

Masterarbeit

# Prozesse, Methoden und Werkzeugen zum Pen-Test b. mob. Applikationen

zur Erlangung des akademisches Grades eines  
Master of Science

angefertigt von  
**Dominik Gunther Florian Schlecht**  
Matrikelnummer: 00032209

## Betreuer

Erstprüfer:	Prof. Hahndel
Zweitprüfer:	Prof. von Koch
Allianz Deutschland AG:	Herr Muncan und Herr Gerhager

Fakultät:	Elektrotechnik und Informatik
Studiengang:	Informatik
Schwerpunkt:	Security & Safety

Abgabedatum:	01. April 2016
--------------	----------------

Ingolstadt, 21. Februar 2017



# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit bis auf die offizielle Betreuung durch die Betreuer selbstständig und ohne fremde Hilfe verfasst habe.

Die verwendeten Quellen sowie die verwendeten Hilfsmittel sind vollständig angegeben. Wörtlich übernommene Textteile und übernommene Bilder und Zeichnungen sind in jedem Einzelfall kenntlich gemacht.

Ingolstadt, 21. Februar 2017



# Inhaltsverzeichnis

<b>Erklärung</b>	<b>i</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Arten von Penetration-Tests</b>	<b>3</b>
2.1. Web-Application . . . . .	3
2.2. Web-Service . . . . .	3
2.3. Mobile-Application . . . . .	3
2.4. Network . . . . .	4
2.5. Wireless Network . . . . .	4
2.6. Social Engineering . . . . .	4
2.7. Physical . . . . .	4
2.8. Redteam . . . . .	5
<b>3. Prozesse zu Penetration-Tests</b>	<b>7</b>
3.1. Vorbereitung . . . . .	7
3.1.1. Aufwandsschätzung . . . . .	7
3.1.2. Rechtliche Aspekte . . . . .	19
3.1.3. Technische Aspekte . . . . .	19
3.2. Durchführung . . . . .	21
3.2.1. Kickoff . . . . .	21
3.2.2. Kategorisierung von Findings . . . . .	23
3.2.3. Bewertung von Findings . . . . .	24
3.2.4. Dokumentation . . . . .	27
3.3. Nachbereitung . . . . .	29
3.3.1. Inhalte eines Pentest-Berichts . . . . .	29
3.3.2. Implementierung in LaTeX . . . . .	30
3.3.3. Implementierung als Webanwendung . . . . .	30
3.4. Kontinuität . . . . .	31
<b>4. Penetrationstests mobiler Anwendungen</b>	<b>33</b>
4.1. Anforderungen . . . . .	33
4.2. Bestehende Anwendungen . . . . .	34
4.2.1. All-In-One-Framework: MobSF . . . . .	34
4.2.2. Einzelanwendungen . . . . .	34
4.3. Aktuelle Situation und Vergleich der Emulation . . . . .	35
4.3.1. iOS . . . . .	35
4.3.2. Windows-Phone . . . . .	41
4.3.3. Android . . . . .	42
4.4. Abgleich der Anforderungen mit MobSF . . . . .	43

---

4.5. Laboraufbau . . . . .	44
4.6. Weiterentwicklung MobSF . . . . .	44
4.6.1. Allgemeine Verbesserungen . . . . .	45
4.6.2. Windows-Apps . . . . .	52
4.6.3. iOS-Apps . . . . .	61
4.7. Abgleich mit Anforderungen . . . . .	66
4.8. Anwendung der Umgebung . . . . .	66
4.8.1. Test Anwendung 1 - Windows-Phone . . . . .	66
4.8.2. Test Anwendung 2 - iOS . . . . .	66
<b>5. Fazit</b>	<b>67</b>
<b>A. Quellcode</b>	<b>69</b>
A.1. Fragebogen . . . . .	69
A.1.1. Latex . . . . .	69

# 1. Einleitung

Smartphones verbreiten sich immer stärker. Dies zeigt auch eine Statistik von Gardner, nach welcher die Absätze von Mobilgeräten die von herkömmlichen Rechnern und Laptops weiter übertreffen werden [8], siehe Tabelle ?? . Mit diesem Fortschritt im Bereich der Verkäufe steigt natürlich auch die Nutzung von Smartphones. Diese können genutzt werden um im Internet zu surfen, Medien-Inhalte wiederzugeben oder zu chatten. Dies sind im Bezug auf Datenschutz und Risiko relativ ungefährliche Anwendungen. Jedoch können auch kritischere Handlungen vollzogen werden, wie zum Beispiel Online-Banking oder das Verwalten von Versicherungs-Verträgen. Das dies gemacht wird, ist eine notwendige Konsequenz aus der Natur des Smartphones und deren User. So ist es einfach praktisch, seinen Kontostand schnell von unterwegs einsehen zu können. Jedoch müssen solche Apps dann so gestaltet sein, dass ein Missbrauch nicht oder nur mit unverhältnismäßig hohem Aufwand möglich ist. Dies ist die Aufgabe der Unternehmen, welche solche Apps bereitstellen. Dies stellt viele vor unerwartet hohe Herausforderungen. Mussten bisher nur lokale oder Web-Anwendungen getestet werden, sind es nun Applikationen auf mobilen Geräten mit einem oft wesentlich höheren Anteil an Web-Services und APIs.

Device Type	2015	2016	2017	2018
Traditional PCs (Desk-Based and Notebook)	244	228	223	216
Ultramobiles (Premium)	45	57	73	90
PC Market	289	284	296	306
Ultramobiles (Basic and Utility)	195	188	188	194
Computing Devices Market	484	473	485	500
Mobile Phones	1,917	1,943	1,983	2,022
Total Devices Market	2,401	2,416	2,468	2,521

Abbildung 1.1.: Worldwide Devices Shipments by Device Type, 2015-2018 (Millions of Units)[8]

Die Allianz Deutschland AG ist ein solches Unternehmen. Bisher waren viele Wege zum Kunden analog, also per Brief. Es gab nicht viele elektronische Schnittstellen. Vor 2 Jahren wurde eine Digitalisierungs-Strategie gegenüber beschlossen und die Web-Anwendung "Meine-Allianz" entwickelt. In dieser können Versicherungsnehmer nicht nur Verträge anschauen, sondern auch Änderungen an diesen vornehmen. Sollte diese Anwendung eine schwerwiegende Schwachstelle aufweisen, hätten dies für die Allianz nicht nur Schäden in Bezug auf die Reputation, sondern eventuell zusätzliche rechtliche Folgen, da Krankendaten unter §203 StGB fallen. Um dies zu verhindern, sind Prozesse in der Anwendungsentwicklung notwendig, welche die Sicherheit einer Anwendung sicherstellen.

Diese Prozesse umfassen Komponenten wie Source-Scanning, Penetrations-Test sowie eine regelmäßig wiederkehrende Prüfung von Anwendungen, selbst nach dem Release.

In dieser Arbeit werden als Hinleitung allgemeine Prozesse und Fakten rund um die Applikations-Sicherheit erläutert. Im weiterführenden Teil wird ein bereits bestehendes Open-Source-Tool zum automatischen Testen mobiler Anwendungen um Features wie TODO erweitert.



## 2. Arten von Penetration-Tests

Als Einführung dient zunächst ein kurzer Überblick über die verschiedenen Arten von Penetration-Tests. An diesen orientiert sich ebenfalls das zweite Kapitel, Prozesse zu Penetration-Tests (3). Die Arten basieren dabei auf der Liste von Pentest-Standard.org<sup>1</sup>, sind jedoch um weitere Arten, basierend auf den Erfahrungen des Autors sowie andere Pen-Tester der Allianz Deutschland AG, ergänzt.

### 2.1. Web-Application

*Web-Application-Tests* sind wohl die häufigste durchgeführten Tests. Dabei wird eine definierte Menge an Webseiten von einem sogenannten Penetration-Tester auf Schwachstellen getestet. Typische Findings im bei *Web-Application-Tests* wären fehlende *Cookie-Flags*, *XSS*, *CSRF* oder auch *SQL-Injection*. Ebenso gehören neben technischen Sicherheitslücken auch Mängel in der Anwendungslogik zur Menge der möglichen Findings.

### 2.2. Web-Service

Ebenfalls sehr häufig und immer öfter gefragt sind Pen-Tests auf Web-Services. Diese haben im Gegensatz zu *Web-Applications* keine Oberfläche sondern sind reine Schnittstellen zur Abfrage von Informationen. Zudem sind sie meist in *REST*, *WSDL* oder *SOAP* geschrieben, weshalb sich das Vorgehen bei Pen-Test grundlegend von anderen Arten unterscheidet.

### 2.3. Mobile-Application

Von einem Penetration-Test auf eine Mobile-Application redet man dann, wenn eine sog. App, also eine Anwendung geschrieben für ein mobiles Gerät, getestet wird. Da der Begriff nicht weiter definiert ist (TODO), ist unklar, ob ein Mobile-Application-Pen-Test ebenfalls die Backend-Systeme einschließt. Der Unterschied wäre dabei, dass bei einem Test ohne Backend-Tests nur der Teil der App auf dem Handy untersucht wird. Hier können Lücken wie *Buffer-Overflows*, mangelnde *TLS*-Prüfungen und ähnliches gefunden werden, welche eher den Benutzer eines Mobiltelefons gefährden. Test auf die Backend-Systeme von Apps dagegen sollen eher den Betreiber der Applikation schützen. Eine Lücke wäre hier beispielsweise eine *SQL-Injektion*, durch welche ein Angreifer im schlimmsten Fall alle Daten des Betreibers abfragen oder auch Code auf dem DB-System ausführen könnte.

---

<sup>1</sup><http://www.pentest-standard.org/index.php/Pre-engagement>

## 2.4. Network

Bei einem *Network-Pen-Test* geht es zumeist darum, in eine Netzwerk einzubrechen, die Bestandteil zu erfassen und anschließend zu analysieren. Oft werden auch bestimmte Systeme als Ziele definiert, welche es zu übernehmen gilt. Typische Findings wären veraltete Systeme oder unerwünschte Komponenten im Netzwerk (sogenannte „Schatten-IT“). Da diese Test meist in der produktiven Umgebung durchgeführt werden, sollte zur Verhinderung von Störungen im Betrieb vorher definiert werden, welche Systeme vom Test ausgenommen sind.

## 2.5. Wireless Network

Bei einem Test von Wireless-Systemen meint man den Angriff auf IT-Systeme, welche Funktechniken nutzen. Oft wird hier die WLAN-Konfiguration eines Unternehmens daraufhin getestet, ob ein Eindringen in das Netzwerk oder das Abfangen von Daten möglich ist. In den letzten Jahren sind jedoch auch Tests auf andere Funktechniken wie NFC/RFID häufiger geworden, seitdem diese oft als Zugangskontrollen oder Zahlungsmittel in Unternehmen verwendet werden. Ebenso sind hier Tests auf Funk-Systeme im 433 oder 900 MHz Bereich angesiedelt, welche häufig bei z.B. Garagentoren oder Auto-Schlüsseln verwendet werden.

## 2.6. Social Engineering

Der BSI-Grundschutz beschreibt *Social Engineering* wie folgt:<sup>2</sup>

Social Engineering ist eine Methode, um unberechtigten Zugang zu Informationen oder IT-Systemen durch Äußerungen zu erlangen. Beim Social Engineering werden menschliche Eigenschaften wie z. B. Hilfsbereitschaft, Vertrauen, Angst oder Respekt vor Autorität ausgenutzt.

Ein Beispiel wäre, dass ein Pen-Tester versucht Zugang zu einem Gebäude der Firma zu erlangen, indem er eine Blumenlieferung an einen Vorstand vortäuscht. Dabei stehen ihm sowohl die Möglichkeit der Autorität („Ich darf die Blumen nur direkt dem Vorstand liefern. Wenn Sie mich nicht durchlassen, wird er das erfahren.“) als auch der Hilfsbereitschaft („Wenn ich diese Lieferung nicht abschließe, werde ich gefeuert.“) zur Verfügung.

## 2.7. Physical

Bei einem *Physical-Pen-Test* versteht man den Versuch, physikalische Sicherheitsmaßnahmen zu überwinden. Diese werden oft in Verbindung mit *Social Engineering* verwendet und sind oft Teil eines *Redteam*-Tests. Da hier bei Sicherheitskomplexen mit zum Beispiel bewaffnetem Wachpersonal ein besonderes Risiko für die Tester vorliegt, sollten sie besonders sorgfältig vorbereitet werden.

---

<sup>2</sup>[https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/\\_content/g/g05/g05042.html](https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/g/g05/g05042.html)

## 2.8. Redteam

Bei einem Redteam/Blueteam wird den Testern, welche das sogenannte Redteam bilden, im Gegensatz zu den anderen Arten von Penetration-Tests keine bestimmte Menge an Webseiten oder Angriffspunkten genannt, sondern nur ein bestimmtes, zu erreichendes Ziel. Dieses können bestimmte Daten sein, wie zum Beispiel die E-Mails eines Vorstands, oder die Kompromittierung eines bestimmten Systems, wie zum Beispiel dem *Active Directory*. Je nach Vereinbarung werden von den Testern Variationen der vorherigen Arten von Angriffen genutzt, um an das definierte Ziel zu gelangen. Dem Redteam gegenüber steht das Blueteam, meist ein SOC oder CERT der Firma, welches versucht die Angriffe zu detektieren und vereiteln. Das Blueteam kann über den bevorstehenden Test informiert werden, in der Praxis wird darauf jedoch bewusst verzichtet, um realitätsnahe Ergebnisse zu erzielen.



## 3. Prozesse zu Penetration-Tests

### 3.1. Vorbereitung

Vor der Durchführung eines Penetrationstests müssen verschiedenste Aufgaben erledigt und Rahmenbedingungen geklärt werden.

#### 3.1.1. Aufwandsschätzung

Ein äußerst wichtiger aber auch sehr schwieriger Punkt ist die Aufwandsschätzung. Natürlich können klassische Methoden der Aufwandsschätzung aus dem Bereich der Projektmanagement verwendet werden, jedoch müssen eine Vielzahl weitere technischer Aspekte betrachtet werden.

Viele Firmen berechnen den Aufwand aus der Anzahl der Eingabefelder. Leider ist diese Vorgehensweise oft nicht zielführend, da die Komplexität des dahinter liegenden Systems nicht in Betracht gezogen wird. Wird zum Beispiel eine Webseite mit einem Framework erstellt, welches Eingaben stetig filtert, ist die Wahrscheinlichkeit eine XSS oder SQLi zu finden unabhängig von der Anzahl der Eingabefelder gering. In diesem Fall sollte man andere Komponenten wie die Authentisierungslogik oder den Webserver selbst angreifen. Dies würde durch die Aufwandsschätzung rein nach Eingabefeldern nicht in Betracht gezogen. Eine weitere Möglichkeit wäre die Einschätzung anhand der Anzahl der zu testenden IP-Adressen. Leider gibt es hier ähnliche Probleme wie bei der Anzahl der Eingabefelder, da auch hier nicht die Komplexität der Anwendung an sich mit einbezogen wird.

Insgesamt müssen viele Teilaspekte beachtet werden, damit die Pentester ein Gefühl für die Anwendung bekommen und den groben Aufwand schätzen können. Um dies zu vereinfachen, wurden im ersten Schritt Fragen erarbeitet. Als Grundlage hierfür dienten die Fragen von <http://www.pentest-standard.org/index.php/Pre-engagement>. Zusätzlich wurden Fragen und Kategorien ergänzt, welche sich während der Tätigkeit des Autors bei der Allianz Deutschland AG als hilfreich oder notwendig erwiesen haben. Die Kategorien und Fragen sind in der Sektion 3.1.1.1 dargestellt.

Anschließend galt es, die Fragen in eine passende Form zu bringen. Dazu wurden im ersten Versuch ein LaTeX-Dokument erstellt. Dies ist in Sektion 3.1.1.2 dargestellt. Trotz der Ansprechenden Form sind LaTeX-Dokumente meistens nicht sehr intuitiv und effizient auszufüllen. Daher wurde eine Webanwendung implementiert, welche den Aufwandsfragebogen darstellt und ein einfaches Ausfüllen ermöglicht. Diese Technik und Vorgehensweise ist in Sektion 3.1.1.3 dargestellt.

TODO Docker

### 3.1.1.1. Fragebogen

Um einen groben Eindruck vom Umfang des Tests zu bekommen, empfiehlt sich die Durchsprache eines standardisierten Fragebogens. Im Rahmen dieser Arbeit wurde der Fragebogen von Pentest-Standard.org [TODO\(Link\)](#) übersetzt und erweitert. Anschließend wurde der Fragebogen mit der Ansprechpartnern der Allianz Deutschland AG diskutiert und ergänzt, sodass sich folgende Fragen ergeben:

#### Allgemein

- Wie ist der Projekt-Name?
- Wer sind die Ansprechpartner?
- Welche Art/en von Pentest/s sollen durchgeführt werden?  
(Web-Application/Web-Service/Mobile-Application/Network/Social Engineering/Wireless/Physical)
- Ist der Test für eine spezielle Compliance-Anforderung notwendig?  
(Ja/Nein)
- Wann soll der Test durchgeführt werden?
- In welchen Zeiträumen soll der Test durchgeführt werden?  
(Bürozeiten/Feierabend/Wochenende)

#### Web Application Penetration Test

- Wie geschieht der Zugriff auf die Anwendung?  
(Aus dem Internet erreichbar/IP-Einschränkung/VPN/Pentest muss intern durchgeführt werden)
- Welche Funktionalitäten hat die Anwendung?  
(CMS/Captcha/Upload/Download/Browser-Plugins/Workflows)
- In welcher Stage befindet sich die Anwendung?  
(Development oder Test/System Integration/Produktion)
- Wird der Quellcode der Applikation/Webseite zugänglich gemacht?  
(Ja/Nein)
- Wie viele Web-Applikationen sind In-Scope?
- Wie viele Login-Systeme sind In-Scope?
- Wie viele statische Seiten sind ca. In-Scope?
- Wie viele dynamische Seiten sind ca. In-Scope?
- Soll Fuzzing gegen die Applikation/en eingesetzt werden?  
(Ja/Nein)

- Soll der Penetrations-Test aus verschiedenen Rollen durchgeführt werden?  
(Ja/Nein)
- Wie ist die Anmeldung gestaltet?  
(Benutzername und Passwort/Zertifikat/Komplexeres System)
- Welche Technologien nutzt die Anwendung?
- Sollen Password-Scans auf die Webseite durchgeführt werden?  
(Ja/Nein)

#### **Web Service Penetration Test**

- Wie geschieht der Zugriff auf die Anwendung?  
(Aus dem Internet erreichbar/IP-Einschränkung/VPN/Pentest muss intern durchgeführt werden)
- In welcher Stage befindet sich die Anwendung?  
(Development oder Test/System Integration/Produktion)
- Wird der Quellcode des Services zugänglich gemacht?  
(Ja/Nein)
- Wie viele Web-Services sind In-Scope?
- Soll Fuzzing gegen die Applikation/en eingesetzt werden?  
(Ja/Nein)
- Soll der Penetrations-Test aus verschiedenen Rollen durchgeführt werden?  
(Ja/Nein)
- Wie ist die Anmeldung gestaltet?  
(Benutzername und Passwort/Zertifikat/Komplexeres System)
- Welche Technologien wurden für den Service genutzt??
- Sollen Password-Scans auf die Webseite durchgeführt werden?  
(Ja/Nein)

#### **Mobile Application Penetration Test**

- Welche Technologien wurden für die App genutzt?
- Wird der Quellcode der App zugänglich gemacht?  
(Ja/Nein)
- Gibt es eine Root-Detection? Wenn ja, welche Konsequenzen hat eine positive Erkennung?  
(Ja/Nein)
- Hat die App einen Login?  
(Ja/Nein)

- Soll der Penetrations-Test aus verschiedenen Rollen durchgeführt werden?  
(Ja/Nein)
- Soll Fuzzing gegen die App eingesetzt werden?  
(Ja/Nein)

### **Network Penetration Test**

- Was ist das Ziel des Penetrations-Test?
- Wie viele IP-Adressen sollen getestet werden?
- Sind Techniken im Einsatz, die die Resultate verfälschen könnten? (WAF, IPS etc.?)  
(Ja/Nein)
- Wie ist das Vorgehen bei einem gelungenen Angriff?
- Soll versucht werden lokale Admin-Rechte zu erlangen und tiefer in das Netz vorzudringen?  
(Ja/Nein)
- Sollen Angriffe auf gefundene Passwort-Hashes durchgeführt werden?  
(Ja/Nein)

### **Social Engineering**

- Gibt es eine vollständige Liste von E-Mail-Adressen, die für den Test verwendet werden können?  
(Ja/Nein)
- Gibt es eine vollständige Liste von Telefon-Nummern, die für den Test verwendet werden können?  
(Ja/Nein)
- Ist das Einsetzen von Social Engineering zum Überwinden physikalischer Sicherheitseinrichtungen erlaubt?  
(Ja/Nein)
- Wie viele Personen sollen ca. getestet werden?

### **Wireless Network Penetration Test**

- Wie viele Funk-Netzwerke sind im Einsatz?
- Gibt es eine Gäste WLAN? Wenn ja, wie ist dieses umgesetzt?  
(Ja/Nein)
- Welche Verschlüsselung wird für die Netzwerke genutzt?
- Sollen nicht-firmen-Geräte im WLAN aufgespürt werden?  
(Ja/Nein)



- Sollen Netz-Attacken gegen Clients durchgeführt werden?  
(Ja/Nein)
- Wie viele Clients nutzen das WLAN ca.?

#### **Physical Penetration Test**

- Wie viele Einrichtungen sollen getestet werden?
- Wird die Einrichtung mit anderen Parteien geteilt?  
(Ja/Nein)
- Muss Sicherheitspersonal umgangen werden?  
(Ja/Nein)
- Wird das Sicherheitspersonal durch einen Dritten gestellt?  
(Ja/Nein)
- Ist das Sicherheitspersonal bewaffnet?  
(Ja/Nein)
- Ist der Einsatz von körperlicher Gewalt durch das Sicherheitspersonal gestattet?  
(Ja/Nein)
- Wie viele Eingänge gibt es zu der/den Einrichtung/en?
- Ist das Knacken von Schlössern oder Fälschen von Schlüsseln erlaubt?  
(Ja/Nein)
- Wie groß ist die Fläche ungefähr?
- Sind alle physikalischen Sicherheitsmaßnahmen dokumentiert und werden zur Verfügung gestellt?  
(Ja/Nein)
- Werden Video-Kameras verwendet?  
(Ja/Nein)
- Werden diese Kameras durch Dritte verwaltet?  
(Ja/Nein)
- Soll versucht werden, die aufgezeichneten Daten zu löschen?  
(Ja/Nein)
- Gibt es ein Alarm-System?  
(Ja/Nein)
- Gibt es einen Stillen Alarm?  
(Ja/Nein)
- Welche Ereignisse lösen den Alarm aus?

