



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**

15. Mai 2017

Bearbeitungszeitraum

6 Monate

Matrikelnummern

9207236, 3788021, 8296876

Kurs

TINF14B2

Ausbildungsfirma

Fiducia & GAD IT AG

Gutachter der Studienakademie

Dr. Martin Johns

Abstract

Daniel

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 15. Mai 2017

Ort, Datum

Samuel Philipp

Karlsruhe, den 15. Mai 2017

Ort, Datum

Daniel Brown

Karlsruhe, den 15. Mai 2017

Ort, Datum

Jan-Eric Gaidusch

Inhaltsverzeichnis

Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VIII
Listings	IX
1 Einleitung	1
1.1 Hintergrund	1
1.2 Aufgabenstellung	1
1.3 Team	2
1.4 webifier	3
2 Grundlagen	4
2.1 Frontend Technologien und Frameworks	4
2.2 Backend Technologien und Frameworks	5
2.3 Technologien und Frameworks der Tests	8
2.4 Angriffstypen	9
2.4.1 Malware	10
2.4.2 User Agent Sniffing	13
2.4.3 JavaScript Port & IP Scanning	14
2.4.4 Phishing	18
3 Konzept	22
3.1 Gesamtkonzept	22
3.1.1 webifier Tests	22
3.1.2 webifier Tester	24
3.1.3 webifier Plattform	26
3.1.4 webifier Mail	26

3.1.5	webifier Data	27
3.1.6	webifier Statistics	27
3.2	Testarten	28
3.2.1	Virensan der Webseite	28
3.2.2	Vergleich in verschiedenen Browsern	29
3.2.3	Überprüfung der Port-Nutzung	29
3.2.4	Überprüfung der IP-Nutzung	29
3.2.5	Prüfung aller verlinkten Seiten	30
3.2.6	Google Safe Browsing	30
3.2.7	Überprüfung des SSL-Zertifikats	30
3.2.8	Erkennung von Phishing	31
3.2.9	Screenshot der Seite	32
4	Umsetzung	33
4.1	Gesamtanwendung	33
4.1.1	webifier Tests	33
4.1.2	webifier Tester	34
4.1.3	webifier Plattform	37
4.1.4	webifier Mail	45
4.1.5	webifier Data	46
4.1.6	webifier Statistics	49
4.2	Tests	51
4.2.1	Virensan der Webseite	51
4.2.2	Vergleich in verschiedenen Browsern	51
4.2.3	Überprüfung der Port-Nutzung	52
4.2.4	Überprüfung der IP-Nutzung	52
4.2.5	Prüfung aller verlinkten Seiten	53
4.2.6	Google Safe Browsing	53
4.2.7	Überprüfung des SSL-Zertifikats	53
4.2.8	Erkennung von Phishing	53
4.2.9	Screenshot der Seite	54
5	Analyse	55
5.1	Gesamtauswertungen	56
5.2	Einzelauswertungen	59
5.2.1	Virensan der Webseite	61
5.2.2	Vergleich in verschiedenen Browsern	61

5.2.3	Überprüfung der Port-Nutzung	62
5.2.4	Überprüfung der IP-Nutzung	62
5.2.5	Prüfung aller verlinkten Seiten	63
5.2.6	Google Safe Browsing	63
5.2.7	Überprüfung des SSL-Zertifikats	64
5.2.8	Erkennung von Phishing	64
5.3	Bewertung der Ergebnisse	64
6	Ausblick	65
6.1	Weitere Tests	65
6.2	Weitere Module	65
7	Fazit	66
	Literaturverzeichnis	XII
	Anhang	XVI

Abkürzungsverzeichnis

WWW World Wide Web

HTML Hypertext Markup Language

CSS Cascading Style Sheets

UI User Interface

JVM Java Virtual Machine

API Application Programming Interface

DRY Don't Repeat Yourself

REST Representational State Transfer

URI Uniform Ressource Identifier

NIDS Network Intrusion Detection System

CLI Command Line Interface

JSON JavaScript Object Notation

XML Extensible Markup Language

Abbildungsverzeichnis

1	Secutitysquad - Logo	2
2	webifier - Logo	3
3	Malware Verbreitung in Deutschland	12
4	Verbreitung neuer Malware in Deutschland	12
5	3-Step-Handshake TCP	17
6	PayPal Phishing Webseite	20
7	PayPal Original Webseite	21
8	webifier Platform - Startseite	38
9	webifier Platform - Ergebnisseite	39
10	webifier Platform - Virensan der Webseite	39
11	webifier Platform - Vergleich in verschiedenen Browsern	40
12	webifier Platform - Überprüfung der Port-Nutzung	41
13	webifier Platform - Überprüfung der IP-Nutzung	41
14	webifier Platform - Prüfung aller verlinkten Seiten	42
15	webifier Platform - Google Safe Browsing	42
16	webifier Platform - Überprüfung des SSL-Zertifikats	43
17	webifier Platform - Erkennung von Phishing	44
18	webifier Platform - Screenshot der Seite	45
19	Generierte Valuebox	50
20	Webifier Statistics Dashboard	55
21	Erkennungen anhand Top-Level-Domains	56
22	prozentuale Erkennungen anhand Top-Level-Domains	56
23	Verteilung der getesteten Top-Level-Domains	57
24	Bedrohliche Funde visualisiert anhand einer Weltkarte	57
25	Testergebnisverteilung	58
26	Visualisierung der Testzusammenhänge	58
27	Top 10: Die bedrohlichsten Webseiten	59
28	Einzelauswertung: Vergleich in verschiedenen Browsern	59
29	Einzelauswertung: Überprüfung der Port-Nutzung	60

30	Einzelauswertung: Virensan der Webseite	60
31	Virensan der Webseite - Testergebnisverteilung	61
32	Vergleich in verschiedenen Browsern - Testergebnisverteilung	61
33	Überprüfung der Port-Nutzung - Testergebnisverteilung	62
34	Überprüfung der IP-Nutzung - Testergebnisverteilung	62
35	Prüfung aller verlinkten Seiten - Testergebnisverteilung	63
36	Google Safe Browsing - Testergebnisverteilung	63
37	Überprüfung des SSL-Zertifikats - Testergebnisverteilung	64
38	Erkennung von Phishing - Testergebnisverteilung	64

Tabellenverzeichnis

1	Beschreibung der einzelnen Tests	23
2	Gewichtungen der einzelnen Tests	24
3	Zuordnung Testergebnis zu Ergebniswert	25

Listings

1	Beispiel.html	5
2	Beispiel eines simplen Java PortScanners	15
3	Phishing Lockmail	18
4	Result JSON	34
5	Hilfe webifier Tester	34
6	Standardausgabe webifier Tester	34
7	Ausschnitt Konfigurationsdatei webifier Tester	35
8	Ausschnitt Ergebnisberechnung webifier Tester	36
9	Konfigurationsdatei webifier Plattform	37
10	Ausschnitt Inhalt push-Request - webifier Data	46
11	Inhalt check-Request - webifier Data	47
12	Verarbeitung check Methode - webifier Data	47
13	Inhalt check-Response - webifier Data	48
14	Beispieldokument webifierTestResultData - webifier Data	49
15	Beispiel R-Grafik	50
16	Vollständige Konfigurationsdatei webifier Tester	XX
17	Vollständige Ergebnisberechnung webifier Tester	XXIII
18	Beispieldokument webifierSingleTestResultData: Virensan der Webseite - webifier Data	XXIV
19	Beispieldokument webifierSingleTestResultData: Vergleich in verschiedenen Browsern - webifier Data	XXV
20	Beispieldokument webifierSingleTestResultData: Überprüfung der Port-Nutzung - webifier Data	XXVI
21	Beispieldokument webifierSingleTestResultData: Überprüfung der IP-Nutzung - webifier Data	XXVI
22	Beispieldokument webifierSingleTestResultData: Prüfung aller verlinkten Seiten - webifier Data	XXVII
23	Beispieldokument webifierSingleTestResultData: Google Safe Browsing - webifier Data	XXVIII

24	Beispieldokument webifierSingleTestResultData: Überprüfung des SSL-Zertifikats - webifier Data	XXIX
25	Beispieldokument webifierSingleTestResultData: Erkennung von Phishing - webifier Data	XXX

1 Einleitung

Daniel

1.1 Hintergrund

Normale Nutzer sind heutzutage im World Wide Web ein gefragtes Angriffsziel für webbasierte Angriffe. Häufig wird hierfür der Nutzer auf maliziöse 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.

1.2 Aufgabenstellung

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.

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*.¹



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 AG.²

¹ Der Name *SecuritySquad* ist angelehnt an den Titel des US-amerikanischen Actionfilms *Suicide Squad*.

² Vgl. Johns (2017)

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.

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 Interaktion des Benutzers visualisieren. 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.³ 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. z.B. Listing 1 Zeile 5, 10, 11). Innerhalb dieser beiden Tags können wiederum neue

³ Vgl. World Wide Web Consortium (W3C) (2014)

