

Masterarbeit

Prozesse, Methoden und Werkzeuge zum Pen-Test bei mobilen Applikationen

zur Erlangung des akademischen 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

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

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

Ingolstadt, 25. März 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, 25. März 2017

Inhaltsverzeichnis

Erklärung	i
1. Einleitung	1
2. Arten von Penetration-Tests	3
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	5
2.8. Redteam	5
3. Prozesse zu Penetration-Tests	7
3.1. Vorbereitung	7
3.1.1. Aufwandsschätzung	7
3.1.2. Rechtliche Aspekte	20
3.1.3. Technische Aspekte	21
3.2. Durchführung	23
3.2.1. Kickoff	23
3.2.2. Kategorisierung von Findings	24
3.2.3. Bewertung von Findings	25
3.2.4. Dokumentation	29
3.3. Nachbereitung	31
3.3.1. Inhalte eines Pen-Test-Berichts	31
3.3.2. Implementierung als Webanwendung	33
3.4. Kontinuität	34
4. Penetration-Tests mobiler Anwendungen	37
4.1. Anforderungen	37
4.2. Bestehende Anwendungen	38
4.2.1. All-In-One-Framework: MobSF	38
4.2.2. Einzelanwendungen	38
4.3. Vergleich der Emulationsumgebungen	40
4.3.1. iOS	40
4.3.2. Windows-Phone	47
4.3.3. Android	48
4.4. Abgleich der Anforderungen mit MobSF	50
4.5. Laboraufbau	51

4.6.	Weiterentwicklung MobSF	51
4.6.1.	Allgemeine Verbesserungen	51
4.6.2.	Windows-Apps	64
4.6.3.	iOS-Apps	73
4.7.	Abgleich mit Anforderungen	82
4.8.	Anwendung der Umgebung	82
4.8.1.	Test Anwendung 1 - Windows-Phone	82
4.8.2.	Test Anwendung 2 - iOS	83
5.	Fazit	85
A.	Quellcode	87
A.1.	Pentest-Helper	87
A.1.1.	Latex	87
A.1.2.	Docker	92
A.2.	iOS-Simulator	93
A.2.1.	Control-Script	93
A.2.2.	MITM-Server	95
A.3.	MobSF	97
A.3.1.	BinScope-Installer	97

1. Einleitung

Smartphones verbreiten sich immer stärker. Dies zeigt auch eine Statistik von Gardner, nach welcher die Absätze von mobilen Geräten die von herkömmlichen Rechnern und Laptops weiter übertreffen werden [8], siehe Tabelle 1.1. Mit diesem Fortschritt im Bereich der Verkäufe steigt natürlich auch die Nutzung von Smartphones. Diese können genutzt werden, um Webseiten aus dem Internet aufzurufen, Medien-Inhalte wiederzugeben oder über Kommunikationsprogramme andere Personen zu kontaktieren. Dies sind in 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. So erfreuen sich Banking-Apps immer größerer Beliebtheit, da es von Nutzern als praktisch empfunden wird, den Kontostand schnell von unterwegs einsehen zu können oder Bankgeschäfte zu erledigen. Jedoch sollten solche Apps als Konsequenz aus der Radikalität der Daten so gestaltet sein, dass ein Missbrauch nicht oder nur mit unverhältnismäßig hohem Aufwand möglich ist. Dies ist die Aufgabe der Unternehmen, welche solche Apps bereit stellen. Aktuelle Beispiele¹ zeigen jedoch, dass dies viele Unternehmen vor unerwartet hohe Herausforderungen stellt. 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 ebenfalls ein Unternehmen, welches sich auf diese Herausforderung vorbereiten muss. Bisher waren viele Wege zum Kunden analog, also per Brief oder direkt über die Außendienstmitarbeiter. Es gab nicht viele elektronische Schnittstellen. Vor zwei Jahren wurde eine Digitalisierungs-Strategie beschlossen und die Web-Anwendung „Meine-Allianz“ entwickelt. In dieser können Versicherungsnehmer nicht nur Verträge anschauen, sondern auch Änderungen an diesen vornehmen. In einer Weiterentwicklung stehen nun auch manche Funktionen in einer iOS-App zur Verfügung. Sollte diese Anwendung eine

¹https://media.ccc.de/v/33c3-7969-shut_up_and_take_my_money

schwerwiegende Schwachstelle aufweisen, hätte dies für die Allianz nicht nur Schäden in Bezug auf die Reputation, sondern eventuell zusätzliche rechtliche Folgen, da Krankendaten unter den §203 des StGB fallen. Um dies zu verhindern, sind Prozesse in der Anwendungsentwicklung notwendig, welche die Sicherheit einer Anwendung auf einen Möglichst hohen Stand heben.

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

In dieser Arbeit werden als Einführung die Arten von Pen-Tests erläutert. Anschließend werden bestehende, etablierte Prozesse um Komponenten für mobile Applikationen erweitert und Werkzeuge entworfen, um Prozessschritte effizienter abzubilden. Abschließend wird ein Teil eines bereits bestehendes Open-Source-Tools zum automatischen Testen mobiler Anwendungen um Funktionen erweitert, um alle Anforderungen der Allianz Deutschland AG zu erfüllen. Dies umfasst eine Verbesserung der Analysen von Android-, iOS- und Windows-Phone-Apps.

2. Arten von Penetration-Tests

Als Einführung dient zunächst ein kurzer Überblick über die verschiedenen Arten von Penetration-Tests (im Folgenden „Pen-Tests“). An diesen orientiert sich ebenfalls das zweite Kapitel, Prozesse zu Penetration-Tests (siehe Kapitel 3). Die Arten basieren dabei auf der Liste von *Pentest-Standard.org*¹, sind jedoch um weitere Arten, basierend auf den Erfahrungen des Autors sowie anderen Pen-Testern der Allianz Deutschland AG, ergänzt.

2.1. Web-Application

Web-Application-Tests finden wohl am häufigsten Anwendung. Dabei wird eine definierte Menge an Webseiten von einem sogenannten Penetration-Tester auf Schwachstellen getestet. Typische Findings 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 und die Tools bei Penetration-Test von anderen Arten unterscheidet.

2.3. Mobile-Application

Von einem *Mobile-Application-Test* redet man dann, wenn eine sog. App, also eine Anwendung geschrieben für ein mobiles Gerät, getestet wird. Da der Begriff nicht weiter definiert ist, ist unklar, ob ein *Mobile-Application-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 den Benutzer eines Mobiltelefons gefährden. Tests auf die Backend-Systeme von Apps dagegen sollen 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.

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

2.4. Network

Bei einem *Network-Pen-Test* geht es zumeist darum, in ein 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 Tests 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-Infrastruktur, welche Funktechniken nutzt. 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 Autoschlüsseln verwendet werden.

