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

14. 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 14. Mai 2017	
Ort, Datum	Samuel Philipp
Karlsruhe, den 14. Mai 2017	
Ort, Datum	Daniel Brown
Karlsruhe, den 14. Mai 2017	
Ort, Datum	Jan-Eric Gaidusch

Inhaltsverzeichnis

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

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Einführung

Daniel

1.2 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.

1.3 Aufgabenstellung

Anbieter von zwielichtigen Web-Angeboten greifen ihre User mit diversen Clientseitigen 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 Untersuchungsprozes 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.

Abbildung 1: Secutitysquad - Logo

1.4 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]

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] Johns (2017), S. Vgl.

1.5 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 unabhänig 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 Testergebisse sammeln zu können. Das Statisitik-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 vorgsetellt 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

Daniel schreiben: CSS

Daniel schreiben: JavaScript

Daniel schreiben: jQuery

Daniel schreiben: Bootstrap

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, klassenbarierte und objektorientierte Programmiersprache. Sie wurde möglichst einfach gestaltet um von vielen Entwicklern genutzt zu werden. In ihrer Syntax ähnelt sie den Programmiersprachen C und C++. Außerdem ist sie stark und statisch typisiert. Vorallem aber zeichnet sich Java durch seine plattformunabhängigkeit aus. Diese wird dadurch umgesetzt, dass Java-Quellcode in plattformunabhängigen Byte-Code kompiliert wird, welcher von einer Java Virtual Machine (JVM) ausgeführt wird. Java ist eine Hochsprache, die mit Hilfe des so genannten „Garbage Collectors“ eine automatische Speicherverwaltung bereitstellt.[5]

[4] Vgl. Jackson (2007), S. 57
[5] Vgl. Gosling u. a. (2014), S. 1

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, Sprint Boot, Spring Integration, Spring Data, Spring Session oder Spring MVC.[7] Im folgenden werden die *Spring*-Mudule naäher erläutert, die für das weitere Verständnis der Arbeit notwendig sind.

Spring Boot

Mit Spring Boot können Anwendungen, welche das *Spring*-Framework nutzen, einfacher eintwickelt 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 die 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 stelt 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 compilieren 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 Ressource Identifier (URI) adressiert werden können. Über die HTTP-Methoden GET, PUT, POST und DELETE können diese Ressourcen geladen, erstellt, geändert oder auch gelöscht werden.[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, das 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 läd 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. Urspünglich wurde es für eine Bibliothek von Phanton 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)

Samuel: Noch allgemeiner auf das Thema eingehen...

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 damalign 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 Passworter, E-Mailadressen, Bankaccounts und Kreditkartennummern oder Softwarelizenzschlüssel 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 Webung 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 Kentniss 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 komprommitiert, 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

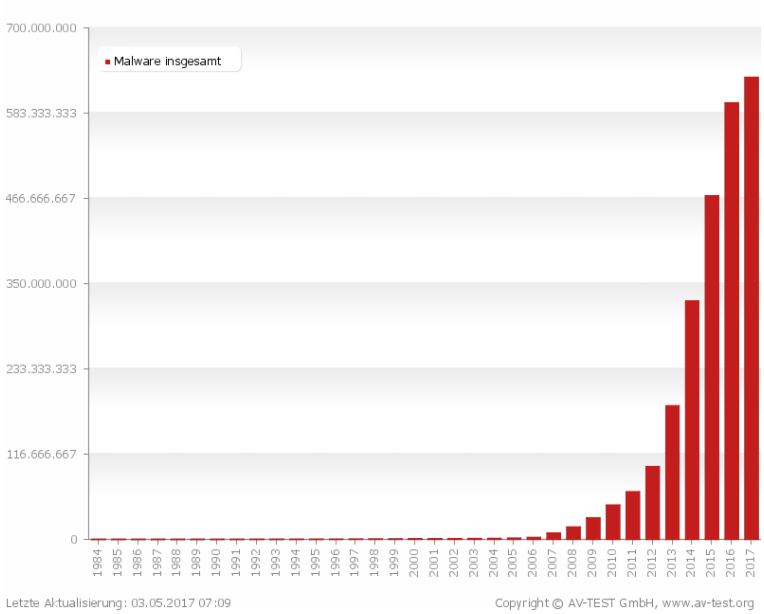

Abbildung 3: Malware Verbreitung in Deutschland[37]

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.

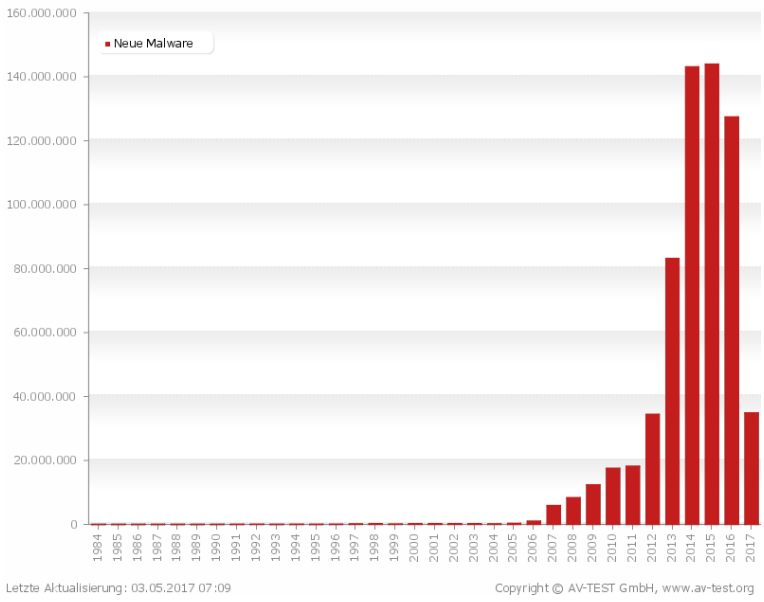

Abbildung 4: Verbreitung neuer Malware in Deutschland[38]