### Fragen an den System-Administrator

- Gibt es Systeme, die als instabil angesehen werden (alte Patch-Stände, Legacy Systeme etc.)?  
(Ja/Nein)
- Gibt es Systeme von Dritten, die ausgeschlossen werden müssen oder für die weitere Genehmigungen notwendig sind?  
(Ja/Nein)
- Was ist die Durchschnittszeit zur Wiederherstellung der Funktionalität eines Services?
- Ist eine Monitoring-Software im Einsatz?  
(Ja/Nein)
- Welche sind die kritischsten Applikationen?
- Werden in einem regelmäßigen Turnus Backups erstellt und getestet?  
(Ja/Nein)
- Fragen an den Business Unit Manager
- Ist die Führungsebene über den Test informiert?  
(Ja/Nein)
- Welche Daten stellen das größte Risiko dar, falls diese manipuliert werden?
- Gibt es Testfälle, die die Funktionalität der Services prüfen und belegen können?  
(Ja/Nein)
- Sind "Disaster Recovery Procedures" vorhanden?  
(Ja/Nein)

Dabei müssen natürlich nur die für die Art des Pentests (Web-Application/Network/Social Engineering/Wireless/Physical) durchgeführt werden.

#### 3.1.1.2. Implementierung in Latex

Nachdem der Festlegung der Fragen, wurde der Fragebogen in LaTeX implementiert. Dabei wurde speziell das Format angepasst, sodass eine ausgedruckte Kopie des Fragebogens möglichst effizient ausgefüllt werden kann. Dafür wurden die Fragen in Boxen integriert, sowie Ja/Nein-Antworten ansprechend dargestellt. Die angepasste Formatierung ist in Abbildung 3.1 dargestellt.

Um eine möglichst einfache Erweiterung der Fragen in LaTeX zu gewährleisten, wurden LaTeX-Makros für die jeweiligen Frage-Arten entworfen. Im Folgenden ist ein kurzes Beispiel für ein solches Makro und zwei Fragen, die dieses nutzen:

```
1 \newcommand{\frageJaNein}[1]{\makebox[\textwidth]{%  
2 \renewcommand{\arraystretch}{2.0}  
3 \begin{tabularx}{\textwidth}{|X|S|S|}
```

## 2 Web Application Penetration Test

Wird der Quellcode der Applikation/Webseite zugänglich gemacht?	Ja	Nein
_____	<input type="checkbox"/>	<input type="checkbox"/>

Wie viele Web-Applikationen sind In-Scope?
_____

Wie viele Login-Systeme sind In-Scope?
_____

Abbildung 3.1.: Ein kurzer Ausschnitt des Fragebogens mit formatierten Fragen

```

4 \hline
5 #1 & Ja & Nein\\
6 \hline
7 \line(1,0){350} & $\square$ & $\square$ \\
8 \hline \hline
9 \end{tabularx}%
10 \renewcommand{\arraystretch}{1.0}
11 }}
12
13 \frageJaNein{Soll der Penetrations-Test aus verschiedenen Rollen
    durchgeführt werden?}
14 \frageJaNein{Sollen Angriffe auf gefundene Passwort-Hashes
    durchgeführt werden?}

```

### 3.1.1.3. Implementierung als Webabwendung

Die LaTeX-Implementierung ist gut geeignet, um den Fragebogen bei einem physikalischen Termin zu besprechen. Für ein Ausfüllen am Computer ist es jedoch weniger geeignet, da man mit einem passenden PDF-Reader die Antworten einfügen müsste. Daher wurde eine Webanwendung entworfen, welche das Ausfüllen am Computer, zum Beispiel parallel zu einem Online-Meeting, vereinfacht.

Um die Anwendung möglichst minimalistisch und einfach zu halten, wurde Python als Programmiersprache und Flask Web-Server verwendet. Auf HTML-Seite wurde Bootstrap 4 zur einfachen Formatierung genutzt.

Das Einfügen der Eingaben in der Weboberfläche wurde anfangs über eine einfache String-Substitution gelöst:

```
1 # Load template from file
2 orig_file = open(tex_path + "/Fragen.tex", 'r')
3 template = LaTeXTemplate(orig_file.read())
4 orig_file.close()
5
6 # Substitute with form-fields
7 new_string = template.substitute(
8     allg_anspr_web_app=__html_to_tex(
9         request.form['allg_anspr_web_app']
10    ),
11     allg_pen_art_wapt=__html_to_checkbox(
12         request, 'allg_pen_art_wapt'
13    ),
14     # Andere Form-Felder
15 )
```

Da dies jedoch einen erhöhten Wartungsaufwand bei zum Beispiel Änderung von Parametern mit sich zieht, wurde die Substitution durch eine einfachere Implementierung in Jinja2 ersetzt. Die Substitutions-Technik von Jinja ist unter 3.1.1.3 aufgezeigt. Unter der Nutzung von Jinja-Makros können so sehr effizient die HTML-Dateien für die Fragen erstellt werden. Im Folgenden ist ein kurzes Beispiel aus dem Fragebogen bezüglich Web-Application Penetration Tests, in welchem einige Fragen definiert werden:

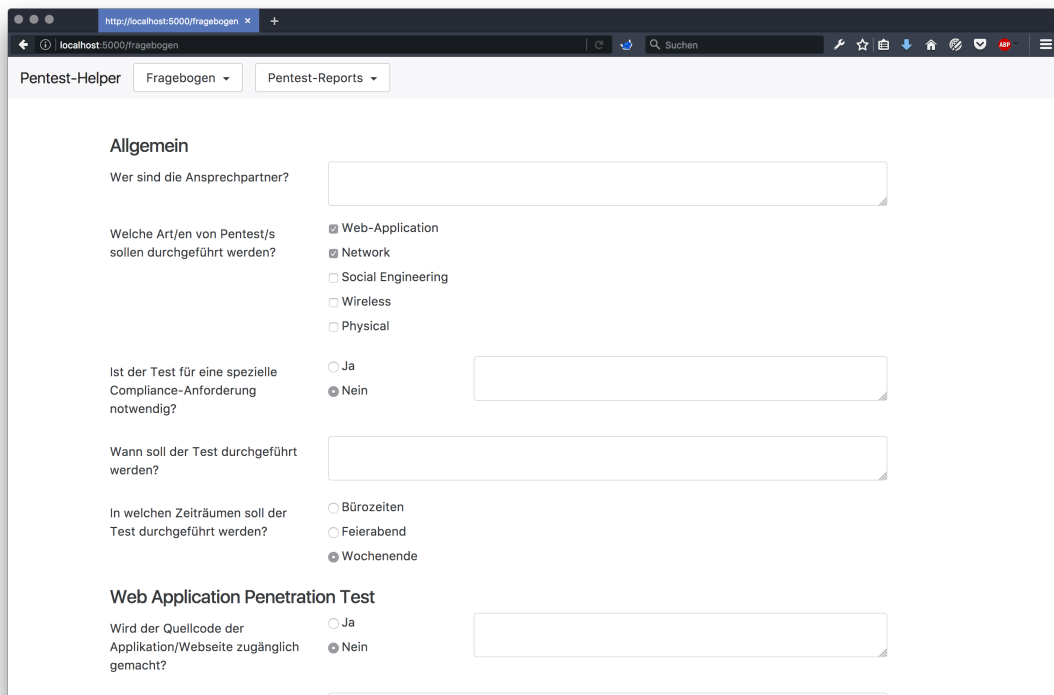
```
1 {{ binary_question("wapt_quell_zug", "Wird der Quellcode der
   Applikation/Webseite zugänglich gemacht?") }}
2 {{ text_question("wapt_anz_web_app", "Wie viele Web-
   Applikationen sind In-Scope?") }}
3 {{ text_question("wapt_anz_login_sys", "Wie viele Login-
   Systeme sind In-Scope?") }}
```

Durch die Verwendung von Bootstrap ist die Anwendung automatisch „responsive“, also unter verschiedenen Auflösungen verwendbar. Ein Screenshot der Anwendung ist unter 3.2 dargestellt.

Um die Daten dauerhaft zu speichern, wurde eine Datenbankanbindung implementiert. Hierzu wurde SQLAlchemy als Engine genutzt, da diese sehr gut zu Flask passt und sich einfach und ohne viel Aufwand implementieren lässt. Mehr zu SQLAlchemy ist de Abschnitt 3.1.1.3 zu entnehmen.

Zum Erstellen des PDFs muss das generierte TeX-Dokument über *pdflatex* umgewandelt werden. Dies geschieht im Quellcode folgendermaßen:

```
1 try:
2     # Double-Compile for ToC and refs
3     subprocess.check_output(
4         [
5             '/usr/local/texlive/2016/bin/x86_64-darwin/pdflatex',
6             '-synctex=1',
```



Pentest-Helfer Fragebogen Pentest-Reports

**Allgemein**

Wer sind die Ansprechpartner?

Welche Art/en von Pentest/s sollen durchgeführt werden?

- ☒ Web-Application
- ☒ Network
- ☐ Social Engineering
- ☐ Wireless
- ☐ Physical

Ist der Test für eine spezielle Compliance-Anforderung notwendig?

☐ Ja ☒ Nein

Wann soll der Test durchgeführt werden?

In welchen Zeiträumen soll der Test durchgeführt werden?

☐ Bürozeiten  
☐ Feierabend  
☒ Wochenende

**Web Application Penetration Test**

Wird der Quellcode der Applikation/Webseite zugänglich gemacht?

☐ Ja ☒ Nein

Abbildung 3.2.: Ein kurzer Ausschnitt der Fragebogen-Webanwendung

```
7         '-interaction=nonstopmode',
8         '--output-directory=' + output_path,
9         tex_file_name
10    ], cwd=tex_path
11 )
12 except subprocess.CalledProcessError as error:
13     print(error)
```

Dabei wird der File-Path für das generierte TeX-File in der Variable *tex\_path* übergeben und das generierte PDF unter *output\_path* gespeichert.

Anschließend wird das von Flask über die *send\_file*-Methode zurückgegeben und im Browser dargestellt.

TODO Code-Sample, Screenshot aktualisieren

## Jinja2

Jinja2 ist eine Template-Engine für Python <sup>1</sup>. Diese ist eigentlich für das Füllen von Inhalten in HTML-Dateien gedacht, kann sich jedoch auch auf andere Sprachen anwenden lassen. Einige Features sind:

- Ausführung in einer Sandbox
- Automatisches Escaping von HTML-Characters
- Template-Vererbung
- Code-Optimierung
- Sprechende Fehlermeldungen
- Anpassbare Syntax

Zudem erlaubt Jinja eine Nutzung von übergebenen Python-Objekten. So wird zum Beispiel im Fragebogen folgendes Code-Segment genutzt:

```
1 # Define template
2 template = latex_jinja_env.get_template('aufwand.tex')
3
4 # Render template
5 rendered_latex = template.render(
6     aufwand=aufwand
7 )
```

Listing 3.1: Aufrufender Python-Code

```
1 \makebox[\textwidth]{%
2 \renewcommand{\arraystretch}{2.0}
3 \begin{tabularx}{\textwidth}{|X|}
4 \hline
```

<sup>1</sup><http://jinja.pocoo.org/docs/2.9/>

```

5  Wie viele Einrichtungen sollen getestet werden? \\
6  \hline
7  \VAR{ aufwand.phys_anz_einr } \\
8  \hline \hline
9  \end{tabularx}%
10 \renewcommand{\arraystretch}{1.0}
11 }

```

Listing 3.2: aufwand.tex

In Zeile 7 des Listings 3.2 ist der Jinja2-Zugriff auf das Attribut *phys\_anz\_einr* des in Zeile 6 des Listings 3.1 übergebenen Objekts *aufwand* zu sehen. Ebenso können Attribute geprüft und Makros implementiert werden. Beides ist in folgendem Code-Snippet zu verwenden:

```

1 \BLOCK{ macro checkbox(feld) }
2   \BLOCK{ if feld == 'checked' }
3     \mbox{\ooalign{\checkmark$\cr\hidewidth$\square$\cr\hidewidth\cr}}
4   \BLOCK{ else }
5     $\square$
6   \BLOCK{ endif }
7 \BLOCK{ endmacro }

```

Dabei wird ein Makro implementiert, welcher ein übergebenes Attribut prüft (Zeile 2) und je nach dem eine Checkbox mit einem Hacken (Zeile 3) oder eine leere Checkbox (Zeile 5) darstellt.

## SQLAlchemy

SQLAlchemy ist ein Modul zum Ausführen von SQL-Statements aus Python. Eine Besonderheit ist der ORM (*Object Relational Mapper*), welcher es erlaubt die Datenbank in Python als normale Klassen zu definieren. Das DB-Schema sowie die Zugriffe passieren zum Großteil transparent.

So konnte zum Beispiel der Fragebogen zur Aufwandsschätzung aus 3.1.1.3 direkt als Klasse definiert und im weiteren Kontext sowohl in Python als auch in Jinja verwendet werden. Das folgende Beispiel verdeutlicht das Zusammenspiel zwischen Flask und SQLAlchemy:

```

1 from flask_sqlalchemy import SQLAlchemy
2 from flask import Flask, request
3 from jinja2 import Template
4
5 APP = Flask(__name__)
6 APP.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
7 APP.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
8 db = SQLAlchemy(APP)
9
10 class Aufwand(db.Model):
11     id = db.Column(db.Integer, primary_key=True)
12     allg_name = db.Column(db.Text)
13     allg_anspr_web_app = db.Column(db.Text)
14     allg_pen_art_wapt = db.Column(db.Text)

```

```
15     [...]
16
17     def __init__(self, aufwand):
18         self.allg_name = aufwand.allg_name
19         self.allg_anspr_web_app = aufwand.allg_anspr_web_app
20         self.allg_pen_art_wapt = aufwand.allg_pen_art_wapt
21         [...]
22
23 @APP.route('/aufwand', methods=['POST'])
24 def save_aufwand():
25
26     aufwand_tuple = AufwandTuple(
27         allg_name=request.form['allg_name'],
28         allg_anspr_web_app=request.form['allg_anspr_web_app'],
29         allg_pen_art_wapt='checked' if 'allg_pen_art_wapt' in
            request.form else '',
30         allg_pen_art_wspt='checked' if 'allg_pen_art_wspt' in
            request.form else '',
31         [...]
32     )
33
34     aufwand = Aufwand(aufwand_tuple)
35     db.session.add(aufwand)
36     db.session.commit()
37
38     file_path = __generate_tex_aufwand(aufwand)
39     return send_file(file_path, 'aufwand.pdf')
```

In den Zeilen 10 bis 21 wird die DB-Tabelle definiert, welche im weiteren Verlauf wie ein normales Python-Objekt benutzt werden kann. In Zeile 34 wird ein DB-Eintrag erstellt und in den Zeilen 35 und 36 der Session hinzugefügt und in die Datenbank geschrieben. In Zeile 38 wird das Template aufgerufen, das Objekt übergeben und ein Template durch Jinja gefüllt. Dies ist den Code-Snippets unter [3.1.1.3](#) zu entnehmen.

#### 3.1.1.4. Portierung nach Docker

Um die Anwendung möglichst einfach auf weitere Rechner portieren zu können, wurde die Anwendung, zusammen mit der Anwendung zur Erstellung von Pentest-Reports (siehe TODO), in einen Docker-Container aufgenommen. Dazu wurde das entsprechende "Dockerfile" erstellt, welches im Anhang unter TODO zu finden ist.

Dabei wird Ubuntu 16.10 als Basis verwendet und anschließend alle Voraussetzungen installiert sowie der Service gestartet. Daraus ergeben sich für Benutzer zwei Möglichkeiten zur einfacheren Einrichtung des Programms. Entweder es kann lokal über das Kommando

```
1 docker build -t dominikschlecht/pentest_helper .
```

erstellt werden oder es kann ein Abbild des Containers auf den Rechner kopiert und ausgeführt werden.



Leider stellte sich heraus, dass die LaTeX-Installation sowie andere Komponenten den Container auf eine Größe ansteigen ließen, welche den Austausch TODO ASCII und PATH und Umsetzung

### 3.1.2. Rechtliche Aspekte

Sollte eine Beauftragung erfolgen, müssen sowohl Auftragnehmer wie Auftraggeber verschiedenen rechtlichen Aspekte beachten. Im Folgenden sind kurz einige der wichtigsten Aspekte dargestellt, welche zusätzlich zum normalen Vertrag beachtet werden sollten.

**NDA:** Im Normalfall der Auftraggeber ein NDA (Non-Disclosure-Agreement) oder auf deutsch eine Vertraulichkeitserklärung vom Dienstleister einfordern. Dieses verpflichtet Auftragnehmer dazu, keine Informationen über den Pentest an Dritte weiterzugeben. Dieses Vertragsstück schützt den Auftraggeber vor Reputationsschäden und sollte auf jeden Fall geschlossen werden. Beachtet werden sollte jedoch, dass die Vertragsstrafe bei Verletzung der Abmachung auch dem wirklich zu erwartenden Schaden entspricht.

**Haftungsausschluss:** Im Gegenzug wird der Auftragnehmer einen Haftungsausschluss vom Auftraggeber einfordern. In diesem ist definiert, dass der Auftragnehmer nicht für entstehende Schäden aufkommt. Dies ist für Pentester äußerst wichtig, da bei Tests oft beiläufig und unabsichtlich Randsysteme in Mitleidenschaft gezogen werden. Ein Beispiel wäre eine Log-Instanz, welche die Log-Meldungen des anzugreifenden Systems verarbeitet, aber unter der Last der Log-Meldungen aufgrund von Security-Scans abstürzt. Wurde kein Haftungsausschluss vereinbart, wäre der Auftraggeber unter Umständen imstande den Auftragnehmer auf den entstandenen Schaden zu verklagen.

**Personenbezogene Daten:** Falls im Laufe des Pentests der Auftragnehmer zugriff auf personenbezogene Daten erlangen könnte, sollte dies mit dem Datenschutzbeauftragten des Auftraggebers besprochen und falls notwendig entsprechende Vereinbarungen getroffen werden.

### 3.1.3. Technische Aspekte

Im Vorfeld zu einem Pentest sollten auch verschiedene technische Aspekte beachtet werden. Diese sind im folgenden in Ausstattung, Infrastruktur und Tools aufgeteilt.

#### 3.1.3.1. Ausstattung

Natürlich brauch ein Pentester eine gewisse Ausrüstung. Dazu sollte auf jeden Fall Rechner mit mittlerer bis hoher Leistung Prozessorleistung sowie ausreichend RAM für das Ausführen von virtuellen Maschinen gehören. Zudem wäre eine Grafikkarte zu erwägen, um falls gewünscht Attacken auf Passwort-Hashes durchführen zu können, welche auf der GPU wesentlich effizienter sind als auf der CPU. Insofern der Auftraggeber jedoch damit einverstanden ist, kann dies ebenfalls z.B. in der Amazon-Cloud durchgeführt werden, da hier innerhalb weniger Minuten Rechner mit 16 GPUs zur Verfügung stehen.

Sollte ein vor-Ort-Einsatz geplant sein, könnte ebenso ein Ethernet-Tap <https://greatscottgadgets.com/throwingstar/>, entsprechend viele LAN-Kabel sowie ein WLAN-Router von Vorteil sein.

Bei einem Team von Pentester sollte auf Homogenität bezüglich der verwendeten Hardware geachtet werden.

### 3.1.3.2. Infrastruktur

Gerade beim Pentesten von Web-Applikationen sollte der Pentester eine statische IP besitzen, von welcher aus er testet. Sollte die Umgebung, von welcher getestet wird, diese Eigenschaft nicht aufweisen, sollte ein Proxy-Server verwendet und alle jeglicher Traffic über diesen geschickt werden. Nur so kann der Auftraggeber feststellen, ob die Angriffe wirklich von Auftragnehmer kommen oder parallel ein echter Angriff stattfindet.

Ebenso sollte eine gemeinsame Ablage erstellt werden, auf welche Pentester alle notwendigen Informationen austauschen können. Dabei sollte die Ablage so dynamisch wie möglich sein. Eine Möglichkeit wäre *etherpad* <http://etherpad.org/>, da es als Open-Source-Software keine weiteren Lizenzen erfordert und einen schnellen, dynamischen Austausch ermöglicht. Zudem können die Daten einfach als *HTML*-File exportiert werden.

### 3.1.3.3. Betriebssystem und Tools

Bei Pentests ist es essentiell, dass die Pentester ihr Laptop sehr genau kennen und keine unerwarteten Aktionen auftreten. Ein Beispiel wäre DHCP, welches in Endbenutzer-Distributionen wie Ubuntu standardmäßig aktiviert ist. Bei einem Network-Penetration Test kann es aber durchaus von nutzen sein, wenn der Rechner keine sofortige DHCP-Anfrage an das Netzwerk stellt. Daher sollte ein möglichst minimales System gewählt und nur notwendige Software installiert werden. Dazu bieten sich als Betriebssysteme hoch konfigurierbare Linux-Distributionen wie Gentoo oder Arch-Linux an.

Bei der Installation von Software sollte immer auf vertrauenswürdige Quellen geachtet werden. Im besten Fall lädt man den Source-Code aus dem offiziellen Code-Verwaltungssystem und kompiliert diesen lokal. Bei einer vollkommen identischen Hardware und Konfiguration des Betriebssystems ergibt sich hier der Vorteil, dass die kompilierte ausführbare Datei zwischen den Pentestern ausgetauscht werden kann. Ist der Quellcode nicht öffentlich, sollte die Software auf jeden Fall nur von der Herstellerseite heruntergeladen und über die Checksumme überprüft werden.

Ein weitere wichtiger Punkt ist die Aktualität der Software. Hier gibt es zwei Aspekte, welche sich manchmal gegenüber stehen. Zum einen sollte gerade Software, welche zum Aufdecken von Schwachstellen genutzt wird, immer aktuell sein um auch neueste Angriffe abprüfen zu können. Auf der anderen Seite steht die Verfügbarkeit und die Vergleichbarkeit. So will man nicht, dass eine Software zum Start eines Pentests wegen zum Beispiel Paketkonflikten nicht mehr funktioniert, oder dass zwei Pentester mit unterschiedlichen Versionen arbeiten und daher unterschiedliche Ergebnisse für den gleichen Test erlangen. Eine Lösung für das Problem könnten virtuelle Maschinen darstellen. So kann man vor

jedem Pentest einen Snapshot erstellen, ein Update durchführen und falls es Probleme gibt, welche sich bis zum nächsten Pentest nicht beheben lassen, auf diesen zurückspringen. Nach dem Pentest kann dann in Ruhe an dem jeweiligen Problem weitergearbeitet werden. Zudem hat dies den Reiz, dass man die virtuelle Maschine nach dem Pentest löschen kann und so garantiert keine Fragmente im Betriebssystem verbleiben.