2.6. Social Engineering

Der BSI-Grundschutz beschreibt *Social Engineering* wie folgt:²

Social Engineering ist eine Methode, um unberechtigten Zugang zu Informationen oder IT-Systemen durch Aushorchen 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. Dazu täuscht der Pen-Test einen Grund vor, in das Gebäude zu müssen (zum Beispiel eine Blumenlieferung an einen Vorstand). Wird er aufgehalten, werden verschiedene soziale Aspekte wie Autorität („Ich darf die Blumen nur direkt dem Vorstand liefern. Wenn Sie mich nicht durchlassen, wird er das erfahren.“) oder Hilfsbereitschaft („Wenn ich diese Lieferung nicht abschließe, werde ich gefeuert.“) genutzt, um das Ziel zu erreichen. Alternativ können ähnliche Erfolge auch über Telefonanrufe erreicht werden, indem man zum Beispiel einen Mitarbeiter dazu überredet, sein Passwort preis zu geben.

Auch hier wird oft ein bestimmtes Ziel definiert, das der Pen-Tester zu erreichen hat.

²https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/g/g05/g05042.html

2.7. Physical

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

2.8. Redteam

Bei einem Redteam/Blueteam wird den Testern, welche das sogenannte Redteam bilden, im Gegensatz zu den anderen Arten von Pen-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 *Domain Controller*. 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

Nach der Erläuterung der verschiedenen Arten von Pen-Tests werden im Folgenden Prozesse und Best-Practices erläutert. Dabei ist das Kapitel aufgeteilt in die Abschnitte Vorbereitung, Durchführung, Nachbereitung und Kontinuität.

3.1. Vorbereitung

Vor der Durchführung eines Pen-Tests 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 des Projektmanagements verwendet werden, jedoch müssen eine Vielzahl weiterer 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*-Lücke oder *SQLi* zu finden, unabhängig von der Anzahl der Eingabefelder, äußerst 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 Pen-Tester 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 dienen die Fragen von Pentest-Standard.org¹. 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

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

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 sind in Sektion 3.1.1.3 dargestellt.

Um eine einfache Installation zu gewährleisten, wurde die Möglichkeit eines *Docker*-Containers getestet. Dies ist in Abschnitt 3.1.1.4 beschrieben.

3.1.1.1. Fragebogen

Um einen groben Eindruck vom Umfang des Tests zu bekommen, empfiehlt sich das Ausfüllen eines standardisierten Fragebogens. Im Rahmen dieser Arbeit wurde der Fragebogen von Pentest-Standard.org² übersetzt und erweitert. Anschließend wurde der Fragebogen mit den Ansprechpartnern der Allianz Deutschland AG diskutiert und ergänzt, sodass sich die folgenden Fragen ergeben. Dabei stehen die Begriffe in den Klammern für Antwortoptionen. Sind keine Antwortoptionen gegeben, ist die Frage eine Freitext-Frage.

Allgemein

- Wie ist der Projekt-Name?
- Wer sind die Ansprechpartner?
- Welche Art/en von Pen-Test/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/Pen-Test 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)

²<http://www.pentest-standard.org/index.php/Pre-engagement>

- 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 Penetration-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/Pen-Test 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 Penetration-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 Penetration-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 Penetration-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 ein 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 der Niederlassung 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?

Dabei müssen natürlich nur die für die Art des Pen-Tests (Web-Application/Network/Social Engineering/Wireless/Physical) notwendigen Fragen bearbeitet werden.

3.1.1.2. Implementierung in LaTeX

Nach 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.

Fragebogen Penetrations-Test



2 Web Application Penetration Test

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

Wie viele Web-Applikationen sind In-Scope?
<hr/>

Wie viele Login-Systeme sind In-Scope?
<hr/>

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

Um eine möglichst einfache Erweiterung der Fragen in *LaTeX* zu gewährleisten, wurden *LaTeX*-Makros für die jeweiligen Frage-Arten entworfen. In Abbildung 3.2 ist ein kurzes Beispiel für ein solches Makro dargestellt sowie zwei Fragen, die dieses nutzen.

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

Abbildung 3.2.: Makro zur automatischen Formatierung in LaTeX

3.1.1.3. Implementierung als Webabwendung

Die *LaTeX*-Implementierung ist gut geeignet, um den Fragebogen vor Ort zu besprechen. Für ein Ausfüllen am Computer ist sie 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. Diese ist Abbildung 3.3 zu entnehmen. 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. Unter 3.4 ist ein kurzes Beispiel aus dem Fragebogen bezüglich Web-Application Penetration Tests aufgezeigt, in welchem einige Fragen definiert werden.

Durch die Verwendung von *Bootstrap* ist die Anwendung automatisch „responsive“, also unter verschiedenen Auflösungen verwendbar. Ein Screenshot der Anwendung ist unter 3.5 dargestellt.

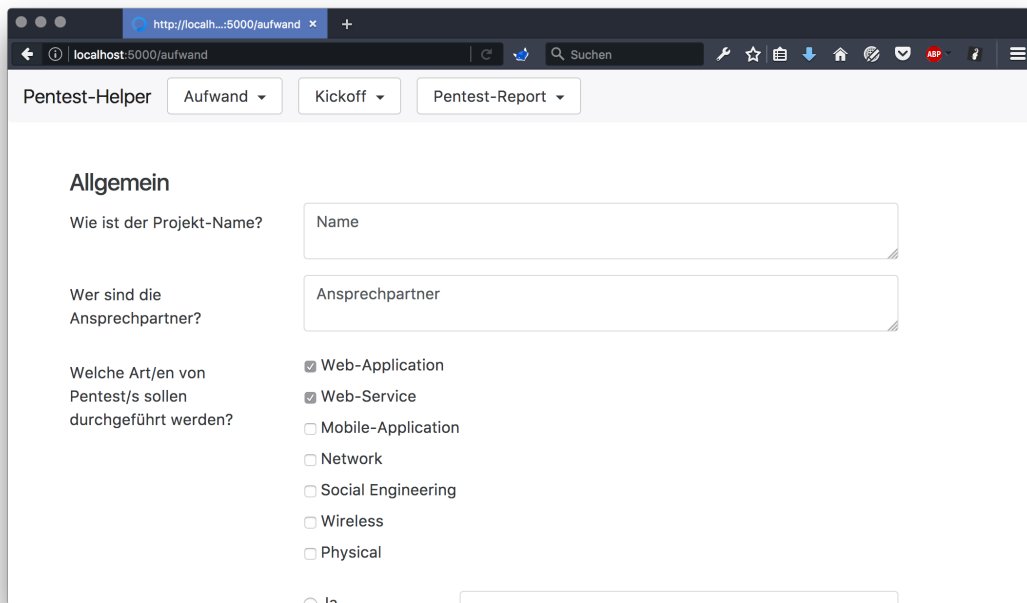
Um die Daten dauerhaft zu speichern, wurde eine Datenbankanbindung implementiert. Hierzu wurde *SQLAlchemy* als Engine genutzt, da diese sehr gut zu *Flask* passt und sich einfach und ohne viel Aufwand implementieren lässt. Mehr zu *SQLAlchemy* ist dem Abschnitt 3.1.1.3 zu entnehmen.

```
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 )
```

Abbildung 3.3.: String-Substitution zum Ersetzen von Inhalten für das TeX-Dokument

```
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?") }}
```