[37] AV-TEST GmbH (2017), Abbildung 1

Die Verbreitung von Malware beginnt größtenteils über Webseiten und E-Mails.[39] Deshalb ist es notwendig mit webifier 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 Pracitces 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 getesten 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
               with a timeout of " + timeout + "ms)");
18     }
19     public static Future<Boolean> portIsOpen(final ExecutorService es, final String ip, final
           int port, final int timeout) {
20         return es.submit(new Callable<Boolean>() {
21             @Override public Boolean call() {
22                 try {
23                     Socket socket = new Socket();
24                     socket.connect(new InetSocketAddress(ip, port), timeout);
25                     socket.close();
26                     return true;
27                 } catch (Exception ex) {
28                     return false;
29                 }
30             }
31         });
32     }
33 }
```

Listing 2: Beispiel eines simplen Java PortScanners[49]

[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

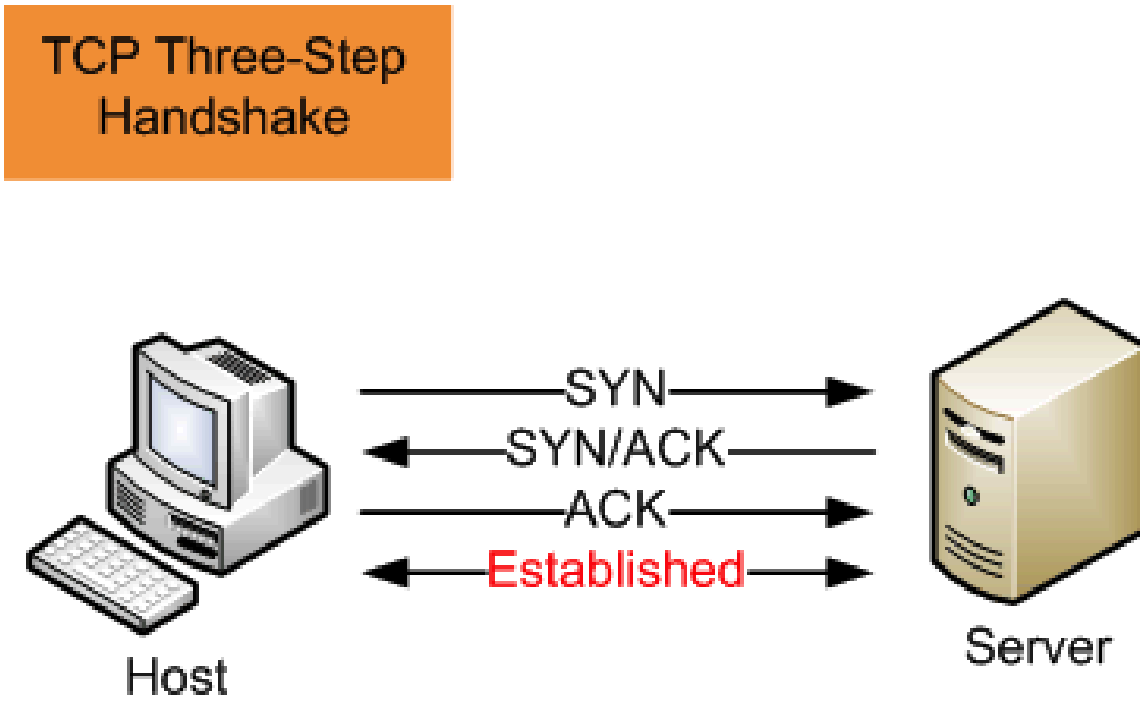

Abbildung 5: 3-Step-Handshake TCP[51]

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 aufgeruft 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 erfolgreicheren 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 gibt 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 Krefitkartennummern 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 Engeneering 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
ungewöhlichen 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 enzige 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 \enquote{Hilfe}-Link im Menü.

PayPal E-Mail ID PP321
```

Listing 3: Phishing Lockmail[57]

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 vermeindlich 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 orginale PayPal-Webseite verweist, sondern auf die Phishing-Seite des Angreifers. Der Hinweiß „Schützen Sie Ihr Passwort“ verleiht der E-Mail noch ein authentischeres Aussehen und würde der Nutzer dem Rat folgen wäre dieser Phishing-Angriff wirkungslos. Viele Nutzer nehmen diesen Rat auch wahr, nutzen aber trotzdem den bereitgestellten Link aus der E-Mail, da diese ja offensichtlich von PayPal stammt und deshalb vertrauenswürdig ist.[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 orginaladresse 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]

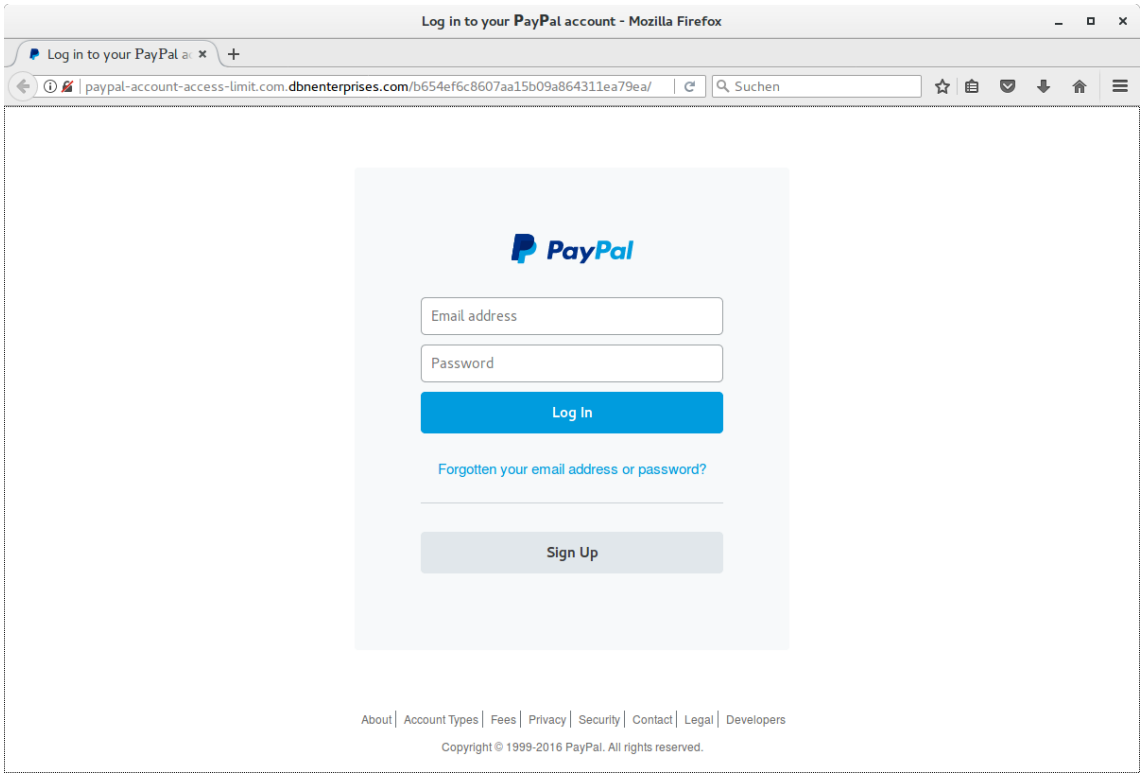

Abbildung 6: PayPal Phishing Webseite

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 orginale 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.