Unabhängig davon, ob die Daten des Pentests nur in einer virtuellen Maschine oder auf dem physikalischen Client gehalten werden, sollte das Laptop an sich geschützt sein. Maßnahme dazu sind die richtige Konfiguration des BIOS, Festplattenverschlüsselung und das deaktivieren von bestimmten Anschlüssen oder Hardware-Features. So sollte das BIOS mit einem Kennwort geschützt und die Boot-Reihenfolge möglichst restriktiv gesetzt sein. Dies auch bei Laptops mit Festplattenverschlüsselung empfohlen, da sonst der Bootloader der Full-Disk-Encryption mit Malware (zum Beispiel über USB) infiziert werden könnte.<sup>2</sup>

## 3.2. Durchführung

Eine gute Vorbereitung vereinfacht in vielerlei Hinsicht die Durchführung von Pentests, zum Beispiel weil funktionierende Hard- und Software bereits zur Verfügung steht und nicht während dem Pentest Zeit auf Update und Konfiguration verwendet wird. Trotzdem gibt es einige bewährte Vorgehensweisen, welche man in Prozessen festhalten oder auf welche man sich Einigen sollte.

### 3.2.1. Kickoff

Unmittelbar vor dem Pen-Test sollte ein Kickoff durchgeführt werden. Zu diesem sollten alle verantwortlichen Stellen eingeladen werden (Business-Unit Manager, System-Administrator, den Sicherheitsverantwortlichen des Projekt, evtl. Informationssicherheit und Hosting-Provider) und Kern-Fragen abgesprochen werden.

Im Rahmen dieser Arbeit wurden folgende Fragend definiert und mit Ansprechpartnern der Allianz Deutschland AG geprüft.

#### Allgemein

- Wie ist der Projekt-Name?
- Wer sind die Teilnehmer?
- Wann soll der Test durchgeführt werden?
- Was ist das Ziel des Tests?
- In welcher Stage befindet sich die Anwendung? Development Test System Integration Produktion
- Welche IP-Adressen sollen getestet werden?

---

<sup>2</sup>[https://www.schneier.com/blog/archives/2009/10/evil\\_maid\\_attac.html](https://www.schneier.com/blog/archives/2009/10/evil_maid_attac.html)

- Welche URLs sollten getestet werden?
- Welche Zugangsdaten sollen genutzt werden?
- Sollen Denial-Of-Service-Angriffe durchgeführt werden? Ja Nein
- Liegt ein Haftungsschluss vor? Ja Nein
- Liegt eine Erlaubnis des Server-Betreibers vor? Ja Nein
- Gibt es eine Deadline für den Bericht? Ja Nein
- Erste Ergebnisse am Ende des Pen-Tests in einfacher Form zukommen lassen (z.B. Excel)? Ja Nein

#### **Fragen an den System-Administrator**

- Gibt es Systeme, die als instabil angesehen werden (alte Patch-Stände, Legacy Systeme etc.)? Ja Nein
- Gibt es Systeme von Dritten, die ausgeschlossen werden müssen oder für die weitere Genehmigungen notwendig sind? Ja Nein
- Was ist die Durchschnittszeit zur Wiederherstellung der Funktionalität eines Services?
- Ist eine Monitoring-Software im Einsatz? Ja Nein
- Welche sind die kritischsten Applikationen?
- Werden in einem regelmäßigen Turnus Backups erstellt und getestet? Ja Nein

#### **Fragen an den Business Unit Manager**

- Ist die Führungsebene über den Test informiert?
- Ja Nein
- Welche Daten stellen das größte Risiko dar, falls diese manipuliert werden?
- Gibt es Testfälle, die die Funktionalität der Services prüfen und belegen können? Ja Nein
- Sind "Disaster Recovery Procedures" vorhanden? Ja Nein

#### **Abschluss**

- Offene TODOs

Um den Kickoff möglichst effizient zu gestalten, wurden auch diese Fragen über die gleiche Technik wie bei TODO in eine Webanwendung integriert. Diese ist auf dem Datenträger unter TODO abgelegt.

### 3.2.2. Kategorisierung von Findings

Um einen besseren Überblick über Findings zu bekommen, werden diese meist in Kategorien eingeteilt. Im Folgenden werden zwei Methoden, die OWASP TOP 10 und die CWE (*Common Weakness Enumeration*), vorgestellt.

#### 3.2.2.1. OWASP TOP 10

Die *OWASP Foundation* (Open Web Application Security Project) ist ein non-Profit Organisation mit dem Ziel es Unternehmen zu vereinfachen sichere Software zu schreiben.

Im Rahmen des Projekts erfasst die OWASP alle 3 Jahre, zuletzt für das Jahr 2013, aktuell für das Jahr 2016, die am meisten aufgetretenen Schwachstellen und veröffentlicht die sogenannte *OWASP TOP 10*. Aus dem Datensatz von 2013 ergeben sich folgende Kategorien<sup>[10]</sup>:

- A1-Injection
- A2-Broken Authentication and Session Management
- A3-Cross-Site Scripting (XSS)
- A4-Insecure Direct Object References
- A5-Security Misconfiguration
- A6-Sensitive Data Exposure
- A7-Missing Function Level Access Control
- A8-Cross-Site Request Forgery (CSRF)
- A9-Using Components with Known Vulnerabilities
- A10-Unvalidated Redirects and Forwards

Diese sind in der Security-Szene weit verbreitet und können gut genutzt werden, um Pen-Test-Findings zu kategorisieren.

#### 3.2.2.2. Common Weakness Enumeration

Eine Alternative ist die von der MITRE Corporation (mit Unterstützung verschiedener anderer Stellen) entwickelte *Common Weakness Enumeration*. Diese ist wesentlich feingranularer als die OWASP TOP 10, so umfasst die Version 2.10 vom 19.01.2017 über 1000 verschiedene Schwachstellen.<sup>3</sup><sup>[6]</sup>

Dies hat den Vorteil, dass man die Schwachstellen meist genau einer Kategorie zuweisen kann. Jedoch ist die Zuordnung wesentlich aufwändiger.

---

<sup>3</sup>[http://cwe.mitre.org/data/published/cwe\\_v2.10.pdf](http://cwe.mitre.org/data/published/cwe_v2.10.pdf)

Risikostufe	CVSS3-Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Tabelle 3.1.: Zuordnung von CVSS3-Score zur Risikostufe [5]

### 3.2.3. Bewertung von Findings

Für die in einem Pentest gefundenen Findings sollte direkt nach deren Entdeckung eine Bewertung durchgeführt werden. Dies ist notwendig, um entsprechende Eskalationsstufen zu informieren, wenn kritische Findings auftreten. Zur Bewertung bieten sich verschiedene Systeme an, im Folgenden werden *CVSS* und *DREAD* vorgestellt und verglichen.

#### 3.2.3.1. CVSS

Das CVSS (Common Vulnerability Scoring System) wurde entwickelt von und gehört der FIRST.Org, Inc., einer in Amerika ansässigen wohltätigen Organisation, welche Lösungen zur Koordination und Unterstützung von IT-Security-Teams entwickelt.

Es ist ein System, welches Anhand von mehreren Attributen die Charakteristiken und Risiko-Gerade von Schwachstellen bestimmt. Dabei gibt es 3 Metriken: *Base*, *Temporal* und *Environmental*. Die *Base*-Metrik beschreibt den Grund-Score einer Schwachstelle, die *Temporal*-Metrik der Score zum aktuellen Zeitpunkt und die *Environmental*-Metrik den Score in einem bestimmten Umfeld. Der Base-Score ist verpflichtend auszufüllen, die anderen Metriken können das Ergebnis verfeinern, sind aber nicht zwingend notwendig.

Die Base-Metrik ergibt einen Score zwischen 0.0 und 10.0, welcher durch die beiden Zusatz-Metriken erhöht oder geschwächt werden kann. Eine komprimierte Darstellungsweise stellt der CVSS-Vector-String dar, welcher die Attribute aus allen drei Metriken sowie deren jeweiligen Werte komprimiert anzeigt. Die Attribute für die Base-Metrik sind im Vergleich unter 3.2.3.3 dargestellt. Die Attribute für die *Temporal*- und *Environmental*-Metrik genauen Formeln können der Spezifikation<sup>4</sup> entnommen werden.[4]

Zusätzlich gib es für den Score eine Zuordnung zu 5 größeren Risikostufen. Diese ist Tabelle 3.1 zu entnehmen.

Ebenso gibt es für CVSS Module für zum Beispiel Python<sup>5</sup>, um CVSS-Vector-Strings auszuwerten und die Scores für die einzelnen Metriken zu berechnen.

<sup>4</sup><https://www.first.org/cvss/specification-document>

<sup>5</sup><https://pypi.python.org/pypi/cvss>

### 3.2.3.2. DREAD

DREAD wie CVSS ein Metrik-System zur Einschätzung des resultierenden Risikos aus einer Schwachstelle. DREAD ist dabei das Akronym für die fünf bewerteten Attribute der Schwachstelle. Jedes Attribut hat einen Wert von 0 bis 10, wobei 10 immer der Schlimmste anzunehmende Fall ist. Im Folgenden sind die Attribute sowie grobe Richtwerte pro Attribut aufgeführt.[11]

**Damage:** Wie viel Schaden würde eine Ausnutzung der Schwachstelle bedeuten?

- 0 Kein Schaden
- 5 Die Daten eines einzelnen Users sind betroffen
- 10 Komplette Zerstörung der Daten oder des Systems

**Reproducibility:** Wie verlässlich funktioniert der Exploit?

- 0 Selbst mit erhöhten Rechten ist ein funktionierender Exploit äußerst unwahrscheinlich
- 5 Mehrstufiges Vorgehen notwendig, es gibt vorgefertigte Scripte oder Tools
- 10 Nicht authentifizierte User können den Exploit trivial ohne Hilfsmittel reproduzierbar durchführen

**Exploitability:** Wie schwer ist es die Schwachstelle auszunutzen?

- 0 Darf nicht vergeben werden, es wird angenommen dass jede Schwachstelle mit genügend Aufwand exploitable ist
- 1 Selbst mit direktem Wissen der Schwachstelle gibt derzeit keine bekannte Methode zur Ausnutzung
- 5 Der Exploit ist öffentlich, mittleres Können durch den Angreifer benötigt, Angreifer muss authentisiert sein
- 7 Der Exploit ist öffentlich, Angreifer muss nicht authentisiert sein
- 10 Triviale Ausnutzung, zum Beispiel über einen Webbrowser

**Affected Users:** Wie viele Nutzer betrifft die Schwachstelle?

- 0 Keine User betroffen
- 5 Einige User betroffen, aber nicht alle
- 10 Alle User betroffen

**Discoverability:** Wie einfach ist die Schwachstelle zu finden?

- 0 Selbst mit Source-Code-Zugriff und erhöhten Rechten schwer zu finden
- 5 Kann durch Netzwerk-Dumps oder Fuzzing gefunden werden
- 9 Schwachstelle ist öffentlich bekannt und kann über Such-Maschinen gefunden werden
- 10 Schwachstelle ist direkt auf der Webseite oder in der Adressleiste des Browser zu erkennen

Sind Werte für die einzelnen Attribute bestimmt, kann der Score über folgende Formel berechnet werden:[9]

$$SCORE_{DREAD} = \frac{DA + R + E + A + DI}{5}$$

### 3.2.3.3. Vergleich von CVSS und DREAD anhand einer XSS-Lücke

Im Folgenden wird CVSS3 mit DREAD verglichen. Als Basis gilt eine Reflected-Cross-Site-Scripting Lücke, wie Sie bereits in einem Beispiel FIRST.Org beschrieben wird.<sup>6</sup>

Die Lücke besteht nur in einer bestimmten Version der Webanwendung. Ebenso muss ein valider Datenbankname in den Exploit integriert werden, damit dieser funktioniert. Das System hat die *HTTPOnly*-Flag gesetzt.

#### CVSS3

Die CVSS3-Methodik würde die Vulnerability wie folgt bewerten:

**Attack Vector: Network** Die Schwachstelle kann über das Netzwerk erreicht werden.

**Attack Complexity: Low** Although an attacker needs to perform some reconnaissance of the target system, a valid session token can be easily obtained and many systems likely use well-known or default database names.

**Privileges Required: None** An attacker requires no privileges to mount an attack.

**User Interaction: Required** A successful attack requires the victim to visit the vulnerable component, e.g. by clicking a malicious URL.

**Scope: Changed** The vulnerable component is the web server running the phpMyAdmin software. The impacted component is the victim's browser.

**Confidentiality Impact: Low** Information maintained in the victim's web browser can be read and sent to the attacker. This is constrained to information associated with the web site running phpMyAdmin, and cookie data is excluded because the *HttpOnly* flag is enabled by default by phpMyAdmin. If the *HttpOnly* flag is not set, the Confidentiality Impact will become High if the attacker has access to sufficient cookie data to hijack the victim's session.

**Integrity Impact: Low** Information maintained in the victim's web browser can be modified, but only information associated with the web site running phpMyAdmin.

**Availability Impact: None** The malicious code can deliberately slow the victim's system, but the effect is usually minor and the victim can easily close the browser tab to terminate it.

Daraus ergibt sich der CVSS3-String *CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N* und ein Rating von 6.1.

---

<sup>6</sup><https://www.first.org/cvss/examples>



## DREAD

Unter DREAD gibt es leider keine Beispiele für eine Reflected-Cross-Site-Scripting Lücke. Daher sind die Werte unten selbst gewählt.

**Damage: 3** Die auf der Webseite angezeigten Daten können gelesen und manipuliert werden. Aufgrund des *HTTPOnly*-Flags des Cookies kann die Session jedoch nicht einfach übernommen werden.

**Reproduceability: 10** Der Angriff kann mit einem Web-Browser jederzeit reproduziert werden.

**Exploitability: 7** Der Angriff benötigt einen gültigen Datenbank-Namen. Da es viele Datenbanken mit Standardnamen gibt, ist der Exploit jedoch weiterhin relativ einfach. Zusätzlich muss ein registrierter User im angemeldeten Zustand auf einen manipulierten Link des Angreifers klicken.

**Affected Users: 10** Alle User sind betroffen.

**Discoverability: 8** Es muss auf die Version der Webanwendung geprüft werden.

Daraus ergibt sich ein Score von 7.6.

## Fazit

Der Score von DREAD ist mit einem Score von 7.6 um 1.5 Punkte höher als CVSS3 und muss als „Hoch“ gewertet werden. Dies beruht darauf, dass die Attribute *Reproduceability* und *Affected Users* auf 10 gesetzt werden müssen. Für eine nur mittel-triviale XSS-Lücke ohne die Möglichkeit zur Session-Übernahme scheint dies etwas zu hoch. Der CVSS3-Score mit 6.1 scheint als mittleres Finding durchaus angemessen. Dies ist jedoch nur ein Beispiel, für andere Schwachstellen ist es durchaus möglich, dass DREAD eine besser passende Klassifizierung darstellt. Im Endeffekt sollte nur Konsistent ein System für Pentests eingesetzt werden, damit die Findings vergleichbar bleiben.

### 3.2.4. Dokumentation

Ein weiterer wichtiger Punkt während der Durchführung eines Pentests ist die fortlaufende Dokumentation. So sollten sowohl Fortschritte bezüglich Schwachstellen sowie alle durchgeführten Aktionen dokumentiert werden. Im Folgenden werden verschiedene Methoden vorgestellt, welche diese Dokumentation vereinfachen.

#### 3.2.4.1. Netzwerkverkehr

Eine der effektivsten Methoden zur Dokumentation bei Web-Applikation und Web-Service Pentests ist das Aufzeichnen des Netzwerkverkehrs. So können jegliche am Server passierenden Aktionen später über den Netzwerkdump einer Aktion des Pentesters zugeordnet werden. Dies ist gerade bei Ausfällen der getesteten Anwendung äußerst hilfreich.

Technisch kann die Aufzeichnung über mehrere Programme leicht realisiert werden. So kann sowohl Wireshark<sup>7</sup> wie auch TCPDump<sup>8</sup> den gesamten Verkehr eines oder mehrerer Netzweradapter aufzeichnen und als PCAP speichern.

Ein Problem ist, wenn die Verbindung über TLS/SSL geschützt ist. Da normalerweise der *Private-Key* des Zielsystems nicht zur Verfügung steht, muss die Verbindung unterbrochen werden. Dies kann über MITMProxy<sup>9</sup> bewerkstelligt werden, welche einen Proxy-Server aufbaut. Dieser Proxy-Server baut zum Ziel eine TLS-Verbindung auf, stellt jedoch zum Pentester eine separate TLS-Verbindung auf, für welche ein privates Zertifikat hinterlegt werden kann. Zeichnet man nun den Netzwerkverkehr zwischen dem Pentester und MITMProxy auf, können die TLS-Verbindungen später mit dem privaten Zertifikat (zum Beispiel in Wireshark) decrypted und gesichtet werden.

#### 3.2.4.2. Konsoleneingaben

Zusätzlich zu Netzwerkdumps kann eine Aufzeichnung der Terminal-Sessions während des Pentests sinnvoll sein. So kann die Ausgabe von Konsolen-Kommandos auch in dem Fall, dass die Bedeutung einer Ausgabe erst später klar wird, einfach in die Dokumentation aufnehmen. Um das Transkript zu erstellen, kann das Linux-Tool *screen*<sup>10</sup> verwendet werden. Ein Beispiel-Aufruf wäre:

```
1 script "$(date +%Y-%m-%d %H:%M:%S").log" -t 2> "$(date +%Y-%m-%d %H:%M:%S").time"
```

Die erstellten Dateien können anschließend über

```
1 scriptreplay -t timestamp.time timestamp.log
```

abgespielt werden.

#### 3.2.4.3. Findings

Neben Netzwerkverkehr und Konsoleneingaben sollten auch Findings, sowie Hinweise darauf, möglichst umgehend dokumentiert werden. Dazu sollte eine standardisierte Form gefunden werden. Wie diese genau aussieht, kann den Penstestern überlassen werden. Ein Beispiel wäre ein Excel-Sheet mit den wichtigsten Spalten (Name, Kategorie, Beschreibung, CVSS3), da diese Infos dort ohne viel Aufwand eingetragen werden können. Zusätzlich sollten Findings, sowie auch schon Hinweise auf mögliche Schwachstellen, umgehend mit einem Screenshot dokumentiert werden. Auf Linux-Laptops können Screenshots unabhängig vom Windows-Manager (solange ein X-Server läuft) über das *import*-Kommando von ImageMagick<sup>11</sup> genommen werden.

---

<sup>7</sup><https://www.wireshark.org/>

<sup>8</sup><http://www.tcpdump.org/>

<sup>9</sup><https://mitmproxy.org/>

<sup>10</sup><http://man7.org/linux/man-pages/man1/script.1.html>

<sup>11</sup><https://www.imagemagick.org/script/import.php>

### 3.3. Nachbereitung

Im Anschluss an die Durchführung des Pentests müssen die Ergebnisse aufbereitet und dem Kunden präsentiert werden. Dies passiert meist in Form eines sogenannten Pentest-Reports. Im Folgenden wird auf die Inhalte sowie die Erstellung eingegangen.

#### 3.3.1. Inhalte eines Pentest-Berichts

Im Folgenden werden die typischen Inhalte eines Pentest-Berichts kurz dargestellt. Der Abschnitt ist unterteilt in Allgemeine Informationen (3.3.1.1), Management Summary (3.3.1.2), Technische Zusammenfassung (3.3.1.3) und Findings (3.3.1.4).

##### 3.3.1.1. Allgemeine Informationen

Der Abschnitt „Allgemeinen Informationen“ hält alle organisatorischen Informationen zum Pen-Test. Dies umfasst Autor und Datum des Berichts, den Testzeitraum, die Pen-Tester sowie die Ansprechpartner im Projekt.

##### 3.3.1.2. Management Summary

Die „Management Summary“ ist, wie der Name schon sagt, als Zusammenfassung für das Management gedacht. Sie hat jedoch auch die Funktionen, dem Leser kurz die kritischen Erkenntnisse des Pen-Tests zu vermitteln. Dazu ist die Summary in die Abschnitte „Ausgangssituation“, „Überblick über die Befunde“ und „Risikomatrix“ unterteilt.

**Ausgangssituation** Die Ausgangssituation beschreibt kurz, wie die Anwendung zustande gekommen ist und in welcher Projektphase sich diese befindet. Alternativ oder zusätzlich kann der Grund für den Pen-Test dargestellt werden.

**Überblick über die Befunde** In diesem Abschnitt wird ein kurzer Überblick über die Befunde gegeben, um den Leser einen ersten Eindruck zu vermitteln. Dabei kann die Aufmerksamkeit auf besonders kritische Findings gelenkt werden.

**Risikomatrix** Die Risikomatrix stellt die Findings in eingeordnet nach Eintrittswahrscheinlichkeit und Schadenspotenzial dar und ermöglicht so dem Leser, sich einen schnellen Überblick über die Verteilung der Findings bezüglich des Gesamtrisikos zu verschaffen.

##### 3.3.1.3. Technische Zusammenfassung

Die „Technische Zusammenfassung“ soll wie die „Management Summary“ einen kurzen Überblick geben, bezieht jedoch bereits technische Aspekte mit ein und ist ausführlicher wie die „Management Summary“. Sie ist in vier Abschnitte unterteilt, welche im Folgenden dargestellt werden.

**Testobjekt** Im Abschnitt „Testobjekt“ wird kurz das oder auch die Testobjekte beschrieben. Dabei kann es sich bei Web-Applikation- oder Web-Service-Pen-Tests zum Beispiel um IP-Ranges oder URLs handeln.

**Verwendete IPs** Unter „verwendete IPs“ ist die externe IP der Pentester festgehalten, welche zum Testen genutzt wurde. Diese sollte sich, wie in 3.1.3.2 beschrieben, auf eine Adresse begrenzen.

**Zugangsdaten und Benutzerkonten** Ebenfalls werden durch den Auftraggeber festgelegte Zugangsdaten und Benutzerkonten im Bericht dokumentiert. Dabei besteht ein Datum im Normalfall aus Benutzername, Passwort und Rollenbeschreibung.

**Überblick der Ereignisse** Am Ende der „Technischen Zusammenfassung“ werden anhand einer Tabelle alle Findings erneut dargestellt. Im Gegensatz zur Risikomatrix sind hier Titel, Kategorie und andere Informationen enthalten.

#### 3.3.1.4. Findings

Als letzter Punkt werden die Findings im Detail dargestellt. Dabei werden verschiedene Daten dargestellt, welche im Folgenden kurz aufgeführt werden.