Tags (Zeile), aber auch einfache Textdaten stehen (s. Zeilen 4, 7, 8, 9). Diese Verschachtelung führt dazu, dass meist ein komplexer Baum von Tagelementen 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.⁴

```
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

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 Maschine (JVM) ausgeführt wird. Java ist eine Hochsprache, die mit Hilfe des so genannten „Garbage Collectors“ eine automatische Speicherverwaltung bereitstellt.⁵

⁴ Vgl. Jackson (2007), S. 57

⁵ Vgl. Gosling u. a. (2014), S. 1

Daniel
schrei-
ben: CSS

Daniel
schrei-
ben: Ja-
vaScript

Daniel
schrei-
ben:
jQuery

Daniel
schrei-
ben:
Boot-
strap

In einigen Teilprojekten wurde das auf Java basierende *Spring*-Framework verwendet. *Spring* stellt eine vereinfachte Möglichkeit auf den Zugriff auf viele Application Programming Interface (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.⁶

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.⁷ 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 lauffähige 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.⁸

Spring MVC

Spring MVC ist sehr gut geeignet um Webanwendungen zu implementieren.⁹ Hierfür können diese in mehrere Abstraktionsschichten gegliedert werden. Beispielsweise in das User Interface (UI), die Geschäftslogik und die Persistenzschicht.¹⁰

Spring Data

Spring Data vereinfacht Datenbankzugriffe ungemein. Das Modul stellt APIs für fast alle gängigen Datenbankzugriffsschichten, wie JDBC (Java Database Connec-

⁶ Vgl. Wolff (2011), S. 2

⁷ Vgl. Cosmina (2016), S. 2

⁸ Vgl. Gutierrez (2016), S. 1

⁹ Vgl. Wolff (2011), S. 3

¹⁰ Vgl. Yates u. a. (2006), S. 21

tivity), 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.¹¹

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.¹²

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.¹³

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.¹⁴ 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.¹⁵

¹¹ Vgl. Pollack u. a. (2012), S. 3f

¹² Vgl. Chodorow/ Dirolf (2010), S. 1f

¹³ Vgl. Wollschläger (2014), S. 1ff

¹⁴ Vgl. Wikipedia (2017)

¹⁵ Vgl. Baumann (2013), S. 6f

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 Resource Identifier (URI) adressiert werden können. Über die HTTP-Methoden GET, PUT, POST und DELETE können diese Ressourcen geladen, erstellt, geändert oder auch gelöscht werden.¹⁶

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.¹⁷

2.3 Technologien und Frameworks der Tests

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

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 einheitliche Version verwendet wird. Python zählt zu den dynamisch typisierten Programmiersprachen, was bedeutet, dass es wie bei JavaScript2.1 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.¹⁸

Daniel schreiben: Python

diesen Nebensatz in Retrospektive, als Punkt zur Verbesserung?

Daniel schreiben: Phantom JS

¹⁶ Vgl. itwissen.info (2017)

¹⁷ Vgl. Roden (2017)

¹⁸ Python Software Foundation (2017)

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 alle HTML-Links entsprechend zu modifizieren. Abschließend bietet *HTTrack* umfassende Konfigurationsoptionen um es für den optimalen Gebrauch anpassen zu können.¹⁹

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.²⁰

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).²¹ Zudem ermöglicht *Bro* dem Nutzer den Netzwerktraffic zu loggen und mittels eigener Skriptsprache zu filtern.²² Die Logging-Möglichkeiten werden für die Analyse des Traffics genutzt um mögliche verdächtige Abfragen zu erkennen.

2.4 Angriffstypen

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

¹⁹ Vgl. Roche/ Kauler (2017)

²⁰ Vgl. Cryer (2017)

²¹ Vgl. Ali A. Ghorbani (2009), S. 199

²² *Bro Network Monitor* (2017)

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.²³

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.²⁴

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.²⁵ 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.²⁶ 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.²⁷ 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.²⁸ 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.²⁹

²³ Vgl. Kappes (2013), S. 95

²⁴ Vgl. ebenda, S. 95 f.

²⁵ Vgl. ebenda, S. 95

²⁶ Vgl. Aycock (2006), S. 14

²⁷ Vgl. ebenda, S. 12 f.

²⁸ Vgl. Kappes (2013), S. 95

²⁹ 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.³⁰ Solche Daten können beispielsweise Benutzernamen und Passwörter, E-Mailadressen, Bankaccounts und Kreditkartennummern oder Softwarelizenzen sein. Mitte der 1990er Jahre war erste Spyware zu finden.³¹ Als Adware werden Programme bezeichnet, welche dem Benutzer Werbeanzeigen einblenden.³² 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.³³ 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.³⁴ Am häufigsten werden Zombie-Rechner eingesetzt um Spam zu versenden oder mit vielen anderen Denial of Service Angriffe auszuführen.³⁵ 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.³⁶

Wie Abbildung 3 zeigt nimmt die Verbreitung von Malware in Deutschland weiterhin zu und verliert deshalb nicht an Bedeutung.

³⁰ Vgl. Kappes (2013), S. 95 f.

³¹ Vgl. Aycock (2006), S. 16

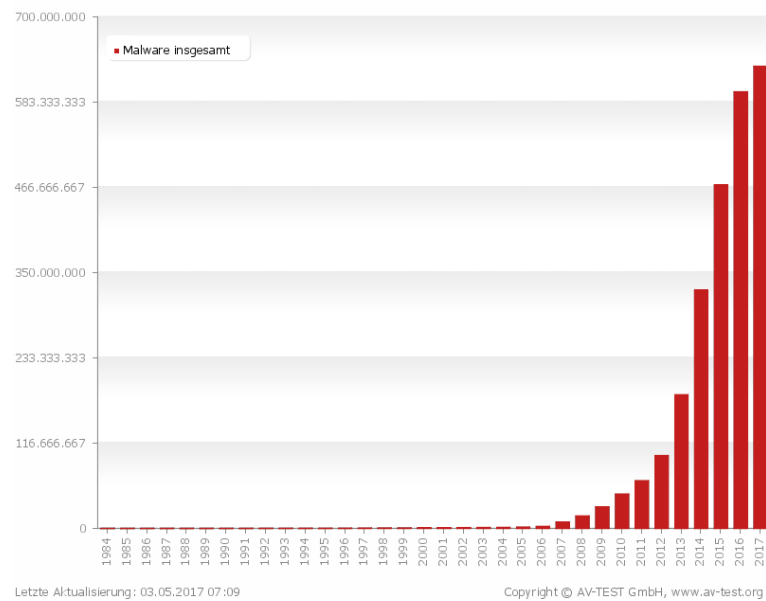
³² Vgl. Kappes (2013), S. 96

³³ Vgl. Aycock (2006), S. 17

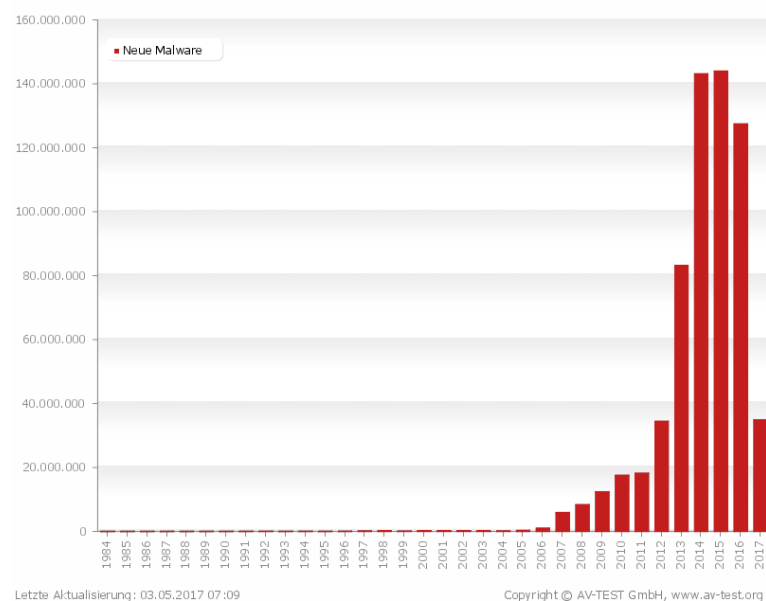
³⁴ Vgl. Kappes (2013), S. 96

³⁵ Vgl. Aycock (2006), S. 18

³⁶ Vgl. Kappes (2013), S. 96

Abbildung 3: Malware Verbreitung in Deutschland³⁷

Interessant ist allerdings dass in Abbildung 4 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 4: Verbreitung neuer Malware in Deutschland³⁸

³⁷ AV-TEST GmbH (2017), Abbildung 1

Die Verbreitung von Malware beginnt größtenteils über Webseiten und E-Mails.³⁹ 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.⁴⁰ HTTP-Requests beinhalten einen Request-Header, der dem Server Auskunft über den Browser des Clients und den gewünschten Ressourcentyp geben kann.⁴¹ 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.⁴²