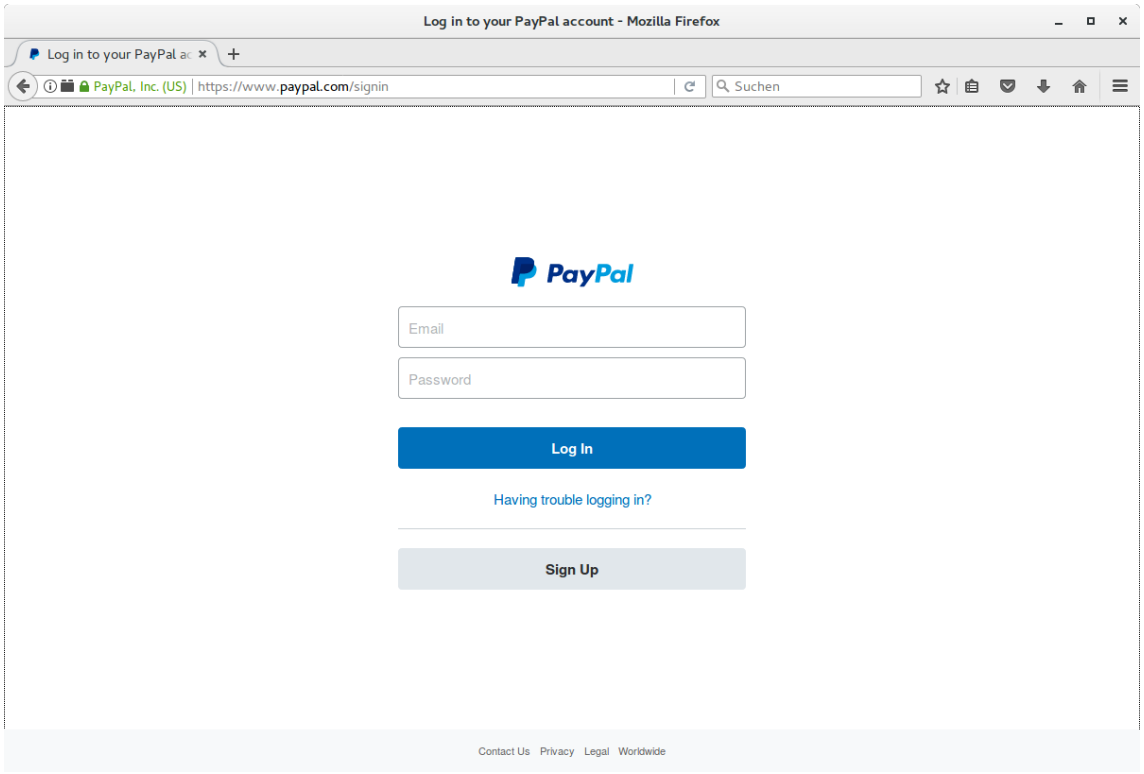

Abbildung 7: PayPal Original Webseite

Das Vorgehen im vorausgegangen 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 Übermittelung 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 Organisisieren 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**
Virenscan 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 Gesamtergebis 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 *Virenscan 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**
Virenscan 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 $\frac{Testgewichtung}{Summe\ der\ Gewichungen\ aller\ Tests}$.

Ein weiterer wichtiger Punkt, der für die Berechung ges 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 Wergebniswert 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**
CLEAN	0
SUSPICIOUS	Prozentuale Gewichtung des Tests
MALICIOUS	1
UNDEFINED	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:

$$0 \leq CLEAN < 0,0434027 \leq SUSPICIOUS < 0,2083 \leq MALICIOUS \leq 1$$

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 Gesamtresultat, welches vom Tester berechnet wurde und zu Anderen sowohl die Ergebnisse der einzelen 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

Daniel

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 Virenscan der Webseite

Der Virenscan der Webseite führt nutzt verschiedene Virenscanner 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 Virenscanner 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 Virenscanner als Malware eingestuft wird diese als *CLEAN* gekennzeichnet. Klassifiziert nur ein Virenscanner die Datei als Malware, wird diese also *SUSPICIOUS* eingestuft. Halten mehr als ein Virenscanner eine Datei für Malware ist diese *MALICIOUS*. Sind alle Dateien als *CLEAN* eingestuft, so ist auch das Endergeblis dieses Tests *CLEAN*. Sollten ein oder mehrer Dateien *SUSPICIOUS* sein wird auch das Endergebnis *SUSPICIOUS*. Das selbe gilt danach für *MALICIOUS*. Sobald eine Datei *MALICIOUS* ist, ist das Endergbnis 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 Weibseite sucht nach einem vorhandenen Zertifikat und validiert dieses, sofern die Webseite eines nutzt. Hierfür liest es die dafür notwendigen Informationen des Zertifikats aus und berechnet anschließend ein Testergebnis.

Stellt die Webseite kein Zertifikat zur Verfügung so ist das Testergebnis *SUSPICIOUS*, da es in Zeiten von Let's Encrypt[61] 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 Tertifikat Fehler auf ist das Ergebnis *MALICIOUS*. Solche Fehler können beispielsweise sein, dass das Zertifikat abgelaufen ist, dass es selbst signiert wurde oder das 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 mehere 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 Schüsselwörter 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 Orginalen, welcher der gegebenen Webseite entsprechen gefilterert 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 Eintrage gekürtzt.

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 er der Quelltext der beiden Webseiten, aber auch Screenshots verglichen. Jeder dieser Vergleiche gibt die prozentuale Übereinstimmung der zu vergleichenden Webseiten zurück. Anschließend werden diese drei Ergebnisse zu einem Gesamtergebnis verrechnet. In diese Rechnung fließt das Ergebnis des Screenshotsvergleichs mitt doppeltem Gewicht ein, da dieser Vergleich am aussagekräftigsten ist.

