9 Screenshot

### 5 Analyse

#### Einfuehrung in Analyse

#### 5.1 Gesamtauswertungen

#### 5.2 Einzelauswertungen

##### 5.2.1 Virensan

##### 5.2.2 Vergleich in verschiedenen Browsern

##### 5.2.3 Test auf Port Scanning

##### 5.2.4 Test auf IP Scanning

##### 5.2.5 Link Checker

##### 5.2.6 Google Safe Browsing

##### 5.2.7 Überprüfung des Zertifikats

##### 5.2.8 Erkennung von Phishing

#### 5.3 Bewertung der Ergebnisse

### 6 Ausblick

### 7 Fazit

## TEIL B: Vollständige Konfigurationsdatei webifier Tester

```
1 {
2   "resolver": {
3     "name": "resolver",
4     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-resolver",
5     "startup_timeout_seconds": 60,
6     "shutdown": "docker stop #ID",
7     "shutdown_timeout_seconds": 30
8   },
9   "tests": [
10    {
11      "name": "VirusScan",
12      "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-virusscan",
13      "startup_timeout_seconds": 600,
14      "shutdown": "docker stop #ID",
15      "shutdown_timeout_seconds": 30,
16      "result_class": "de.securitysquad.webifier.output.result.virusscan.
17        TestVirusScanResultInfo",
18      "weight": 5,
19      "enabled": true
20    },
21    {
22      "name": "HeaderInspection",
23      "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-header-
24        inspection",
25      "startup_timeout_seconds": 300,
26      "shutdown": "docker stop #ID",
27      "shutdown_timeout_seconds": 30,
28      "result_class": "de.securitysquad.webifier.output.result.headerinspection.
29        HeaderInspectionResultInfo",
30      "weight": 1,
31      "enabled": true
32    },
33    {
34      "name": "PortScan",
35      "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-portscan",
36      "startup_timeout_seconds": 300,
37      "shutdown": "docker stop #ID",
38      "shutdown_timeout_seconds": 30,
39      "result_class": "de.securitysquad.webifier.output.result.portscan.TestPortScanResultInfo
40        ",
41      "weight": 3,
42      "enabled": true
43    },
44    {
45      "name": "IpScan",
46      "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-ipscan",
47      "startup_timeout_seconds": 300,
48      "shutdown": "docker stop #ID",
49      "shutdown_timeout_seconds": 30,
50      "result_class": "de.securitysquad.webifier.output.result.ipscan.TestIpScanResultInfo",
```

```
47     "weight": 3,
48     "enabled": true
49 },
50 {
51     "name": "Screenshot",
52     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-screenshot",
53     "startup_timeout_seconds": 300,
54     "shutdown": "docker stop #ID",
55     "shutdown_timeout_seconds": 30,
56     "result_class": "de.securitysquad.webifier.output.result.screenshot.
        TestScreenshotResultInfo",
57     "weight": 0,
58     "enabled": true
59 },
60 {
61     "name": "LinkChecker",
62     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-linkchecker",
63     "startup_timeout_seconds": 300,
64     "shutdown": "docker stop #ID",
65     "shutdown_timeout_seconds": 30,
66     "result_class": "de.securitysquad.webifier.output.result.linkchecker.
        TestLinkCheckerResultInfo",
67     "weight": 1,
68     "enabled": true
69 },
70 {
71     "name": "CertificateChecker",
72     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-
        certificatechecker",
73     "startup_timeout_seconds": 300,
74     "shutdown": "docker stop #ID",
75     "shutdown_timeout_seconds": 30,
76     "result_class": "de.securitysquad.webifier.output.result.certificatechecker.
        TestCertificateCheckerResultInfo",
77     "weight": 3,
78     "enabled": true
79 },
80 {
81     "name": "PhishingDetector",
82     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-
        phishingdetector",
83     "startup_timeout_seconds": 300,
84     "shutdown": "docker stop #ID",
85     "shutdown_timeout_seconds": 30,
86     "result_class": "de.securitysquad.webifier.output.result.phishingdetector.
        TestPhishingDetectorResultInfo",
87     "weight": 5,
88     "enabled": true
89 },
90 {
91     "name": "GoogleSafeBrowsing",
92     "startup": "docker run --rm --name #ID -e URL=#URL -e ID=#ID -e API_KEY=INSERT_API_KEY
        webifier-test-google-safe-browsing",
93     "startup_timeout_seconds": 300,
94     "shutdown": "docker stop #ID",
```

```
95     "shutdown_timeout_seconds": 30,  
96     "result_class": "de.securitysquad.webifier.output.result.googleSafeBrowsing.  
    TestGoogleSafeBrowsingResultInfo",  
97     "weight": 3,  
98     "enabled": true  
99   }  
100 ],  
101 "preferences": {  
102   "push_result_data": true  
103 }  
104 }
```

Listing 20: *webifier Tester* - Vollständige Konfigurationsdatei



## TEIL C: Vollständige Ergebnisberechnung webifier Tester

```
1 private WebifierOverallTestResult calculateOverallResult() {
2     int weightSum = tests.stream().map(WebifierTest::getData).mapToInt(WebifierTestData::
    getWeight).sum();
3     int mostWeighted = tests.stream().map(WebifierTest::getData).mapToInt(WebifierTestData::
    getWeight).max().orElse(weightSum / 2);
4     double maliciousMin = (double) mostWeighted / (double) weightSum;
5     double suspiciousMin = Math.pow(maliciousMin, 2);
6
7     int undefinedTestSum = tests.stream().filter(test -> test.getResult().getResultType() ==
    WebifierResultType.UNDEFINED)
8         .map(WebifierTest::getData).mapToInt(WebifierTestData::getWeight).sum();
9     double undefinedPercentage = (double) undefinedTestSum / (double) weightSum;
10    if (undefinedPercentage > MAX_UNDEFINED_TEST_PERCENTAGE) {
11        return new WebifierOverallTestResult(WebifierResultType.UNDEFINED);
12    }
13    double result = 0;
14    for (WebifierTest<TestResult> test : tests) {
15        double testWeight = (double) test.getData().getWeight() / (double) weightSum;
16        result += getTestResultValue(test.getResult().getResultType(), testWeight) *
            testWeight;
17    }
18    if (result >= maliciousMin) {
19        return new WebifierOverallTestResult(WebifierResultType.MALICIOUS, result);
20    }
21    if (result >= suspiciousMin) {
22        return new WebifierOverallTestResult(WebifierResultType.SUSPICIOUS, result);
23    }
24    return new WebifierOverallTestResult(WebifierResultType.CLEAN, result);
25 }
26
27 private double getTestResultValue(WebifierResultType type, double testWeight) {
28     if (type == WebifierResultType.MALICIOUS) {
29         return 1;
30     }
31     if (type == WebifierResultType.SUSPICIOUS) {
32         return testWeight;
33     }
34     return 0;
35 }
```