Abbildung 3.4.: Definition von drei Fragen in HTML über Jinja2



The screenshot shows a web browser window with the address bar displaying `http://localhost:5000/aufwand`. The page title is "Pentest-Helfer". Below the title, there are three tabs: "Aufwand", "Kickoff", and "Pentest-Report". The "Allgemein" section is active. It contains the following form elements:

- A text input field labeled "Wie ist der Projekt-Name?" with the placeholder text "Name".
- A text input field labeled "Wer sind die Ansprechpartner?" with the placeholder text "Ansprechpartner".
- A section titled "Welche Art/en von Pentest/s sollen durchgeführt werden?" with a list of checkboxes:
 - ☒ Web-Application
 - ☒ Web-Service
 - ☐ Mobile-Application
 - ☐ Network
 - ☐ Social Engineering
 - ☐ Wireless
 - ☐ Physical

Abbildung 3.5.: Ein kurzer Ausschnitt der Fragebogen-Webanwendung

Zum Erstellen des PDFs muss das generierte TeX-Dokument über *pdflatex* umgewandelt werden. Der dazu genutzte Code ist in Listing 3.6 dargestellt. 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 PDF von *Flask* über die *send_file*-Methode zurückgegeben und im Browser dargestellt. Das Ergebnis ist in Grafik 3.7 dargestellt.

```
1 try:
2     subprocess.check_output(
3         [
4             '/usr/local/texlive/2016/bin/x86_64-darwin/pdflatex',
5             '-synctex=1',
6             '-interaction=nonstopmode',
7             '--output-directory=' + output_path,
8             tex_file_name
9         ], cwd=tex_path
10    )
11 except subprocess.CalledProcessError as error:
12     print(error)
```

Abbildung 3.6.: Aufruf von *pdflatex* aus *python* über das *subprocess*-Modul

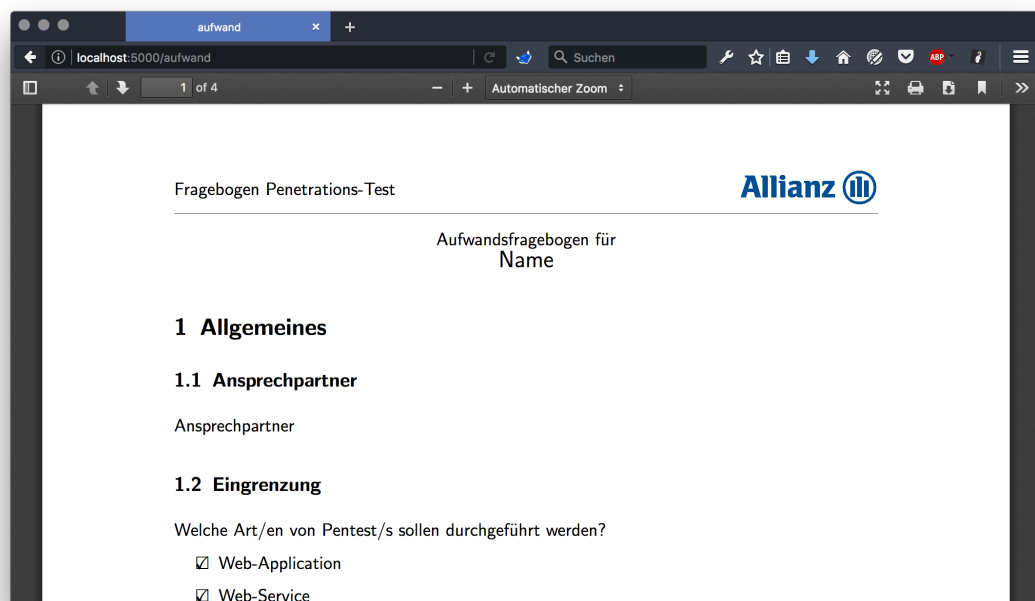


Abbildung 3.7.: Ausschnitt des Fragebogens nach Umwandlung über LaTeX zu PDF

Jinja2

*Jinja2*³ ist eine Template-Engine für *Python*. 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 das unter Listing 3.8 aufgezeigt 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 )
```

Abbildung 3.8.: Aufrufender Python-Code

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

Abbildung 3.9.: Aufgerufenes Template

³<http://jinja.pocoo.org/docs/2.9/>

```
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 }
```

Abbildung 3.10.: Jinja2-Macro zur Erstellung einer Frage mit Checkboxes

In Zeile 7 der Abbildung 3.9 ist der *Jinja2*-Zugriff auf das Attribut *phys_anz_einr* des in Zeile 6 der Abbildung 3.8 übergebenen Objekts *aufwand* zu sehen. Ebenso können Attribute geprüft und Makros implementiert werden. Beides ist im Code-Snippet 3.10 dargestellt.

Dabei wird ein Makro implementiert, welcher ein übergebenes Attribut prüft (Zeile 2, 3.10) und je nach dem eine Checkbox mit einem Hacken (Zeile 3, 3.10) oder eine leere Checkbox (Zeile 5, 3.10) 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. Die Definition des DB-Schemas sowie die Zugriffe auf die Datenbank 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 Beispiel in Listing 3.11 verdeutlicht das Zusammenspiel zwischen *Flask* und *SQLAlchemy*. 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 Möglichkeit getestet, die Applikation, zusammen mit der Anwendung zur Erstellung von Pen-Test-Reports (siehe 3.3.2) und dem Kickoff-Fragebogen (siehe 3.2.1), in einen *Docker*-Container aufzunehmen. Dazu wurde (für den damals aktuellen Stand) ein entsprechendes „Dockerfile“ erstellt, welches im Anhang unter A.1.2 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 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')
```