[61] https://letsencrypt.org/

Aufgrund der Komplexität wird die genaue Berechnung des Ergebnisses wird 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, derern 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 mit 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

4.1 Gesamtanwendung

Daniel

4.1.1 webifier Tests

In diesem Kapitel 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 Ergebnisübermittelung der Tests an den Tester wird mittels JSON-Strings realisiert. Wie in Beispiel (...) zu sehen besteht das JSON aus dem Testergebnis und einer ResultInfo. Die ResultInfo varriert 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

Samuel

4.1.3 webifier Plattform

Samuel

4.1.4 webifier Mail

Daniel

4.1.5 webifier Data

Samuel

4.1.6 webifier Statistics

Webifier Statistics wird in R implementiert. Hierzu werden Flexdashboards[62] verwendet. Zur Generierung der Grafiken wurde auf verschiedene Librarys, wie beispielsweise Plot.ly, zurückgegriffen um den Entwicklungsaufwand für die Visualisierungen zu minimieren. Die Anordnung der Grafiken wird über ein bestimmtes Layout definiert. Jede Grafik wird prinzipiell in 3 Schritten erstellt:

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

[62] Siehe http://rmarkdown.rstudio.com/flexdashboard/index.html

```
### Durchschnittliche Analysezeit

```{r}
result <- dbGetQueryForKeys(mg1, 'webifierTestResultData',"{}", "{durationInMillis:1}",skip=0,limit=Inf)
mean.dur <- mean(result$durationInMillis)/1000
mean.dur <- round(mean.dur)
tp <- seconds_to_period(mean.dur)
valueBox(paste(minute(tp),'min ',second(tp),'s',sep=""), icon="fa-hourglass-half",color="grey")
```
```

Listing 5: Beispiel R-Grafik

Im Codebeispiel 5 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 8: Generierte Valuebox

In Abbildung 8 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 Virenscan 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 intialisiert und es werden Filter angelegt um lediglich die Ports, der eingehenden Anfragen, mitzuloggen und in der Logdatei abzuspeichern. Ist Bro vollständig initialisiert und einsatzbereit startet PhantomJS mit dem Aufrufen der Seite und Ausführen des JavaScript-Codes. Währenddessen speichert Bro alle Netzwerkaktivitäten. Sobald der Durchlauf von PhantomJS abgeschlossen ist wird mittels Python die Validierung des Ergebnisses gestartet. Hier werden die angefragten Ports aus der Logdatei geladen und klassifiziert. Die Ports 80 und 443 werden verworfen, da diese die HTTP und SSL Ports sind und somit als harmlos klassifiziert werden können. Die weiteren Ports werden in einer Liste an riskanten Ports gespeichert. Die Anzahl an Ports in dieser Liste bestimmt nun das Ergebnis des Testes. Wurden keine verdächtigen Portanfragen gefunden wird das Ergebnis *unbedenklich* übermittelt. Bei 1 oder 2 Ports in der Liste gibt der Test *verdächtig* als Ergebnis zurück. Sollte die Anzahl größer gleich 3 sein wird die Seite von diesem Test als *bedrohlich* eingestuft. Zusätzlich zum Ergebnis wird die Liste der riskanten Ports in der Ergebnisinformation weitergeleitet.

4.2.4 Überprüfung der IP-Nutzung

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

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

4.2.5 Prüfung aller verlinkten Seiten

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
  - Gesamtauswetungen
  - Kleine Abschnitte für Einzelauswertungen der Tests
- Disskusion
- Bewertung