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.⁴³ 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

³⁸ AV-TEST GmbH (2017), Abbildung 2

³⁹ Vgl. Kappes (2013), S. 97

⁴⁰ Vgl. Gourley/ Totty (2002), S. 19

⁴¹ Vgl. Wong (2000), S. 9

⁴² Vgl. Gourley/ Totty (2002), S. 259, 528 f.

⁴³ 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.⁴⁴ 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.⁴⁵

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.⁴⁶ UDP und TCP sind zwei verschiedene Internet Protokolle. Sie unterscheiden sich zum einen darin, dass TCP verbindungsorientiert arbeitet, während UDP verbindungslose Kommunikation nutzt.⁴⁷ Ein Portscanner nutzt die verschiedenen Eigenschaften der Protokolle aus um festzustellen ob ein Port offen oder geschlossen ist.⁴⁸ 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.

⁴⁴ Vgl. Shepherd (2016)

⁴⁵ Vgl. Harold F. Tipton (2007), S. 937

⁴⁶ Vgl. ebenda, S. 937

⁴⁷ Vgl. nixCraft (2017)

⁴⁸ Vgl. Messier (2016), S. 31

In Listing 2 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 2: Beispiel eines simplen Java PortScanners⁴⁹

⁴⁹ 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 5 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.⁵⁰

⁵⁰ Vgl. Messier (2016), S. 32

TCP Three-Step Handshake

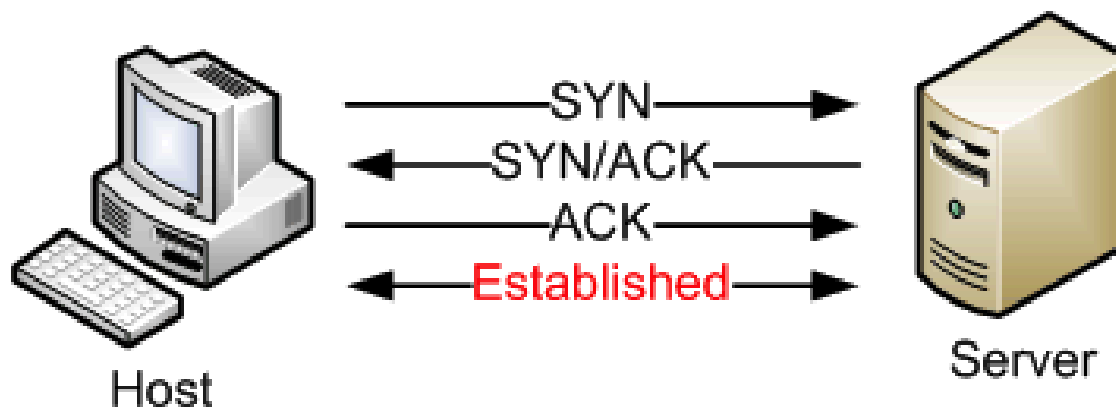


Abbildung 5: 3-Step-Handshake TCP⁵¹

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.⁵²

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. Beispielsweise kann dieser in Image-Tags versteckt werden, welche dann beim Laden der Seite aufgerufen werden.

⁵¹ Aung (2017), Abbildung 1

⁵² Vgl. Messier (2016), S. 33

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.⁵³

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.⁵⁴

Im Folgenden wird ein beispielhafter Phishingangriff auf PayPal geschildert. PayPal ist ein Online-Bezahldienst mit über 18 Millionen Nutzern alleine in Deutschland.⁵⁵ Am häufigsten wird PayPal genutzt um Internetkäufe zu bezahlen. Listing 3 zeigt eine E-Mail, mit der ein PayPal-Nutzer auf eine Phishing-Seite gelockt werden soll.⁵⁶

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.

⁵³ Vgl. Jakobsson/ Myers (2006), S. 1

⁵⁴ Vgl. ebenda, S. 1 f.

⁵⁵ Vgl. PayPal (2017)

⁵⁶ Vgl. Jakobsson/ Myers (2006), S. 10

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

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

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 3: Phishing Lockmail⁵⁷

Die in Listing 3 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 authentisches 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.⁵⁸

⁵⁷ Jakobsson/ Myers (2006), S. 11 Abbildung 1.4, Übersetzung Samuel Philipp

⁵⁸ Vgl. ebenda, S. 10

Üblicher Weise wird auch die Absenderadresse der E-Mail gefälscht und eine original-adresse 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 6 dargestellte Webseite geleitet, welche ihn zur eingabe seiner Anmeldedaten auffordert.⁵⁹

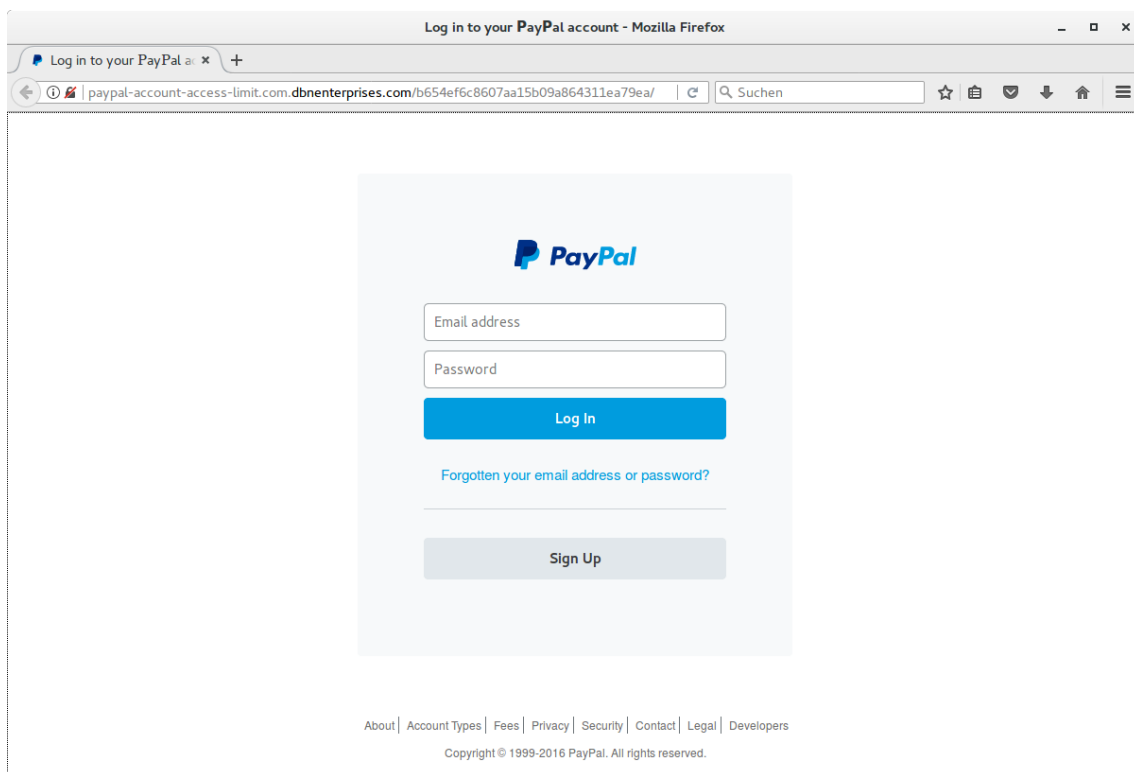


Abbildung 6: PayPal Phishing Webseite

Das Aussehen der Phishing-Webseite ist dem des Originals (Abbildung 7) 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 7) weitergeleitet. Damit ist der Phishing-Angriff abgeschlossen und der Angreifer wird keine Zeit verlieren, die Daten zu missbrauchen.⁶⁰

⁵⁹ Vgl. Jakobsson/ Myers (2006), S. 10

⁶⁰ Vgl. ebenda, S. 10 ff.

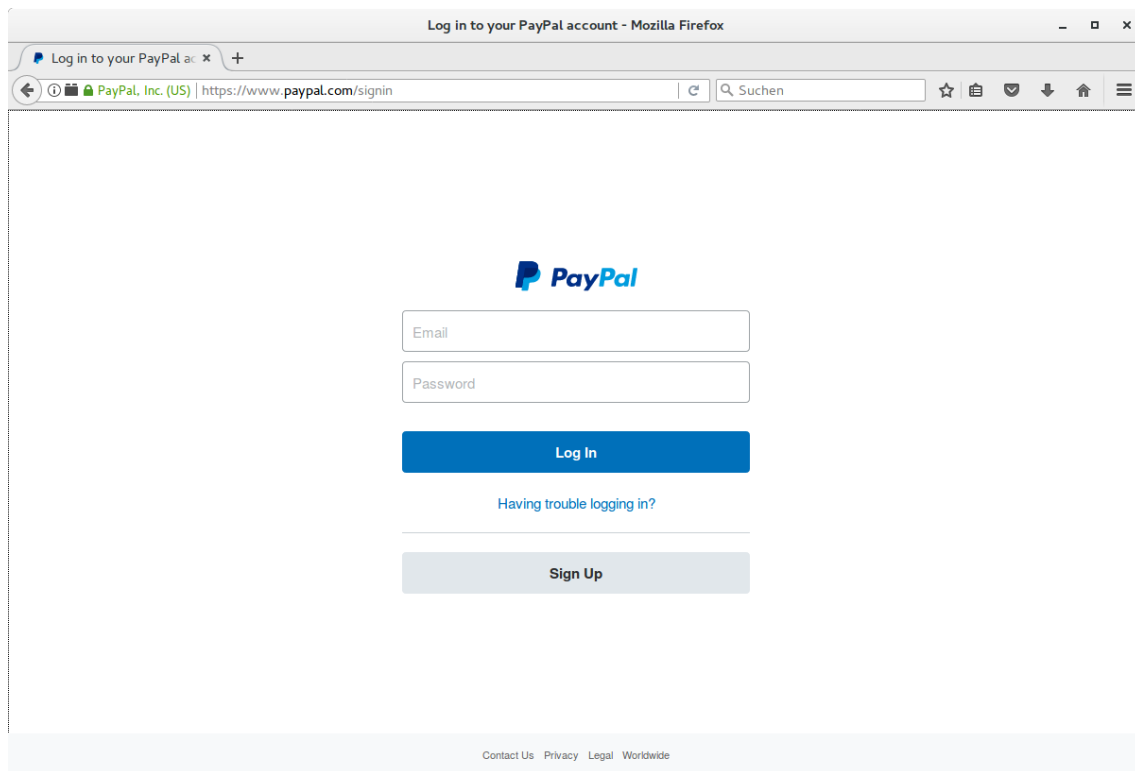


Abbildung 7: PayPal Original Webseite

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

Daniel

- Grafik
- Erklärung der Ergebnistypen
 - unbedenklich (CLEAN)
 - verdächtig (SUSPICIOUS)
 - bedrohlich (MALICIOUS)
 - unbekannt (UNDEFINED)

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.

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 $\frac{\text{Testgewichtung}}{\text{Summe der Gewichtungen aller Tests}}$.

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 Ergebniswert 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:

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

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 Gesamteresultat, 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

3.1.5 webifier Data

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

Ein Testergebnis, welches in dem Datamodul gespeichert wird enthält einmal die eingebene 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 mehrere 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

Daniel

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 das 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

Daniel

- herausfiltern aller Links und nachgeladenen Ressourcen
- Schnittstelle in webifier-data

3.2.6 Google Safe Browsing

Daniel

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 Encrypt⁶¹ 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.

3.2.8 Erkennung von Phishing

webifier enthält auch einen sehr einfachen Tests 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

⁶¹ <https://letsencrypt.org/>

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 werden 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

Daniel

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 Beispiel (...) 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 4: Result JSON

- Beschreiben der Startparameter URL und ID

4.1.2 webifier Tester

Der webifier Tester wurde als Anwendung für das Command Line Interface (CLI) in Java implementiert. Der Tester kann mit Hilfer verschiedener Parameter in seinem Verhalten gesteuert werden. Die Option `-h` gibt beispielsweise die in Listing 5 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 5: Hilfe webifier Tester

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 vor allem 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 6 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 6: Standardausgabe webifier Tester

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 8 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.
      TestVirusScanResultInfo",
17     "weight": 5,
18     "enabled": true
19   }
20   ...
21 ],
22 "preferences": {
23   "push_result_data": true
24 }
25 }