**Allgemeines** Anfangs werden die allgemeinen Informationen zu dem Finding dargestellt. Dies umfasst die Nummer, den Namen, die Kategorie sowie den Status des Findings.

**Beschreibung** Anschließend an den allgemeinen Teil folgt eine genaue Beschreibung des Findings. Dabei können und sollen durchaus technische Details und Angriffsszenarien dargestellt werden.

**Belege** Unter Belege sollten alle die Beschreibung ergänzenden Screenshots oder Textstücke (zum Beispiel die Ausgabe eines Programms) angehängt werden. Sie dienen dazu, das Finding zu erklären und verdeutlichen.

**Kritikalität** In diesem kurzen Teil wird erneut die Eintrittswahrscheinlichkeit, das Schadenspotenzial sowie das daraus resultierende Gesamtrisiko dargestellt.

**CVSS** Ergänzend zur Kritikalität wird der CVSS-Score (oder falls anders beschlossen, der DREAD-Score) mit den einzelnen Metriken und dem CVSS-Score-Vector dargestellt.

**Empfehlung** Abschließend folgt eine Empfehlung, wie mit dem Finding umgegangen werden sollte.

### 3.3.2. Implementierung in LaTeX

TODO: Implementierung in Latex?

### 3.3.3. Implementierung als Webanwendung

Traditionell werden Pen-Test-Reports in LaTeX geschrieben. Dies hat den Vorteil, dass die Reports eine einheitliche Formatierung vorweisen und professionell aussehen. Leider ist die Report-Erstellung oft mühselig und erfordert einigen Aufwand, sodass oft, abhängig von der Länge des Tests, mehrere Tage für den Report in Anspruch genommen werden.

Um diesen Aufwand zu minimieren, wurde auch hier eine Web-Anwendung entwickelt. Diese baut auf die gleiche Technologie auf wie die, welche für den Fragebogen genutzt

wurde (siehe Abschnitt 3.1.1.3. So werden eingaben ebenfalls über eine HTML5-Oberfläche aufgenommen und im Hintergrund in ein LaTeX-Dokument eingefügt. Anschließend wird das LaTeX-Dokument automatisch kompiliert und als PDF an den User präsentiert.

Ein Feature, welches besonders Zeit spart, ist die Möglichkeit Templates für Findings zu nutzen. So wurden im Rahmen dieser Masterarbeit mehrere Templates für häufig vorkommende Findings angelegt. Diese sind:

- Session-Timeout nicht gesetzt
- Offene Ports
- Fehlende Passwort-Richtlinie
- Ungeschützte Cookies
- Kein Bruteforce-Schutz im Login
- Reflected-XSS in diversen Eingabefeldern
- Persistent-XSS in diversen Eingabefeldern
- Kein Schutz gegen CSRF

Für jedes Finding-Template wurde jeweils Name, Kategorie, Beschreibung, Empfehlung, Schadenspotenzial, Eintrittswahrscheinlichkeit, Gesamtrisiko sowie einen Standard-CVSS3-Score vor ausgefüllt. Der User kann die Texte und Metriken in der Weboberfläche anpassen und Belege hinzufügen.

### 3.4. Kontinuität

Ein weitere wichtiger Teil der Prozesse um Pen-Tests ist Kontinuität. So reicht es nicht, nur einen Pen-Test durchzuführen, wenn eine Anwendung das erste Mal online genommen wird. Denn durch Weiterentwicklung sowie neue entdeckte Schwachstelle kann sich das Sicherheitsniveau ständig ändern.

Allgemein gilt, dass das Sicherheitsniveau einer Anwendung immer mindestens der Kritikalität des Anwendung für das Unternehmen entsprechen sollte. Hat eine Anwendung eine hohe Kritikalität, so sollte auch mindestens ein hohes Sicherheitsniveau gelten.

Natürlich sollte dafür festgelegt werden, wie die Kritikalität und das Sicherheitsniveau festgelegt sind. Die Kritikalität pro Anwendung ist meist Abhängig vom Unternehmen. So würde für einen Automobilhersteller wohl eine Produktionsanlage als hoch kritisch gesehen, bei einem Online-Vertrieb wohl eher der Online-Shop.

Auch das Sicherheitsniveau sowie dessen Aufrechterhaltung muss im Unternehmen definiert werden. Dabei sollte zuerst Maßnahmen definiert werden.

**Pen-Tests** sind eine äußerst effektive, aber kostspielige Maßnahme.

	Jahr 1	Jahr 2	Jahr 3	Jahr 4
Sehr Hoch	Pen-Test	Pen-Test	Pen-Test	Pen-Test
Hoch	Code-Audit Security-Scan	Pen-Test	Code-Audit	Pen-Test Security-Scan
Mittel	Security-Scan Sourcecode-Scan	Code-Audit	Security-Scan Sourcecode-Scan	Pen-Test
Niedrig	Security-Scan	Security-Scan	Security-Scan	Security-Scan
Sehr Niedrig	-	Security-Scan	-	Security-Scan

Tabelle 3.2.: Einteilung von Maßnahmen zum Sicherheitsniveau über bestimmte Zeiträume

**Code-Audits** sind von Personen ausgeführte Analysen des Quellcodes auf Schwachstellen. Das das Personal äußerst gut geschult sein muss, sollten hierfür entweder im Unternehmen Experten eingestellt oder extern gebucht werden. Beim Einkauf von externen Dienstleistern, sind die Audits ähnlich teuer wie Pentests, sind aber ebenfalls eine sehr effektive Maßnahme.

**Security-Scanner** meint automatisierte Sicherheits-Scans durch Software wie *IBM Security AppScan*<sup>12</sup> oder der *Nessus Vulnerability Scanner*<sup>13</sup>. Security-Scanner sind nicht so genau wie Pen-Tests, dafür sind diese, aber einer gewissen Anzahl von Anwendungen, wesentlich günstiger.

**Sourcecode-Scans** ebenso wie Security-Scanner sind Sourcecode-Scanner Produkte, welche automatisiert Analysen durchführen. Produkte wären beispielsweise *Fortify Static Code Analyzer*<sup>14</sup> oder *Veracode Static Analysis*<sup>15</sup>. Sourcecode-Scans sind ähnlich effektiv wie Security-Scanner.

Hat ein Unternehmen die passenden Maßnahmen aufgebaut, sollten diese über eine zeitliche Einteilung den Sicherheitsniveaus zugeordnet werden. Ein frei gewähltes Beispiel dafür ist in Tabelle ?? zu sehen. Dabei ist Jahr 1 das erste Jahr, nachdem die Anwendung (nach einem Pen-Test) online genommen wurde.

<sup>12</sup><http://www-03.ibm.com/software/products/de/appscan>

<sup>13</sup><https://www.tenable.com/products/nessus-vulnerability-scanner>

<sup>14</sup><http://www8.hp.com/de/de/software-solutions/static-code-analysis-sast/>

<sup>15</sup><https://www.veracode.com/products/binary-static-analysis-sast>

## 4. Penetrationstests mobiler Anwendungen

Durch eine zunehmende Anfrage des Marktes auf mobile Anwendungen entwickelt die Allianz Deutschland AG zunehmende mobile Applikationen. Dafür müssen nicht nur die bestehenden Prozesse angepasst, sondern auch neue Werkzeuge geschaffen werden, welche die Security-Prozesse unterstützen.

Im Folgenden sind die Anforderungen an ein solches Werkzeug festgehalten, gefolgt mit einer Evaluierung bestehenden Programme. Daraufhin wurde ein passendes Werkzeug ausgewählt und die Weiterentwicklung dieses unter Abschnitt 4.6 beschrieben. Abschließend werden die erreichten mit den ursprünglichen Anforderungen verglichen.

### 4.1. Anforderungen

Im Folgenden sind die Anforderungen festgehalten, welche für einen echt-Einsatz in der Allianz Deutschland AG notwendig sind.

**Einfache Einrichtung** Ebenso sollte das Tool möglichst unkompliziert einem Benutzer zur Verfügung gestellt werden können. Dies könnte entweder über eine einfache Installation oder durch die Ausführung auf einer zentralen, erreichbaren Instanz realisiert werden.

**Einfache Handhabung** Im Optimalfall soll ein Scan bereits nach einer kurzen Einführungszeit durch einen zuvor ungeschulten Mitarbeiter durchgeführt werden können. Dazu muss das Tool einfach bedienbar sein.

**Verständliche Ergebnisse** Die Ergebnisse des Werkzeugs sollten für einen in der Informationssicherheit arbeitenden Angestellten verständlich sein. Hier kann eine gewisse Fachkenntnis vorausgesetzt werden und Grundlagen (wie zum Beispiel warum *memcpy* auf eine mögliche Schwachstelle hinweist) müssen durch das Werkzeug nicht vermittelt werden.

**Unterstützung für Android/iOS/Windows Phone** Um zu verhindern, dass für jede der aktuell geläufigen Plattformen ein eigenes Werkzeug genutzt werden muss, sollte das Werkzeug die Analyse von Apps für *Android*, *iOS* und *Windows Phone* unterstützen.

**Niedrige False-Positive-Rate** Das Werkzeug sollte eine möglichst geringe Rate an *False-Positives*, also falschen Meldungen, aufweisen. Jedoch sollten auch keine Hinweise auf Schwachstellen verworfen werden, sodass das Werkzeug eine Einstufung der Ergebnisse vornehmen sollte.

**Reproduzierbarkeit** Das Werkzeug sollte reproduzierbare Ergebnisse liefern, also bei gleichen Eingangswert das gleiche Ergebnis generieren. Dies für die Nachverfolgung von Meldungen äußert wichtig.

## 4.2. Bestehende Anwendungen

Trotz der relativ neuen Thematik der Mobilen Applikationen gibt es schon einige Programme und Applikationen, die bei der Identifizierung von Schwachstellen helfen können. Im Folgenden sind diese unterteilt in *All-In-One-Framework* und Einzelanwendungen. Die Namen sind hierbei sprechend: Sogenannte *All-In-One-Frameworks* bündeln mehrere kleine Anwendungen und automatisieren den Ablauf oder vereinfachen die Bedienung.

### 4.2.1. All-In-One-Framework: MobSF

*MobSF* ist das einzige, derzeit öffentlich Verbreitete All-In-One-Framework zur Analyse von Mobilen Applikationen. Es ist eine Plattform zur statischen Analyse von Android und iOS-Apps sowie zur dynamischen Analyse von Android Apps. Es bündelt viele kleinere Anwendungen, welche unter 4.2.2 aufgeführt sind, in einer einfachen Weboberfläche. Es ist Open-Source, in *Python* geschrieben und steht in *GIT* frei zur Verfügung.<sup>1</sup> Die aktuelle Version ist *0.9.2 beta*.

Es unterstützt die statische Analyse von Apps in den Formaten *APK* und *IPA* sowie aus einfach komprimierten Archiven (*zip*). Zusätzlich beinhaltet *MobSF* einen eingebauten API Fuzzer und ist in der Lage, API-spezifische Schwachstellen wie *XXE*, *SSRF* oder *Path Traversal* zu erkennen (TODO Auflisten).

### 4.2.2. Einzelanwendungen

Das All-In-One-Framework *MobSF* greift im Hintergrund oft auf eigenständige Tools zurück. Da es für Penetration-Test oft hilfreich ist, diese ohne ein umgebendes Framework nutzen zu können, sind im Folgenden die wichtigsten Tools kurz aufgeführt.

Für Android-Apps:

**jd-core** ist eine Java Decompiler für Java 5 und spätere Versionen. Er steht unter <http://jd.benow.ca/> zur Verfügung und kann zum Beispiel über das auf der selben Seite zur Verfügung gestellte JD-GUI genutzt werden.

**Dex2Jar (d2j)** ist ein Tool zum Umwandeln von *.dex*-Dateien (Dalvik-Bytecode) zu normalen Java-Bytecode (gepackt in einem Jar-File). Anschließend können normale Java-Tools zur Analyse verwendet werden. Das Tool ist kostenlos, Open-Source und in Github<sup>2</sup> verfügbar.

**enjarify** ist eine modernere alternative zu *Dex2Jar*. *enjarify* wurde von Google entwickelt, ist jedoch trotzdem unter der Apache-Lizenz in Github veröffentlicht<sup>3</sup>.

**Dex2Smali** stammt unterstützt ebenfalls bei der Konvertierung von *.dex*-Files in andere Formate. In diesem Fall ist das Zielformat *smali*. Dieses Tool ist ebenfalls Open-Source und in Github<sup>4</sup> zu finden.

---

<sup>1</sup><https://github.com/ajinabraham/Mobile-Security-Framework-MobSF>

<sup>2</sup><https://github.com/pxb1988/dex2jar>

<sup>3</sup><https://github.com/google/enjarify>

<sup>4</sup><https://github.com/JesusFreke/smali>



**procyon** ist ein Framework zur Analyse von Java-Bytecode. Insbesondere ist ein Decompiler enthalten, welcher den Bytecode in lesbaren Java-Code umwandelt. Das Tool ist kostenlos auf Bitbucket<sup>5</sup> verfügbar.

Für iOS-Apps:

**otool**, auch „object file displaying tool“ genannt, ist ein Tool zur Analyse von Object-Files. Es ist auf Mac OS X bei der Installation von XCode enthalten. Es bietet viele brauchbare Funktionen wie die Auflistung der *shared libraries* oder der „indirect symbol table“.

Für Windows-Phone-Apps:

**BinScope** ist Security-Analyse-Tool für Windows-Applikationen. Es wurde von Microsoft für den *Secure Development Lifecycle* entwickelt und steht auf der Microsoft-Webseite<sup>6</sup> zur Verfügung.

**BinSkim** ist der Nachfolger von BinSkim. Jedoch wurden in dieser Masterarbeit mit BinScope oft bessere Ergebnisse erzielt. BinSkim wurde ebenfalls von Microsoft für den *Secure Development Lifecycle* entwickelt und kann über *nuget*<sup>7</sup> bezogen werden.

### 4.3. Aktuelle Situation und Vergleich der Emulation

Ein wichtiger Bestandteil in der dynamischen Analyse von Apps ist die Möglichkeit Applikationen in einer emulierten Umgebung auszuführen. Im Folgenden werden diese Möglichkeiten für die iOS, Windows-Phone und Android getestet.

#### 4.3.1. iOS

Im Folgenden wurde speziell die der in XCode enthaltene, offizielle iPhone-Simulator in seiner Funktionsweise untersucht.

##### 4.3.1.1. Emulation

Die Emulation von iOS-Geräten ist derzeit mit der Verwendung von XCode möglich. XCode wiederum ist nur unter Mac OS X erhältlich. Da Mac OS X laut EULA nur auf „Apple-branded computers“ verwendet werden darf [3], ist die Simulation von iOS-Geräten nur unter Apple-Hardware möglich. Nach der Installation über den in Mac OSX enthaltenen App-Store kann ein virtualisiertes iPhone über die Schritte XCode → Open Developer Tools → Simulator oder über

```
1 /Applications/Xcode.app/Contents/Developer/Applications/  
   Simulator.app
```

<sup>5</sup><https://bitbucket.org/mstrobel/procyon/overview>

<sup>6</sup><https://blogs.microsoft.com/microsoftsecure/2012/08/15/microsofts-free-security-tools-binscope-binary-analyzer/>

<sup>7</sup><https://www.nuget.org/packages/Microsoft.CodeAnalysis.BinSkim/>

gestartet werden.

#### 4.3.1.2. Debugging

Als Debugger unter *Mac OS X* hat sich *LLDB* etabliert und stellt das Pendant zu GDB unter Linux dar. *LLDB* ist kostenlos verfügbar, Open-Source und steht unter der University of Illinois/NCSA Open Source License<sup>8</sup>, welche die Vervielfältigung und Veränderung des Quellcodes unter Hinweis auf *LLVM* erlaubt.

*LLDB* sollte auf jedem *Mac OS X* System mit *XCode* automatisch installiert sein und lässt sich im Terminal über das Kommando

```
1 lldb
```

aufrufen. Eine Gegenüberstellung von GDB-Kommandos zu *LLDB* steht auf der *LLDB*-Webseite<sup>9</sup> zur Verfügung.

Kompiliert man eine Applikation in *XCode*, wird diese in einem emulierten iPhone gestartet und direkt ein Fenster *LLDB* hergestellt. Die ausgeführte Applikation ist in *LLDB* automatisch ausgewählt.

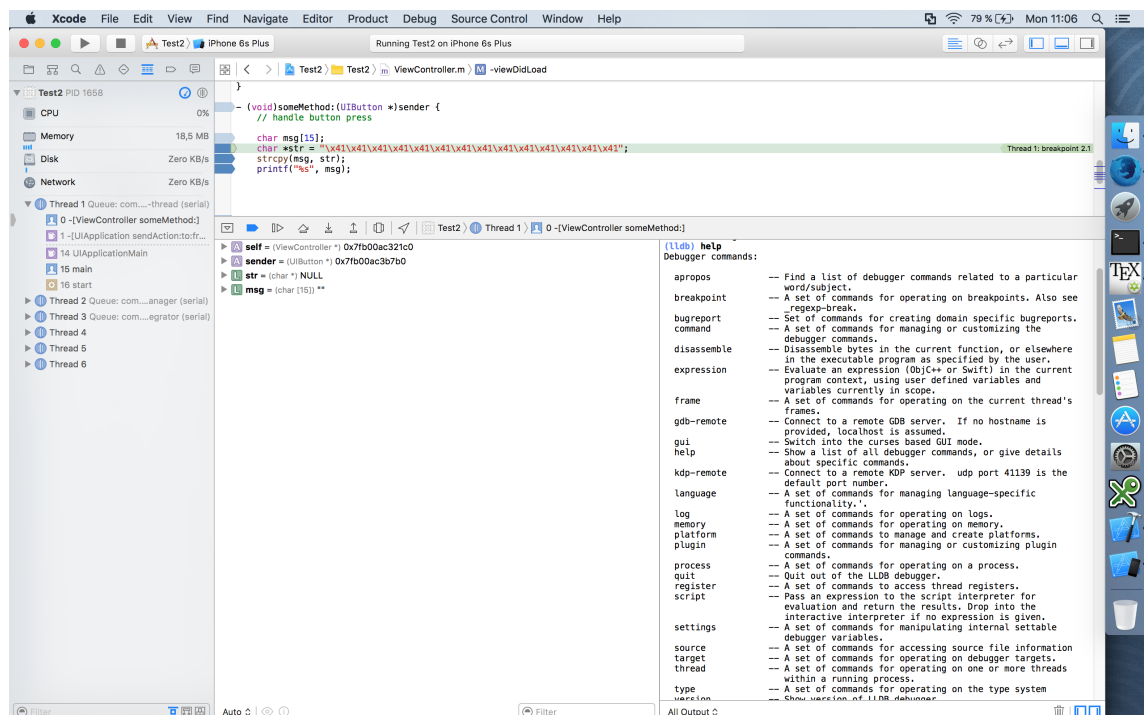


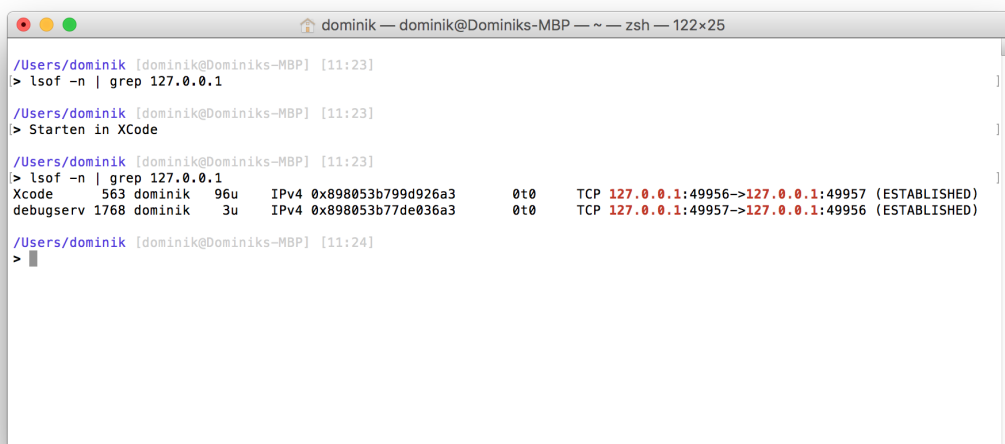
Abbildung 4.1.: LLDB in XCode

<sup>8</sup><https://opensource.org/licenses/UoI-NCSA.php>

<sup>9</sup><http://lldb.llvm.org/lldb-gdb.html>

Ein Ziel dieser Arbeit ist jedoch das Automatisieren von Analysen, weshalb das Ausführen der grafischen Oberfläche nicht optimal ist.

Leider ist nicht erkennbar, wie *LLDB* und das emulierte iPhone eine Verbindung herstellen. Eine Auflistung der offenen Sockets auf dem System legt jedoch nahe, dass auf dem iPhone das Programm *debugserver* gestartet wird, welches Remote-Debugging mit *LLDB* erlaubt. Es bleibt herauszufinden, wie die Debugging-Session auf dem simulierten iPhone ohne XCode hergestellt werden kann.



```
/Users/dominik [dominik@Dominiks-MBP] [11:23]
> lsof -n | grep 127.0.0.1

/Users/dominik [dominik@Dominiks-MBP] [11:23]
> Starten in XCode

/Users/dominik [dominik@Dominiks-MBP] [11:23]
> lsof -n | grep 127.0.0.1
Xcode      563  dominik    96u   IPv4  0x898053b799d926a3    0t0  TCP 127.0.0.1:49956->127.0.0.1:49957 (ESTABLISHED)
debugserv 1768  dominik     3u   IPv4  0x898053b77de036a3    0t0  TCP 127.0.0.1:49957->127.0.0.1:49956 (ESTABLISHED)

/Users/dominik [dominik@Dominiks-MBP] [11:24]
>
```

Abbildung 4.2.: Vergleich der offenen Pipes vor und nach der Ausführung der Applikation in XCode


Nach einem Artikel von Apple<sup>10</sup>, ist es möglich, mit LLDB eine App auch als „Standalone Debugger“, also ohne XCode, zu verwenden. Dies ist in Abbildung 4.3 aufgezeigt.

Um zu verifizieren, dass die App auf einem simulierten iPhone ausgeführt wird, können entweder die geöffneten Prozesse (siehe Abbildung ??) oder die geladenen Bibliotheken der Programme (siehe Abbildung 4.14) verglichen werden.

Bei Methoden zeigen, dass die App auf dem simulierten iPhone gestartet wird. Bei den Prozesse ist zu beobachten, dass vor Start von LLDB nur der Hintergrund-Service ausgeführt wurde. Nach dem Start von LLDB dagegen läuft der gesamte Simulator.