Abbildung 3.11.: Minimales Beispiel für *SQLAlchemy*

```
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 lassen, welche den Austausch schwierig gestalten würden. Daher wurde vorerst auf eine Portierung auf *Docker* verzichtet.

3.1.2. Rechtliche Aspekte

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

NDA: Im Normalfall wird der Auftraggeber ein NDA (Non-Disclosure-Agreement, zu deutsch Vertraulichkeitserklärung) vom Auftragnehmer einfordern. Dieses verpflichtet Auftragnehmer dazu, keine Informationen über den Pen-Test 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 Pen-Tester 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 Pen-Tests 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 Pen-Test sollten auch verschiedene technische Aspekte beachtet werden. Diese sind im folgenden in Ausstattung, Infrastruktur und Tools aufgeteilt.

3.1.3.1. Ausstattung

Natürlich benötigt ein Pen-Tester eine gewisse Ausrüstung. Dazu sollten auf jeden Fall Rechner mit mittlerer bis hoher 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 einer Cloud-Umgebung durchgeführt werden, da hier innerhalb weniger Minuten Rechner mit einer Vielzahl an GPUs zur Verfügung stehen. Der Nachteil ist natürlich, dass die Hashes dadurch in das Rechenzentrum des Cloud-Betreibers geladen werden müssen.

Sollte ein Vor-Ort-Einsatz geplant sein, könnte ebenso ein Ethernet-Tap ⁴, entsprechend viele LAN-Kabel sowie ein WLAN-Router von Vorteil sein.

Sollen Pen-Tests auf mobile Applikationen durchgeführt werden, sollte für iOS-Apps ein Mac Book zur Verfügung stehen. Zwar können alle mobilen Betriebssysteme emuliert werden, dies ist jedoch zum Teil mit Nachteilen verbunden (siehe Abschnitt 4.3). Daher empfiehlt sich ein physikalisches Endgerät für jeweils *iOS*, *Android* und *Windows-Phone*. Diese sollten jeweils mit einem *Jailbreak*/I ausgestattet sein, um alle Analysen vornehmen zu können.

Bei einem Team von Pen-Tester sollte auf Homogenität bezüglich der verwendeten Hardware geachtet werden.

3.1.3.2. Infrastruktur

Gerade beim Pen-Testen von Web-Applikationen sollte der Pen-Tester 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 Netzwerkverkehr über diesen geleitet 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, über welche Pen-Tester alle notwendigen Informationen austauschen können. Dabei sollte die Ablage so dynamisch wie möglich sein. Eine Möglichkeit wäre *etherpad* ⁵, 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.

⁴<https://greatscottgadgets.com/throwingstar/>

⁵<https://etherpad.org/>

3.1.3.3. Betriebssystem und Tools

Bei Pen-Tests ist es essentiell, dass die Pen-Tester ihr Laptop sehr genau kennen und keine unerwarteten Aktionen auftreten. Ein Beispiel wäre *DHCP*, welches in Endbenutzer-Distributionen wie *Ubuntu* standardmäßig aktiviert ist und direkt beim Einstecken des Netzkabels versucht, eine IP zu beziehen. Bei einem Network-Penetration-Test kann es aber durchaus von Nutzen sein, wenn der Rechner keine sofortige Kommunikation mit dem Netzwerk aufbaut. Daher sollte ein möglichst minimales System gewählt und nur notwendige Software installiert werden. Dazu bieten sich als Betriebssysteme hoch konfigurierbare Linux-Distributionen wie *Gentoo*⁶ oder *Arch-Linux*⁷ 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 Pen-Testern 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 weiterer 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 hohe Anforderung an Verfügbarkeit der Programme im Test und die Vergleichbarkeit zwischen verschiedenen Tests. So will man vermeiden, dass eine Software zum Start eines Pen-Tests wegen zum Beispiel Paketkonflikten nicht mehr funktioniert, oder dass zwei Pen-Tester 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 Pen-Test einen Snapshot erstellen, ein Update durchführen und falls es Probleme gibt, welche sich bis zum nächsten Pen-Test nicht beheben lassen, auf diesen zurückspringen. Nach dem Pen-Test kann dann in Ruhe an dem jeweiligen Problem weitergearbeitet werden. Zudem hat dies den Vorteil, dass man die virtuelle Maschine nach dem Pen-Test löschen kann und so garantiert keine Fragmente des Pen-Test und damit des Kunden im Betriebssystem verbleiben.

Unabhängig davon, ob die Daten des Pen-Tests nur in einer virtuellen Maschine oder auf dem physikalischen Client gehalten werden, sollte das Laptop an sich geschützt sein. Maßnahmen 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 ist auch bei Laptops mit Festplattenverschlüsselung empfohlen, da sonst der Bootloader der Full-Disk-Encryption mit Malware (zum Beispiel über USB) infiziert werden könnte.⁸

⁶<https://www.gentoo.de/>

⁷<https://www.archlinux.de/>

⁸https://www.schneier.com/blog/archives/2009/10/evil_maid_attac.html

3.2. Durchführung

Eine gute Vorbereitung vereinfacht in vielerlei Hinsicht die Durchführung von Pen-Test - beispielsweise insofern, dass bereits funktionierende Hard- und Software zur Verfügung steht und während des Pen-Tests somit keine Zeit auf Update und Konfiguration verwendet werden muss. 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, der Sicherheitsverantwortliche des Projekts, eventuell Informationssicherheit und Hosting-Provider) und Kern-Fragen abgesprochen werden.

Im Rahmen dieser Arbeit wurden die folgenden Fragen definiert und mit Ansprechpartnern der Allianz Deutschland AG geprüft. Diese sollten im Kickoff bearbeitet werden.

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?
- Welche URLs sollten getestet werden?
- Welche Zugangsdaten sollen genutzt werden?
- Sollen Denial-Of-Service-Angriffe durchgeführt werden?
(Ja/Nein)
- Liegt ein Haftungsausschluss 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 3.1.1.3 in eine Webanwendung integriert. Diese ist auf dem Datenträger unter „pentest_helper“ 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 eine non-Profit Organisation, welche es sich zum Ziel gemacht hat, das Schreiben sicherer Software für Unternehmen einfacher zu gestalten.

Im Rahmen des Projekts erfasst die *OWASP* alle 3 Jahre die am meisten aufgetretenen Schwachstellen und veröffentlicht die sogenannte *OWASP TOP 10*. Da die *TOP 10* aus dem Jahr 2016 zu diesem Zeitpunkt noch nicht veröffentlicht sind, ist die aktuellste Version aus dem Jahr 2013. Aus dieser ergeben sich folgende Kategorien^[12]:

- 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.⁹^[6]

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

3.2.3. Bewertung von Findings

Für die in einem Pen-Test 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.

⁹http://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]

3.2.3.1. CVSS

FIRST.Org, Inc., eine in Amerika ansässige wohltätige Organisation, welche Lösungen zur Koordination und Unterstützung von IT-Security-Teams entwickelt, ist sowohl Eigentümer als auch Entwickler des *CVSS* (*Common Vulnerability Scoring System*).