```

Listing 7: Ausschnitt Konfigurationsdatei webifier Tester⁶²

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.

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 ?? 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);

```

⁶² Der vollständige Inhalt der Konfigurationsdatei befindet sich in Anhang B.


```
13     }
14     if (result >= suspiciousMin) {
15         return new WebifierOverallTestResult(WebifierResultType.SUSPICIOUS, result);
16     }
17     return new WebifierOverallTestResult(WebifierResultType.CLEAN, result);
18 }
```

Listing 8: Ausschnitt Ergebnisberechnung webifier Tester⁶³

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. Genauer 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.

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 9 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 9: Konfigurationsdatei webifier Plattform

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

⁶³ Der vollständige Inhalt der Ergebnisberechnung befindet sich in Anhang C.

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 8 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.

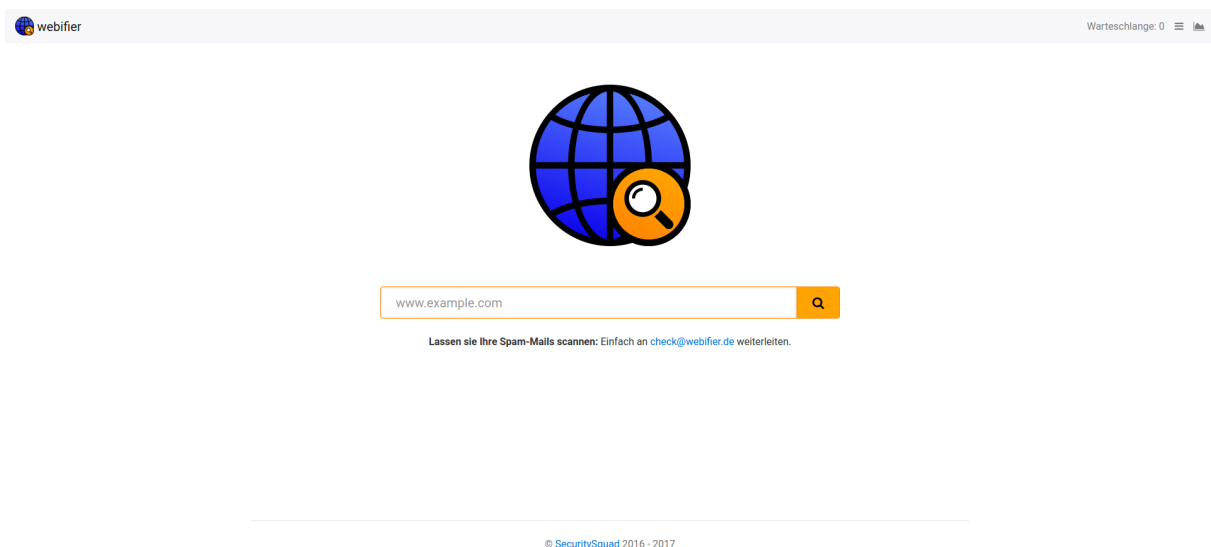


Abbildung 8: webifier Plattform - Startseite

Nachdem die Überprüfung einer Webseite gestartet wurde wird der Nutzer auf die in Abbildung 9 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.

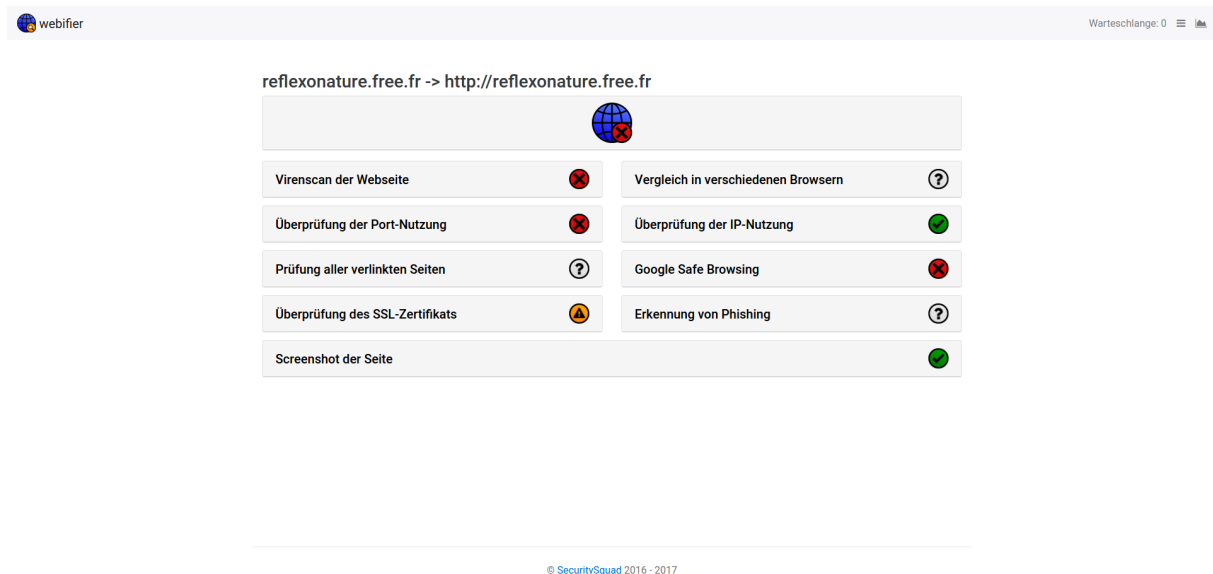


Abbildung 9: webifier Plattform - Ergebnisseite

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.

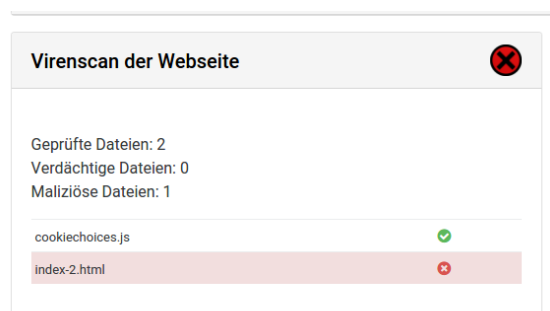



Abbildung 10: webifier Plattform - Virensan der Webseite

Die Detailansicht des Virenskans, welche in Abbildung 10 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.

Vergleich in verschiedenen Browsern			
Durchschnittliche Abweichung:	0 Zeichen	0.00 %	
Maximale Abweichung:	0 Zeichen	0.00 %	
Geprüfte Systeme:			
Android 4.3			
Android 5.1			
iPhone 6, iOS 8.4			
iPhone 6, iOS 9.3			
Windows 10, Chrome 54.0			
Windows XP, Internet Explorer 6.0			
Windows XP, Firefox 4.0			
OS X 10.11, Safari 10.0			
OS X 10.11, Safari 6.0			


Vergleich in verschiedenen Browsern			
Durchschnittliche Abweichung:	303 Zeichen	0.67 %	
Maximale Abweichung:	484 Zeichen	1.43 %	
Geprüfte Systeme:			
Android 4.3			
Android 5.1			
iPhone 6, iOS 8.4			
iPhone 6, iOS 9.3			
Windows 10, Chrome 54.0			
Windows XP, Internet Explorer 6.0			
Windows XP, Firefox 4.0			
OS X 10.11, Safari 10.0			
OS X 10.11, Safari 6.0			

Abbildung 11: 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 11 zu sehen.

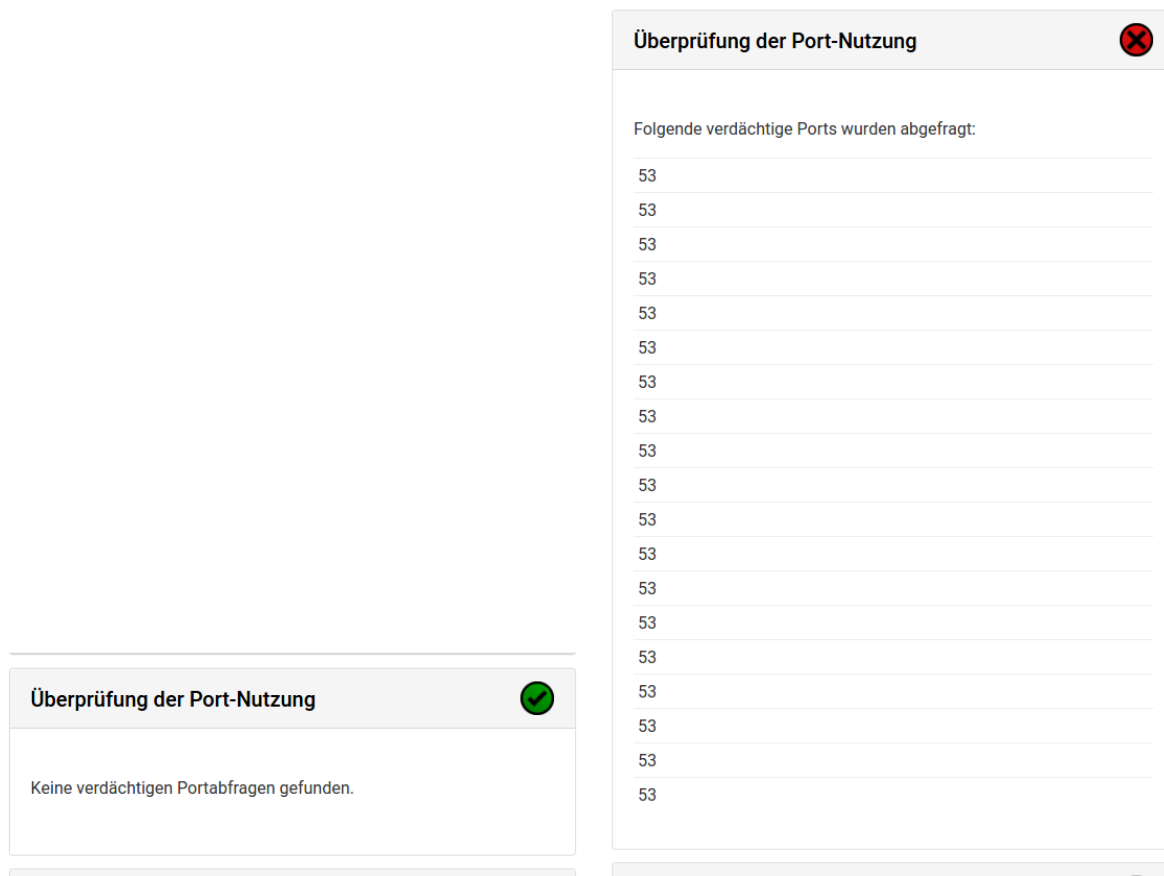


Abbildung 12: webifier Plattform - Überprüfung der Port-Nutzung

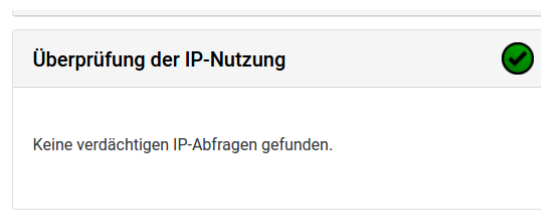


Abbildung 13: webifier Plattform - Überprüfung der IP-Nutzung

Abbildung 12 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 in Abbildung 13 abgebildet. Es stellt bei entsprechenden Funden eine Liste aller IP-Adressen bereit.

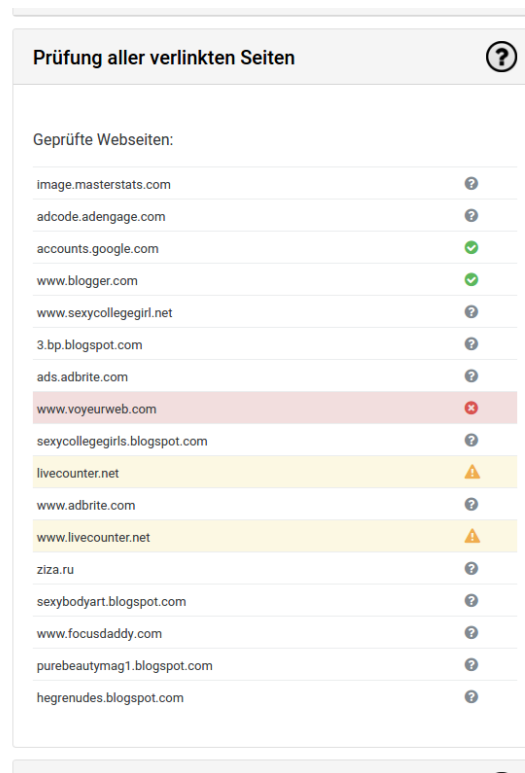


Abbildung 14: webifier Platform - Prüfung aller verlinkten Seiten

Das Ergebnis der Prüfung aller verlinkten Seiten enthält eine einfache Liste mit allen gefundenen Links und dem entsprechenden Ergebnis aus webifier Data. Abbildung 14 zeigt eine solche Liste.

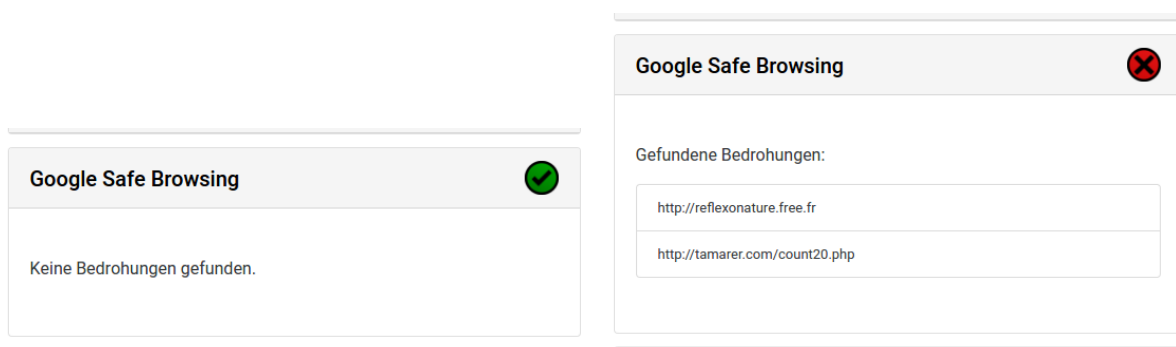


Abbildung 15: webifier Platform - Google Safe Browsing

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 15 enthält zwei Beispielresultate.

Überprüfung des SSL-Zertifikats

Zertifikat:

Ausgestellt für:

Name	securitysquad.de
Organisation	
Organisationseinheit	

Ausgestellt von:

Name	Let's Encrypt Authority X3
Organisation	Let's Encrypt
Organisationseinheit	

Gültigkeitszeitraum:

Gültig ab	Sonntag, 12. März 2017, 02:02
Gültig bis	Samstag, 10. Juni 2017, 03:02

Überprüfung des SSL-Zertifikats

Zertifikat:

Fehler: 10 (certificate has expired)

Ausgestellt für:

Name	*.badssl.com
Organisation	Lucas Garron
Organisationseinheit	

Ausgestellt von:

Name	DigiCert Secure Server CA
Organisation	DigiCert Inc
Organisationseinheit	

Gültigkeitszeitraum:

Gültig ab	Dienstag, 9. Juni 2015, 02:00
Gültig bis	Donnerstag, 5. Januar 2017, 13:00

Abbildung 16: webifier Plattform - Überprüfung des SSL-Zertifikats

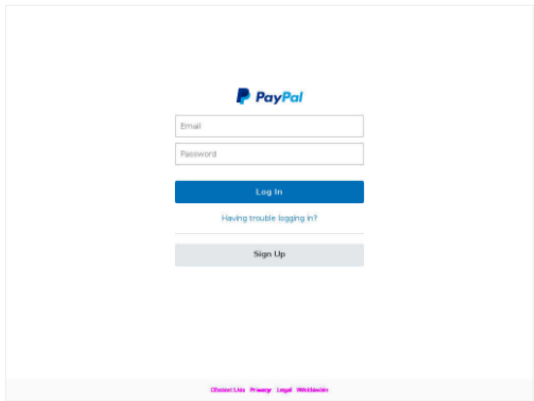
Das Ergebnis der Überprüfung des SSL-Zertifikates enthält alle Informationen des Zertifikats, sofern die Webseite eines nutzt. Wie in Abbildung 16 dargestellt, zeigt die Detailansicht einmal für wen das Zertifikat ausgestellt wurde, aber auch wer es ausgestellt hat. Außerdem wird der Gültigkeitszeitraum des Zertifikats angezeigt und im Fehlerfall der gefundene Fehler.

Erkennung von Phishing

Schlagwörter: paypal, your, account, required

Übereinstimmungen:
<https://www.paypal.com/signin/>

Resultat	
Übereinstimmung	95.05 %
-> Inhalt	100.00 %
-> Quelltext	80.45 %
-> Aussehen	99.88 %



Erkennung von Phishing

Schlagwörter: webifier, securitysquad, jan-eric, gaidusch

Abbildung 17: webifier Plattform - Erkennung von Phishing

Die Erkennung von Phishing stellt ebenfalls einige Informationen zur Verfügung, wie Abbildung 17 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.

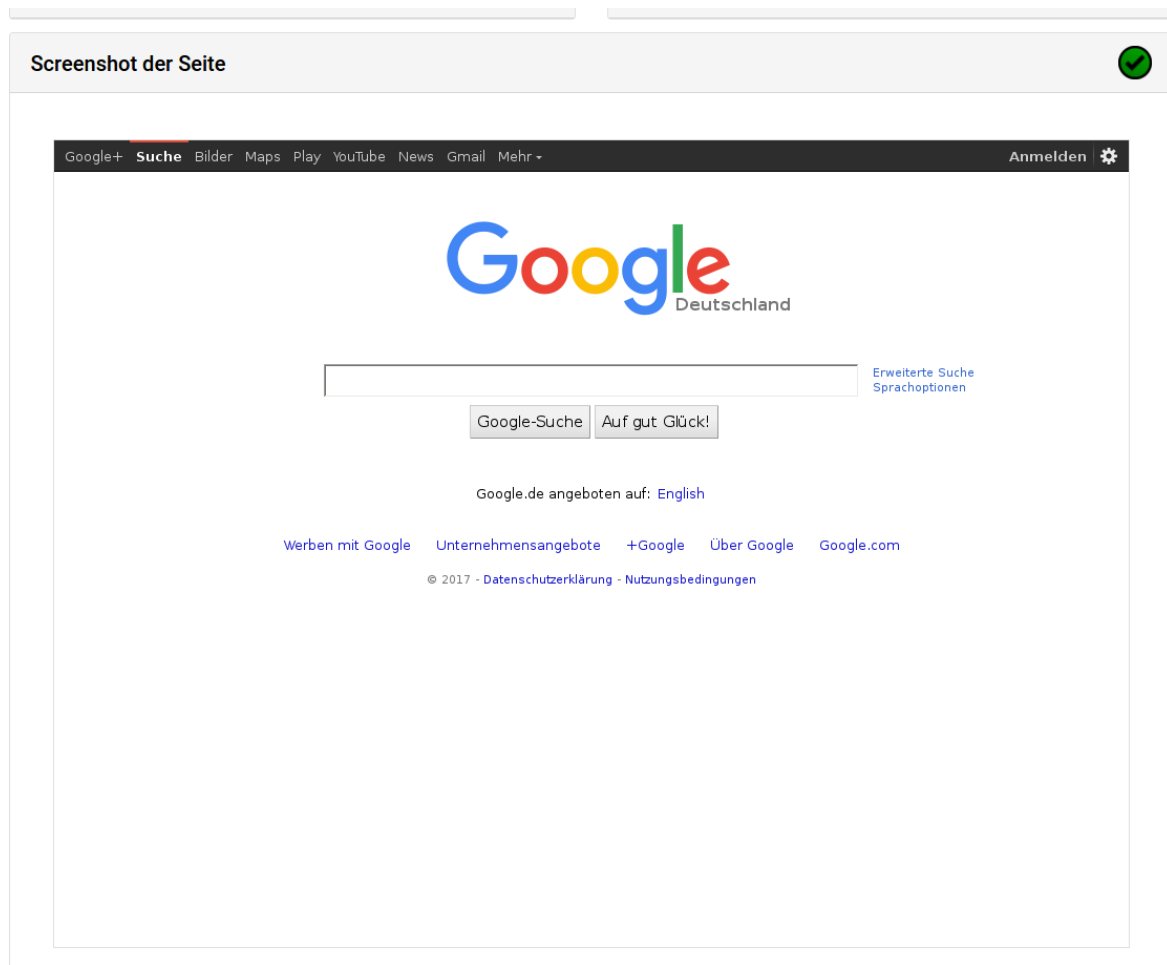


Abbildung 18: webifier Plattform - Screenshot der Seite

Der Screenshot der Seite wird beim Ausklappen des Panels einfach angezeigt, so wie es in Abbildung 18 dargestellt. Dies gibt dem Nutzer die Möglichkeit sich die Webseite anzusehen, ohne sie selbst zu besuchen.

Wie nun gezeigt wird, 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

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 10 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 10: Ausschnitt Inhalt push-Request - webifier Data

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 11 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 11: Inhalt check-Request - webifier Data

Wie Listing 12 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 13 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);
```

```

6         } catch (MalformedURLException e) {
7             return null;
8         }
9     }).distinct().filter(StringUtils::isEmpty).collect(toList());
10    Map<String, WebifierTestResultDetails> hostResults = new HashMap<>();
11    hosts.forEach(host -> {
12        List<WebifierTestResultData> data = dataPersistenceService.getTestResultDataByHost(
13            host)
14            .stream().filter(d -> d.getOverallResultType() != WebifierTestResult.UNDEFINED)
15            .collect(toList());
16        if (data.isEmpty()) {
17            hostResults.put(host, WebifierTestResultDetails.UNDEFINED);
18        } else {
19            double resultValue = data.stream().mapToDouble(this::mapDataResultToIndex).average()
20                .orElse(1);
21            hostResults.put(host, mapResultValueToResult(resultValue));
22        }
23    });
24    if (hostResults.isEmpty()) {
25        return new WebifierCheckTestResultsResponse(false);
26    }
27    return new WebifierCheckTestResultsResponse(true, hostResults);
28 }