Auch der Vergleich der geladenen Bibliotheken legt nahe, dass die LLDB Standalone und XCode in der gleichen Umgebung ausgeführt werden. Die Adressen im RAM variieren aufgrund von ASLR zwar, aber es werden die selben Bibliotheken verwendet (am Pfad zu

<sup>10</sup>[https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/gdb\\_to\\_lldb\\_transition\\_guide/document/lldb-terminal-workflow-tutorial.html](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/gdb_to_lldb_transition_guide/document/lldb-terminal-workflow-tutorial.html)



```

/Users/dominik [dominik@Dominiks-MBP] [12:06]
> lldb Library/Developer/Xcode/DerivedData/Test2-dzkepdccqdrcebcldbvysnxcbrmp/Build/Products/Debug-iphonesimulator/Test2.app
(lldb) target create "Library/Developer/Xcode/DerivedData/Test2-dzkepdccqdrcebcldbvysnxcbrmp/Build/Products/Debug-iphonesimulator/Test2.app"
Current executable set to 'Library/Developer/Xcode/DerivedData/Test2-dzkepdccqdrcebcldbvysnxcbrmp/Build/Products/Debug-iphonesimulator/Test2.app' (x86_64).
(lldb) run --stop-at-entry
Process 2750 launched: '/Users/dominik/Library/Developer/Xcode/DerivedData/Test2-dzkepdccqdrcebcldbvysnxcbrmp/Build/Products/Debug-iphonesimulator/Test2.app/Test2' (x86_64)
2016-06-27 12:06:59.691 Test2[2750:72230] *** Assertion failure in Boolean BKSDisplayServicesStart(), /BuildRoot/Library/Caches/com.apple.xbs/Sources/BackBoardServicesFramework_Sim/backboarddemon-2.34/BackBoardServices/BKSDisplayServices.m:38
2016-06-27 12:06:59.696 Test2[2750:72230] *** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'backboardd isn't running, result: 268435459 isAlive: 0'
*** First throw call stack:
(
    0  CoreFoundation                0x0000000101104d85 __exceptionPreprocess + 165
    1  libobjc.A.dylib                0x0000000100b78deb objc_exception_throw + 48
    2  CoreFoundation                0x0000000101104bea +[NSError raise:format:arguments:] + 106
    3  Foundation                    0x00000001007c2e1e -[NSAssertionHandler handleFailureInFunction:file:lineNumber:description:] + 169
    4  BackBoardServices              0x00000001045aeeeb BKSDisplayServicesStart + 279
    5  UIKit                          0x00000001014bafcf UIApplicationMainPreparations + 164
    6  UIKit                          0x00000001014baeda UIApplicationMain + 124
    7  Test2                          0x0000000100691a8f main + 111
    8  libdyld.dylib                 0x00000001038d492d start + 1
    9  ???                            0x0000000000000002 0x0 + 2
)
libc++abi.dylib: terminating with uncaught exception of type NSError
Process 2750 stopped
* thread #1: tid = 0x11a26, 0x0000000103c18f06 libsystem_kernel.dylib`__pthread_kill + 10, queue = 'com.apple.main-thread', stop reason = signal SIGABRT
    frame #0: 0x0000000103c18f06 libsystem_kernel.dylib`__pthread_kill + 10
libsystem_kernel.dylib`__pthread_kill:
-> 0x103c18f06 <+10>: jae    0x103c18f10          ; <+20>
    0x103c18f08 <+12>: movq    %rax, %rdi
    0x103c18f0b <+15>: jmp     0x103c137cd          ; cerror_nocancel
    0x103c18f10 <+20>: retq
(lldb)

```

Abbildung 4.3.: LLDB als Standalone Debugger

```

[0] lldb image list
[0] A4680B5E-D086-3461-8C99-49039AFC63 0x00007ff69641000 /usr/lib/dyld
[1] 49268249-F1CD-35FC-BFFD-84B8F351B00 0x000000018365000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libSystem.dylib
[2] 570299A4-9342-3433-9388-0808B0A7491 0x000000018365000 /Users/dominik/Library/Developer/CoreData/Test2-3433-dkxpdcqczebcbvysnxcwmp/Build/Products/Debug-iphonesimulator/Test2.app/Test2
[3] F05317-3A9F-307D-9182-6E811198B76F 0x000000018362000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/Foundation.framework/Foundation
[4] E68BF8F6-2E97-3EC0-A283-ABE046C2539 0x000000018365000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/Foundation.framework/Foundation
[5] E68BF8F6-2E97-3EC0-A283-ABE046C2539 0x000000018365000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libobjc.A.dylib
[6] 2A3E2D6A-32B8-3B68-BF16-808B378DF43C 0x0000000183492000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libSystem.dylib
[7] 19E86284-8583-397E-BC5C-15F8ED50794 0x000000018349000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation
[8] 628B9919-66FF-348E-A399-E8BF04D7285F 0x000000018445000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/UIKit.framework/UIKit
[9] 7F20EDCB-DC65-3AEA-AC6A-8F2DF082446 0x00000001059F2000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/MobileCoreServices.framework/MobileCoreServices
[10] 3FAB46E8-2C9C-33CA-9E84-615AC83022F 0x0000000105952000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libExtension.dylib
[11] D19C592E-2C8A-39C1-AA02-09234095158A 0x000000010591000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libArchive.2.dylib
[12] 89720B89-7886-3880-A065-A135478B8480 0x00000001058C7000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libCurlcore.A.dylib
[13] 1E04A821-09F2-3133-8C84-9A91B789FA18 0x0000000105932000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libXML2.2.dylib
[14] C8BF6978-C316-37E9-8584-81320A16123 0x000000010547000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libCurlcore.A.dylib
[15] 6A661AFF-F096-325E-8E58-D56036ACB16 0x00000001054E000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/Cocoa.framework/Cocoa
[16] 8735AFB8-5818-3C9E-8076-43EE290EF86 0x0000000105955000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/

```

Abbildung 4.4.: Vergleich der geladenen Bibliotheken

erkennen). Dies ist in Grafik 4.14 dargestellt.

#### 4.3.1.3. Architektur

Auffällig ist, dass der Simulator nicht die CPU-Architektur eines iPhones simuliert, sondern den Code auf dem x86-Prozessor des Mac-Books ausführt. Dies hat den Nachteil, dass Apps aus dem Apple-App-Store, welche für ARM-Prozessoren kompiliert sind, nicht auf dem Simulator ausgeführt werden können. Lediglich Apps, welche für den Simulator in XCode kompiliert wurden, können im Simulator ausgeführt werden. Dadurch ist eine Analyse von Apps ohne Quellcode-Zugriff nicht möglich.

#### 4.3.1.4. Sicherheits-Aspekte

Bei Analyse von iOS-Apps wurden zwei mögliche Sicherheits-Lücken testweise in eine App implementiert. Die Lücken sind „unsichere Funktionen“, welche eine eventuelle Memory Corruption mit sich ziehen, und Verbindungen ohne TLS-Absicherung.

## Unsicher Funktionen

In Objective C für iOS-Apps sind Funktionen, welche für Memory Corruption-Schwachstellen bekannt sind, leider noch vorhanden.

So führt folgendes Code-Segment zwar zum Absturz der App, aber könnte bei einer dynamischen Eingabe des zu kopierenden Strings durchaus eine echte Schwachstelle einführen.

[illegible]

```
1 /Users/dominik [dominik@Dominiks-MacBook-Pro] [12:22]
2 > ps aux | grep Simulator
3 dominik          567      0.0   0.0   2546312    3344   ??   U
   8:51PM    0:00.21 /Applications/Xcode.app/Contents/Developer/
   Library/PrivateFrameworks/CoreSimulator.framework/Versions/A/
   XPCServices/com.apple.CoreSimulator.CoreSimulatorService.xpc/
   Contents/MacOS/com.apple.CoreSimulator.CoreSimulatorService
4 dominik          3330      0.0   0.0   2434840     776  s006  R+
   12:22PM    0:00.00 grep --color=auto --exclude-dir=.bzd --
   exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --
   exclude-dir=.svn Simulator
5
6 [Ausführung der App mit LLDB]
7
8 /Users/dominik [dominik@Dominiks-MacBook-Pro] [12:22]
9 > ps aux | grep Simulator
10 dominik          3361    82.6   0.6   2937152  100892   ??   Ss
   12:22PM    0:08.23 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/System/Library/CoreServices/SpringBoard.
   app/SpringBoard
11 dominik          3371    38.4   0.0   2525776    2896   ??   Rs
   12:22PM    0:05.15 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/usr/sbin/notifyd
12 dominik          3344     7.8   0.0   2546992    4560   ??   S
   12:22PM    0:00.97 launchd_sim /Users/dominik/Library/
   Developer/CoreSimulator/Devices/F379F302-76DE-43BA-A6A9-27
   F85C97ED6C/data/var/run/launchd_bootstrap.plist
13 dominik          3389     0.0   0.1   2600072   10244   ??   Ss
   12:22PM    0:00.05 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/usr/libexec/nanoregistrylaunchd
14 dominik          3388     0.0   0.3   2790820   44792   ??   Us
   12:22PM    0:00.24 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/System/Library/PrivateFrameworks/
   ManagedConfiguration.framework/Support/profiled
15 dominik          3387     0.0   0.1   2584432   12680   ??   Ss
   12:22PM    0:00.07 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/usr/libexec/networkd
16 dominik          3386     0.0   0.1   2614364   15684   ??   Ss
   12:22PM    0:00.11 /Applications/Xcode.app/Contents/Developer/
   Platforms/iPhoneSimulator.platform/Developer/SDKs/
   iPhoneSimulator.sdk/System/Library/Frameworks/Accounts.
   framework/accountsd
```

Listing 4.1: Gestartete Prozesse nach LLDB Aufruf



Dies ist auch in der Apple-Online-Dokumentation beschrieben<sup>11</sup>.

## Ungesicherte Verbindungen

Ein Sicherheits-Feature ab iOS 9.0 ist der Umgang von iOS mit Netzwerk-Verbindungen. So können ohne weitere Konfiguration keine Verbindungen aufgebaut werden, welche nicht dem RFC-Standard 2818<sup>12</sup> folgen. So wird folgender Aufruf einer HTTP-Seite

```
1 NSURL *url = [NSURL URLWithString:@"http://api.ipify.org"];
2   NSData *data = [NSData dataWithContentsOfURL:url];
3   NSString *ret = [[NSString alloc] initWithData:data encoding
4     :NSUTF8StringEncoding];
5   NSLog(@"ret=%@", ret);
```

blockiert und folgende Fehlermeldung generiert:

```
1 2016-06-28 08:42:56.518 Test2[4789:140270] App Transport
   Security has blocked a cleartext HTTP (http://) resource load
   since it is insecure. Temporary exceptions can be configured
   via your app's Info.plist file.
```

Sollten unsicher Verbindungen benötigt werden, muss ein entsprechender Eintrag in der „Info.plist“ angelegt werden. Dieser Eintrag muss sehr genau auf die App angepasst werden, da zum Beispiel die Domains festgelegt werden müssen. Ein Eintrag muss laut Apple[2] folgenden Inhalt haben:

```
1 NSAppTransportSecurity : Dictionary {
2   NSAllowsArbitraryLoads : Boolean
3   NSAllowsArbitraryLoadsForMedia : Boolean
4   NSAllowsArbitraryLoadsInWebContent : Boolean
5   NSAllowsLocalNetworking : Boolean
6   NSExceptionDomains : Dictionary {
7     <domain-name-string> : Dictionary {
8       NSIncludesSubdomains : Boolean
9       NSExceptionAllowsInsecureHTTPLoads : Boolean
10      NSExceptionMinimumTLSVersion : String
11      NSExceptionRequiresForwardSecrecy : Boolean //
          Default value is YES
12      NSRequiresCertificateTransparency : Boolean
13    }
14  }
15 }
```

Eine Überprüfung auf solche Ausnahmen kann dem Abschnitt 4.6.3.3 entnommen werden.

### 4.3.2. Windows-Phone

<sup>11</sup><https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html>

<sup>12</sup><https://tools.ietf.org/html/rfc2818>

#### 4.3.2.1. Emulation vs. Hardware

VS

#### 4.3.2.2. Debugging

VS

### 4.3.3. Android

Android ist ein Ursprünglich 2003 von der Android, Inc. entwickeltes mobiles Betriebssystem. 2005 wurde es durch Google übernommen und wird seit dem weiterentwickelt. 2015 liegt es bei einem Marktanteil von TODO %. Aufgrund der Quelloffenheit des Systems wird von vielen Herstellern auf verschiedensten Plattformen genutzt. Jedoch bringt die weitführende Fragmentierung des Betriebssystems auch Nachteile mit sich. So sind in 2015 nur TODO % der Android-Devices auf einer aktuellen Version.[7]

#### 4.3.3.1. Android-Studio und SDK

Das Android-Studio ist eine umfassende IDE. Sie ermöglicht unter anderem das schnelle Entwickeln und Testen von Apps, sowie die Emulation von beliebigen Android-Versionen. Außerdem ist Android Studio kostenlos, Open-Source und für Linux, Mac und Windows erhältlich. Die aktuelle Version kann unter <http://developer.android.com/sdk/index.html> heruntergeladen werden. Die Installation unter Linux ist vergleichsweise einfach, da nur ein Archiv über das Kommando

```
1 unzip android-studio-ide-143.2739321-linux.zip
```

entpackt werden muss. Für alle anderen Betriebssysteme werden entsprechende Installationsroutinen zur Verfügung gestellt. Anschließend kann die IDE über die Datei „bin/studio.sh“ gestartet werden. Neben dem Android-Studio gibt es noch das Android SDK, welches über die gleiche URL heruntergeladen werden kann. Es enthält wichtige Kommandozeilen-Tools wie *adb*, *fastboot* oder *logcat*, auf welche im weiteren Verlauf noch detailliert eingegangen wird.

#### 4.3.3.2. Compatibility Testing Suite

[7] Seite 18

#### 4.3.3.3. Emulation vs. Hardware

Android SDK; AVD Im Gegensatz zum iOS-Simulator hat AVD die Möglichkeit CPU wie auch GPU eines Handys zu emulieren. Dabei besteht die Auswahl zwischen verschiedenen Architekturen, wie Intel x86 oder ARM. Zudem gibt es viele andere Möglichkeiten zur Konfiguration der einzelnen Maschinen, wie in Grafik 4.5 zu sehen ist. Daher hat Android den Vorteil, dass gerade tiefgreifende Analysen aufgrund der Emulation der Architektur näher an der echten Hardware sind als bei iOS.

TODO: Android-Hackers Handbook



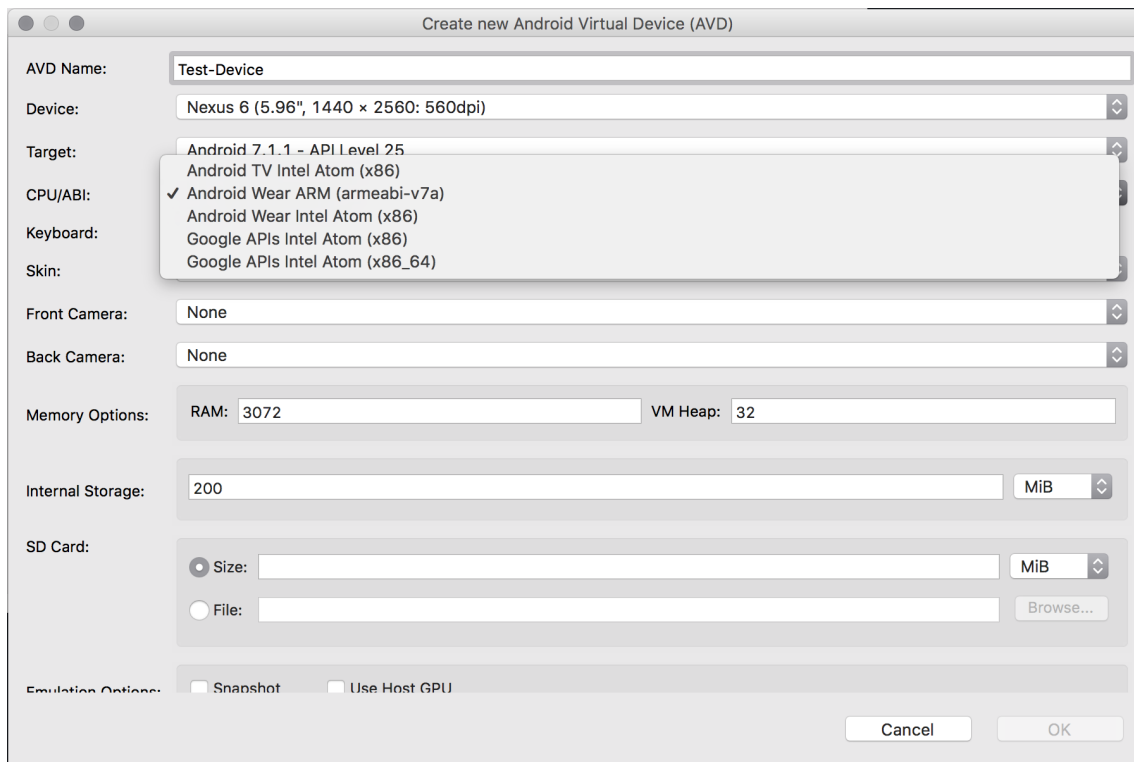


Abbildung 4.5.: Der Konfigurations-Bildschirm des AVD-Managers

#### 4.3.3.4. Debugging

Android Debug Bridge[1]

#### 4.3.3.5. Logcat

Android Debug Bridge[1]

### 4.4. Abgleich der Anforderungen mit MobSF

Im Folgenden Abschnitt werden die Anforderungen aus 4.1 mit den bestehenden Eigenschaften von MobSF abgeglichen.

**Einfache Einrichtung** MobSF benötigt keine echte Installation, sondern kann einfach aus dem Github-Repository heruntergeladen werden. Die anschließenden Konfigurationsschritte unterscheiden sich leicht nach Betriebssystem, besteht jedoch grundsätzlich immer aus der Installation von Python sowie die Installation der Abhängigkeiten über das Python-Tool *pip*. Abschließend sollten die Einstellungen an die Umgebung angepasst werden. Damit ist die Installation nicht trivial, aber gut dokumentiert und durch eine Person mit Fachkenntnis in kurzer Zeit durchzuführen.

**Einfache Handhabung** Die Handhabung von MobSF wird über eine HTML-Oberfläche abgebildet. Samples können per Drag-n-Drop zur Analyse eingereicht werden. Die

Realisierung als Web-Anwendung hat den Vorteil, dass das Tool zentral installiert und von verschiedenen Orten über das Netzwerk genutzt werden kann.

**Verständliche Ergebnisse** Die Ergebnisse einer Analyse werden ebenfalls in der HTML-Anwendung dargestellt. Dabei über Farb-Codes zumindest einfach zu erkennen, ob Probleme mit der App bestehen. Eine genaue Analyse der Ergebnisse und weiterführende Untersuchungen sollten jedoch durch Experten durchgeführt werden.

**Unterstützung für Android/iOS/Windows Phone** Vor der Weiterentwicklung von MobSEF unterstütze das Framework die statische und dynamische Analyse von Android-Apps sowie die statische Analyse von iOS-Apps.

**Niedrige False-Positive-Rate** MobSEF beschreibt nicht direkt Schwachstellen, sondern Indikatoren für zum Beispiel die Verwendung von verwundbaren Funktionen oder angeforderte Rechte. Daher sind die gegebenen Warnungen stets korrekt, jedoch wird womöglich auch in machen Bereichen nicht alles entdeckt.

**Reproduzierbarkeit** MobSEF benutzt zumeist Tools, welche in MobSEF selbst enthalten sind. Daher sollte die selbe Version von MobSEF auch auf verschiedenen Rechnern gleiche Ergebnisse liefern.

Somit sind alle Anforderungen bis auf die Unterstützung von Windows-Phone-Apps ausreichend erfüllt.

## 4.5. Laboraufbau

Zur Weiterentwicklung des *Mobile Security Frameworks* wurde unterschiedliche Hardware mit verschiedenen Werkzeugen genutzt.

Als Hardware wurde durch die Allianz Deutschland AG ein Apple Mac-Book Pro mit ausreichenden Ressourcen gestellt (i7, 16GB RAM). Als Betriebssystem wurde anfangs Mac OS X Maverick und später Sierra verwendet.

Um die Entwicklung für Windows-spezifische Anwendungen zu realisieren, wurde eine virtuelle Maschine mit *Windows 10* über *VMWare Fusion* verwendet.

Als Software wurde Python in Version 2.7 oder 3.6 sowie die jeweiligen Abhängigkeiten in Form von Modulen genutzt.

Als Entwicklungsumgebung wurde *Atom* oder *VIM* mit Python 2/3 genutzt.

## 4.6. Weiterentwicklung MobSEF

Ein Kernelement dieser Arbeit ist die Weiterentwicklung des Mobile Security Frameworks *MobSEF*. Die Funktionen des Frameworks sind bereits unter TODO aufgezeigt. Im Folgenden sind die Änderungen dargestellt, welche an dem Framework vorgenommen und veröffentlicht wurden.

```
Mobile-Security-Framework-MobSF/  
├── .git/  
├── APITester/  
├── downloads/  
├── DynamicAnalyzer/  
├── LICENSES/  
├── logs/  
├── MalwareAnalyzer/  
├── MobSF/  
├── static/  
├── StaticAnalyzer/  
├── templates/  
└── uploads/
```

Abbildung 4.6.: Struktur MobSF auf der ersten Ebene

### 4.6.1. Allgemeine Verbesserungen

Neben Verbesserungen, welche einem genauen Bereich (*Windows-Phone*, *iOS*, *Android*) zuzuordnen sind, gibt es auch einige allgemeine Erweiterungen am *Mobile Security Framework*. Diese sind im Folgenden dargestellt.

#### 4.6.1.1. Struktur

Die Struktur von MobSF war bisher relativ flach. Auf der ersten Ebene findet man die Übergeordneten Module wie den *ApiTester*, *StaticAnalyzer*, *DynamicAnalyzer* sowie den *statischen Content*, *Templates* und Kern-Module des *MobSF*. Dies ist in der Abbildung ?? verdeutlicht. Jedoch hatte die Struktur in den Modulen oft keine saubere Trennung der Aufgaben. So waren im *StaticAnalyzer*-Modul sowohl *iOS* wie auch *Android*-Analyse in der *views.py* zusammengefasst. Um hier eine klarere Trennung zu schaffen, wurde die *views.py* aufgegliedert in drei Dateien:

**shared\_func.py:** Die *shared\_func.py* enthält alle Funktionen, welche sowohl für *iOS* als auch *Android* gebraucht werden. Beispiele sind die Erstellung von Hashes, das Generieren von PDFs oder das Entpacken von Archiven.