Samuel

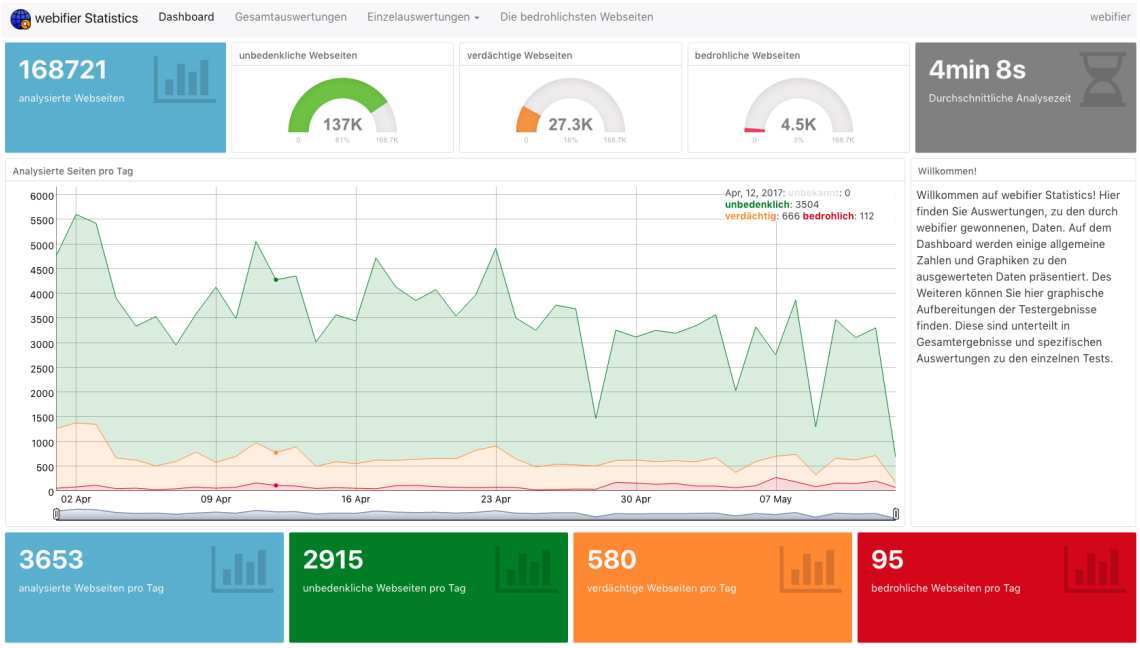

Abbildung 9: Webifier Statistics Dashboard

5.1 Gesamtauswertungen

Jani

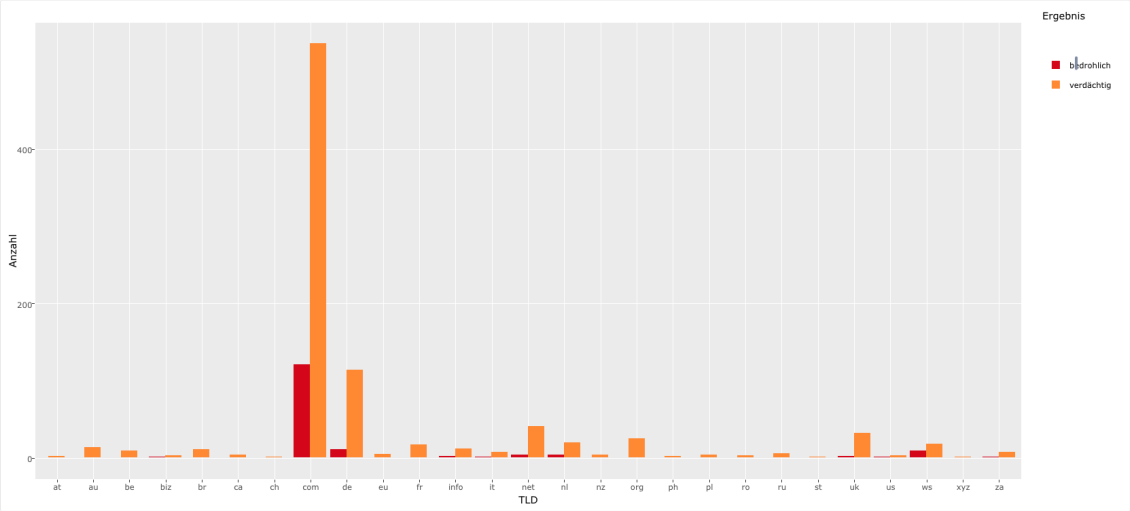

Abbildung 10: Erkennungen anhand Top-Level-Domains

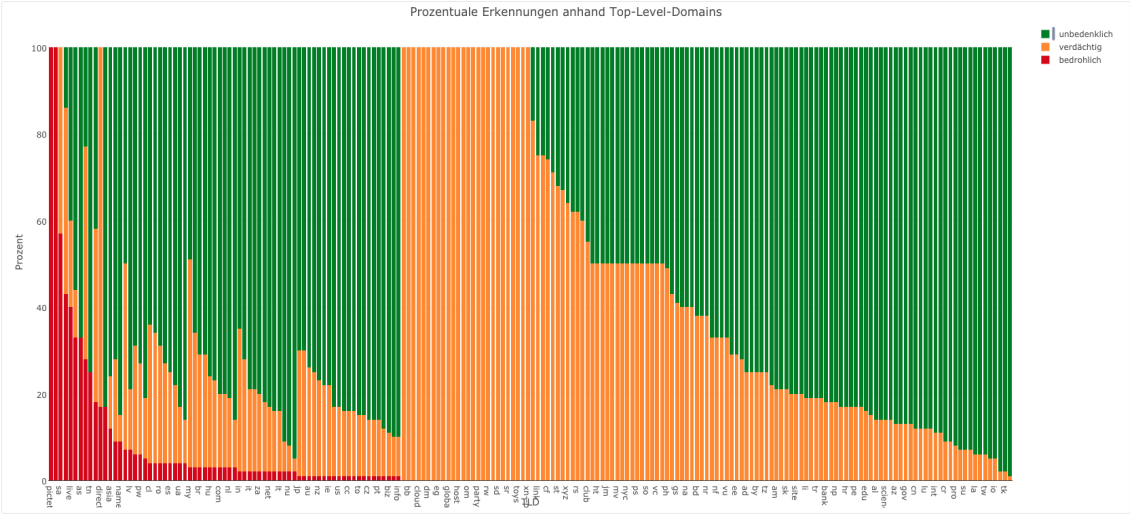

Abbildung 11: prozentuale Erkennungen anhand Top-Level-Domains

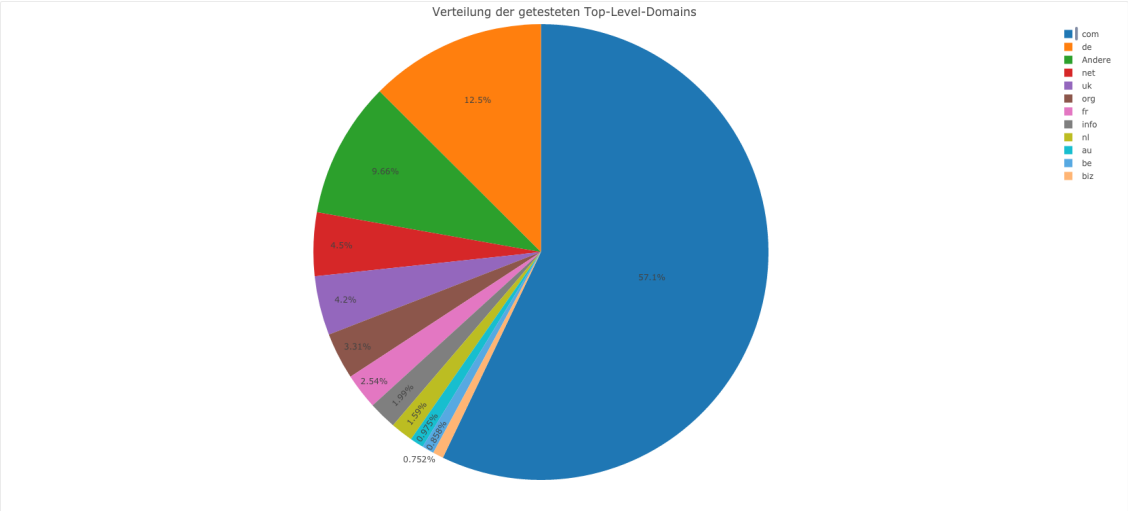

Abbildung 12: Verteilung der getesteten Top-Level-Domains

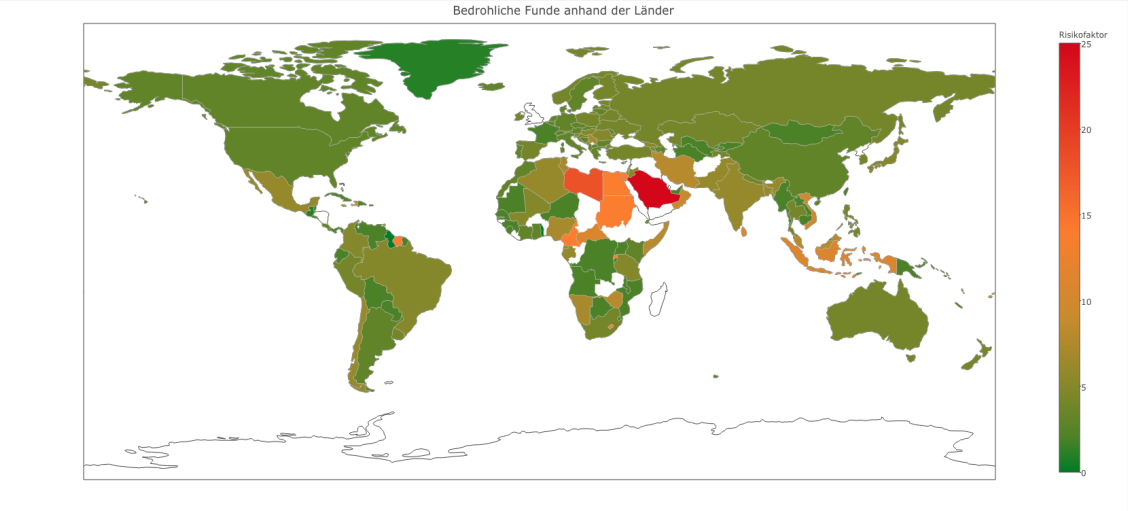

Abbildung 13: Bedrohliche Funde visualisiert anhand einer Weltkarte

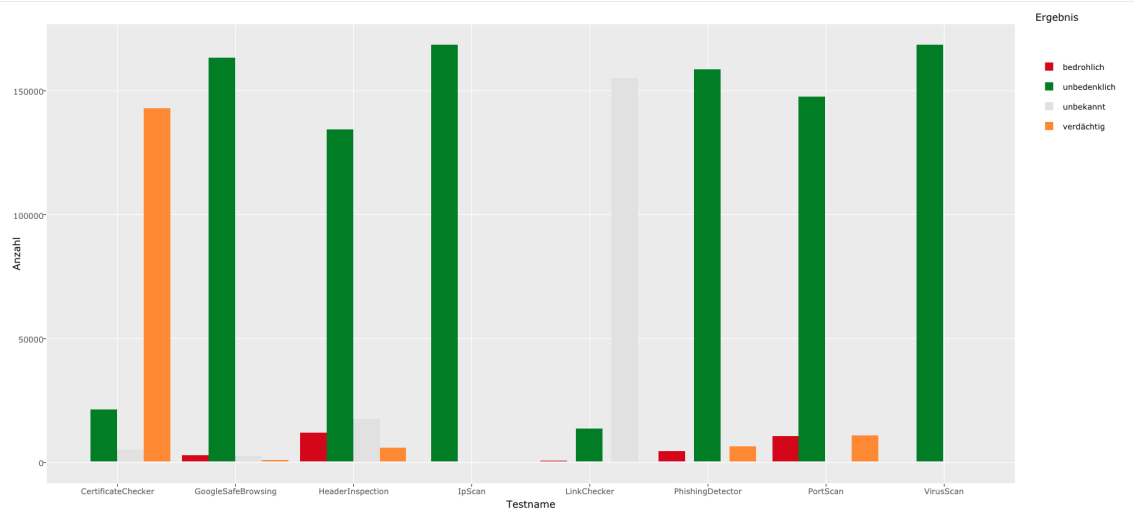

Abbildung 14: Testergebnisverteilung

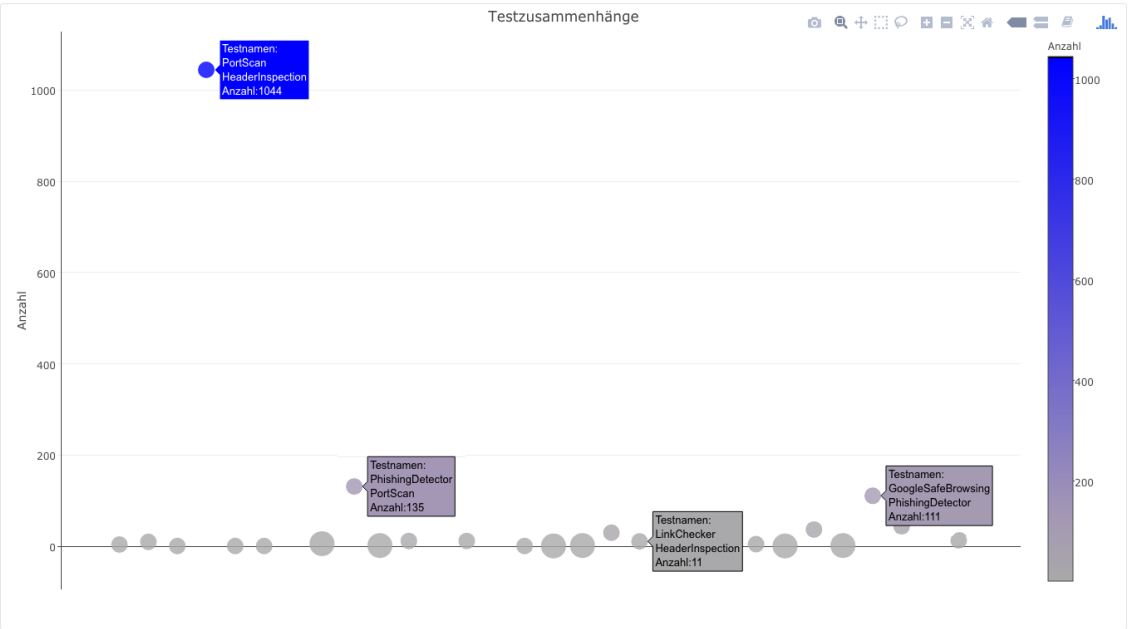