Listing 21: *webifier Tester* - Vollständige Ergebnisberechnung

## TEIL D: webifier Plattform - Screenshots

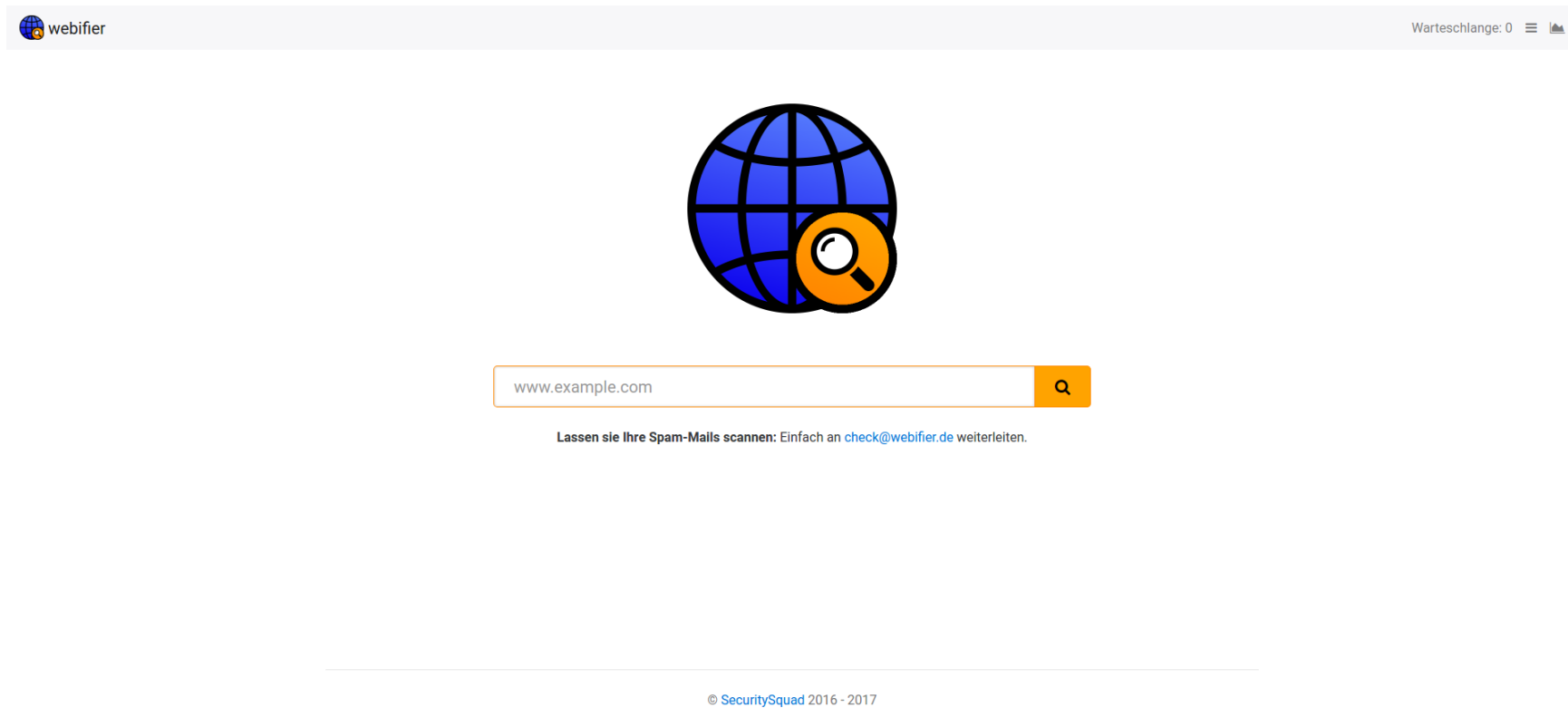


Abbildung 47: *webifier Plattform* - Startseite

reflexonature.free.fr -> http://reflexonature.free.fr















|   |   |
|---|---|
|                                  |   |
| Virensan der Webseite            | Vergleich in verschiedenen Browsern  |
| Überprüfung der Port-Nutzung     | Überprüfung der IP-Nutzung           |
| Prüfung aller verlinkten Seiten  | Google Safe Browsing                 |
| Überprüfung des SSL-Zertifikats  | Erkennung von Phishing               |
| Screenshot der Seite  |                                      |










Abbildung 48: *webifier* Plattform - Ergebnisseite für reflexonature.free.fr

webifier

Warteschlange: 0  

securitysquad.de -> <https://www.securitysquad.de/>



|                                 |   |                                     |   |
|---------------------------------|---|-------------------------------------|---|
| Virenscan der Webseite          |  | Vergleich in verschiedenen Browsern |  |
| Überprüfung der Port-Nutzung    |  | Überprüfung der IP-Nutzung          |  |
| Prüfung aller verlinkten Seiten |  | Google Safe Browsing                |  |
| Überprüfung des SSL-Zertifikats |  | Erkennung von Phishing              |  |
| Screenshot der Seite            |   |                                     |  |

© SecuritySquad 2016 - 2017

Abbildung 49: *webifier* Plattform - Ergebnisseite für securitysquad.de

google.com -> http://www.google.de/?gfe\_rd=cr&ei=JYD\_WKvfLOHM8gelgpKgAw











|   |   |
|---|---|
|                                  |   |
| Virensan der Webseite            | Vergleich in verschiedenen Browsern  |
| Überprüfung der Port-Nutzung     | Überprüfung der IP-Nutzung           |
| Prüfung aller verlinkten Seiten  | Google Safe Browsing                 |
| Überprüfung des SSL-Zertifikats  | Erkennung von Phishing               |
| Screenshot der Seite             |   |

Abbildung 50: *webifier* Plattform - Ergebnisseite für google.com

## TEIL E: Beispieldokumente

### webifierSingleTestResultData - webifier Data

```
1 {
2   "_id" : "2429f999-3168-4070-b085-7887ee64d8a0",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "61574eb4-e9ac-42e2-9140-4c1ce23a2891_5b8bcdbf-2d89-4930-915a-d65c5367bd28",
5   "name" : "VirusScan",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-virusscan",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 5,
10  "startupTimeoutInSeconds" : 600,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "MALICIOUS",
13  "resultInfo" : {
14    "scanned_files" : 4,
15    "malicious_files" : 3,
16    "suspicious_files" : 0,
17    "files" : [
18      {
19        "name" : "index98e1.html",
20        "result" : "MALICIOUS"
21      },
22      {
23        "name" : "index-2.html",
24        "result" : "CLEAN"
25      },
26      {
27        "name" : "indexec9e.html",
28        "result" : "MALICIOUS"
29      },
30      {
31        "name" : "index3547.html",
32        "result" : "MALICIOUS"
33      }
34    ]
35  },
36  "durationInMillis" : NumberLong(142805),
37  "overallResult" : DBRef("webifierTestResultData", "82879e46-b286-4bdf-9876-875dc67e5708")
38 }
```