**ios.py:** Die Datei *ios.py* enthält alle *iOS* spezifischen Funktionen zur statischen Analyse.

**android.py:** Die Datei *android.py* enthält alle *Android* spezifischen Funktionen zur statischen Analyse.

**windows.py:** Die Datei *windows.py* enthält alle *Windows-Phone* spezifischen Funktionen zur statischen Analyse. Sie wurde nachträglich hinzugefügt (siehe 4.6.2), weshalb die Funktionen in der altern Struktur nicht auftauchen.

Dies ist im Detail in der Abbildung 4.7 dargestellt.

Im weiteren Verlauf der Weiterentwicklung und mit der Einführung von Code-Standards (siehe 4.6.1.3) wurde die Struktur weiter verfeinert.

TODO

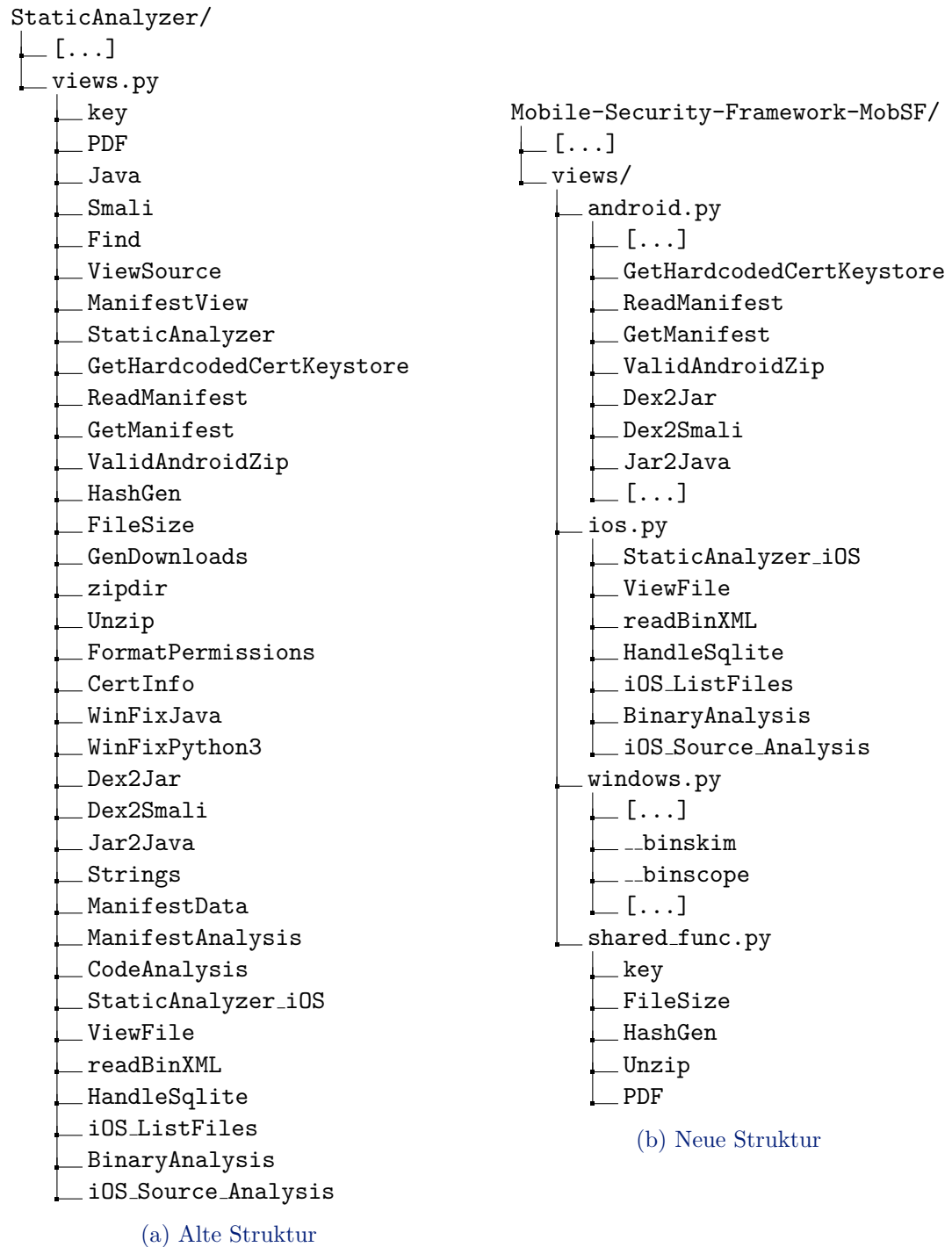


Abbildung 4.7.: Vergleich der Struktur von *StaticAnalyzer*

#### 4.6.1.2. Namedtuple vs. DICT

TODO

#### 4.6.1.3. Code-Standards

Bei der Neu- oder Reimplementierung wurde auf die Verwendung von offiziellen Style-Guides geachtet. Insbesondere wurde der *PEP 8* Standard<sup>13</sup> für Python verwendet, welcher die Lesbarkeit und Wartbarkeit von Python-Code verbessern soll. Um die Einhaltung des Standards zu gewährleisten, wurde das Tool *Pylint* verwendet. Dieses prüft einen gegebenen Quellcode gegen den Code-Standard *PEP 8* und kreiert entsprechende Warnungen für Abweichungen. Ursprünglich musste *Pylint* auf der Konsole extra ausgeführt werden, jedoch können moderne Entwicklungsumgebungen wie *Atom* *Pylint* direkt einbinden und nutzen. Dies hat den Vorteil, dass bereits während dem Programmieren Verstöße gegen Standard oder Fehler wie zum Beispiel falsche Variablennamen entdeckt werden.

Für *MobSF* wurde nicht von Anfang an mit Code-Standards entwickelt, weshalb zum Beispiel die Datei *android.py* über 2000 Code-Zeilen mit über 1600 *Pylint*-Fehler aufwies. Durch aufwendige Refaktorisierungs- und Umstrukturierungsarbeiten konnte die Anzahl der Abweichungen massiv reduziert werden.

Daraus ergibt sich erhöhte Wartbarkeit sowie eine einfachere Entwicklung aufgrund weniger Merge-Konflikte. Dies ist möglich, da die jeweiligen Methoden je nach Funktionalität in entsprechende Module ausgelagert wurden und somit nur die eine, für die Funktion benötigte Datei, verändert und wieder in das Haupt-Projekt eingegliedert werden muss. Zuvor musste bei paralleler Entwicklung am Projekt auch nur eine kleine Änderung in die übergreifende Datei mit Änderungen einer parallel arbeitenden Partei zusammengeführt werden, was oftmals viel Arbeit bedeutet.

TODO pep8 vs flake8 vs xxx..

#### 4.6.1.4. strings

Zuerst wurde das *MobSF* um die Fähigkeit erweitert, eine iOS-Applikation mit dem *strings*-Programm zu untersuchen. *strings* durchsucht, insofern keine zusätzlichen Parameter übergeben werden, eine binäre Datei nach 4 aufeinander folgende ASCII-Elemente.

Dies hilft oft bei einer ersten Einschätzung der Anwendung, da oft eine grundlegende Funktionsweise und der Zweck der Software abgeleitet werden kann. Ebenso können eventuell unbeabsichtigt im Programm vergessene Strings in einer App aufgedeckt werden. Auch sind gelegentlich Funktionsnamen oder Kernel-Calls als String in einem Binary enthalten, was unter *TODO* zum Entdecken von verwundbaren Funktionen genutzt wird.

Sowohl *Mac-OSX* wie auch *Linux* haben ein integriertes *string*-Kommando, welches jedoch *Windows* fehlt. Um die Multi-Platform-Fähigkeit weiterhin zu gewährleisten, wurde die Funktionsweise in Python-Code abgebildet. Als Vorlage wurde ein bestehender Code

---

<sup>13</sup><https://www.python.org/dev/peps/pep-0008/>

von *Stackoverflow* <sup>14</sup> genutzt. Dieser lieferte jedoch eine wesentlich höhere Anzahl von Ergebnissen, da bestimmte Whitespace-Character ebenfalls beachtet wurden:

```
1 > wc -l strings_test_*
2          85149 strings_test_orig
3          541393 strings_test_pyth
4          626542 total
```

Da viele der zusätzlich aufgedeckten Strings jedoch nicht bei der Analyse geholfen, sondern eher das Auffinden relevanter Strings erschwert haben, wurde der Originalcode wie folgt angepasst:

```
1 <<<<<< Vor Anpassung
2 import string
3 =====
4 >>>>>> Nach Anpassung
5
6 def strings(filename, min=4):
7     """Print out all connected series of readable chars longer
8         than min."""
9     with open(filename, "rb") as f:
10         result = ""
11         <<<<<< Vor Anpassung
12         for c in f.read():
13             <<<<<< Vor Anpassung
14             if c in string.printable:
15                 =====
16                 if c in (
17                     '0123456789',
18                     'abcdefghijklmnopqrstuvwxyz',
19                     'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
20                     '!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ ',
21                 ):
22                     >>>>>> Nach Anpassung
23                     result += c
24                     continue
25                     if len(result) >= min and result[0].isalnum():
26                         yield '"' + result + '"'
27                     result = ""
```

Durch die Reduzierung der ausschlaggebenden Zeichen konnten die Ergebnisse optimiert werden, sodass eine effiziente Suche über Strings wieder möglich ist.

#### 4.6.1.5. PDF-Generation

Um eine effiziente Weitergabe der Ergebnisse zu ermöglichen, besitzt das *Mobile Security Framework* eine PDF-Export-Funktion. Die Implementierung ist dabei relativ einfach. Es wird eine neue HTML-Sicht geschaffen, welche anschließend über das Python-Modul *xhtml2pdf.pisa* als PDF geöffnet wird.

Eine solche Sicht wurde jeweils für alle Erweiterungen implementiert, welche in dieser Arbeit vorgenommen wurden.

<sup>14</sup><http://stackoverflow.com/a/17197027>

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     print("Execute command!")
7     return 'Hello World!'
8
9 @app.route('/second_command/')
10 def not_hello_world():
11     print("Execute second command!")
12     return 'Goodbye World!'
13
14 if __name__ == '__main__':
15     app.run()
```

Abbildung 4.8.: `rpc_client.py`

#### 4.6.1.6. RPC-Service

Um eine Kommunikation zwischen verschiedenen Virtuellen Maschinen zu ermöglichen, wurde ein minimaler *RPC-Server* in Python 3.5 entwickelt. Hierzu wurden verschiedene Ansätze untersucht. Getestet wurden hierzu die Python-Module *Flask*, *Requests*, *xmlrpc* sowie *RSA* zum Hinzufügen einer Authentisierung.

##### Flask

Flask ist ein schneller, minimaler Webserver. Mit nur sehr wenig Code es möglich, eine Schnittstelle bereit zu stellen. Ein Code mit zwei akzeptierenden Funktionen, basierend auf der Schnellstart-Anleitung<sup>15</sup>, ist in Abbildung ?? dargestellt.

Es wird eine minimale Anwendung erstellt, welche auf dem Pfad `/` lokal das *print*-Statement ausführt und den Text 'Hello World!' zurück gibt. Wird die Anwendung unter dem Pfad `/second_command/` angesprochen, wird ein anderes *print*-Statement ausgeführt und ein anderer Wert zurück gegeben. Auf diese Weise können schnell API-Funktionen auf verschiedene Pfade gelegt und angesprochen werden.

##### Requests

*Requests* ist ein Python-Modul, welche einfache Anfragen (sogenannte *Requests*) über HTTP(S) ermöglicht. So ist es über ein kurzes Code-Snippet, dargestellt in Abbildung ??, möglich die unter ?? aufgezeigt Schnittstelle anzusprechen.

Wird zuerst der Code ?? und anschließend der Code ?? ausgeführt, wird auf Server-Seite folgende Ausgabe erzeugt:

```
1 $ python3 rpc_server.py
2 Hello World!
3 Goodbye World!
```

<sup>15</sup><http://flask.pocoo.org/docs/0.10/quickstart/>

```
1 import requests
2
3 r = requests.get('http://localhost:5000')
4 print(r.text)
5
6 r = requests.get('http://localhost:5000/second_command/')
7 print(r.text)
```

Abbildung 4.9.: `rpc_server.py`

Auf der Client-Seite erfolgt folgende Ausgabe:

```
1 $ python3 rpc_client.py
2 * Running on http://127.0.0.1:5000/
3 Execute command!
4 127.0.0.1 - - [18/May/2016 19:10:56] "GET / HTTP/1.1" 200 -
5 Execute second command!
6 127.0.0.1 - - [18/May/2016 19:10:56] "GET /second_command/ HTTP
  /1.1" 200 -
```

Aufgrund dieser Basis wurde die erste Version des RPC-Servers implementiert.

### xmlrpc mit rsa

Bei der Eingliederung der Verbindung zwischen der virtuellen Maschine und dem Host wurde jedoch die Anforderung nach einer Möglichkeit zur Authentifizierung gefordert.

Zur Implementierung dieser Anforderung wurde das Python-Modul *rsa* verwendet, welches sowohl für Python 2 wie auch Python 3 existiert. Die Funktionsweise ist dabei wie folgt.

TODO Zustandsdiagramm

Durch das Signieren einer bei jedem Funktionsaufruf neu generierten Challenge ist sichergestellt, dass nur der echte Host die Funktion aufruft und ist ebenfalls gegen Replay-Attacken geschützt TODO zu Replay.

Jedoch gab es bei der Implementierung über Flask sowie mit den verschiedenen Python-Versionen zwischen Server und Client verschiedene Probleme. So gibt es in Python 2 einen dedizierten Variablen-Typ namens *string*, wohingegen Python 3 Strings in codierten Byte-Objekten speichert. Eine Konvertierung ist an sich möglich und wurde später auch umgesetzt. Jedoch legt *Flask* eine weitere Ebene des Encodings über den übertragenen Inhalt, weshalb es leider nicht möglich war, die kryptographische Signatur fehlerfrei zu übertragen. Aus diesem Grund wurde nach weiteren und eventuell besser geeigneten Alternativen gesucht.

Nach einer kurzen Suche bot sich das Modul *xmlrpclib* (Python 2)/*xmlrpc* (Python 3) an. Es ermöglicht die Kommunikation über das standardisierte TODO XML-RPC-Protokoll und ist somit unabhängig von der Python-Version. Zudem können Daten transparent zwischen Server und Client übergeben werden.



Es folgt ein kurzes Beispiel, in welcher ein Client eine *hello\_world*-Funktion am Server aufruft.

Server-Code:

```
1 from xmlrpc.server import SimpleXMLRPCServer
2
3 def hello_world(name):
4     """Return an Hello-World for a name"""
5     return "Hello World {}".format(name)
6
7 if __name__ == '__main__':
8     # Open the Server on port 8000
9     server = SimpleXMLRPCServer(("0.0.0.0", 8000))
10    server.register_function(hello_world, "hello_world")
```

Client-Code (*TARGET\_IP* sollte mit der richtigen IP-Adresse ersetzt werden):

```
1 import xmlrpclib
2
3 proxy = xmlrpclib.ServerProxy(
4     "http://{}:{ {}".format(
5         TARGET_IP, 8000
6     )
7 )
8 print proxy.hello_world("John Doe")
```

Output des Clients:

```
1 Hello World John Doe!
```

Durch den minimalen eingriff von *xmlrpc* in die Kommunikation, konnte die kryptographische Signatur als *base64* codiertes Datum übergeben werden. Im Folgenden sind kurz die Kern-Funktionen des RSA-Checks aufgezeigt.

Client-Code:

```
1 def _get_token():
2     """Get the authentication token for windows vm xmlrpc client
3     ."""
4     challenge = proxy.get_challenge()
5     priv_key = rsa.PrivateKey.load_pkcs1(
6         open(settings.WINDOWS_VM_SECRET).read()
7     )
8     signature = rsa.sign(challenge, priv_key, 'SHA-512')
9     sig_b64 = base64.b64encode(signature)
10    return sig_b64
11 print proxy.test_challenge(_get_token())
```

Server-Code:

```
1 def _check_challenge(signature):
2     signature = base64.b64decode(signature)
3     try:
4         rsa.verify(challenge.encode('utf-8'), signature, pub_key
5                     )
6         print("[*] Challenge successfully verified.")
7         _revoke_challenge()
8     except rsa.pkcs1.VerificationError:
9         print("[!] Received wrong signature for challenge.")
10        raise Exception("Access Denied.")
11    except (TypeError, AttributeError):
12        print("[!] Challenge already unset.")
13        raise Exception("Access Denied.")
14
15 def get_challenge():
16     """Return an ascii challenge to validate authentication in
17     _check_challenge."""
18     global challenge
19     # Not using os.urandom for Python 2/3 transfer errors
20     challenge = ''.join(
21         random.SystemRandom().choice(string.ascii_uppercase +
22                                     string.digits) for _ in range(256)
23     )
24     return "{}".format(challenge)
25
26 def test_challenge(signature):
27     """Test function to check if rsa is working."""
28     _check_challenge(signature)
29     print("Check complete")
30     return "OK!"
```

Durch diese Art der Implementierung ist eine sichere, zuverlässige, effiziente und leicht erweiterbare Kommunikation zwischen Host und virtueller Maschine möglich.

#### 4.6.2. Windows-Apps

Auch wenn die Zukunft der Handy-Sparte von Windows derzeit oftmals in Diskussion steht, geht der Trend zur Entwicklung auf gerade sein Windows 8 und der damit eingeführten TODO Window Unified Platform hin zur Entwicklung von Apps. Der Vorteil, dass diese Apps sowohl auf Windows auf einem PC wie auch auf Handy lauffähig sind, ist schwer bestreiten, macht jedoch eine Umfassende Prüfung der Sicherheit umso wichtiger.

Bisher hat das *Mobile Security Framework* noch keine Möglichkeit zur Prüfung von Windows-Apps bereitgestellt. Im Folgenden sind die im Rahmen dieser Arbeit implementierten Features beschrieben.

#### 4.6.2.1. Windows Phone Formats

Um eine App analysieren zu können, muss zu aller erst das File-Format betrachtet und verarbeitet werden. Leider sind im Windows-Umfeld diverse Formate gängig, von welchen im Folgenden einige in Hinblick auf Aufbau und Schutz analysiert werden.

##### xap

XAP ist ein Format für Windows-Phone-Apps ab *Windows Phone 7* und enthält oft Silverlight-Applikationen.

Ursprünglich war ein XAP-File einfach ein ein ZIP-Archiv, mit allen Dateien der App. Aus einem solchen XAP-File konnte der Inhalt über folgende Schritte einfach gewonnen werden:

1. XAP-File herunterladen
2. evtl. Dateiendung von „.xap“ auf „.zip“ ändern
3. mit einem gängigen Archiv-Programm (z.B. 7-Zip) entpacken

Anschließend liegen allen Dateien der App im Extraktions-Ordner.

Leider sind XAP-Files häufig durch sogenanntes *DRM* (*Digital Rights Management*) geschützt. So lässt sich bei vielen aktuellen XAP-Files folgender Header feststellen:

```
1 <WRMHEADER xmlns="http://schemas.microsoft.com/DRM/2007/03/
  PlayReadyHeader" version="4.0.0.0">
2     <DATA>
3         <PROTECTINFO>
4             <KEYLEN>16</KEYLEN>
5             <ALGID>AESCTR</ALGID>
6         </PROTECTINFO>
7         <KID>5zhQkM1z5kq6HCCYD9nceQ==</KID>
8         <LA_URL>http://microsoft.com/</LA_URL>
9         <CUSTOMATTRIBUTES xmlns="">
10             <S>rtXfkbbz4yuPNGrzjQc9yA==</S>
11             <KGV>0</KGV>
12         </CUSTOMATTRIBUTES>
13         <CHECKSUM>TpkeZrwUjIY=</CHECKSUM>
14     </DATA>
15 </WRMHEADER>
```

<http://forum.xda-developers.com/showpost.php?p=34246750&postcount=3>

The encrypted XAPs use AESCTR, PlayReady. First 32 bytes are a header, last two dwords are a length, the first half (BigEndian) is the 'payload' (actual app(+more?)) and the second half is always -8 bytes. Then comes the 'mal'-formed XML containign PlayReady definitions etc, including base64 encoded key ids etc. I'd like to see someone on custom rom P7 wireshark their network while using a MITM attack (SSL obviously) using an SSL cert accepted as root ca in the phone to see if we can dump any plaintext DRM keys per app, maybe a solid method can be made. For now, not sure. I've been working on it. Key lengths are 16 bytes. I can brute 12 bytes in one month but 16 would take a lot longer. You could also install a custom rom with wdmc on the phone and copy the entire folders off, restructured

a little though, unpacked. Hope this helps someone else. All based on PlayReady DRM bs. Smooth Streaming in IIS can support PR too for instance (WRMHEADER aka PlayReady headers): <http://forum.xda-developers.com/showthread.php?t=2046702>

VM + extraction-tool?

Mime-Type: application/x-silverlight-app

### appx

APPX ersetzt ab Windows 8.1 das XAP-Format.

Extraktion kann über das in Python integrierte Modul *zipfile* erfolgen.