Abbildung 15: Visualisierung der Testzusammenhänge

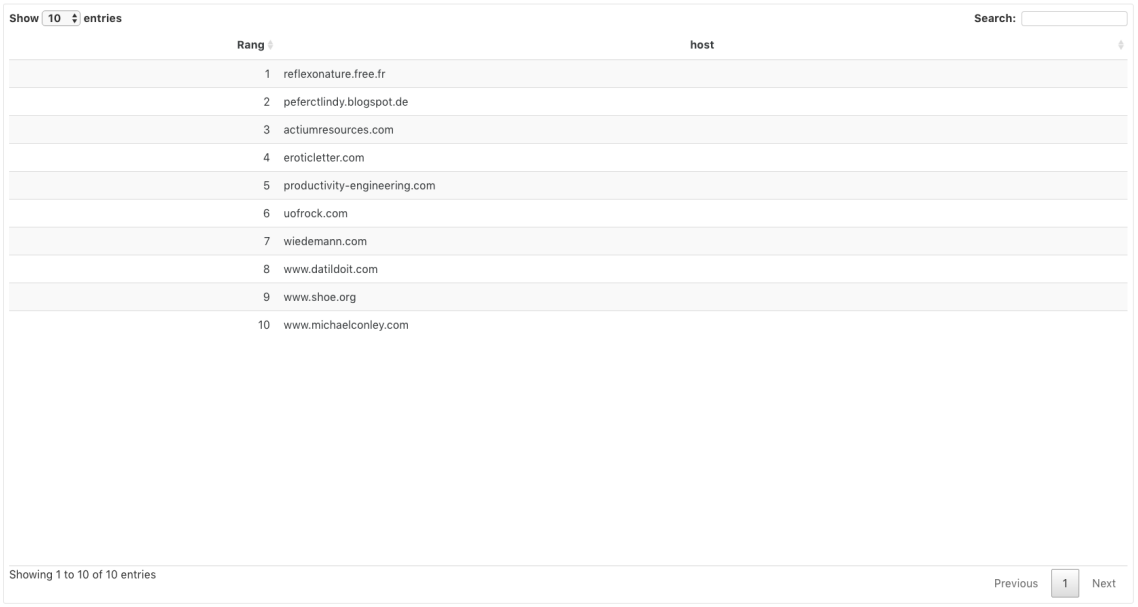

Abbildung 16: Top 10: Die bedrohlichsten Webseiten

5.2 Einzelauswertungen

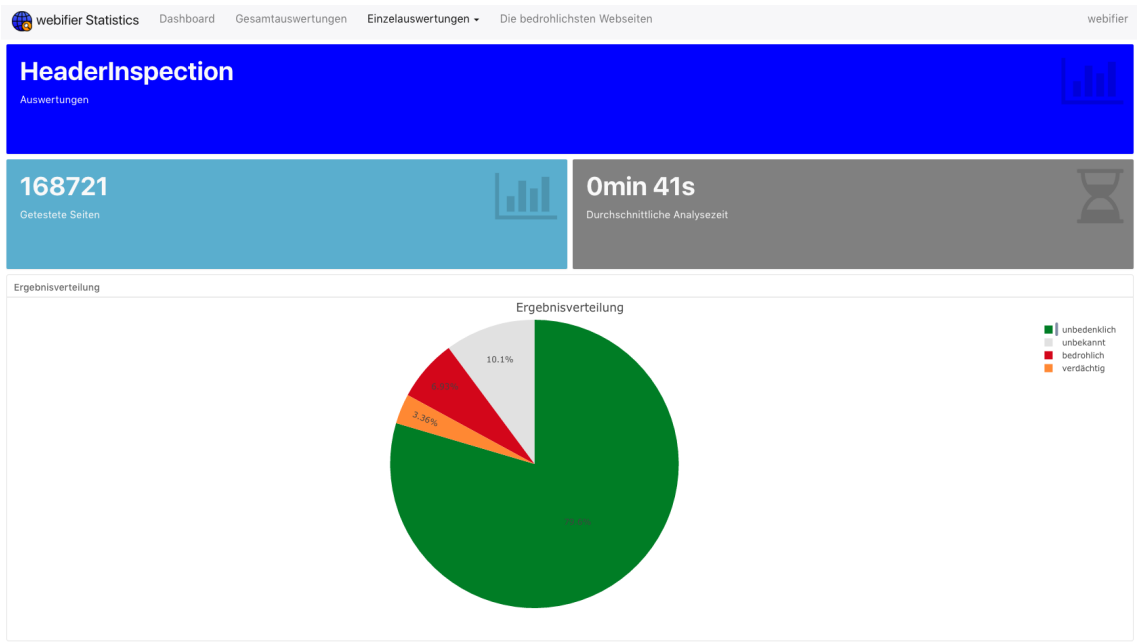

Abbildung 17: Einzelauswertung: Vergleich in verschiedenen Browsern

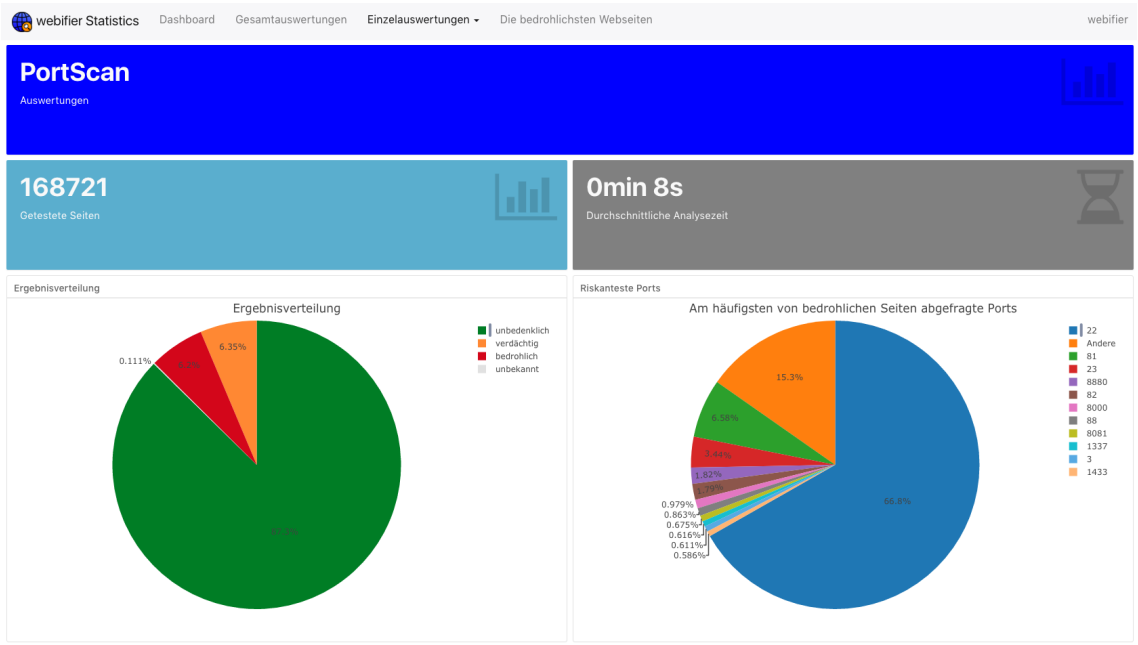

Abbildung 18: Einzelauswertung: Überprüfung der Port-Nutzung

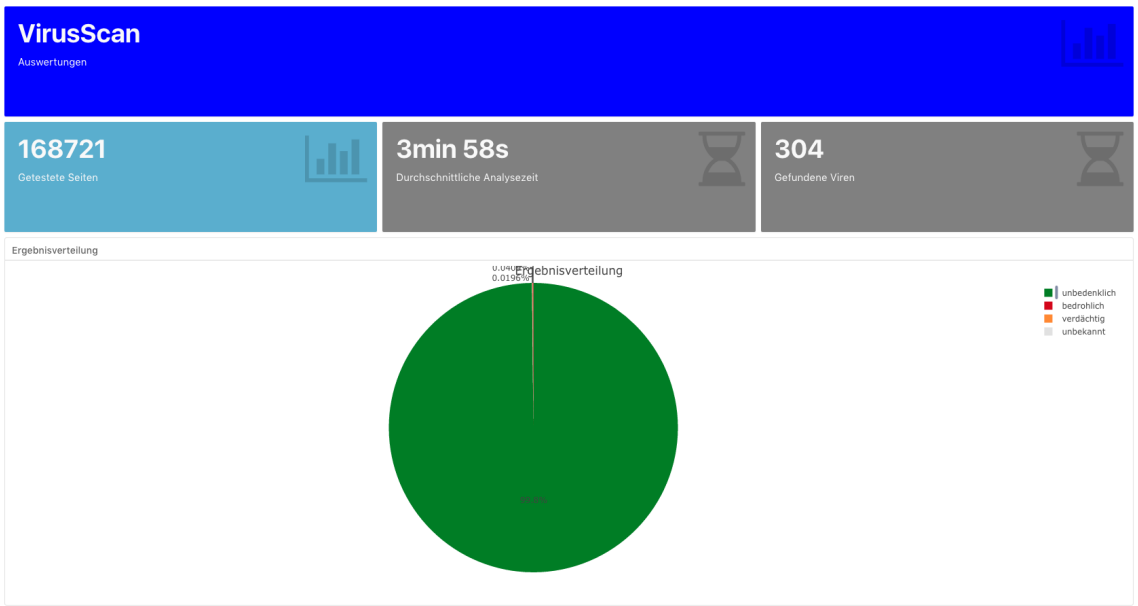

Abbildung 19: Einzelauswertung: Virenscan der Webseite

5.2.1 Virenscan der Webseite

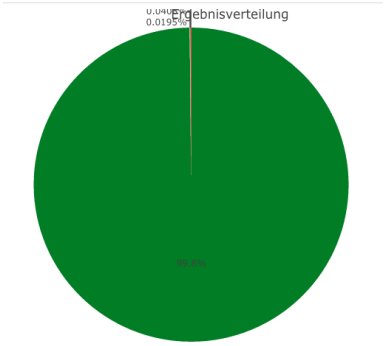

Abbildung 20: Virenscan der Webseite - Testergebnisverteilung

Samuel

5.2.2 Vergleich in verschiedenen Browsern