Das *CVSS* bestimmt anhand mehrerer Attribute die Charakteristika und Risiko-Stufen von Schwachstellen. 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.[4]

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 sowie die genauen Formeln können der Spezifikation¹⁰ entnommen werden.[4]

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

Ebenso gibt es *CVSS*-Module für verschiedene Programmiersprachen, zum Beispiel *Python*¹¹. Über diese kann ein *CVSS-Vector-String* ausgewertet und die Scores für die einzelnen Metriken berechnet werden.

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.[13]

¹⁰<https://www.first.org/cvss/specification-document>

¹¹<https://pypi.python.org/pypi/cvss>

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, da angenommen wird, 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 wird 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:[11]

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

Dabei entspricht 0 dem geringsten und 10 dem höchsten Risiko.

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 der *FIRST.Org* beschrieben wird.¹²

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 Schwachstelle wie folgt bewerten:

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

Attack Complexity: Low Auch wenn der Angreifer vor dem Angriff eine kurze Aufklärungsphase erfordert, ist der Angriff aufgrund von Standard-Datenbanknamen einfach durchführbar.

Privileges Required: None Ein Angreifer braucht keine besonderen Berechtigungen.

User Interaction: Required Damit der Angriff erfolgreich ist, muss das Opfer eine Aktion ausführen, zum Beispiel das Klicken auf einen manipulierten Link.

Scope: Changed Es gibt eine Veränderung im *Scope*, da die Lücke zwar in der Webapplikation ist, aber der Angriff im Browser des Users Wirkung zeigt.

Confidentiality Impact: Low Informationen aus dem Browser des Opfers können abgegriffen und an den Angreifer geschickt werden. Da aber die Cookies aufgrund der *HTTP-Only*-Flag ausgeschlossen sind, ist der *Impact* nur niedrig.

Integrity Impact: Low Es können zwar Informationen im Web-Browser des Opfers verändert werden, jedoch nur solche, welche in Verbindung mit der verwundbaren Webapplikation stehen.

Availability Impact: None Zwar könnte über bösartigen Code der Browser des Users verlangsamt werden, jedoch kann der User den Browser jederzeit beenden.

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.

DREAD

Unter *DREAD* gibt es leider keine Beispiele für eine *Reflected-Cross-Site-Scripting* Lücke. Daher wurden die Werte durch den Autor 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.

¹²<https://www.first.org/cvss/examples>

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 Da der Angriff auf alle User anwendbar ist, sind auch alle User 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 Pen-Tests eingesetzt werden, damit die Findings vergleichbar bleiben.

3.2.4. Dokumentation

Ein weiterer wichtiger Punkt während der Durchführung eines Pen-Tests ist die fortlaufende Dokumentation. So sollten sowohl Zwischenergebnisse 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 Pen-Tests ist das Aufzeichnen des Netzwerkverkehrs. So können jegliche am Server passierenden Aktionen später über den Netzwerkdump einer Aktion des Pen-Testers 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*¹³ wie auch *TCPDump*¹⁴ den gesamten Verkehr eines oder mehrerer Netzwerkadapter aufzeichnen und als *PCAP*¹⁵ speichern.

¹³<https://www.wireshark.org/>

¹⁴<http://www.tcpdump.org/>

¹⁵<http://xml2rfc.tools.ietf.org/cgi-bin/xml2rfc.cgi?url=https://raw.githubusercontent.com/pcapng/pcapng/master/draft-tuexen-opsawg-pcapng.xml&modeAsFormat=txt/pdf&type=ascii>

Eine Problemstellung für den Pen-Test stellt eine geschützte *TLS/SSL*-Verbindung dar. Da normalerweise der *Private-Key* des Zielsystems nicht zur Verfügung steht, muss die Verbindung unterbrochen werden. Dies kann über *MITMProxy*¹⁶ bewerkstelligt werden, welche einen Proxy-Server aufbaut. Dieser Proxy-Server baut zum Ziel eine *TLS*-Verbindung auf, stellt jedoch zum Pen-Tester eine separate *TLS*-Verbindung auf, für welche ein privates Zertifikat hinterlegt werden kann. Zeichnet man nun den Netzwerkverkehr zwischen dem Pen-Tester und *MITMProxy* auf, können die *TLS*-Verbindungen später mit dem privaten Zertifikat (zum Beispiel in *Wireshark*) entschlüsselt und gesichtet werden.

3.2.4.2. Konsoleneingaben

Zusätzlich zu Netzwerkdumps kann eine Aufzeichnung der Terminal-Sessions während des Pen-Tests 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 aufgenommen werden. Um das Transkript zu erstellen, kann das Linux-Tool *screen*¹⁷ 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 ohne zeitliche Verzögerung dokumentiert werden. Dazu sollte eine standardisierte Form gefunden werden. Wie diese genau aussieht, kann den Pen-Testern 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*¹⁸ genommen werden.

¹⁶<https://mitmproxy.org/>

¹⁷<http://man7.org/linux/man-pages/man1/script.1.html>

¹⁸<https://www.imagemagick.org/script/import.php>

3.3. Nachbereitung

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

3.3.1. Inhalte eines Pen-Test-Berichts

Im Folgenden werden die typischen Inhalte eines Pen-Test-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“ enthält 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

Das „Management Summary“ ist, wie der Name schon sagt, als Zusammenfassung für das Management gedacht. Es hat jedoch auch die Funktion, dem Leser kurz die kritischen Erkenntnisse des Pen-Tests zu vermitteln. Dazu ist das 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 dem Leser einen ersten Eindruck zu vermitteln. Dabei kann die Aufmerksamkeit auf besonders kritische Findings gelenkt werden.

Risikomatrix Die Risikomatrix stellt die Findings 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 das „Management Summary“ einen kurzen Überblick geben, bezieht jedoch bereits technische Aspekte mit ein und ist ausführlicher als das „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 Pen-Tester 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 Datensatz 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 sämtliche die Beschreibung ergänzenden Screenshots oder Text-Stücke (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 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, mit mehrere Tage für die Erstellung des Report gerechnet werden muss.

Um diesen Aufwand zu minimieren, wurde auch hier eine Web-Anwendung entwickelt. Diese baut auf die gleiche Technologie auf wie die Anwendung, 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 übergeben.

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. Einige Beispiele 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 wurden jeweils Name, Kategorie, Beschreibung, Empfehlung, Schadenspotenzial, Eintrittswahrscheinlichkeit, Gesamtrisiko sowie einen Standard-*CVSS3-Score* vorbereitend bereits ausgefüllt. Der User kann die Texte und Metriken in der Web-Oberfläche anpassen und Belege hinzufügen.