```
1 import zipfile
2 files=[]
3 with zipfile.ZipFile(APP_PATH, "r") as z:
4     z.extractall(EXT_PATH)
5     files=z.namelist()
6 return files
```

Mime-Type:

```
1 APPX_MIME = [
2     'application/octet-stream',
3     'application/vnd.ms-appx',
4     'application/x-zip-compressed'
5 ]
```

### appxbundle

```
1 ['AugmentendApplicationAppx.appx', 'DumbSmash.WindowsPhone_1
   .1.0.15_language-it.appx', 'DumbSmash.WindowsPhone_1.1.0.15
   _scale-100.appx', 'DumbSmash.WindowsPhone_1.1.0.15_scale-120.
   appx', 'DumbSmash.WindowsPhone_1.1.0.15_scale-140.appx', '
   AppxMetadata/AppxBundleManifest.xml', 'AppxBlockMap.xml', '[
   Content_Types].xml', 'AppxSignature.p7x']
```

MIME-Type file -mime-type -b filename

#### 4.6.2.2. Virtuelle Maschine zur Analyse von Windows-Apps

Die virtuelle Maschine zur Analyse von Windows Applikationen setzt sich aus TODO Komponenten zusammen. Für die Software-Komponenten wird ein Installer im Rahmen dieser Arbeit erstellt und zur Verfügung gestellt.

Als erstes muss ein Betriebssystem für die Analyse-VM gewählt werden. Normalerweise wäre Windows 7 aufgrund der geringeren Ressourcenauslastung die erste Wahl für Windows-Analyse-Maschinen. Da jedoch der Windows-Phone-Simulator erst ab Windows 8.1 64-Bit unterstützt wird, sollte in Hinsicht auf die dynamische Analyse davon abgesehen und Windows 8.1 64-Bit oder höher verwendet werden. Hier wurde Windows 10 64-Bit

verwendet.

Eine weitere Voraussetzung ist Python 3. Getestet wurde in dieser Arbeit mit Python 3.5.2 32-Bit.

Für die restliche Installation wurde ein Installer angefertigt, der die restlichen notwendigen Programme herunterlädt und die Installationen anstößt. Das File ist im Anhang unter TODO zu finden. Genutzt wurden die Bibliotheken *urllib*, *configparser* und *zipfile* sowie *os* zur Ausführung.

Heruntergeladen und Installiert werden folgende Komponenten:

- binskim
- binscope

Diese werden im Abschnitt 4.6.2.5 vorgestellt.

Die Ordnerstruktur wurde dabei anfangs wie folgt aufgebaut:

```
C:/
├── [...]
└── MobSF/
    ├── Config/
    │   └── config.txt
    ├── Download/
    └── Tools/
```

Um die Kompatibilität mit verschiedenen Systemkonfigurationen zu erhöhen, wurde später statt C die Ordner des Benutzers für die Speicherung der Programme verwendet.

Die *config.txt* enthält Inhalte, welche zentral Abgelegt und für verschiedene Skripte eine wichtige Rolle spielen. Ein Beispiel wäre der Pfad zum Verzeichnis, in welchem die Tools gespeichert werden.

Der Downloads-Ordner hält die durch das Setup-Skript heruntergeladenen Binaries, der Tools-Ordner die installierten Tools.

#### 4.6.2.3. setup.py

Die Datei *setup.py* wird über zwei verschiedene Wege gerufen, je nach dem ob MobSF vollständig auf Windows installiert wird oder nur die statische Analyse auf dem Windows-System ausgeführt werden soll. Je nach Aufrufweise werden verschiedene Arbeitsschritte ausgeführt und auch verschiedene Python-Versionen verwendet, weshalb manche Funktionen sowohl Python 2 wie 3 kompatibel gestaltet sind.

#### MobSF auf Windows

Wird MobSD vollständig auf Windows installiert, wird das Setup-Script einmalig aus der *settings.py* aufgerufen. Dabei wird Python 2 verwendet. Anfangs wird das mit dem

Download mitgelieferte Config-File in den richtigen Ordner des User-Kontext kopiert. Anschließend wird das Config-File über das ConfigParser-Modul geladen. Daraufhin werden alle notwendigen Ordner berechnet, in die Config geschrieben und angelegt. Damit sind die Vorbereitungen bezüglich der Config und der Ordner abgeschlossen.

Anschließend wird *nuget* heruntergeladen. *nuget* ist ein Paket-Manager für Windows, über welchen später *BinSkim* installiert wird. Der Download ist in Python relativ einfach zu implementieren, wie in folgendem Code-Fragment dargestellt ist.

```
1 # Open File
2 nuget_file_local = open(
3     os.path.join(mobsf_subdir_tools, nuget_file_path),
4     "wb"
5 )
6
7 # Downloading File
8 print("[*] Downloading nuget..")
9 nuget_file = urlopen(nuget_url)
10
11 # Save content
12 print("[*] Saving to File {}".format(nuget_file_path))
13
14 # Write content to file
15 nuget_file_local.write(bytes(nuget_file.read()))
16
17 # Aaaand close
18 nuget_file_local.close()
```

Anschließend wird *BinSkim* über *nuget* installiert. Dazu wird *nuget* mit verschiedenen Parametern aufgerufen.

```
1 # Execute nuget to get binkim
2 output = subprocess.check_output(
3     [
4         nuget,
5         "install", binskim_nuget, '-Pre',
6         '-o', mobsf_subdir_tools
7     ]
8 )
```

Anschließend wird aus den installierten Dateien die Binaries für *BinSkim x86* sowie *x64* gesucht und in der Config-Datei gespeichert.

Als letztes Tool wird *BinScope* installiert. Dies ist leider nicht über *nuget* möglich. Daher muss eine MSI-Datei heruntergeladen und anschließend installiert werden. Dabei muss der Installationspfad entsprechend gesetzt werden. Der Quellcode kann dem Appendix unter TODO entnommen werden.

Um sicherzustellen, dass das Setup-Script nicht mehrmals ausgeführt wird, wird zuletzt ein Lock-File platziert. Bei erneutem Start der Anwendung wird auf dieses geprüft und falls es existiert eine erneute Installation übersprungen.

## Statische Analyse auf Windows

Wird nur die statische Analyse auf Windows durchgeführt, wird das Setup-Script direkt mit Python 3 ausgeführt. Dabei wird ähnlich zur Installation auf Windows zuerst das Config-File initialisiert und anschließend die Ordner erstellt. Auch werden *nuget*, *BinSkim* und *BinScope* installiert.

Zusätzlich zur lokalen Installation wird jedoch ein XMLRPC-Server installiert und konfiguriert. Die genaue Funktionsweise dieses Services ist unter 4.6.1.6 beschrieben.

Von Ajin Abraham, eines Contributors zu MobSF, wurde nach der Implementierung ein ergänzendes Video zur Installation erstellt und auf Youtube veröffentlicht<sup>16</sup>.

### 4.6.2.4. Statische Analyse

Um die drei marktführenden mobilen Betriebssysteme mit *MobSF* abzudecken, wurden Funktionen für Windows-Phone-Apps hinzugefügt.

Zur statischen Analyse wurde das Tool *binskim* von Microsoft getestet<sup>17</sup>. Das Tool analysiert Compiler-Flags und verschiedenste andere statisch feststellbare Eigenschaften, bewertet diese und gibt die Ergebnisse im SARIF-Format<sup>18</sup> zurück. Leider ist das Tool nur unter Windows ausführbar. Da *MobSF* jedoch auch auf Linux und Mac OS X lauffähig sein soll, wird im folgenden eine virtuelle Windows-Maschine zur statischen und dynamischen Analyse verwendet.

\_CRT\_SECURE\_NO\_WARNINGS

## Files

TODO

## Bad Functions

Ebenso wie bei iOS wurde auch hier eine Extraktion der Strings aus dem Binary implementiert. Genutzt wurde dafür die unter 4.6.1.4 entwickelte Implementierung des Strings-Kommandos in Python.

In diesen Strings sind ebenfalls die Namen der verwendeten Funktionen vorhanden, sodass auch hier eine Suche nach bekannterweise verwundbaren Funktionen implementiert wurde.

### 4.6.2.5. Eingebundene Tools

Im Folgenden werden die für die Analyse von Windows-Apps eingebundenen Tools kurz vorgestellt.

TODO: <https://blog.netspi.com/verifying-aslr-dep-and-safeseh-with-powershell/>

<sup>16</sup><https://www.youtube.com/watch?v=17ilENuMj58>

<sup>17</sup><https://github.com/Microsoft/binskim/releases>

<sup>18</sup><https://github.com/sarif-standard/sarif-spec/>

## Binskim

nuget download A binary static analysis tool that provides security and correctness results for Windows portable executables.

This repository contains source code for BinSkim, a portable executable scanner that validates compiler/linker settings and other security-relevant binary characteristics.

TODO übersetzen

**LoadImageAboveFourGigabyteAddress** 64-bit images should have a preferred base address above the 4GB boundary to prevent triggering an Address Space Layout Randomization (ASLR) compatibility mode that decreases security.

**DoNotIncorporateVulnerableDependencies** Binaries should not take dependencies on code with known security vulnerabilities.

**DoNotShipVulnerableBinaries** Do not ship obsolete libraries for which there are known security vulnerabilities.

**BuildWithSecureTools** Application code should be compiled with the most up-to-date tool sets possible to take advantage of the most current compile-time security features.

**EnableCriticalCompilerWarnings** Binaries should be compiled with a warning level that enables all critical security-relevant checks.

**EnableControlFlowGuard** Binaries should enable the compiler control guard feature (CFG) at build time to prevent attackers from redirecting execution to unexpected, unsafe locations.

**EnableAddressSpaceLayoutRandomization** Binaries should be linked as DYNAMICBASE to be eligible for relocation by Address Space Layout Randomization (ASLR).

**DoNotMarkImportsSectionAsExecutable** PE sections should not be marked as both writable and executable.

**EnableStackProtection** Binaries should be built with the stack protector buffer security feature (/GS) enabled to increase the difficulty of exploiting stack buffer overflow memory corruption vulnerabilities.

**DoNotModifyStackProtectionCookie** Application code should not interfere with the stack protector.

**InitializeStackProtection** Binaries should properly initialize the stack protector (/GS) in order to increase the difficulty of exploiting stack buffer overflow memory corruption vulnerabilities.

**DoNotDisableStackProtectionForFunctions** Application code should not disable stack protection for individual functions.

**EnableHighEntropyVirtualAddresses** Binaries should be marked as high entropy Address Space Layout Randomization (ASLR) compatible.



**MarkImageAsNXCompatible** Binaries should be marked as NX compatible to help prevent execution of untrusted data as code.

**EnableSafeSEH** X86 binaries should enable the SafeSEH mitigation to minimize exploitable memory corruption issues.

**DoNotMarkWritableSectionsAsShared** Code or data sections should not be marked as both shared and writable.

**DoNotMarkWritableSectionsAsExecutable** PE sections should not be marked as both writable and executable.

**SignSecurely** Images should be correctly signed by trusted publishers using cryptographically secure signature algorithms.

Der originale Text sowie längere Beschreibungen sind dem Output des Kommandos

```
1 BinSkim.exe exportRules output.json
```

zu entnehmen. Ebenfalls ist der Output angehängt unter TODO.

### Binscope

x64 vs x68 <https://github.com/DominikSchlecht/Mobile-Security-Framework-MobSF/issues/10>  
Fehler-Beschreibungen

**ATLVersionCheck** Verifies that ATL headers used to build the binary are known good.  
For COM only.

**ATLVulnCheck** Detects classes implementing IPersistStreamInit that have potentially vulnerable property map entries. For COM only.

**APTCACheck** Reports a failure if the binary being verified is a managed assembly, has a strong name signature, and bears the APTCA attribute (AllowPartiallyTrusted-CallersAttribute). Such assemblies can be potentially dangerous and should not be shipped without a thorough security review.

**SectCheck** Reports a failure if the PE-format binary being verified has sections marked as shared and writable. Having such sections is a potential security vulnerability, and their use should be avoided.

**GSCheck** Verifies that the /GS compiler flag was used to compile all components of the binary and shows detailed information per object in the binary. It is possible that only part of the object files in a binary were built with /GS. In this case you will need to find the owner of non-/GS-compiled objects or libraries and request a compliant version. Note: GSCheck needs access to the debug symbols for the binary. Ensure that the correct symbols are able to be located (for example by setting  $NTSYMBOLPATH = SRV * symbols$ ).

**SafeSEHCheck** Verifies that the image was linked using /SAFESEH. Not using /SAFESEH undermines the protection provided by /GS. Note - In order to link an image with /SAFESEH, all object files and lib files must be /SAFESEH-compatible.

**FPCheck** Identifies images having global function pointers. Overriding static buffers can cause global function pointers to be overwritten, which may expose a security vulnerability. This is not covered by /GS protection; therefore, if you have global function pointers, you may want to examine use of static/global buffers in your code to make sure there are no possible security issues. This check is not enforced by SDL, but it is strongly recommended that you check use of your static buffers to make sure no buffer overruns are possible. FPCheck requires that symbols be present (see the /GS check note above).

**SicCheck** Identifies images that have non-/GS-friendly initialization. When using /GS, the executable needs some way to initialize the /GS infrastructure at load time, and usually it is done in CRT startup or similar functions. However, if the executable is linked in such a way that no standard code is executed at startup (such as with /NOENTRY linker option) and no custom /GS-startup routine is provided, that image will be left unprotected. This check identifies these images.

**CompilerCheck** Identifies images that contain C or C++ modules compiled with a compiler older than the version required by the SDL. Specifically, BinScope checks that the C/C++ compiler (cl.exe) is at least version 14.00.50727 and the CVTRES compiler, the MASM compiler and the linker are at least version 8.00.50727. Those are the versions of the tools contained in Visual Studio 2005.

**DBCheck** Checks if a binary has opted into the ASLR feature.

**SNCheck** Checks for use of strong-named assemblies. A strong name is a digital signature representing a cryptographically unique name for a managed assembly. Integrity of information is protected by digital signature. No piece of the strong name and no bits in the assembly body can be modified without rebuilding the assembly.

**NXCheck** Checks if a binary has opted into Hardware Data Execution Prevention.

<http://www.dotnetpark.com/kb/3784-binscope-binary-analyzer.aspx>

### Decompiler?

<http://www.telerik.com/products/decompiler.aspx>

#### 4.6.2.6. Dynamische Analyse von Windows-Apps

Für das Remote-System sollte beachtet werden, dass für den Phone-Simulator von Microsoft Windows 8.1 Pro 64-Bit oder höher benötigt wird.

Verwendet wird ein *Windows 10 64-Bit* mit *Python 3.5* und *Visual Studio 2015* Community Edition verwendet. Bis auf das Betriebssystem sind keine kostenpflichtigen Programme beteiligt.

windows driver kit für windbg

```
DerivedData/  
├── Appname-random/  
├── Appname2-random/  
│   ├── Build/  
│   │   ├── Products/  
│   │   │   └── Platform/  
│   │   │       └── Appname.app  
│   ├── Index/  
│   ├── Logs/  
│   ├── TextIndex/  
│   ├── info.plist  
│   └── scm.plist
```

Abbildung 4.10.: Struktur des DerivedData-Ordners

### 4.6.3. iOS-Apps

TODO

#### 4.6.3.1. Umwandeln app zu ipa

Die Umwandlung von *.app* zu *.ipa* ist über wenige händische Schritte zu verwirklichen. Kompiliert man eine App in XCode, wird diese unter einem bestimmten Pfad abgelegt. Der Standardpfad unter XCode 8 ist */Users/dominik/Library/Developer/Xcode/DerivedData/*. Alternativ kann der Pfad den Projekt-Einstellung in XCode über *File → Projects Settings* entnommen werden.

Die Struktur des *DerivedData*-Ordner ist unter ?? dargestellt.

Die *.app*-Datei kann nun per Drag-n-Drop in den App-Bildschirm von *ITunes* gezogen werden. Anschließend wird die App in *ITunes* angelegt (siehe Grafik 4.11). Wird die App per Drag-n-Drop wieder aus *ITunes* in den Finder gezogen, wird die App als IPA-File abgelegt.

#### 4.6.3.2. iOS-Permissions

In den aktuellen Version von iOS müssen Berechtigungen, welche die App zur Laufzeit anfordert, in der *info.plist* angekündigt werden. Im Folgenden sind die derzeit möglichen Berechtigungen aufgeführt.

**NSAppleMusicUsageDescription** Specifies the reason for your app to use the media library. See *NSAppleMusicUsageDescription* for details.

**NSBluetoothPeripheralUsageDescription** Specifies the reason for your app to use Bluetooth. See *NSBluetoothPeripheralUsageDescription* for details.

**NSCalendarsUsageDescription** Specifies the reason for your app to access the user's calendars. See *NSCalendarsUsageDescription* for details.

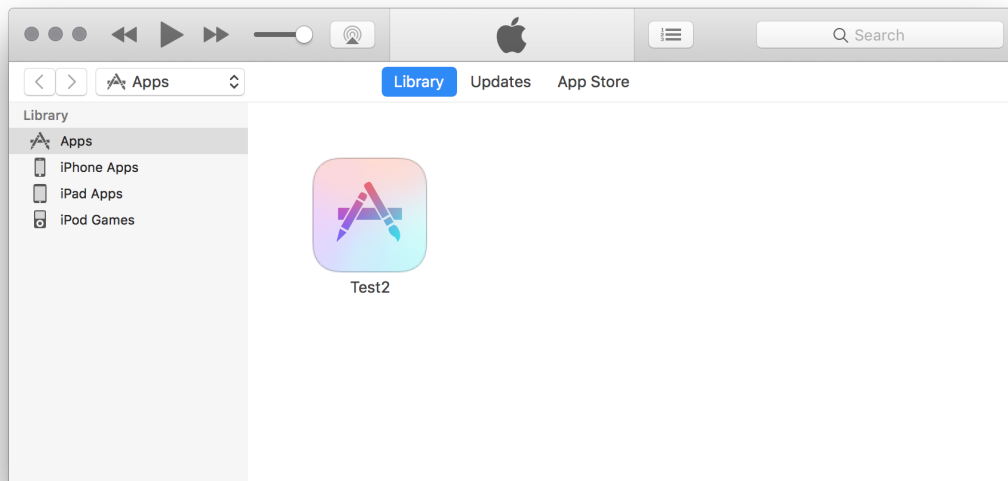


Abbildung 4.11.: iTunes-App-Fenster

**NSCameraUsageDescription** Specifies the reason for your app to access the device's camera. See `NSCameraUsageDescription` for details.

**NSContactsUsageDescription** Specifies the reason for your app to access the user's contacts. See `NSContactsUsageDescription` for details.

**NSHealthShareUsageDescription** Specifies the reason for your app to read the user's health data. See `NSHealthShareUsageDescription` for details.

**NSHealthUpdateUsageDescription** Specifies the reason for your app to make changes to the user's health data. See `NSHealthUpdateUsageDescription` for details.

**NSHomeKitUsageDescription** Specifies the reason for your app to access the user's HomeKit configuration data. See `NSHomeKitUsageDescription` for details.

**NSLocationAlwaysUsageDescription** Specifies the reason for your app to access the user's location information at all times. See `NSLocationAlwaysUsageDescription` for details.

**NSLocationUsageDescription** Unused. Use `NSLocationWhenInUseUsageDescription` or `NSLocationAlwaysUsageDescription` instead. See `NSLocationUsageDescription` for details.

**NSLocationWhenInUseUsageDescription** Specifies the reason for your app to access the user's location information while your app is in use. See `NSLocationWhenInUseUsageDescription` for details.

**NSMicrophoneUsageDescription** Specifies the reason for your app to access any of the device's microphones. See `NSMicrophoneUsageDescription` for details.

**NSMotionUsageDescription** Specifies the reason for your app to access the device's accelerometer. See `NSMotionUsageDescription` for details.

**NSPhotoLibraryUsageDescription** Specifies the reason for your app to access the user's photo library. See `NSPhotoLibraryUsageDescription` for details.

**NSRemindersUsageDescription** Specifies the reason for your app to access the user's reminders. See `NSRemindersUsageDescription` for details.

**NSVideoSubscriberAccountUsageDescription** Specifies the reason for your app to access the user's TV provider account. See `NSVideoSubscriberAccountUsageDescription` for details.

[https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple\\_ref/doc/uid/TP40009251-SW47](https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW47)

Um in *MobSF* die Berechtigungen auszulesen, wurde dementsprechend die *info.plist* ausgelesen und auf die entsprechenden Einträge geprüft. Da die Datei jedoch im Binärformat vorliegt, muss diese zuerst umgewandelt werden. Dies kann über das in XCode enthaltene Tool *plutil* erreicht werden. Der Aufruf ist dabei wie folgt:

```
1 plutil -convert xml1 info.plist
```

Anschließend kann über das in Python enthaltene Modul *plistlib* wie folgt auf die Einträge zugegriffen werden:

```
1 p_list = plistlib.readPlistFromString(read_bin_xml(  
    converted_info_plist_file))  
2 if "NSBluetoothPeripheralUsageDescription" in p_list:  
3     print("Bluetooth-Permission found!")
```

in *MobSF* wird auf diese Art und Weise die *info.plist*-Datei verarbeitet und die Ergebnisse in der Web-Oberfläche dargestellt. Das Ergebnis für eine App mit Bluetooth-Berechtigung ist in Grafik 4.12 dargestellt.

#### 4.6.3.3. Erkennung von ungesicherten Verbindungen

Wie in 4.3.1.4 unter „Ungesicherte Verbindungen beschrieben“, müssen ab *iOS* 9.0 Apps den RFC-Standard 2818<sup>19</sup> nutzen, um Verbindungen zu Webseiten oder APIs aufzubauen.

Daher wurde *MobSF* um ein Feature ergänzt, welches die *Info.plist* auf Ausnahmen überprüft. Der simple Code ist unter Listing 4.13 dargestellt.

Werden Ausnahmen gefunden, werden diese in der HTML-Oberfläche dargestellt. Ein Beispiel ist in TODO zu sehen.

#### 4.6.3.4. Dynamische Analyse über Simulator

Im Rahmen dieser Masterarbeit wurden ebenfalls Möglichkeiten getestet, wie eine dynamische Analyse von *iOS*-Apps über den „Simulator“ abgebildet werden könnten.

<sup>19</sup><https://tools.ietf.org/html/rfc2818>

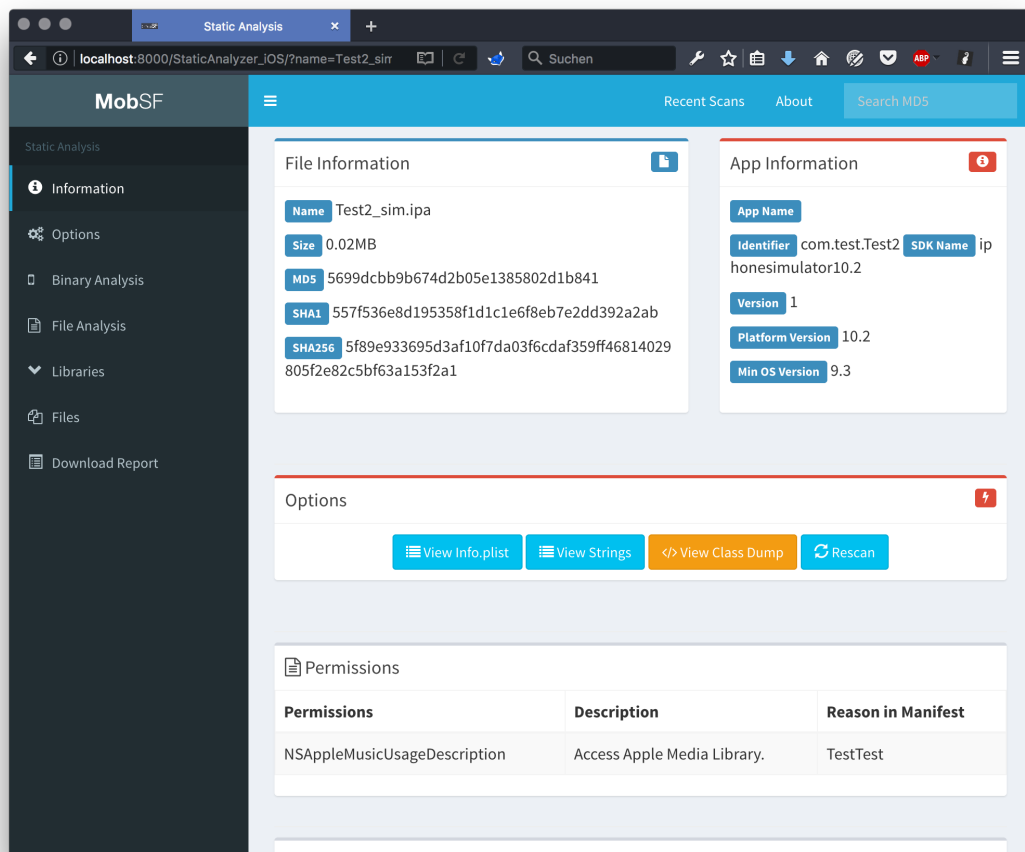


Abbildung 4.12.: Permission-Check-Ergebnis für eine App mit Bluetooth-Berechtigung