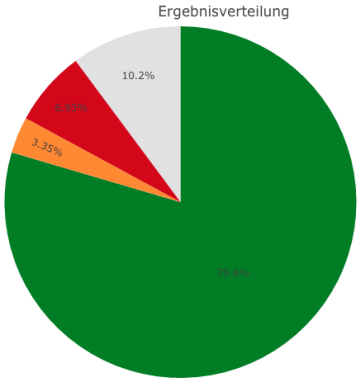

Abbildung 21: Vergleich in verschiedenen Browsern - Testergebnisverteilung

Daniel

5.2.3 Überprüfung der Port-Nutzung

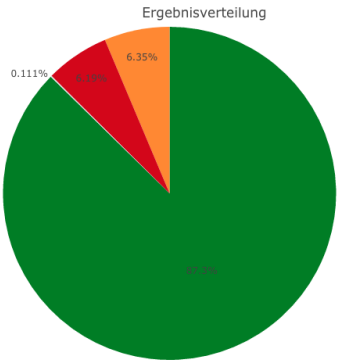

Abbildung 22: Überprüfung der Port-Nutzung - Testergebnisverteilung

Jani

5.2.4 Überprüfung der IP-Nutzung

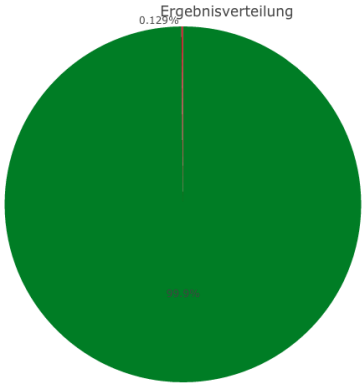

Abbildung 23: Überprüfung der IP-Nutzung - Testergebnisverteilung

Jani

5.2.5 Prüfung aller verlinkten Seiten

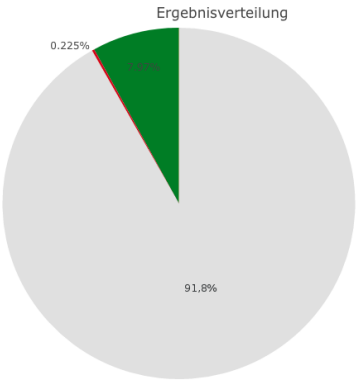

Abbildung 24: Prüfung aller verlinkten Seiten - Testergebnisverteilung

Samuel

5.2.6 Google Safe Browsing

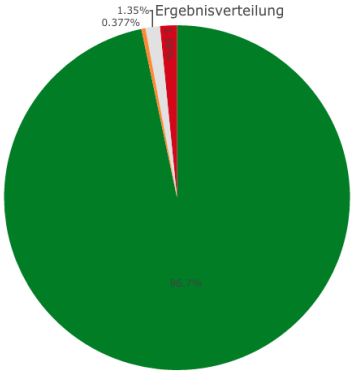