Listing 22: *webifier Data* - Beispieldokument webifierSingleTestResultData: *Virenskan der Webseite*

```
1 {
2   "_id" : "70210b9f-e4fc-475b-a40a-5c98f49a18d3",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "b19e3a9c-7bca-4d4f-b03b-a87e086d1592_2dd8aa86-b6b5-434a-a627-3ae11a0c1b9e",
5   "name" : "HeaderInspection",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-header-
   inspection",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 1,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "MALICIOUS",
13  "resultInfo" : {
14    "medianRatio" : 0.9945236763609246,
15    "worstRatio" : 0.993661446681581,
16    "medianDiff" : 5522,
17    "worstDiff" : 5589,
18    "browsers" : [
19      "Android 4.3",
20      "Android 5.1",
21      "iPhone 6, iOS 8.4",
22      "iPhone 6, iOS 9.3",
23      "Windows 10, Chrome 54.0",
24      "Windows XP, Internet Explorer 6.0",
25      "Windows XP, Firefox 4.0",
26      "OS X 10.11, Safari 10.0",
27      "OS X 10.11, Safari 6.0"
28    ]
29  },
30  "durationInMillis" : NumberLong(137104),
31  "overallResult" : DBRef("webifierTestResultData", "77f77195-f32e-4305-bff3-c29893a4d2f5")
32 }
```

Listing 23: *webifier Data* - Beispieldokument *webifierSingleTestResultData*: Vergleich in verschiedenen Browsern

```
1 {
2   "_id" : "8908d41a-449d-4760-a9a2-32e6603dc6f1",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "ff4109af-6e5a-4824-ae6a-93d4a06b9077_6b9f0762-92ea-452a-a10f-e469f7884591",
5   "name" : "PortScan",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-portscan",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 3,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "SUSPICIOUS",
13  "resultInfo" : {
14    "unknown_ports" : [
15      22
16    ]
17  },
18  "durationInMillis" : NumberLong(12740),
19  "overallResult" : DBRef("webifierTestResultData", "d15e97c7-3687-4261-b1f1-549ba94666ef")
20 }
```

Listing 24: *webifier* Data - Beispieldokument webifierSingleTestResultData: Überprüfung der Port-Nutzung

```
1 {
2   "_id" : "2c0b3f33-b120-4b74-b9a1-75ac9dlad269",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "33e76954-dc94-48a9-a816-99ddeb647887_8cae7586-8360-4d6b-a855-343a23aff0df",
5   "name" : "IpScan",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-ipscan",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 3,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "CLEAN",
13  "resultInfo" : {
14    "risky_hosts" : [ ]
15  },
16  "durationInMillis" : NumberLong(9803),
17  "overallResult" : DBRef("webifierTestResultData", "e3d35b18-332d-4c16-b1a9-d116850509e9")
18 }
```

Listing 25: *webifier* Data - Beispieldokument webifierSingleTestResultData: Überprüfung der IP-Nutzung



```
1 {
2   "_id" : "4d3362e0-e22a-4daa-a7e9-bee4151d1e04",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "cf7fb17e-edfa-4f44-91cf-ebc60a5e67f6_d63a0f52-ba63-45fb-b9f1-9e51f3639de3",
5   "name" : "LinkChecker",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-linkchecker",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 1,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "MALICIOUS",
13  "resultInfo" : {
14    "hosts" : [
15      {
16        "host" : "alt.com",
17        "result" : "MALICIOUS"
18      },
19      {
20        "host" : "adultfriendfinder.com",
21        "result" : "UNDEFINED"
22      },
23      {
24        "host" : "outpersonals.com",
25        "result" : "UNDEFINED"
26      },
27      {
28        "host" : "cams.com",
29        "result" : "CLEAN"
30      },
31      {
32        "host" : "accounts.google.com",
33        "result" : "SUSPICIOUS"
34      },
35      {
36        "host" : "secureimage.securedaimages.com",
37        "result" : "UNDEFINED"
38      },
39      {
40        "host" : "fonts.gstatic.com",
41        "result" : "UNDEFINED"
42      }
43    ]
44  },
45  "durationInMillis" : NumberLong(4512),
46  "overallResult" : DBRef("webifierTestResultData", "822bde3c-92bc-4980-8ef9-7f07fce49d57")
47 }
```

Listing 26: *webifier* Data - Beispieldokument *webifierSingleTestResultData*: Prüfung aller verlinkten Seiten

```
1 {
2   "_id" : "0ba4f3e3-ff1a-4056-b2e4-87028af128c1",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "7872a958-9e3b-4c44-bd02-38665e06edd5_61c8deb7-80ed-400c-b4d7-64645201397d",
5   "name" : "GoogleSafeBrowsing",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID -e API_KEY=XYZ webifier-test
   -google-safe-browsing",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 3,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "MALICIOUS",
13  "resultInfo" : {
14    "matches" : [
15      "http://www.pizzotti.net/",
16      "http://heirem-art.de/crpzw3bh.php?id=19579352",
17      "http://mardhavi.com/krmvcpgg.php?id=19346074",
18      "http://mardhavi.com/krmvcpgg.php?id=19346073"
19    ]
20  },
21  "durationInMillis" : NumberLong(5709),
22  "overallResult" : DBRef("webifierTestResultData", "9f362427-d34a-456a-b72b-919f26e5de40")
23 }
```

Listing 27: *webifier* Data - Beispieldokument *webifierSingleTestResultData*: *Google Safe Browsing*

```
1 {
2   "_id" : "dlaec3e7-139e-41f8-ab4b-9f4cfd3d382e",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "d712d875-da26-4ecf-a81a-66f902644067_173cc3bd-71c3-445e-939f-e234deb24b2a",
5   "name" : "CertificateChecker",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-
   certificatechecker",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 3,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "CLEAN",
13  "resultInfo" : {
14    "certificate" : {
15      "subject" : {
16        "name" : "www.paypal.com",
17        "organisation" : "PayPal, Inc.",
18        "organisation_unit" : "CDN Support"
19      },
20      "issuer" : {
21        "name" : "Symantec Class 3 EV SSL CA - G3",
22        "organisation" : "Symantec Corporation",
23        "organisation_unit" : "Symantec Trust Network"
24      },
25      "validity" : {
26        "from" : NumberLong("1454371200000"),
27        "to" : NumberLong("1509407999000")
28      },
29      "return_code" : "0 (ok)"
30    }
31  },
32  "durationInMillis" : NumberLong(995),
33  "overallResult" : DBRef("webifierTestResultData", "e8d3cc31-1140-4efd-a11b-cd501ad19952")
34 }
```

Listing 28: *webifier Data* - Beispieldokument *webifierSingleTestResultData*: Überprüfung des SSL-Zertifikats

```
1 {
2   "_id" : "6ebf08e4-7d3b-49ce-be02-76599ffca44d",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierSingleTestResultData",
4   "testId" : "138a8a96-3b80-42aa-93eb-365a82ee2b85_718c9b2d-3e82-4638-a596-cd9b3486b2c7",
5   "name" : "PhishingDetector",
6   "startup" : "docker run --rm --name #ID -e URL=#URL -e ID=#ID webifier-test-
   phishingdetector",
7   "shutdown" : "docker stop #ID",
8   "enabled" : true,
9   "weight" : 5,
10  "startupTimeoutInSeconds" : 300,
11  "shutdownTimeoutInSeconds" : 30,
12  "result" : "MALICIOUS",
13  "resultInfo" : {
14    "keywords" : [
15      "paypal",
16      "bezahlen",
17      "senden",
18      "oder"
19    ],
20    "matches" : [
21      {
22        "url" : "https://www.paypal.com/de/home",
23        "result" : "MALICIOUS",
24        "ratio" : 0.9989644191977116,
25        "html_ratio" : 0.9958576767908464,
26        "content_ratio" : 1,
27        "screenshot_ratio" : 1,
28        "comparison" : null
29      },
30      {
31        "url" : "https://www.paypal.com/de/webapps/mpp/home",
32        "result" : "MALICIOUS",
33        "ratio" : 0.9990458916024976,
34        "html_ratio" : 0.9961835664099902,
35        "content_ratio" : 1,
36        "screenshot_ratio" : 1,
37        "comparison" : null
38      }
39    ]
40  },
41  "durationInMillis" : NumberLong(157961),
42  "overallResult" : DBRef("webifierTestResultData", "cf39cafb-8e2e-4a86-9497-e4fe60e5f996")
43 }
```