3.4. Kontinuität

Ein weiterer 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 Angriffsmöglichkeiten kann sich das Sicherheitsniveau ständig ändern.

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

Natürlich sollte dafür festgelegt werden, wie die Kritikalität und das Sicherheitsniveau bestimmt werden. Die Kritikalität pro Anwendung ist meist abhängig vom Unternehmen. So würde für einen Automobilhersteller wohl eine Produktionsanlage als hoch kritisch gesehen werden, bei einem Online-Vertrieb wohl eher der Online-Shop. Auch sollte die Angriffsfläche betrachtet werden. So kann man argumentieren, dass eine auf einem abgeschotteten Rechner laufende Anwendung mit hoher Kritikalität eventuell aufgrund der geringen Angriffsfläche mit einem niedrigeren Sicherheitsniveau wie normal betrieben werden darf. Ein weiteres Beispiel wäre eine Anwendung, welche lediglich im Intranet läuft und nicht aus dem Internet erreichbar ist. Für diese Anwendung könnte, aufgrund der geringeren Angriffsfläche, ein geringeres Sicherheitsniveau gefordert werden.

Auch das Sicherheitsniveau selbst sowie Maßnahmen zu dessen Aufrechterhaltung müssen im Unternehmen definiert werden. Dabei sollte zuerst die Maßnahmen definiert werden.

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

Code-Audits sind von Personen ausgeführte Analysen des Quellcodes auf Schwachstellen.

Da 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 Pen-Tests, sind aber ebenfalls eine sehr effektive Maßnahme.

Security-Scans meint automatisierte Sicherheits-Scans durch Software wie *IBM Security AppScan*¹⁹ oder der *Nessus Vulnerability Scanner*²⁰. Diese erkennen bekannte Schwachstellen und sind gut automatisierbar. Security-Scanner sind nicht so genau wie Pen-Tests, dafür sind diese, ab einer gewissen Anzahl von Anwendungen, wesentlich günstiger.

Sourcecode-Scans sind automatisierte Analysen auf Basis des Quellcodes der Anwendung. Produkte wären beispielsweise *Fortify Static Code Analyzer*²¹ oder *Veracode Static Analysis*²². Sourcecode-Scans sind ähnlich effektiv wie Security-Scanner, bedürfen aber, abhängig von der Komplexität der Anwendung, einem größeren Einrichtungsaufwand.

¹⁹<http://www-03.ibm.com/software/products/de/appscan>

²⁰<https://www.tenable.com/products/nessus-vulnerability-scanner>

²¹<http://www8.hp.com/de/de/software-solutions/static-code-analysis-sast/>

²²<https://www.veracode.com/products/binary-static-analysis-sast>

Hat ein Unternehmen die passenden Maßnahmen aufgebaut, sollten diese über eine zeitliche Einteilung der Maßnahmen zum Sicherheitsniveau vorgenommen werden. Ein frei gewähltes Beispiel dafür ist in Tabelle 3.2 zu sehen. Dabei ist „Jahr 1“ das erste Jahr, nachdem die Anwendung (nach einem Pen-Test) online genommen wurde.

gef. Sicherheitsniveau	Zeit				
		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

4. Penetration-Tests mobiler Anwendungen

Durch die zunehmende Nachfrage des Marktes nach mobilen Anwendungen entwickelt die Allianz Deutschland AG zunehmend mobile Applikationen. Dafür müssen nicht nur die bestehenden Prozesse angepasst, sondern auch neue Werkzeuge zur Unterstützung der Security-Prozesse geschaffen werden.

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

4.1. Anforderungen

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

Einfache Einrichtung Das Tool sollte 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 im Bereich Informationssicherheit zuvor ungeschulten oder nur rudimentär geschulten Mitarbeiter durchgeführt werden können. Dazu muss das Tool einfach zu bedienen sein.

Verständliche Ergebnisse Es sollte eine Übersicht geben, die auch einen Mitarbeiter ohne Erfahrungen im Bereich Informationssicherheit eine erste Einschätzung ermöglicht. Die genauen Ergebnisse des Werkzeugs sollten für einen in der Informationssicherheit arbeitenden Angestellten verständlich sein. Hier kann eine gewisse Fachkenntnis vorausgesetzt werden. 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 gleichem Eingangswert das gleiche Ergebnis generieren. Dies ist für die Nachverfolgung von Meldungen äußerst 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-Frameworks* 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 auf Open-Source Basis. Diese Plattform dient 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 auf *Github* frei zur Verfügung.¹ Die aktuelle Version ist *0.9.4 beta*, wobei teilweise mehrmals pro Woche Code-Änderungen vorgenommen werden.

MobSF 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.

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 ein Java Decompiler für Java 5 und spätere Versionen. Er steht zum Download² zur Verfügung und kann zum Beispiel über das auf derselben Seite zur Verfügung gestellte *JD-GUI* genutzt werden.

Dex2Jar (d2j) ist ein Tool zum Umwandeln von *.dex*-Dateien (Dalvik-Bytecode) zu normalem Java-Bytecode (gepackt in einem *Jar*-File). Anschließend können normale Java-Tools zur Analyse verwendet werden. Das Tool ist kostenlos, Open-Source und in Github³ verfügbar.

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

²<http://jd.benow.ca/>

³<https://github.com/pxb1988/dex2jar>

enjarify ist eine modernere Alternative zu *Dex2Jar*. *enjarify* wurde von Google entwickelt, ist jedoch trotzdem unter der Apache-Lizenz in Github veröffentlicht⁴.

Dex2Smali 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 Github⁵ zu finden.

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 Bitbucket⁶ 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 brauche Funktionen wie die Auflistung der *shared libraries* oder der „indirect symbol table“.

Für Windows-Phone-Apps:

BinScope ist ein Security-Analyse-Tool für Windows-Applikationen. Es wurde von Microsoft für den *Secure Development Lifecycle* entwickelt und steht auf der Microsoft-Webseite⁷ zur Verfügung. Eine genauere Beschreibung ist dem Abschnitt 4.6.2.5 zu entnehmen.

BinSkim ist der Nachfolger von *BinScope*. 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*⁸ bezogen werden. Eine genauere Beschreibung ist dem Abschnitt 4.6.2.5 zu entnehmen.

⁴<https://github.com/google/enjarify>

⁵<https://github.com/JesusFreke/smali>

⁶<https://bitbucket.org/mstrobel/procyon/overview>

⁷<https://blogs.microsoft.com/microsoftsecure/2012/08/15/microsofts-free-security-tools-binscope-binary-analyzer/>

⁸<https://www.nuget.org/packages/Microsoft.CodeAnalysis.BinSkim/>

4.3. Vergleich der Emulationsumgebungen

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 der in *Xcode* enthaltene, offizielle *iOS*-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 OS X* 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
```

