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

23. April 2017

Bearbeitungszeitraum	6 Monate
Matrikelnummern	9207236, 3788021, 8296876
Kurs	TINF14B2
Ausbildungsfirma	Fiducia & GAD IT AG
Gutachter der Studienakademie	Dr. Martin Johns

Abstract *TODO* 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 23. April 2017	
Ort, Datum	Samuel Philipp
Karlsruhe, den 23. April 2017	
Ort, Datum	Daniel Brown
Karlsruhe, den 23. April 2017	
Ort, Datum	Jan-Eric Gaidusch

Inhaltsverzeichnis

Abkürzungsverzeichnis

WWW World Wide Web

UI User Interface

JVM Java Virtual Machine

API Application Programming Interface

DRY Don't Repeat Yourself

REST Representational State Transfer

URI Uniform Ressource Identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Einführung

TODO Samuel

1.2 Hintergrund

TODO Jani

1.3 Team

TODO Daniel

Abbildung 1: Secutitysquad - Logo

1.4 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 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.

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 Platform 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

TODO Daniel

- HTML
- CSS
- JavaScript
- jQuery
- 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.[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.[2]

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.[3] 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.[4]

[1] Vgl. Gosling u. a. (2014), S. 1
[2] Vgl. Wolff (2011), S. 2
[3] Vgl. Cosmina (2016), S. 2
[4] Vgl. Gutierrez (2016), S. 1

Spring MVC

Spring MVC ist sehr gut geeignet um Webanwendungen zu implementieren.[5] Hierfür können die diese in mehrere Abstraktionsschichten gegliedert werden. Beispielsweise in das User Interface (UI), die Geschäftslogik und die Persistenzschicht.[6]

Spring Data

Spring Data vereinfacht Datenbankzugriffe ungemein. Das Modul stelt APIs für fast alle gängigen Datenbankzugriffsschichten, wie JDBC (Java Database Connectivity), Hibernate, JDO (Java Data Objects) zur Verfügung. Aber nicht nur relationale Datenbanken werden unterstützt, sondern beispielsweise auch NoSQL-Datenbanken und Key/Value-Stores können Problemlos eingesetzt werden.[7]

In Verbindung mit Spring Data wurde eine *MongoDB* zur Speicherung der Ergebnisse eingesetzt. *MongoDB* ist eine Dokument orientierte anpassungsfähige, skalierbare Datenbank. Dennoch vereint sie 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.[8]

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.[9] 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

[5] Vgl. Wolff (2011), S. 3
[6] Vgl. Yates u. a. (2006), S. 21
[7] Vgl. Pollack u. a. (2012), S. 3f
[8] Vgl. Chodorow/ Dirolf (2010), S. 1f
[9] Vgl. Wikipedia (2017)

sich Don't Repeat Yourself. 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.[10]

Die Kommunikation zwischen Server und Client erfolgt über Representational State Transfer. Hierbei wird jedes Objekt in Representational State Transfer (REST) als Ressource definiert, welche über einen eindeutigen Uniform Ressource Identifier adressiert werden können. Über die HTTP-Methoden GET, PUT, POST und DELETE können diese Ressourcen geladen, erstellt, geändert oder auch gelöscht werden.[11]

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.[12]

- R
  TODO Jani

2.3 Technologien und Frameworks der Tests

- Python
  TODO Daniel
- Node JS
- Phantom JS
  TODO Daniel
- Bro
  TODO Jani

[10] Vgl. Baumann (2013), S. 6f
[11] Vgl. itwissen.info (2017)
[12] Vgl. Roden (2017)

- HTtrack
  TODO Samuel
- Resemble JS
  TODO Samuel

2.4 Angriffstypen

2.4.1 Malware

TODO Samuel

2.4.2 Request Header Investigation

TODO Daniel

2.4.3 JavaScript Port Scanning

TODO Jani

2.4.4 JavaScript IP Scanning

TODO Jani

2.4.5 Clickjacking

TODO Jani

2.4.6 Phishing

TODO Samuel

3 Konzept

3.1 Gesamtkonzept

3.1.1 webifier Tests

TODO Jani

3.1.2 webifier Tester

TODO Samuel

3.1.3 webifier Platform

TODO Daniel

3.1.4 webifier Mail

TODO Daniel

3.1.5 webifier Data

TODO Samuel

3.1.6 webifier Statistics

TODO Jani

3.2 Testarten

3.2.1 Virenscan

TODO Samuel

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

3.2.2 Vergleich in verschiedenen Browsern

TODO Daniel

3.2.3 Test auf Port Scanning

TODO Jani

3.2.4 Test auf IP Scanning

TODO Jani

3.2.5 Link Checker

TODO Daniel

- herausfiltern aller Links und nachgeladenen Ressourcen

3.2.6 Google Safe Browsing

TODO Daniel

3.2.7 Überprüfung des Zertifikats

TODO Samuel

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

3.2.8 Erkennung von Phishing

TODO 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

3.2.9 Screenshot

TODO Jani

4 Umsetzung

4.1 Gesamtanwendung

4.1.1 webifier Tests

TODO Jani

4.1.2 webifier Tester

Der Tester verwaltet und steuert alle Tests. Da die Konfigration aller Tests in einer einzelnen Datei erfolgt ist es sehr einfach ihn neue Tests zu erweitern. ...

TODO Samuel

4.1.3 webifier Platform

TODO Daniel

4.1.4 webifier Mail

TODO Daniel

4.1.5 webifier Data

TODO Samuel

4.1.6 webifier Statistics

TODO Jani

4.2 Tests

4.2.1 Virenscan

TODO Samuel

4.2.2 Vergleich in verschiedenen Browsern

TODO Daniel

4.2.3 Test auf Port Scanning

TODO Jani

4.2.4 Test auf IP Scanning

TODO Jani

4.2.5 Linkchecker

TODO Daniel

4.2.6 Google Safe Browsing

TODO Daniel

4.2.7 Überprüfung des Zertifikats

TODO Samuel

4.2.8 Erkennung von Phishing

TODO Samuel

4.2.9 Screenshot

TODO Jani

5 Analyse

6 Ausblick

6.1 Weitere Tests

6.2 Weitere Module

7 Fazit

7.1 Zusammenfassung

7.2 Bewertung der Ergebnisse

Literaturverzeichnis

Baumann, Joachim (2013):
Gradle - ein kompakter Einstieg in das Build-Management-System, 1. Auflage, dpunkt.verlag

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

Gosling, James u. a. (2014):
The Java Language Specification - Java SE 8 Edition, 5. Auflage, Addison-Wesley

Gutierrez, Felipe (2016):
Pro Spring Boot, 1. Auflage, Apress

itwissen.info (2017):
REST (representational state transfer), `http://www.itwissen.info/REST-representational-state-transfer.html`, Einsichtnahme: 22. 04. 2017

Pollack, Mark u. a. (2012):
Spring Data: Modern Data Access for Enterprise Java, 1. Auflage, O'Reilly Media

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

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

Yates, Colin u. a. (2006):
Expert Spring MVC and Web Flow, 1. Auflage, Apress