Listing 29: *webifier* Data - Beispieldokument *webifierSingleTestResultData*: Erkennung von Phishing

## TEIL F: webifier Statistics - Screenshots

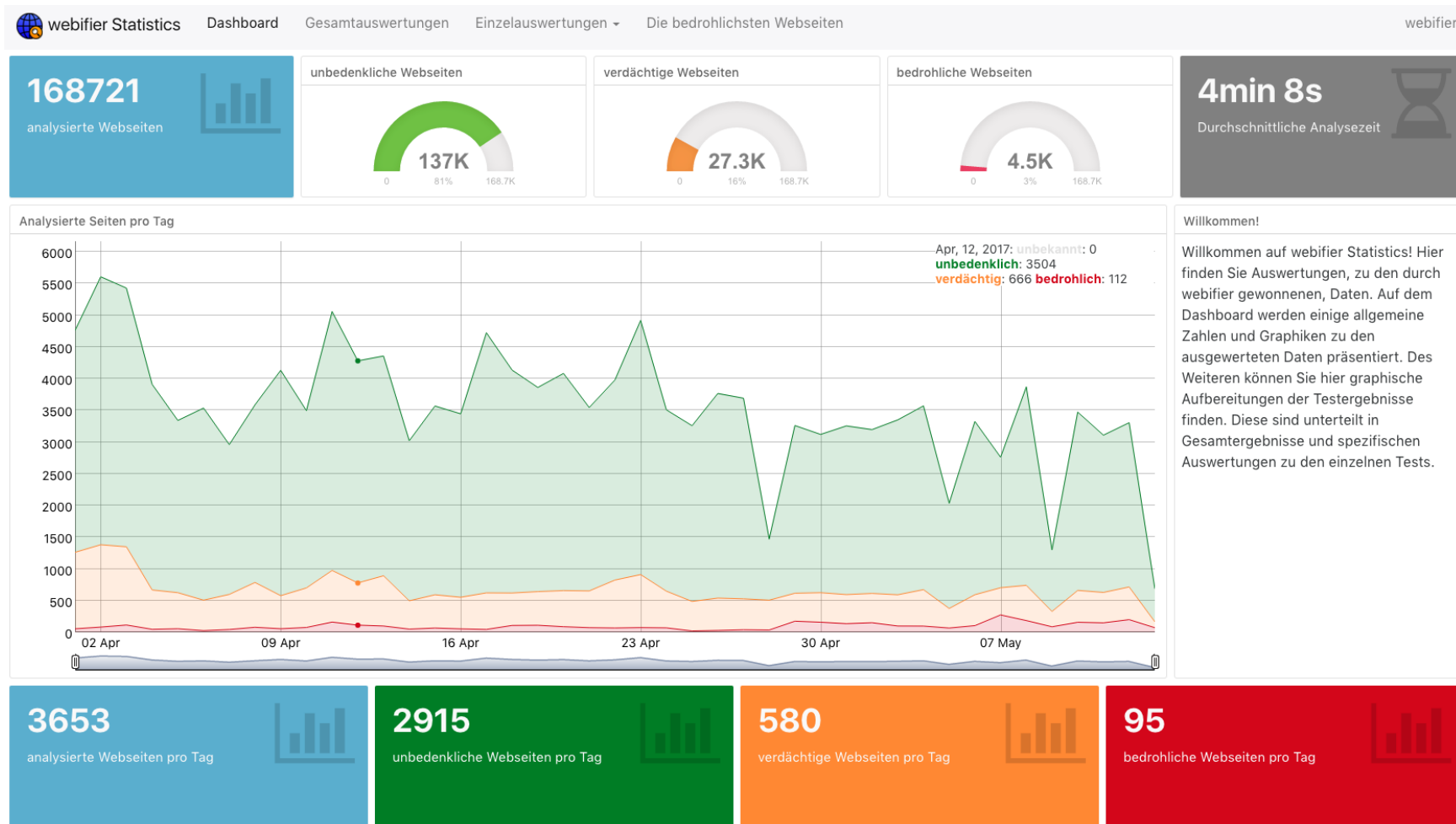


Abbildung 51: webifier Statistics Dashboard

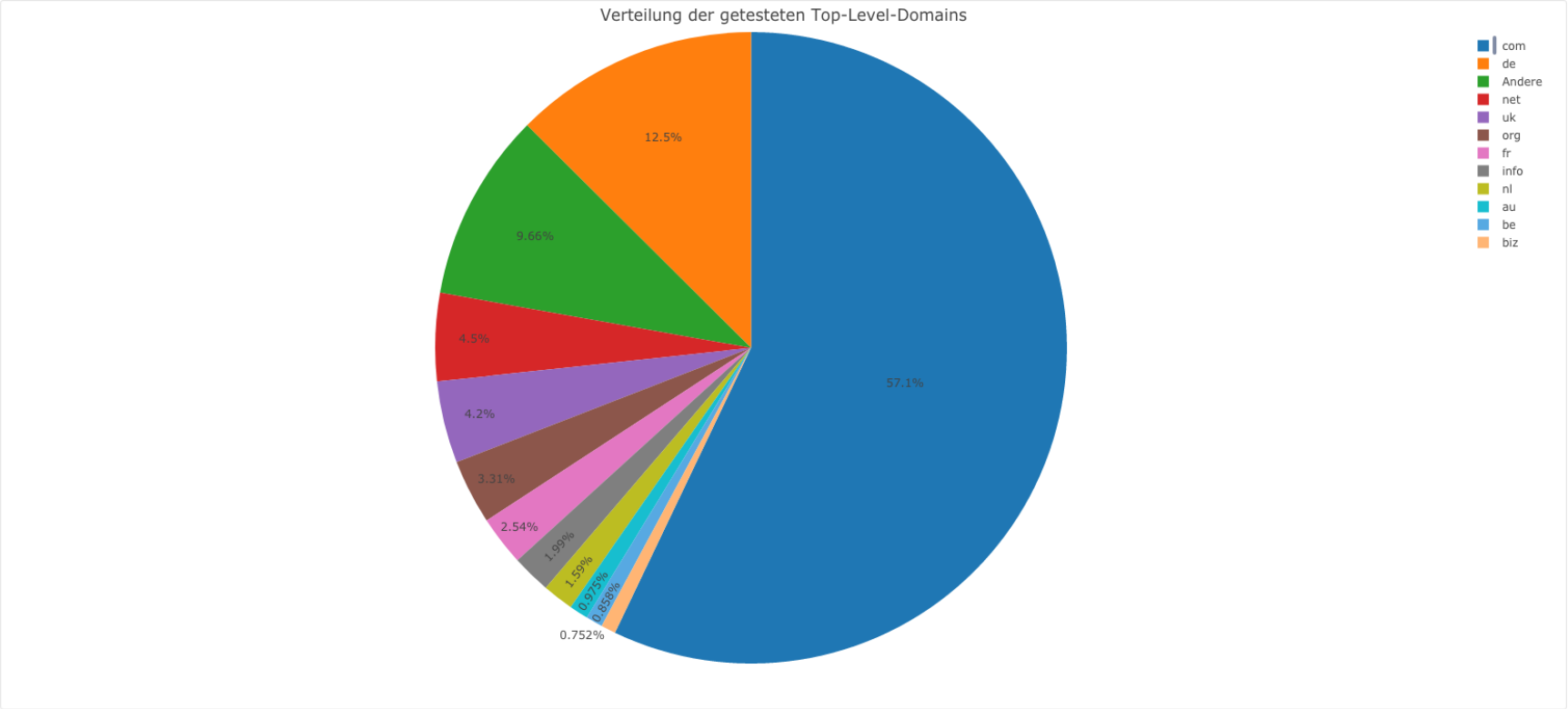
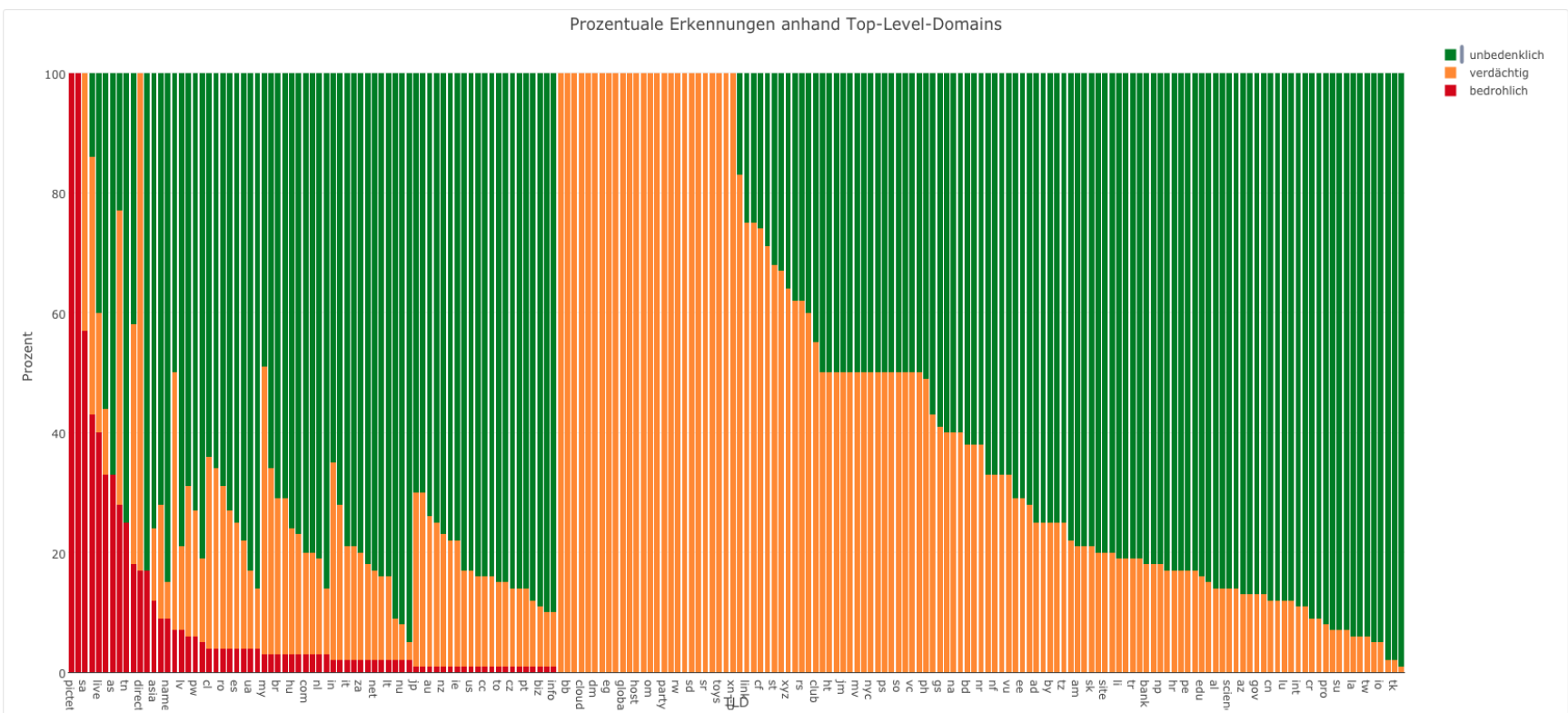