```

Listing 12: Verarbeitung check Methode - webifier Data

```

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 13: Inhalt check-Response - webifier Data

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 Einzel-

ergebnisse des Resultats. Deshalb werden pro Testergebnis ein Dokument in `webifierTestData` und acht in `webifierSingleTestData` gespeichert.

Zur Persistierung werden alle Informationen der `\push` Methode (Listing 10) auf die entsprechenden Dokumente gemappt. Listing 14 zeigt die transformierten Daten aus `webifierTestData`. Anhang D enthält die transformierten Dokumente aller Einzelergebnisse in `webifierSingleTestData`.

```

1 {
2   "_id" : "d62e8c94-21c9-4d69-9337-186b38f73cce",
3   "_class" : "de.securitysquad.webifier.persistence.domain.WebifierTestData",
4   "testerId" : "56821bd8-f652-4b59-b7ec-12faa7cf6cae",
5   "enteredUrl" : "ukauktionline.co.uk",
6   "testedUrl" : "http://ukauktionline.co.uk",
7   "host" : "ukauktionline.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("webifierSingleTestData", "5ecb25e6-457b-41ab-9241-a60f8a484131"),
14    DBRef("webifierSingleTestData", "abb70569-64d3-47ae-b3e3-ff8086c8f937"),
15    DBRef("webifierSingleTestData", "56dbbcb3-9628-4dd4-bb20-692bd86c5410"),
16    DBRef("webifierSingleTestData", "64ae09d4-a79e-4194-b77c-3a7891342567"),
17    DBRef("webifierSingleTestData", "05ba8488-4b69-40f9-8a9d-661c09daab65"),
18    DBRef("webifierSingleTestData", "e0ce8955-feb7-4f7a-adfb-b7b8116a9d28"),
19    DBRef("webifierSingleTestData", "9affd5e9-37eb-4d83-b471-ceab0c19155c"),
20    DBRef("webifierSingleTestData", "c2b02bf9-2073-451b-a566-295fe1801e51")
21  ]
22 }
```

Listing 14: Beispieldokument `webifierTestData` - webifier Data

4.1.6 webifier Statistics

Webifier Statistics wird in R implementiert. Hierzu werden Flexdashboards⁶⁴ 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

⁶⁴ Siehe <http://rmarkdown.rstudio.com/flexdashboard/index.html>

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
    =0, limit=Inf)
5  mean.dur <- mean(result$durationInMillis)/1000
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 15: Beispiel R-Grafik

Im Codebeispiel 15 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 19: Generierte Valuebox

In Abbildung 19 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

4.2.1 Virensan der Webseite

Samuel

- Httrack (Umsetzung)
- Download aller Dateien der Webseite
- Scannen der Heruntergeladenen Dateien
 - Clamav (Umsetzung)
 - AVG (Umsetzung)
 - CAV (Umsetzung)

4.2.2 Vergleich in verschiedenen Browsern

Daniel

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 Auf-

ruf 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

Daniel

4.2.6 Google Safe Browsing

Daniel

4.2.7 Überprüfung des SSL-Zertifikats

Samuel

- Auslesen der relevanten Informationen des Zertifikates der Webseite
- Validierung des Zertifikates

4.2.8 Erkennung von Phishing

Samuel

- Herausfiltern der Schlagwörter
- Finden möglicher Duplikate der Webseite
 - Erstes Schlagwort zu Top Level Domains
 - * com

- * ru
- * net
- * org
- * de
- Websuche nach den Schlagwörtern mittels Suchmaschinen
 - * DuckDuckGo
 - * Ixquick
 - * Bing
- Berechnung Teilergebnisse

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

- Daten
 - Welche Listen wurden verwendet?
 - Woher kommen die?
- Statistische Auswertung
 - Gesamtauswertungen
 - Kleine Abschnitte für Einzelauswertungen der Tests
- Diskussion
- Bewertung

Samuel

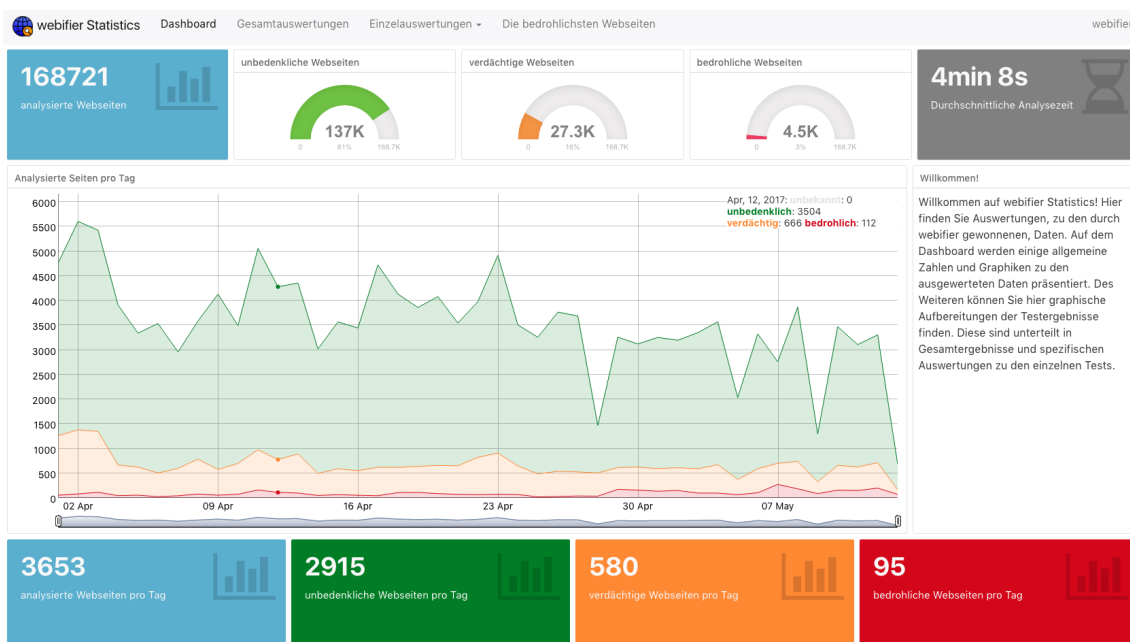


Abbildung 20: Webifier Statistics Dashboard

5.1 Gesamtauswertungen

Jani

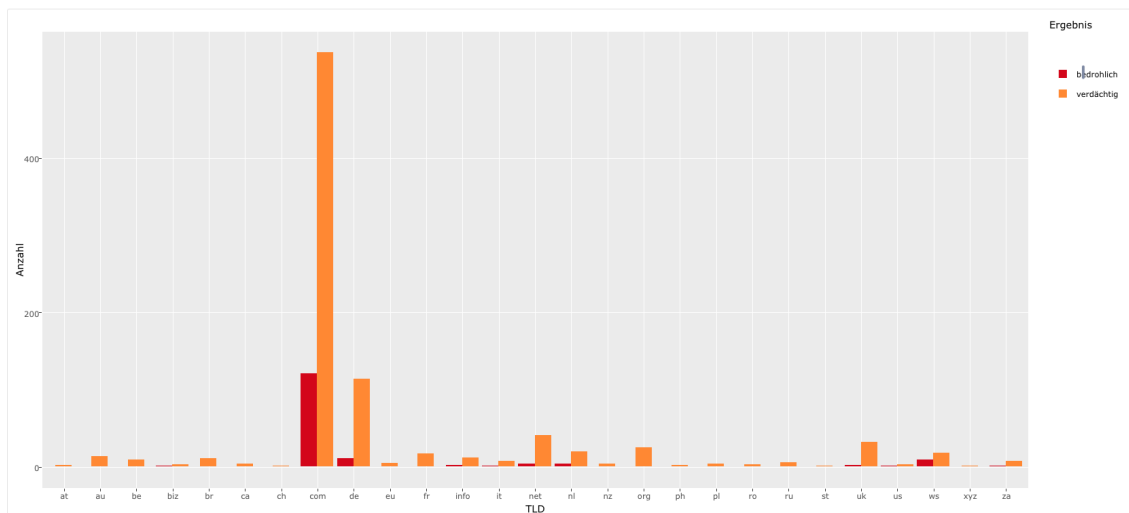


Abbildung 21: Erkennungen anhand Top-Level-Domains

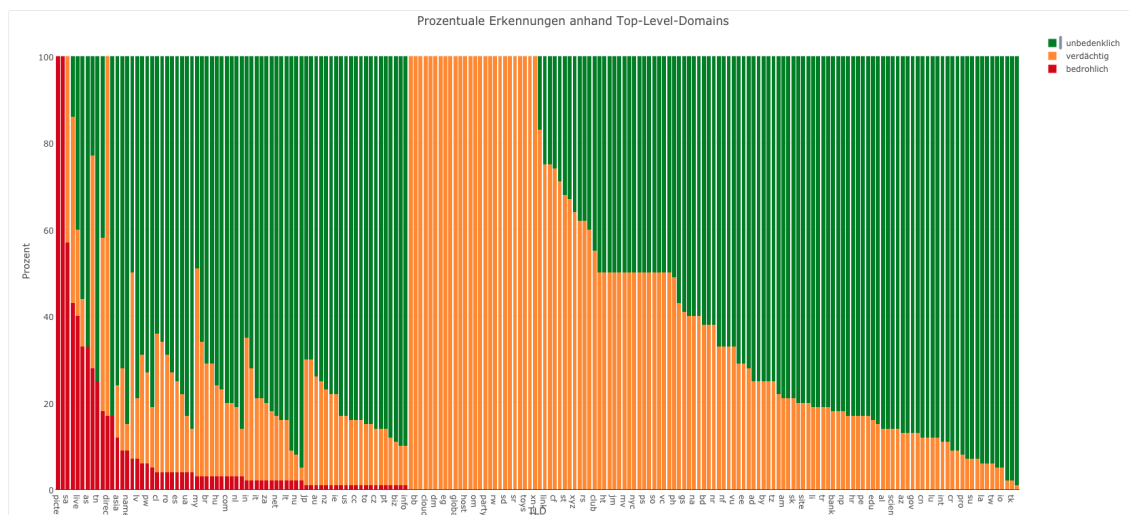