Abbildung 25: Google Safe Browsing - Testergebnisverteilung

Daniel

5.2.7 Überprüfung des SSL-Zertifikats

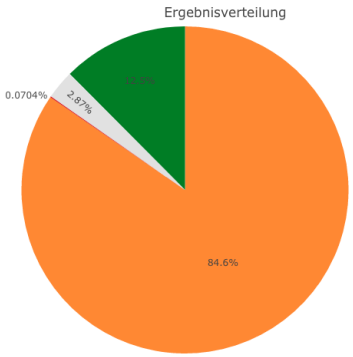

Abbildung 26: Überprüfung des SSL-Zertifikats - Testergebnisverteilung

Samuel

5.2.8 Erkennung von Phishing

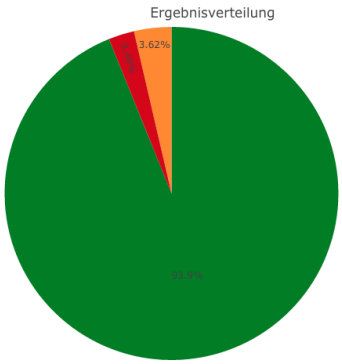

Abbildung 27: Erkennung von Phishing - Testergebnisverteilung

Samuel

5.3 Bewertung der Ergebnisse

6 Ausblick

6.1 Weitere Tests

6.2 Weitere Module

Browserplugin

7 Fazit

7.1 Zusammenfassung

7.2 Bewertung der Ergebnisse

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, 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 Virenscan