Abbildung 52: Verteilung der getesteten Top-Level-Domains



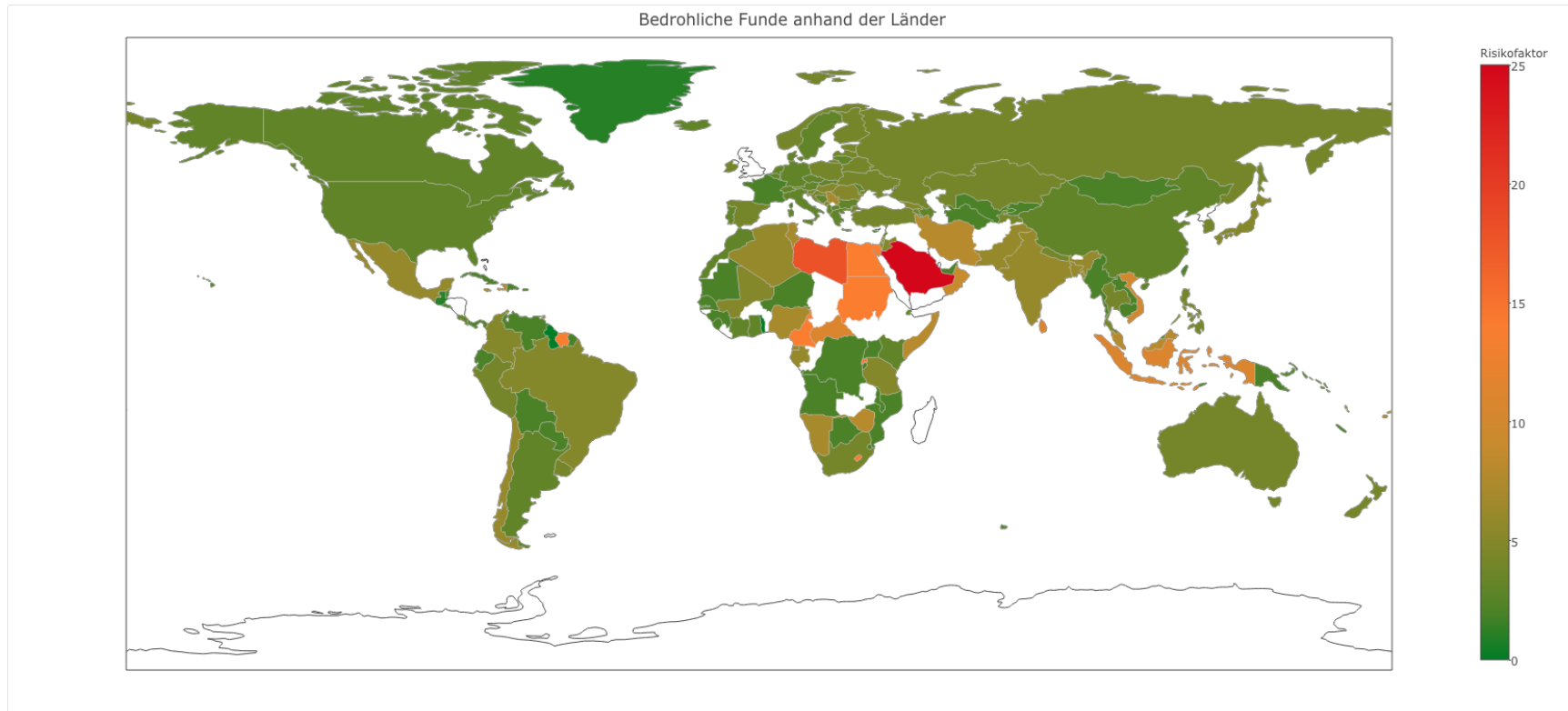


Abbildung 54: Bedrohliche Funde visualisiert anhand einer Weltkarte



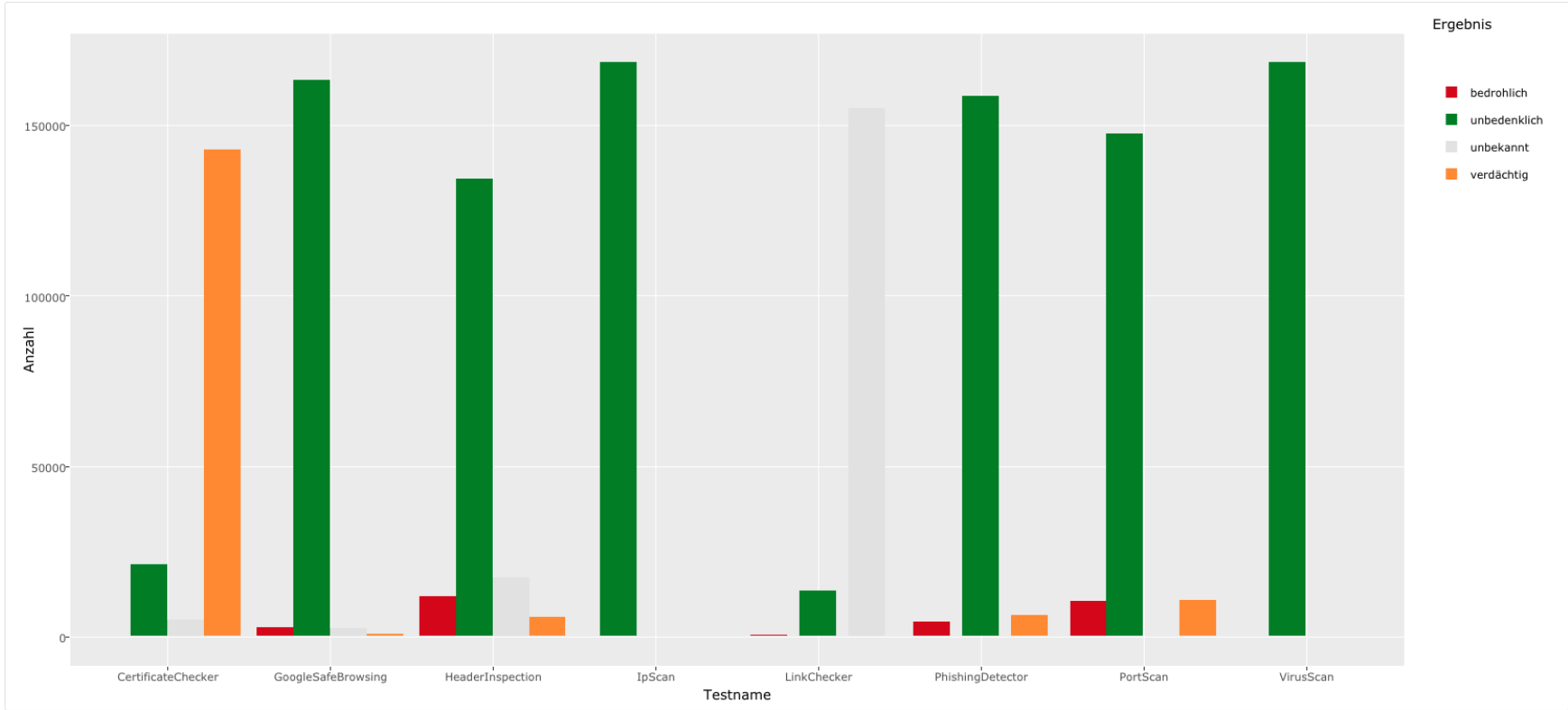


Abbildung 55: Testergebnisverteilung

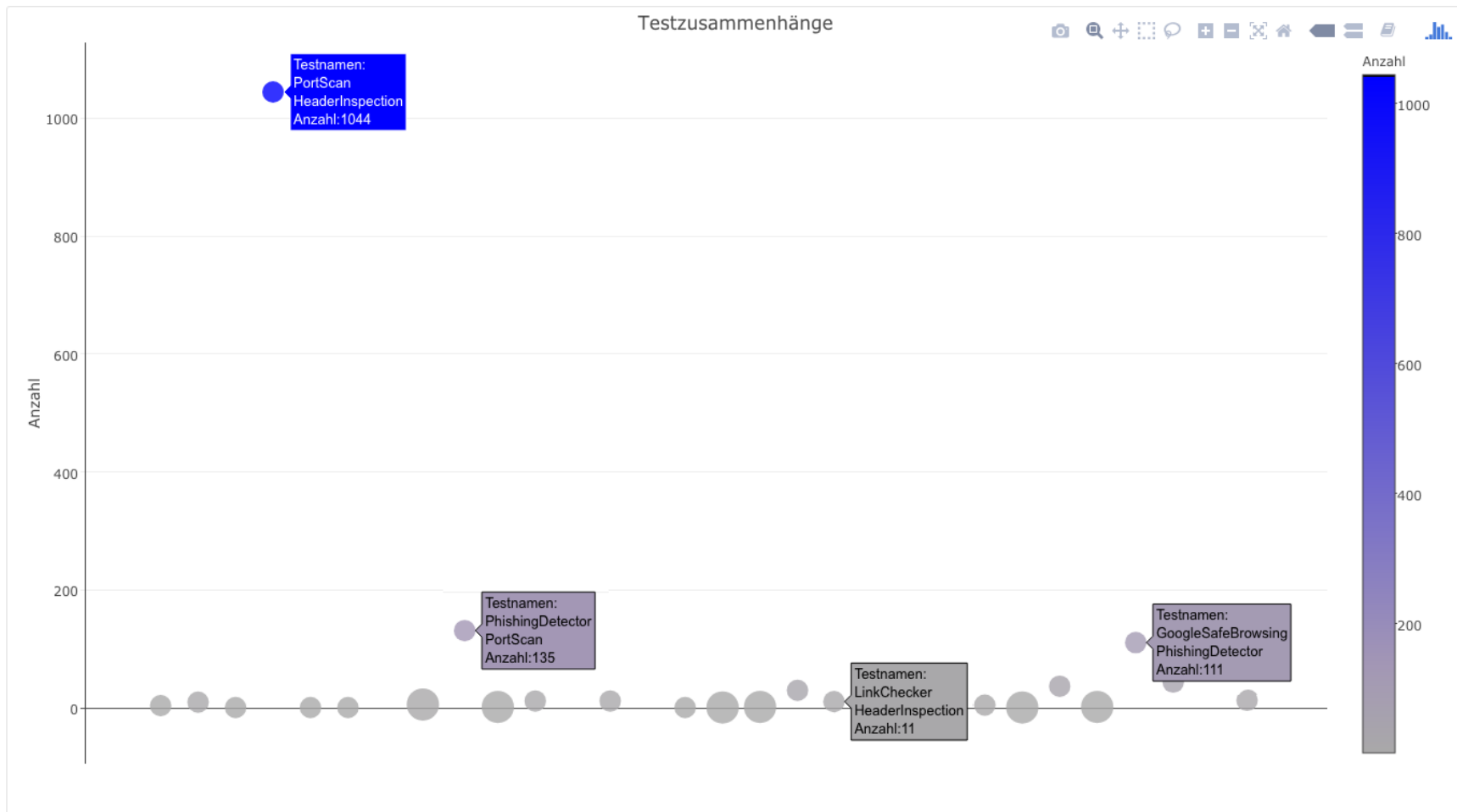


Abbildung 56: Visualisierung der Testzusammenhänge

Show **10** entries Search:

| Rang | host                         |
|------|------------------------------|
| 1    | reflexonature.free.fr        |
| 2    | peferctlindy.blogspot.de     |
| 3    | actiumresources.com          |
| 4    | eroticletter.com             |
| 5    | productivity-engineering.com |
| 6    | uofrock.com                  |
| 7    | wiedemann.com                |
| 8    | www.datidoit.com             |
| 9    | www.shoe.org                 |
| 10   | www.michaelconley.com        |