Abbildung 22: prozentuale Erkennungen anhand Top-Level-Domains

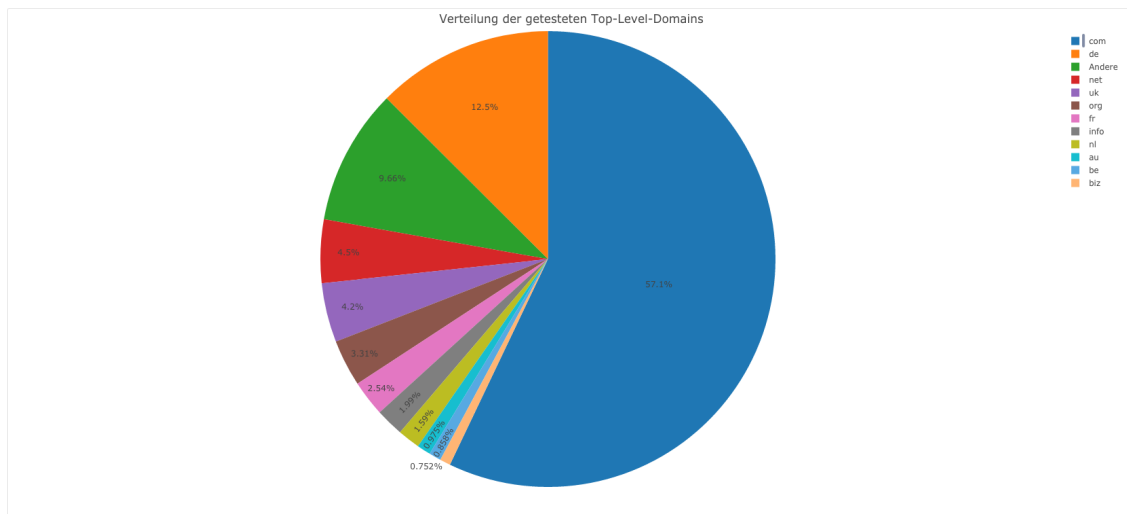


Abbildung 23: Verteilung der getesteten Top-Level-Domains

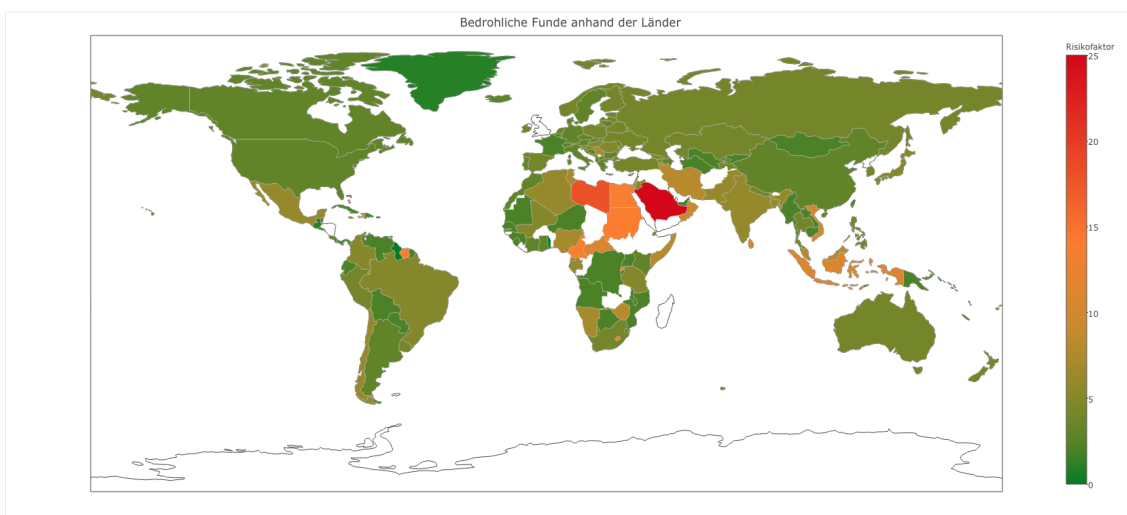


Abbildung 24: Bedrohliche Funde visualisiert anhand einer Weltkarte

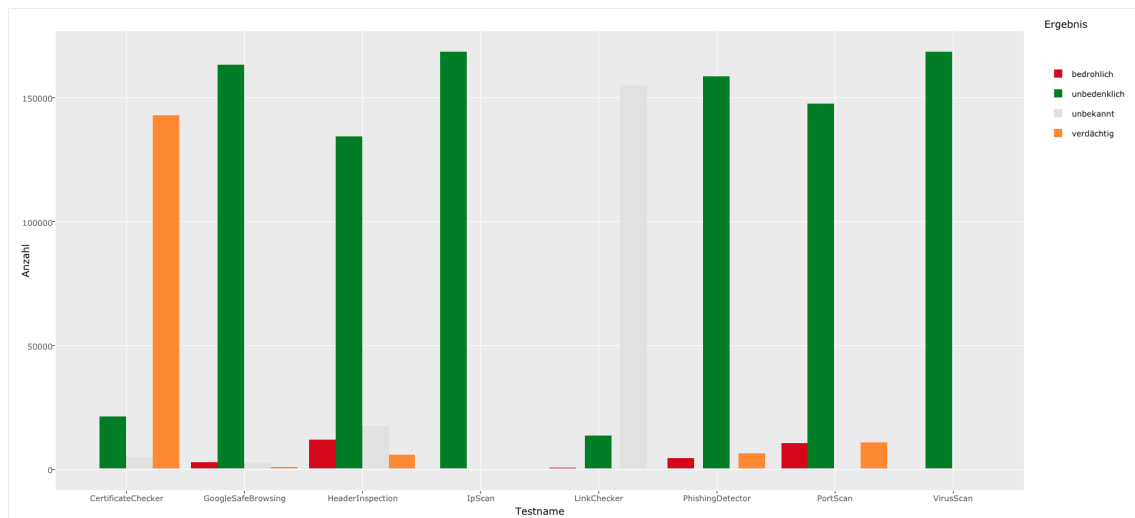


Abbildung 25: Testergebnisverteilung

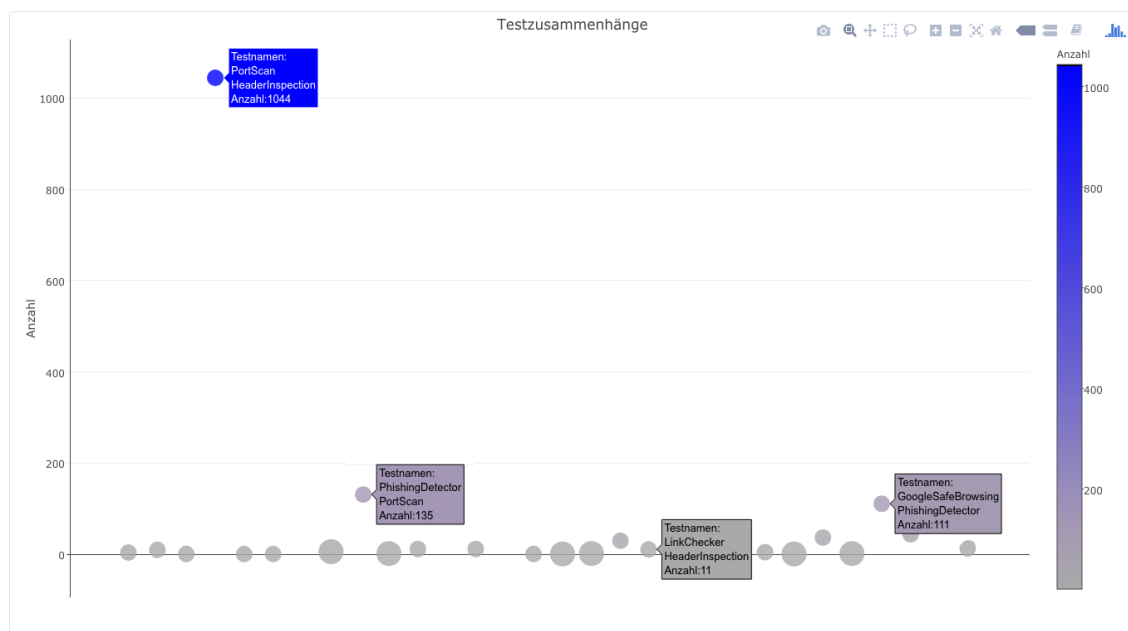


Abbildung 26: Visualisierung der Testzusammenhänge

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.datilddoit.com
9	www.shoe.org
10	www.michaelconley.com

Showing 1 to 10 of 10 entries

Previous 1 Next

Abbildung 27: Top 10: Die bedrohlichsten Webseiten

5.2 Einzelauswertungen

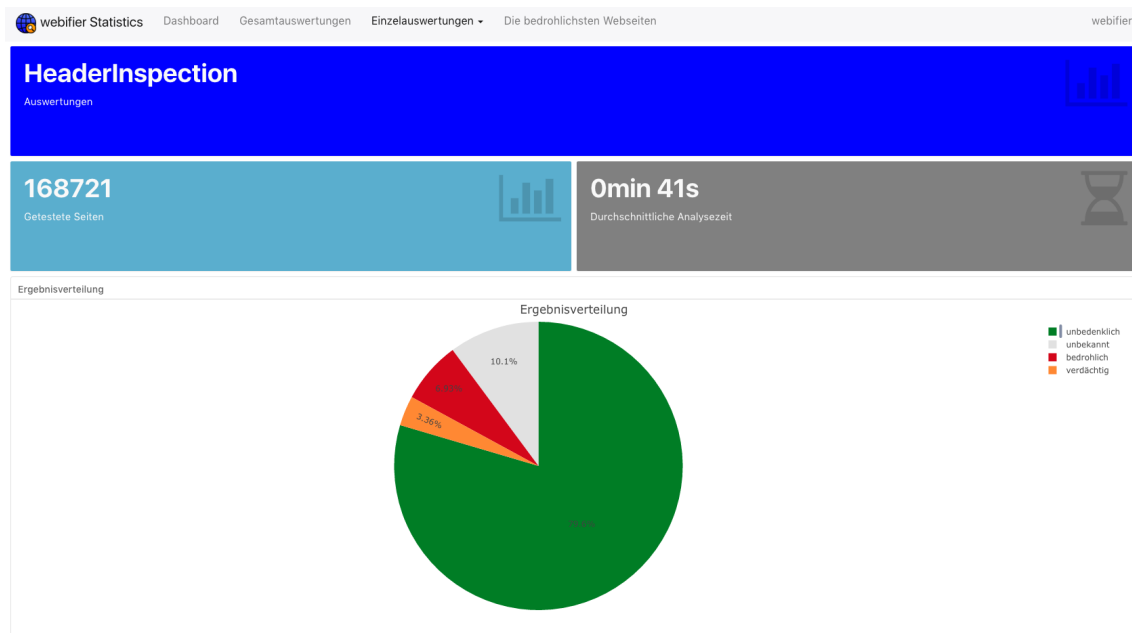


Abbildung 28: Einzelauswertung: Vergleich in verschiedenen Browsern

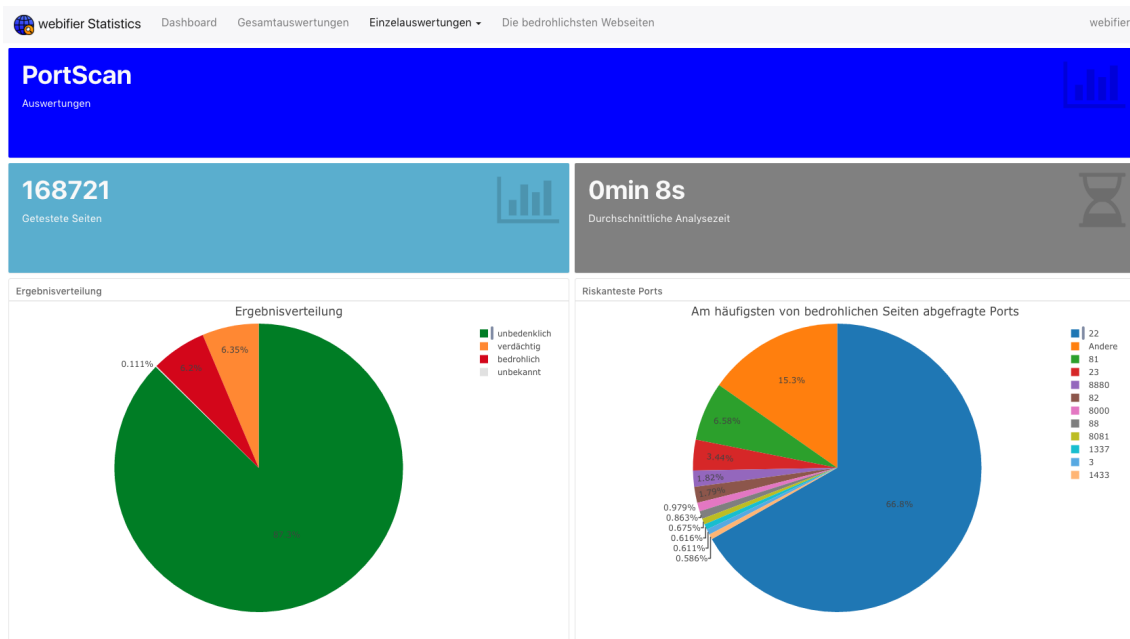


Abbildung 29: Einzelauswertung: Überprüfung der Port-Nutzung

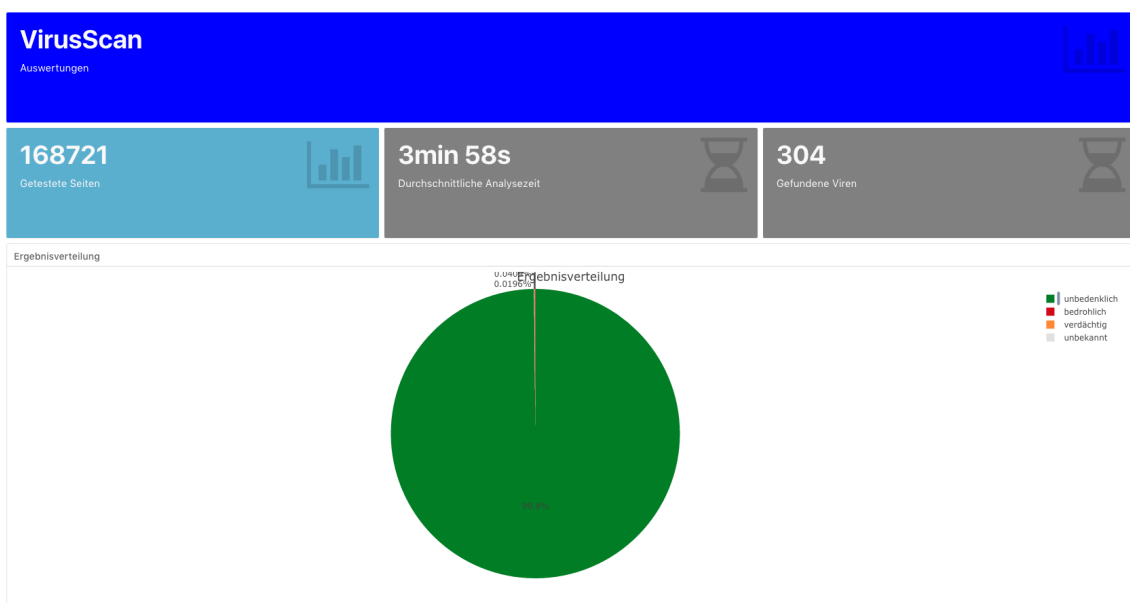


Abbildung 30: Einzelauswertung: Virensan der Webseite

5.2.1 Virensan der Webseite

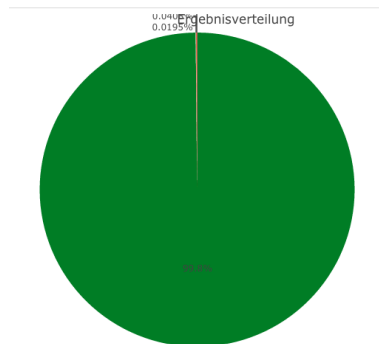


Abbildung 31: Virensan der Webseite - Testergebnisverteilung

Samuel

5.2.2 Vergleich in verschiedenen Browsern

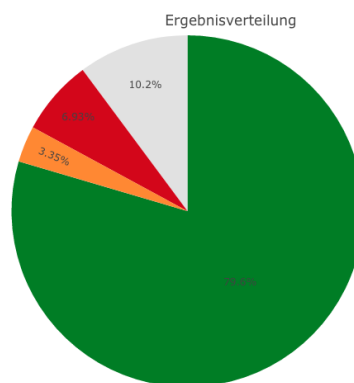


Abbildung 32: Vergleich in verschiedenen Browsern - Testergebnisverteilung

Daniel

5.2.3 Überprüfung der Port-Nutzung

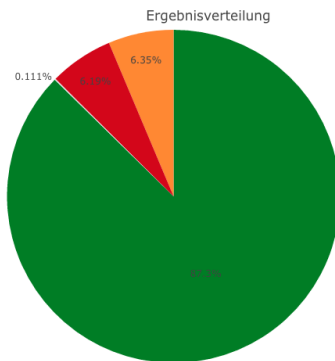


Abbildung 33: Überprüfung der Port-Nutzung - Testergebnisverteilung

Jani

5.2.4 Überprüfung der IP-Nutzung

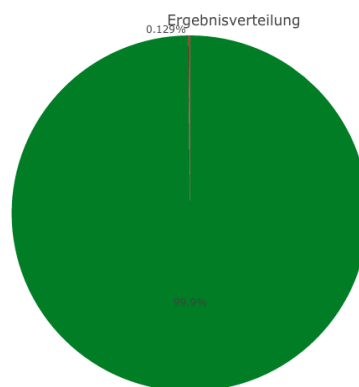


Abbildung 34: Überprüfung der IP-Nutzung - Testergebnisverteilung

Jani

5.2.5 Prüfung aller verlinkten Seiten

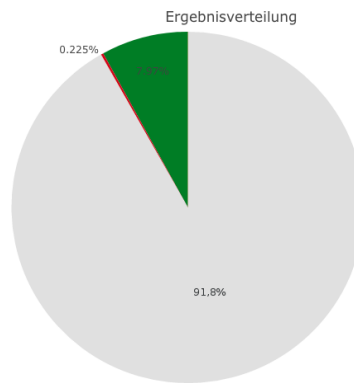


Abbildung 35: Prüfung aller verlinkten Seiten - Testergebnisverteilung

Samuel

5.2.6 Google Safe Browsing

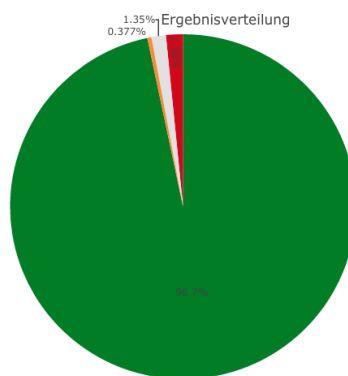


Abbildung 36: Google Safe Browsing - Testergebnisverteilung

Daniel

5.2.7 Überprüfung des SSL-Zertifikats

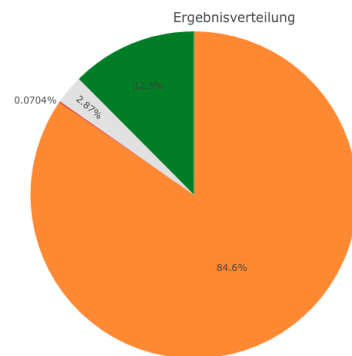


Abbildung 37: Überprüfung des SSL-Zertifikats - Testergebnisverteilung

Samuel

5.2.8 Erkennung von Phishing

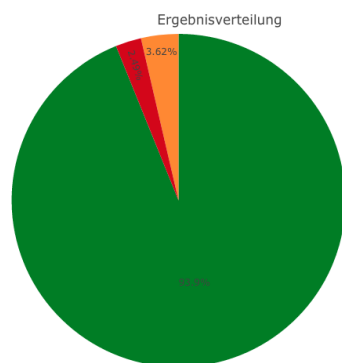


Abbildung 38: Erkennung von Phishing - Testergebnisverteilung

Samuel

5.3 Bewertung der Ergebnisse

6 Ausblick

6.1 Weitere Tests

6.2 Weitere Module

Browserplugin

7 Fazit

Zusammenfassung

Schluss

Literaturverzeichnis

Ali A. Ghorbani Wei Lu, Mahbod Tavallaee (2009):

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

Aung, Ye Htet (2017):

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

Aycock, John (2006):

Computer Viruses and Malware, 1. Auflage, Springer US

Baumann, Joachim (2013):

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

Bro Network Monitor (2017):

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

Chodorow, Kristina/ Michael Dirolf (2010):

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

Cosmina, Iuliana (2016):

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

Cryer, James (2017):

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

Gosling, James u. a. (2014):

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

Gourley, David/ Brian Totty (2002):

HTTP: The Definitive Guide, 1. Auflage, O'Reilly Media, ISBN: 9781565925090

Gutierrez, Felipe (2016):

Pro Spring Boot, 1. Auflage, Apress

Harold F. Tipton, Micki Krause (2007):

Information Security Management Handbook, 6. Auflage, Auerbach Publications

itwissen.info (2017):

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

Jackson, J. C. (2007):