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 License⁹, 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*-Webseite¹⁰ 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.

⁹<https://opensource.org/licenses/UoI-NCSA.php>

¹⁰<http://lldb.llvm.org/lldb-gdb.html>

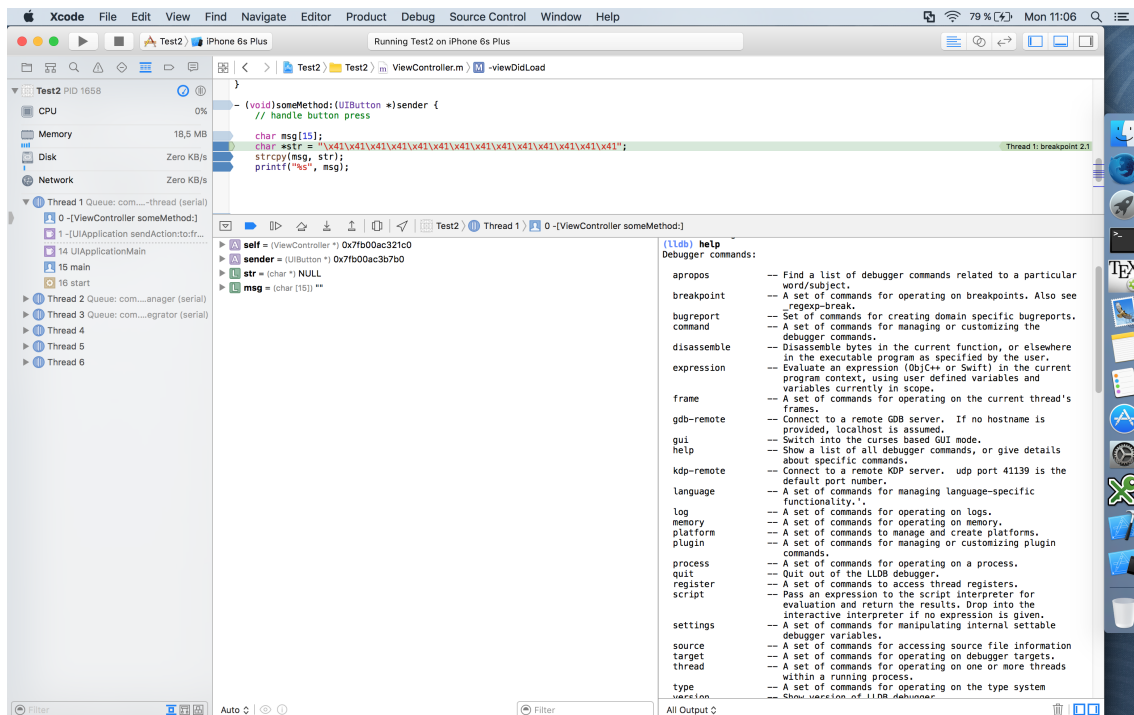


Abbildung 4.1.: LLDB in XCode

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

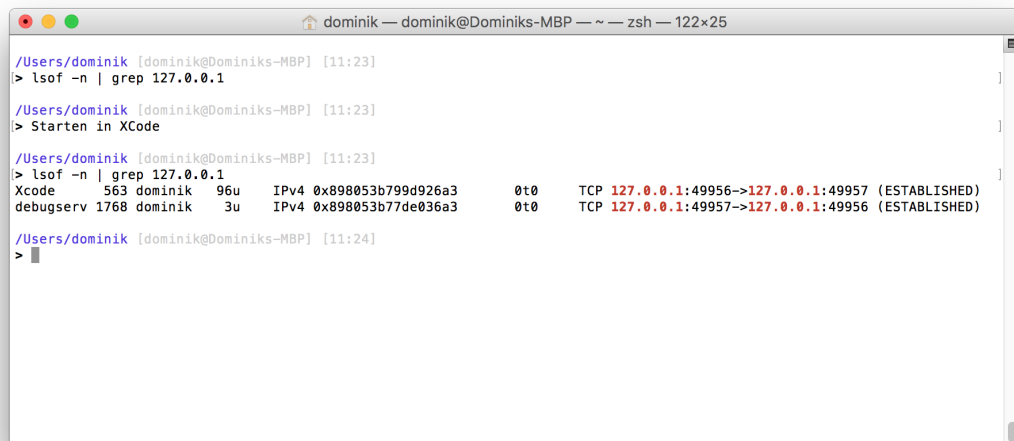
Leider ist nicht erkennbar, wie *LLDB* und das emulierte *iPhone* eine Verbindung herstellen. Eine Auflistung der offenen Sockets auf dem System legt jedoch nahe, dass auf dem *iPhone* das Programm *debugserver* gestartet wird, welches Remote-Debugging mit *LLDB* erlaubt. Es bleibt herauszufinden, wie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

Nach einem Artikel von Apple¹¹ 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 4.5) oder die geladenen Bibliotheken der Programme (siehe Abbildung 4.4) verglichen werden.

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

¹¹https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/gdb_to_lldb_transition_guide/document/lldb-terminal-workflow-tutorial.html



```
/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  0x898053b777de036a3    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

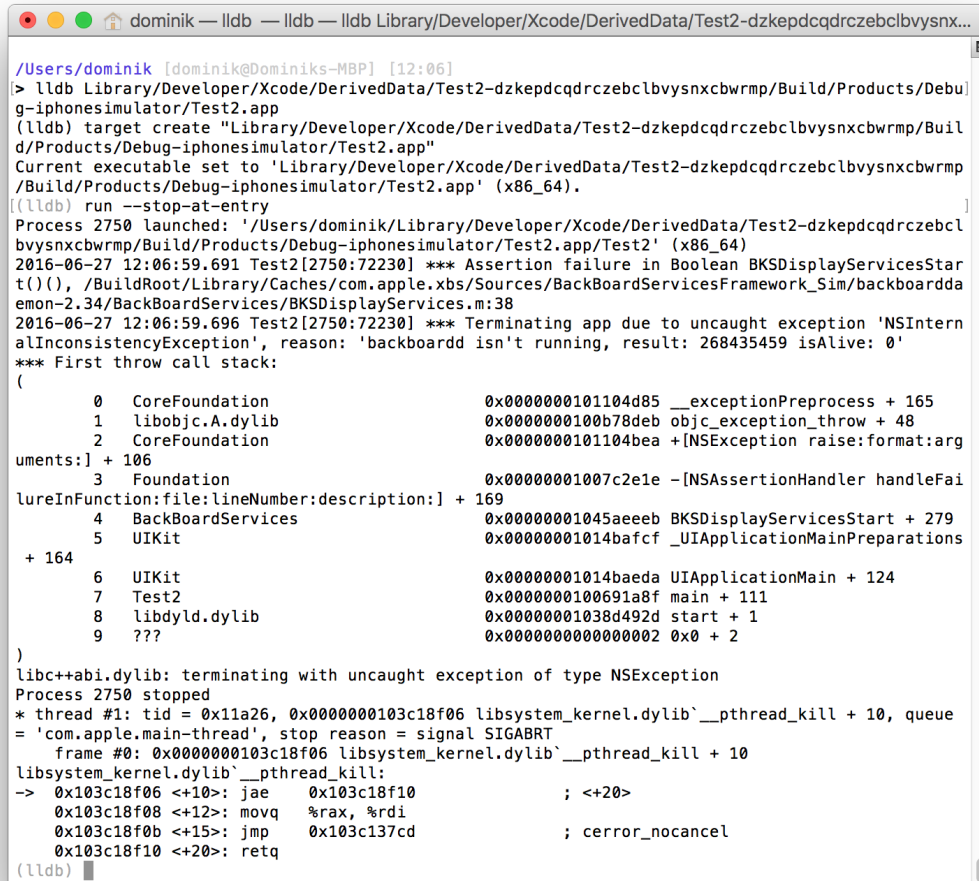
Auch der Vergleich der geladenen Bibliotheken legt nahe, dass die *LLDB* Standalone und *Xcode* in der gleichen Umgebung ausgeführt werden. Die Adressen im RAM variieren aufgrund von *ASLR* zwar, aber es werden dieselben Bibliotheken verwendet (am Pfad zu erkennen). Dies ist in Grafik 4.4 dargestellt.

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