Showing 1 to 10 of 10 entries

Previous **1** Next

Abbildung 57: Top 10: Die bedrohlichsten Webseiten

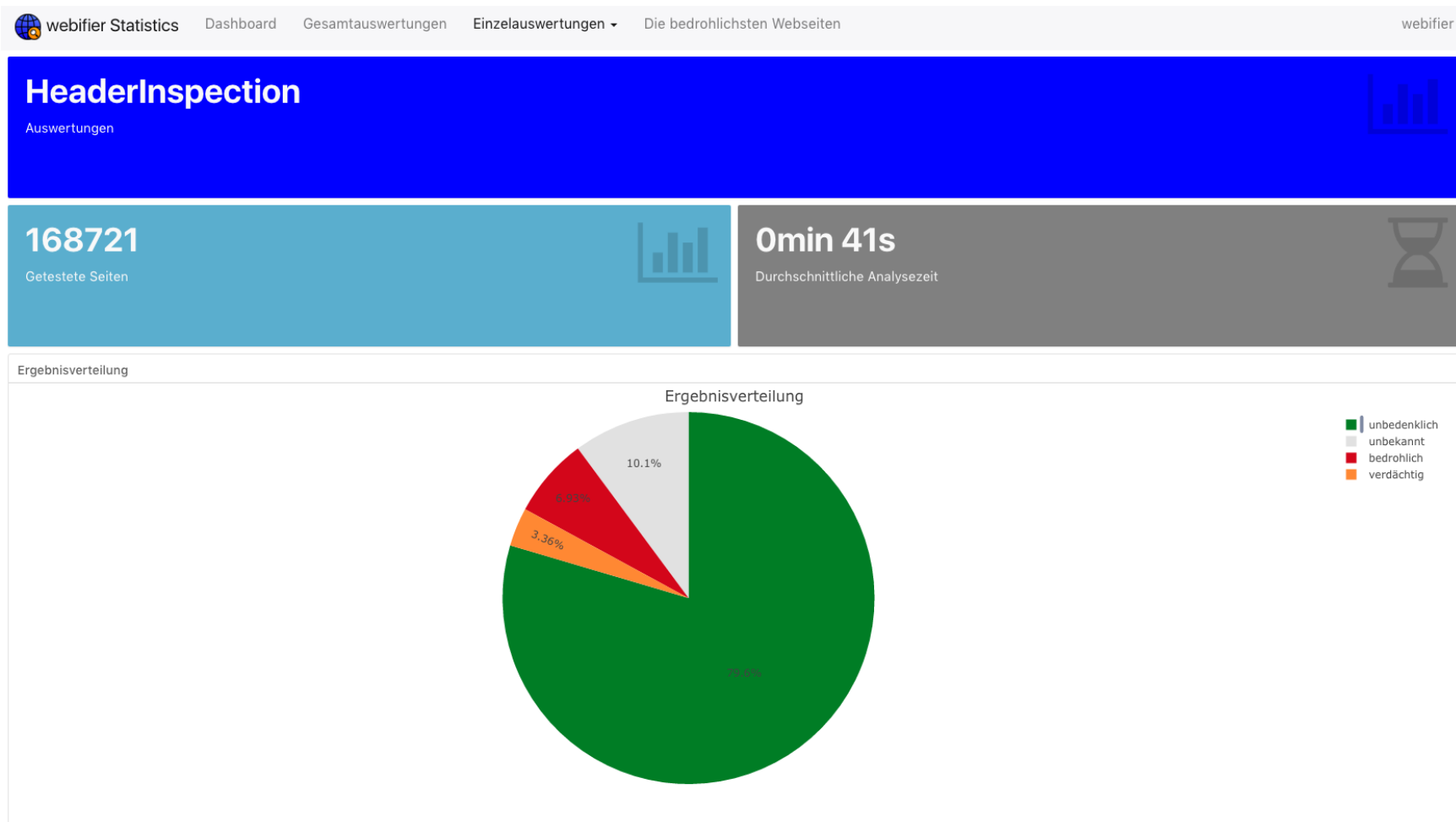


Abbildung 58: Einzelauswertung: *Vergleich in verschiedenen Browsern*