Web Technologies: A Computer Science Perspective, Englisch, Pearson/Prentice Hall, ISBN: 9780131856035, [http://pdfpoint.com/admin/supercategory_content/1469306509-aab008e7ca1d715326928dade3196b2d-Web %20Technologies %20-%20A %20Computer %20Science %20Perspective %20-%20J.%20Jackson%20\(Pearson,%202007\)%20BBS.pdf](http://pdfpoint.com/admin/supercategory_content/1469306509-aab008e7ca1d715326928dade3196b2d-Web%20Technologies%20-%20A%20Computer%20Science%20Perspective%20-%20J.%20Jackson%20(Pearson,%202007)%20BBS.pdf), Einsichtnahme: 03.05.2017

Jakobsson, Markus/ Steven Myers (2006):

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

Johns, Martin (2017):

Martin Johns, www.martinjohns.com, Einsichtnahme: 24.04.2017

Kappes, Martin (2013):

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

Messier, Ric (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

PayPal (2017):

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

Pollack, Mark u. a. (2012):

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

Python Software Foundation (2017):

PhantomJS - Wikipedia, Englisch, Python Software Foundation, <https://www.python.org/>, Einsichtnahme: 21.04.2017

Roche, Xavier/ Leto Kauler (2017):

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

Roden, Golo (2017):

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

Shepherd, Eric (2016):

Browser detection using the user agent - HTTP | MDN, Mozilla Developer Network, https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent, Einsichtnahme: 10.05.2017

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

Wikipedia (2017):

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

Wolff, Eberhard (2011):

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

Wollschläger, Daniel (2014):

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

Wong, Clinton (2000):

HTTP Pocket Reference: Hypertext Transfer Protocol, 1. Auflage, O'Reilly Media, ISBN: 9781449379605

World Wide Web Consortium (W3C) (2014):

PhantomJS - Wikipedia, Englisch, World Wide Web Consortium (W3C), <https://www.w3.org/TR/2014/REC-html5-20141028/single-page.html>, Einsichtnahme: 24.04.2017

Yates, Colin 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 Virenskan

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

3.2.1 Virensan

3.2.2 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

6 Ausblick

6.1 Weitere Tests

6.2 Weitere Module

7 Fazit

7.1 Zusammenfassung

7.2 Bewertung der Ergebnisse

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 16: Vollständige Konfigurationsdatei webifier Tester

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 17: Vollständige Ergebnisberechnung webifier Tester

TEIL D: Vollständiger Inhalt push-Aktion - 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 18: Beispieldokument webifierSingleTestResultData: Virenskan der Webseite - webifier Data

```
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 19: Beispieldokument webifierSingleTestResultData: Vergleich in verschiedenen Browsern - webifier Data

```
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 20: Beispieldokument webifierSingleTestResultData: Überprüfung der Port-Nutzung - webifier Data

```
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 21: Beispieldokument webifierSingleTestResultData: Überprüfung der IP-Nutzung - webifier Data

```
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 22: Beispieldokument webifierSingleTestResultData: Prüfung aller verlinkten
Seiten - webifier Data

```
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 23: Beispieldokument webifierSingleTestResultData: Google Safe Browsing - webifier Data

```
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 24: Beispieldokument webifierSingleTestResultData: Überprüfung des SSL-Zertifikats - webifier Data


```
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 25: Beispieldokument webifierSingleTestResultData: Erkennung von Phishing - webifier Data