```
1 def __check_insecure_connections(p_list):
2     '''Check info.plist for insecure connection configurations'''
3     print "[INFO] Checking for Insecure Connections"
4
5     insecure_connections = []
6
7     if 'NSAppTransportSecurity' in p_list:
8         ns_app_trans_dic = p_list['NSAppTransportSecurity']
9         if 'NSExceptionDomains' in ns_app_trans_dic:
10             for key in ns_app_trans_dic['NSExceptionDomains']:
11                 insecure_connections.append(key)
12
13     return insecure_connections
```

Abbildung 4.13.: Auslesen von Ausnahmen bezüglich der TLS-Konfiguration aus der Info.plist

Erster Versuch Gestartet aus xcode, app:

```
1 NSURL *url = [NSURL URLWithString:@"https://api.ipify.org"];
2 NSData *data = [NSData dataWithContentsOfURL:url];
3 NSString *ret = [[NSString alloc] initWithData:data encoding:
    NSUTF8StringEncoding];
```

Wireshark output:

No.	Time	Source	Destination	Protocol	Length	Info
179	105.443855	192.168.178.30	192.168.178.1	DNS	87	Standard query 0x3770 AAAA api.ipify.org.herokudns.com
180	105.444819	192.168.178.30	192.168.178.1	DNS	87	Standard query 0x9317 A api.ipify.org.herokudns.com
181	105.466520	192.168.178.1	192.168.178.30	DNS	168	Standard query response 0x3770 AAAA api.ipify.org.herokudns.com SOA ns-955.awsdns-55.net
182	105.467195	192.168.178.1	192.168.178.30	DNS	103	Standard query response 0x9317 A api.ipify.org.herokudns.com A 23.23.107.79
183	105.471645	192.168.178.30	23.23.107.79	TCP	78	50357 → 443 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=724572195 TSecr=0
184	105.593358	23.23.107.79	192.168.178.30	TCP	74	443 → 50357 [SYN, ACK, ECN] Seq=0 Ack=1 Win=17898 Len=0 MSS=1452 SACK_PERM=1 TSval=1204797388
185	105.593411	192.168.178.30	23.23.107.79	TCP	66	50357 → 443 [ACK] Seq=1 Ack=1 Win=132480 Len=0 TSval=724572316 TSecr=1204797351
186	105.598051	192.168.178.30	23.23.107.79	TLSv1.2	288	Client Hello
187	105.720517	23.23.107.79	192.168.178.30	TCP	66	443 → 50357 [ACK] Seq=1 Ack=223 Win=19200 Len=0 TSval=1204797383 TSecr=724572320
188	105.738686	23.23.107.79	192.168.178.30	TLSv1.2	1506	Server Hello
189	105.738705	23.23.107.79	192.168.178.30	TCP	1506	[TCP segment of a reassembled PDU]
190	105.738706	23.23.107.79	192.168.178.30	TCP	1506	[TCP segment of a reassembled PDU]
191	105.738790	192.168.178.30	23.23.107.79	TCP	66	50357 → 443 [ACK] Seq=223 Ack=2881 Win=129600 Len=0 TSval=724572460 TSecr=1204797388
192	105.738894	192.168.178.30	23.23.107.79	TCP	66	50357 → 443 [ACK] Seq=223 Ack=4321 Win=131072 Len=0 TSval=724572460 TSecr=1204797388
193	105.740993	23.23.107.79	192.168.178.30	TLSv1.2	1506	Certificate
194	105.740998	23.23.107.79	192.168.178.30	TLSv1.2	175	Server Key Exchange
195	105.741075	192.168.178.30	23.23.107.79	TCP	66	50357 → 443 [ACK] Seq=223 Ack=5870 Win=129504 Len=0 TSval=724572462 TSecr=1204797388
196	105.746638	192.168.178.30	23.23.107.79	TLSv1.2	141	Client Key Exchange
197	105.746639	192.168.178.30	23.23.107.79	TLSv1.2	72	Change Cipher Spec

Abbildung 4.14.: Verbindung zu https://api.ipify.org aus dem iOS-Simulator

Öffnen aus console

mitmproxy vs. <http://useyourloaf.com/blog/remote-packet-capture-for-ios-devices/> <http://www.binarytides.com/packet-sniffer-code-linux/>

mitmproxy installieren, Proxy umstellen, <https://github.com/ADVTOOLS/ADVTrustStore>

ausführen und adden, im Iphone aktivieren, neu starten mitmproxy starten und anfangen zu

schreib, lesen und in db speichern <https://github.com/mitmproxy/mitmproxy/blob/0.18.x/examples/flowbasic>

LLDB + Code Simulator (siehe weiter oben)

ausführen von aktionen <https://developer.apple.com/reference/xctest> xcrun simctl install

booted /Users/dominik/Library/Developer/Xcode/DerivedData/Test2-dzkepdcdrezeblbvysnxcbrmp/Build/Products

iphonesimulator/Test2.app xcrun simctl launch 4A4B2FF5-C435-4ECB-AC49-A4684F133910

Test2

simulator starten einfach über öffnen von App (IPA = Simulator?)

<http://www.libimobiledevice.org/> <https://github.com/facebook/FBSimulatorControl> <http://blog.manbolo.com/2014/04/01/automated-tests-with-uiautomation>

automated-tests-with-uiautomation

<http://eightbit.io/post/64319534191/how-to-set-up-an-ios-pen-testing-environment>

<http://www.testplant.com/dlds/eggplant-mobile-eggplant/> ? <http://www.sikuli.org/>

remote anzeige git://git.saurik.com/cydia.git <http://kanaka.github.io/noVNC/> als front

interface <http://sharedinstance.net/2013/10/running-tweaks-in-simulator/> vnc server auf

Iphone installieren, z.B. <https://github.com/niknah/veency>

TODO: Öffnen aus Console schreiben TODO: Netzwerkverkehr mitschneiden schreiben

TODO: Installation VNC-Server auf Simulator testen j- Kein Chance TODO: VNC in

Web-Oberfläche testen

## 4.7. Abgleich mit Anforderungen

Durch die Weiterentwicklung des MobSF wurde vor allem der Punkt „Unterstützung von Android/iOS/Windows Phone“ verbessert. So ist nun eine statische Analyse von Windows-Phone-Apps über die Tools des Windows SDL möglich. Zudem wurde die Grundlage für eine dynamische Analyse von iOS-Apps geschaffen. Auch wurde die statische Analyse von iOS-Apps erweitert, was bessere Ergebnisse zur Folge hat.

Somit sind alle Anforderungen aus 4.1 ausreichend erfüllt.

## 4.8. Anwendung der Umgebung

Im Folgenden wird die weiterentwickelte Version des *MobSF* auf zwei selbstgeschriebene Applikationen angewendet. Dabei handelt es sich bei Anwendung 1 um eine Windows-Phone-App und bei Applikation 2 um eine iOS-App.

### 4.8.1. Test Anwendung 1 - Windows-Phone

TODO Unsichere Funktionen

### 4.8.2. Test Anwendung 2 - iOS

TODO Zur Entwicklung der Test-Applikation wurde XCode-8 verwendet. Die vollständigen Source-Dateien sind dabei unter TODO oder auf dem beigelegten Datenträger zu finden.

Die Analyse der App in *MobSF* ergibt das in Grafik TODO dargestellte Ergebnis.  
TODO Bild der Analyse Quellcode Memory-Corruption Unsichere Verbindungen



## 5. Fazit

In dieser Arbeit wurden Anfangs die verschiedenen Arten von Pen-Tests mit deren jeweiligen Charakteristika und Besonderheiten aufgezeigt. Anschließend wurden die Prozesse zu Pen-Tests, aufgeteilt nach Vorbereitung, Durchführung, Nachbereitung und Kontinuität, vorgestellt und Best-Practices erläutert. Zudem wurden für manche Prozessschritte Tools entwickelt, welche die Effizienz sowohl auf Seiten der Pen-Tester als auch des Unternehmens erhöhen können. Daraufhin wurde auf Tools und Frameworks eingegangen und eine bestimmte Anwendung so weiterentwickelt, dass diese alle gesetzten Anforderungen erfüllt. Abschließend wurde das Framework anhand von 2 Beispielen vorgestellt.

TODO mehr



# A. Quellcode

## A.1. Fragebogen

### A.1.1. Latex

```
1 \usepackage{amssymb}
2 \usepackage{array}
3 \usepackage{tabularx}
4 \usepackage[backend=biber,]{biblatex}
5 \usepackage[utf8]{luainputenc}
6
7 \renewcommand{\familydefault}{\sfdefault}
8 \usepackage{graphicx}
9
10 \usepackage{scrpage2} % Kopf & Fußzeile im KOMA Stil
11 \pagestyle{scrheadings} % Aktiviert Verwendung vordefinierter
    Kolumnentitel
12 \clearscrheadfoot % alle Standard-Werte und
    Formatierungen löschen
13 \setkomafont{pagehead}{} % Schriftart in Kopfzeile, \
    scshape = Kapitelchen
14 \automark[chapter]{section} % [linke Seite]{rechte Seite}
15 %\ohead{\def\pagestyle{PDTS}{\hrulefill\includegraphics[width =
    6cm]{bilder/thi_logo_quer_cropped}}}
16 \ohead{\includegraphics[width = 3cm]{allianz_logo.jpg}}
17 \ihead{Fragebogen Penetrations-Test}
18
19 \setlength{\headsep}{7mm} % Textabstand zur
    Kopfzeile
20 \setlength{\footskip}{7mm} % Abstand zur Fußzeile
21
22 \ofoot{\vspace{-0.3cm} \pagemark}
23
24 \setheadsepline{.2pt}
25 \setfootsepline{.4pt} % Trennlinie Fußzeile und Textkörper
26
27 \newcolumntype{S}{>{\centering\arraybackslash}m{2em}}
28
29 \ifoot{\vspace{-1.7cm}}
30 \begin{tabular}{1 || 1}
31 % Change me!
32 \textit{Unternehmen} & \textit{Pentester}\\
33 \textit{Straße Nr} & \textit{Tel}
    \\
\end{tabular}
```

```

34 \textit{PLZ Ort} & \
    \textit{E-Mail}\\
35 \textit{USTID} & \
    \textit{Webseite}
36 \end{tabular}
37 }

```

Listing A.1: packages.tex

```

1 \newcommand{\frage}[1]{\makebox[\textwidth]{%
2 \renewcommand{\arraystretch}{2.0}
3 \begin{tabularx}{\textwidth}{|X|}
4 \hline
5 #1 \\
6 \hline
7 \line(1,0){420}\\
8 \hline \hline
9 \end{tabularx}%
10 \renewcommand{\arraystretch}{1.0}
11 }}
12
13 \newcommand{\frageJaNein}[1]{\makebox[\textwidth]{%
14 \renewcommand{\arraystretch}{2.0}
15 \begin{tabularx}{\textwidth}{|X|S|S|}
16 \hline
17 #1 & Ja & Nein\\
18 \hline
19 \line(1,0){350} & $\square$ & $\square$ \\
20 \hline \hline
21 \end{tabularx}%
22 \renewcommand{\arraystretch}{1.0}
23 }}
24
25 \newcommand{\frageOpt}[4]{\makebox[\textwidth]{%
26 \renewcommand{\arraystretch}{2.0}
27 \begin{tabularx}{\textwidth}{|X|X|X|}
28 \hline
29 \multicolumn{3}{|c|}{#1}\\
30 \hline
31 #2 $\square$ & #3 $\square$ & #4 $\square$\\
32 \hline \hline
33 \end{tabularx}%
34 \renewcommand{\arraystretch}{1.0}
35 }}
36
37 \newcommand{\frageJaNeinKurz}[1]{\makebox[\textwidth]{%
38 \renewcommand{\arraystretch}{1.5}
39 \begin{tabularx}{\textwidth}{|X|S|S|}
40 \hline
41 #1 & Ja $\square$ & Nein $\square$\\
42 \hline \hline
43 \end{tabularx}%

```

```

44 \renewcommand{\arraystretch}{1.0}
45 }}

```

Listing A.2: question\_templates.tex

```

1 \documentclass[11pt,DIV=11]{scrartcl} %
2 \input{packages.tex}
3 \input{question_templates.tex}
4
5 \begin{document}
6 \section{Allgemeines}
7 \subsection{Ansprechpartner}
8 \begin{itemize}
9     \item Ansprechpartner
10     \begin{itemize}
11         \item \line(1,0){350}
12         \item \line(1,0){350}
13         \item \line(1,0){350}
14     \end{itemize}
15 \end{itemize}
16
17 \subsection{Informationen}
18 \begin{itemize}
19     \item Alle Inhalte, auch die dieses Gesprächs, werden
20         vertraulich behandelt
21     \item Schäden werden ausgeschlossen
22     \item Ein Pentest hat nie einen Anspruch auf Vollständigkeit
23 \end{itemize}
24 \subsection{Eingrenzung}
25 Welche Art/en von Pentest/s sollen durchgeführt werden?
26 \begin{itemize}
27     \item[$\square$] Web-Application $\rightarrow$ Punkt \ref{ref:WebAppPenTest} ausfüllen
28     \item[$\square$] Network $\rightarrow$ Punkt \ref{ref:NetPenTest} ausfüllen
29     \item[$\square$] Social Engineering $\rightarrow$ Punkt \ref{ref:SocEngi} ausfüllen
30     \item[$\square$] Wireless $\rightarrow$ Punkt \ref{ref:WirNetPen} ausfüllen
31     \item[$\square$] Physical $\rightarrow$ Punkt \ref{ref:PhysPen} ausfüllen
32 \end{itemize}
33
34 Die Punkte \ref{ref:allg_frag}, \ref{ref:QuesSysAdmin} und \ref{ref:QuesBUM} sollten immer ausgefüllt werden, unabhängig von
35 der/den oben gewählten Art/en.
36
37 \subsection{Allgemeine Fragen}\label{ref:allg_frag}
38 \frageJaNein{Ist der Test für eine spezielle Compliance-

```

```

    Anforderung notwendig?}
39 \frage{Wann soll der Test statt finden?}
40 \frageOpt{In welchen Zeiträumen soll der Test durchgeführt
    werden?}{Bürozeiten}{Feierabend}{Wochenende}
41 \newpage
42
43 \section{Web Application Penetration Test}\label{ref:
    WebAppPenTest}
44
45 \frageJaNein{Wird der Quellcode der Applikation/Webseite zugä
    nglich gemacht?}
46 \frage{Wie viele Web-Applikationen sind In-Scope?}
47 \frage{Wie viele Login-Systeme sind In-Scope?}
48 \frage{Wie viele statische Seiten sind ca. In-Scope?}
49 \frage{Wie viele dynamische Seiten sind ca. In-Scope?}
50 \frageJaNeinKurz{Soll Fuzzing gegen die Applikation/en
    eingesetzt werden?}
51 \frageJaNein{Soll der Penetrations-Test aus verschiedenen Rollen
    durchgeführt werden?}
52 \frageJaNeinKurz{Sollen Password-Scans auf die Webseite durchgef
    ührt werden?}
53
54 \section{Network Penetration Test}\label{ref:NetPenTest}
55
56 \frage{Was ist das Ziel des Penetrations-Test?}
57 \frage{Wie viele IP-Adressen sollen getestet werden?}
58 \frageJaNein{Sind Techniken im Einsatz, die die Resultate verfä
    lschen könnten? (WAF, IPS etc.)}
59 \frage{Wie ist das Vorgehen bei einem gelungenen Angriff?}
60 \frageJaNein{Soll versucht werden lokale Admin-Rechte zu
    erlangen und tiefer in das Netz vorzudringen?}
61 \frageJaNeinKurz{Sollen Angriffe auf gefundene Passwort-Hashes
    durchgeführt werden?}
62
63 \section{Social Engineering}\label{ref:SocEngi}
64
65 \frageJaNeinKurz{Gibt es eine vollständige Liste von E-Mail-
    Adressen, die für den Test verwendet werden können?}
66 \frageJaNeinKurz{Gibt es eine vollständige Liste von Telefon-
    Nummern, die für den Test verwendet werden können?}
67 \frageJaNeinKurz{Ist das Einsetzen von Social Engineering zum Ü
    berwinden physikalischer Sicherheitseinrichtungen erlaubt?}
68 \frage{Wie viele Personen sollen ca. getestet werden?}
69
70
71 \section{Wireless Network Penetration Test}\label{ref:WirNetPen}
72 \frage{Wieviele Funk-Netzwerke sind im Einsatz?}
73 \frageJaNein{Gibt es eine Gäste WLAN? Wenn ja, wie ist dieses
    Umgesetzt?}
74 \frage{Welche Verschlüsselung wird für die Netzwerke genutzt?}
75 \frageJaNeinKurz{Sollen nicht-firmen-Geräte im WLAN aufgespürt
    werden?}

```

```
76 \frageJaNeinKurz{Soll Netz-Attacken gegen Clients durchgeführt
    werden?}
77 \frage{Wie viele Clients nutzen das WLAN ca.?}
78
79 \section{Physical Penetration Test}\label{ref:PhysPen}
80
81 \frage{Wie viele Einrichtungen sollen getestet werden?}
82 \frageJaNeinKurz{Wird die Einrichtung mit anderen Parteien
    geteilt?}
83 \frageJaNeinKurz{Muss Sicherheitspersonal umgangen werden?}
84 \frageJaNein{Wird das Sicherheitspersonal durch einen Dritten
    gestellt?}
85 \frageJaNeinKurz{Ist das Sicherheitspersonal bewaffnet?}
86 \frageJaNeinKurz{Ist der Einsatz von körperlicher Gewalt durch
    das Sicherheitspersonal gestattet?}
87 \frage{Wie viele Eingänge gibt es zu der/den Einrichtung/en?}
88 \frageJaNeinKurz{Ist das Knacken von Schlössern oder Fälschen
    von Schlüsseln erlaubt?}
89 \frage{Wie groß ist die Fläche ungefähr?}
90 \frageJaNein{Sind alle physikalischen Sicherheitsmaßnahmen
    dokumentiert und werden zur Verfügung gestellt?}
91 \frageJaNeinKurz{Werden Video-Kameras verwendet?}
92 \frageJaNein{Werden diese Kameras durch Dritte verwaltet?}
93 \frageJaNeinKurz{Soll versucht werden, die aufgezeichneten Daten
    zu löschen?}
94 \frageJaNeinKurz{Gibt es ein Alarm-System?}
95 \frageJaNeinKurz{Gibt es einen Stillen Alarm?}
96 \frageJaNeinKurz{Welche Ereignisse lösen den Alarm aus?}
97
98 \section{Questions for Systems Administrators}\label{ref:
    QuesSysAdmin}
99
100 \frageJaNein{Gibt es Systeme, die als instabil angesehen werden
    (alte Patch-Stände, Legacy Systeme etc.)?}
101 \frageJaNein{Gibt es Systeme von Dritten, die ausgeschlossen
    werden müssen oder für die weitere Genehmigungen notwendig
    sind?}
102 \frage{Was ist die Durchschnittszeit zur Wiederherstellung der
    Funktionalität eines Services?}
103 \frageJaNeinKurz{Ist eine Monitoring-Software im Einsatz?}
104 \frage{Welche sind die kritischsten Applikationen?}
105 \frageJaNeinKurz{Werden in einem regelmäßigen Turnus Backups
    erstellt und getestet?}
106
107 \section{Questions for Business Unit Managers}\label{ref:QuesBUM
    }
108
109 \frageJaNeinKurz{Ist die Führungsebene über den Test informiert
    ?}
110 \frage{Welche Daten stellen das größte Risiko dar, falls diese
    manipuliert werden?}
111 \frageJaNeinKurz{Gibt es Testfälle, die die Funktionalität der
```

```
Services prüfen und belegen können?}  
112 \frageJaNeinKurz{Sind "Disaster Recovery Procedures" vorhanden?}  
113  
114 \renewcommand{\arraystretch}{1.0}  
115 \end{document}
```

Listing A.3: Frage.tex



# Literatur

- [1] *Android Debug Bridge*. URL: <http://developer.android.com/tools/help/adb.html> (besucht am 22.02.2016).
- [2] *Apple zu AppleNSAppTransportSecurity*. URL: [https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple\\_ref/doc/uid/TP40009251-SW33](https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33) (besucht am 16.02.2017).
- [3] Inc. Apple. *OSX 10.11 EULA*. URL: <http://images.apple.com/legal/sla/docs/OSX1011.pdf> (besucht am 16.05.2016).
- [4] *CVSS*. URL: <https://www.first.org/cvss> (besucht am 21.02.2017).
- [5] *CVSS Spezifikation*. URL: <https://www.first.org/cvss/specification-document> (besucht am 21.02.2017).
- [6] *CWE*. URL: <http://cwe.mitre.org/about/index.html> (besucht am 21.02.2017).
- [7] Joshua J. Drake u. a. *Android Hacker's Handbook*. John Wiley & Sons, Inc., Indianapolis, Indiana, 2014.
- [8] Gartner. *Gartner Sales PC/Mobile Phones*. URL: <http://www.gartner.com/newsroom/id/3270418> (besucht am 16.05.2016).
- [9] *OenStack über DREAD*. URL: <https://wiki.openstack.org/wiki/Security/OSSA-Metrics#DREAD> (besucht am 21.02.2017).
- [10] *OWASP Top 10*. URL: [https://www.owasp.org/index.php/About\\_The\\_Open\\_Web\\_Application\\_Security\\_Project#The\\_OWASP\\_Foundation](https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project#The_OWASP_Foundation) (besucht am 21.02.2017).
- [11] *OWASP über DREAD*. URL: [https://www.owasp.org/index.php/Threat\\_Risk\\_Modeling](https://www.owasp.org/index.php/Threat_Risk_Modeling) (besucht am 21.02.2017).