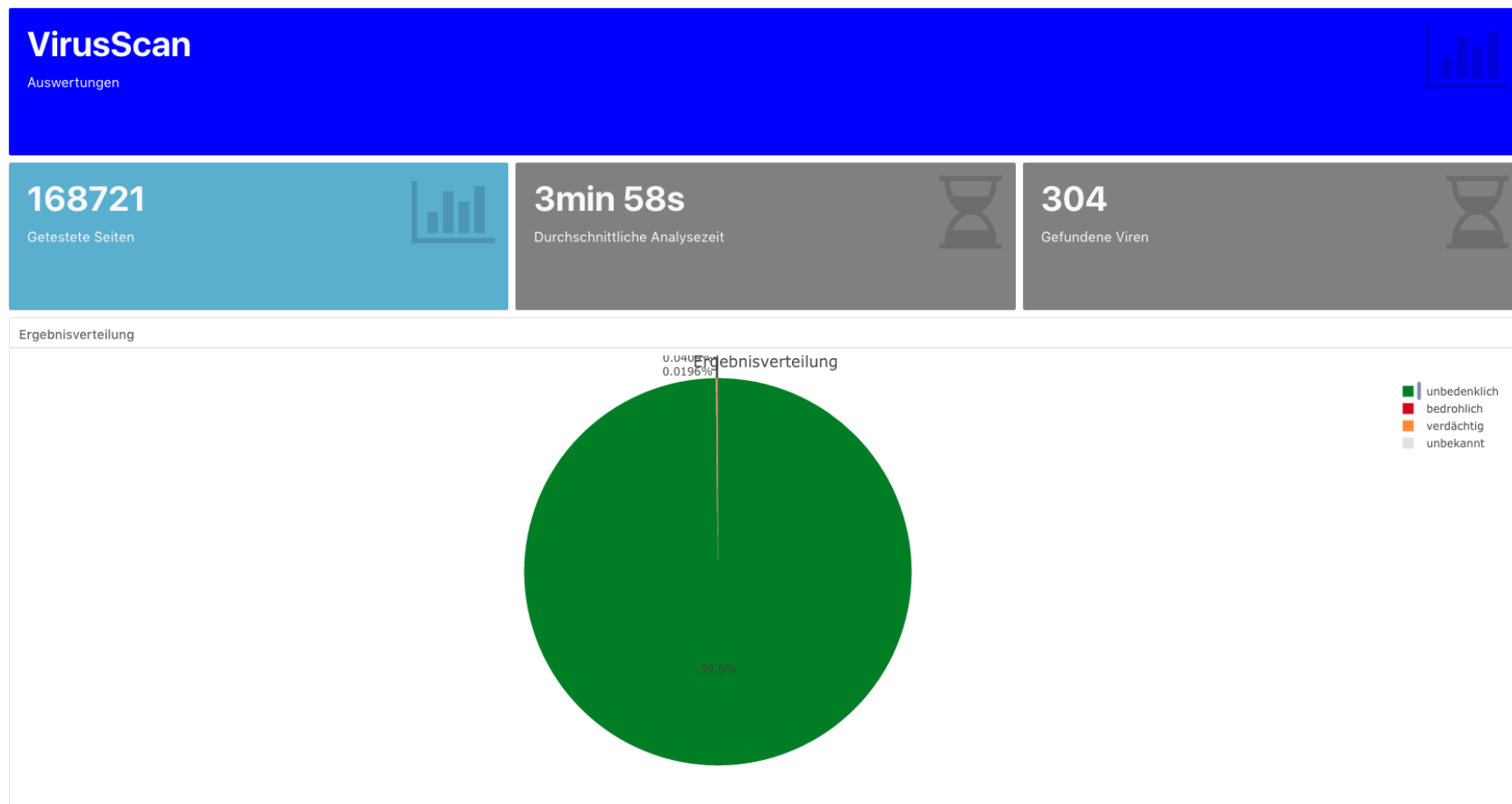


Abbildung 59: Einzelauswertung: *Virensan der Webseite*

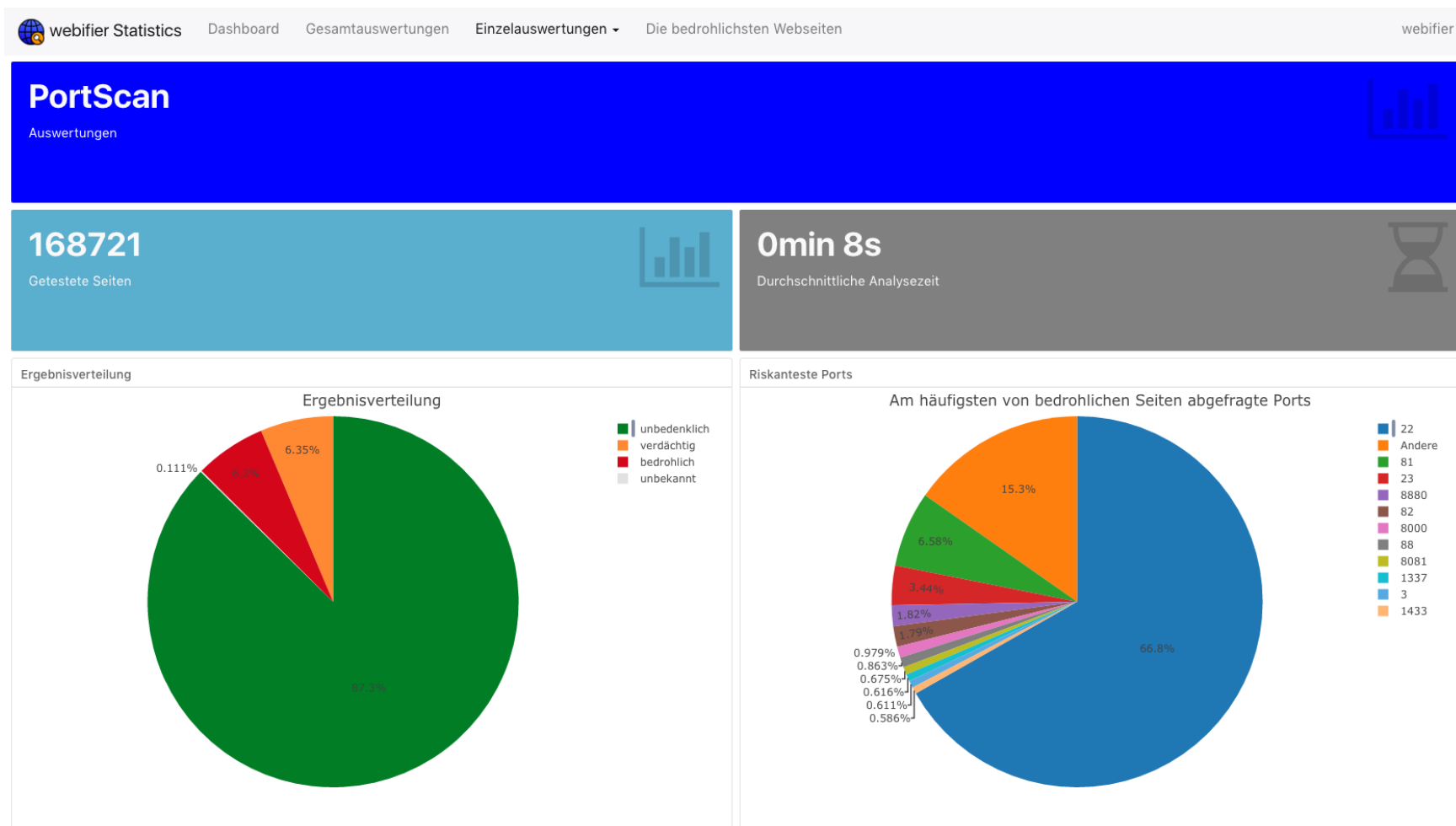


Abbildung 60: Einzelauswertung: Überprüfung der Port-Nutzung