```

/Users/dominik [dominik@Dominiks-MBP] [12:06]
> lldb Library/Developer/Xcode/DerivedData/Test2-dzkepdqdrceblbvynxcbwrmp/Build/Products/Debug-iphonesimulator/Test2.app
(lldb) target create "Library/Developer/Xcode/DerivedData/Test2-dzkepdqdrceblbvynxcbwrmp/Build/Products/Debug-iphonesimulator/Test2.app"
Current executable set to 'Library/Developer/Xcode/DerivedData/Test2-dzkepdqdrceblbvynxcbwrmp/Build/Products/Debug-iphonesimulator/Test2.app' (x86_64).
(lldb) run --stop-at-entry
Process 2750 launched: '/Users/dominik/Library/Developer/Xcode/DerivedData/Test2-dzkepdqdrceblbvynxcbwrmp/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 +[NSException raise:format:arguments:] + 106
    3  Foundation               0x00000001007c2e1e -[NSAssertionHandler handleFailureInFunction:file:lineNumber:description:] + 169
    4  BackBoardServices        0x00000001045aeceb 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 NSException
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

```
[*] ntlb) image list
[0] 46A80E58-D086-3461-8C99-40939ACFC63 @0x00007fff69641000 /usr/lib/dyld
[1] 49268249-F1CD-35FC-BFFD-BA8BF3751800 @0x000000103650000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/dyld_slim
[2] 57D299AA-9342-3433-3388-0808A0747491 @0x000000103650000 /Users/dominik/Library/Developer/Xcode/DerivedData/Test2-dkpedcdrczbebvsnxcwrbp/Build/Products/Debug-iPhoneSimulator/Test2.app/Test2
[3] F05DA171-3A9F-3D70-9102-6E81198B76F @0x000000102be3000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/Foundation.framework/Foundation
[4] EE88F8F6-2E97-3EC0-A2D3-AED04C2C5939 @0x000000103b34000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libobjc.A.dylib
[5] 2A3E2E2C-3A6A-32B8-BF68-BF16-888B3780F43C @0x000000103711000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libSystem.dylib
[6] 1986E624-8583-397E-BC5C-15F8ED508794 @0x0000001037f8000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation
[7] 628B9919-66FF-348E-A439-E8BF04D7205F @0x000000104455000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/UIKit.framework/UIKit
[8] 7F2DECDB-DC65-3AEA-AC6A-BF27B982446E @0x0000001059F2000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/MobileCoreServices.framework/MobileCoreServices
[9] 3F84A6E8-2C9C-33CA-9CE4-615AC83022F @0x000000108552000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libtextbundle.dylib
[10] D19C5926-2C8A-39C1-AD02-09234A95158A @0x000000105910000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libarchive.2.dylib
[11] 87D20B99-7886-3080-A065-A13547888480 @0x0000001050e0000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libcubicore.A.dylib
[12] 1ED4A821-09F2-3133-8C84-9A91879BF91A @0x000000105362000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/lib/libc.2.dylib
[13] C8B7F97B-C316-37E9-8504-C813201A6123 @0x000000105470000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks/CFNetwork.framework/CFNetwork
[14] 87345A8B-325E-8658-056036ACEB16 @0x000000105595000 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/
```

Abbildung 4.4.: Vergleich der geladenen Bibliotheken

Unsichere 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]

Dies ist auch in der Apple-Online-Dokumentation beschrieben¹².

Ungesicherte Verbindungen

Ein Sicherheitsfeature ab *iOS 9.0* ist der Umgang von *iOS* mit Netzwerk-Verbindungen. So können ohne erweiterte Konfiguration keine Verbindungen aufgebaut werden, welche nicht dem RFC-Standard 2818¹³ folgen. So führt der in Listing 4.6 gezeigte Aufruf einer HTTP-Seite zu der in 4.7 gezeigten Fehlermeldung.

Sollten unsichere 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] den in 4.8 dargestellten Inhalt haben. Eine Überprüfung auf solche Ausnahmen kann dem Abschnitt 4.6.3.3 entnommen werden.

¹²<https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html>

¹³<https://tools.ietf.org/html/rfc2818>

```

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=.bzip --
   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

```

Abbildung 4.5.: Geöffnete Prozesse vor und während der Ausführung des Emulators

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

Abbildung 4.6.: Aufbau einer nicht mit TLS gesicherten Verbindung

```
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.
```

Abbildung 4.7.: Fehler bei Aufbau einer ungesicherten Verbindung ohne Definition einer Ausnahme

```
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 }
```

Abbildung 4.8.: XML-Definition eines einer Ausnahme der ATS

4.3.2. Windows-Phone

Im Folgenden wurde der in *Visual Studio* enthaltene Emulator für *Windows-Phone*-Apps in seiner Funktionsweise untersucht.

4.3.2.1. Emulation

Zur Emulation von *Windows-Phones* ist *Visual Studio* notwendig. Im Folgenden wurde *Visual Studio 15* in der *Community Edition*, also einer kostenlosen Version, verwendet.

Nach der Installation von *Visual Studio* sollte das Framework für die *Universal Windows Platform* sowie der Emulator installiert werden. Beide können als Optionen bei der Installation gewählt werden. Anschließend kann eine App sowohl im Emulator, als auch lokal auf der Windows-Maschine gestartet werden. Alternativ kann auch ein physikalisches Windows-Gerät für die Ausführung von Apps genutzt werden.

4.3.2.2. Debugging

Debugging kann direkt über *Visual Studio* durchgeführt werden. Dazu können einfach die jeweiligen Breakpoints im Code gesetzt werden.

Eine Möglichkeit zum Debugging außerhalb von *Visual Studio* wurde nicht gefunden, da Microsoft keine öffentlich beschriebene Schnittstelle zur Kommunikation mit Apps innerhalb des Betriebssystems zur Verfügung stellt.

4.3.2.3. Unsichere Funktionen

Unsichere Funktionen wie „strcpy“ oder „memcpy“ sind in den Bibliotheken für *C++*-Apps der *Universal Windows Platform* noch enthalten, können aber ohne erweiterbare Konfiguration nicht genutzt werden.

Um diese Funktionen zu nutzen, muss dem Compiler die Flag

```
1 _CRT_SECURE_NO_WARNINGS
```

übergeben werden. Anschließend können auch Funktionen, welche zu *Memory-Corruption* führen können, frei genutzt werden.

4.3.2.4. Unsichere Verbindungen

Bezüglich unsicherer Verbindungen schreibt die *Universal Windows Platform* nichts vor. So kann über folgenden Aufruf eine Seite ohne TLS aufgerufen werden.

```
1 auto uri = ref new Windows::Foundation::Uri("http://api.ipify.  
    org");  
2 webView->Navigate(uri);
```

4.3.3. Android

Android ist ein ursprünglich 2003 von der *Android, Inc.* entwickeltes mobiles Betriebssystem. 2005 wurde es durch Google übernommen und wird seitdem weiterentwickelt. 2015 liegt es in der EU bei einem Marktanteil von 75,6%^[10]. Aufgrund der Quelloffenheit des Systems wird es von vielen Herstellern auf verschiedensten Plattformen genutzt. Jedoch bringt die weitführende Fragmentierung des Betriebssystems auch Nachteile mit sich. So sind im Februar 2017 nur 1,2% der Android-Devices auf einer aktuellen Version (Nougat).^{[14][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 von der offiziellen Webseite¹⁴ 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 zum Teil im weiteren Verlauf noch detailliert eingegangen wird.

4.3.3.2. Emulation vs. Hardware

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 4.9 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*.^[7]

4.3.3.3. Debugging

Zum Debugging unter Android kann die *Android Debug Bridge*, kurz *ADB*, genutzt werden. Dabei bietet *ADB* nicht nur die Funktionen für emulierte Geräte, sondern auch für physikalische. Zusätzlich kann für physikalische Geräte sogar Debugging über das Netzwerk durchgeführt werden. Für kabelgebundene oder emulierte Handys können mit dem Aufruf *adb devices* die dem Rechner zur Verfügung stehenden Geräte abgerufen werden. Anschließend können über *adb shell* beliebige Kommandos, inklusive des Aufrufs des internen Debuggers, gegeben werden.^[1]

Eine App kann über das Kommando

¹⁴<http://developer.android.com/sdk/index.html>

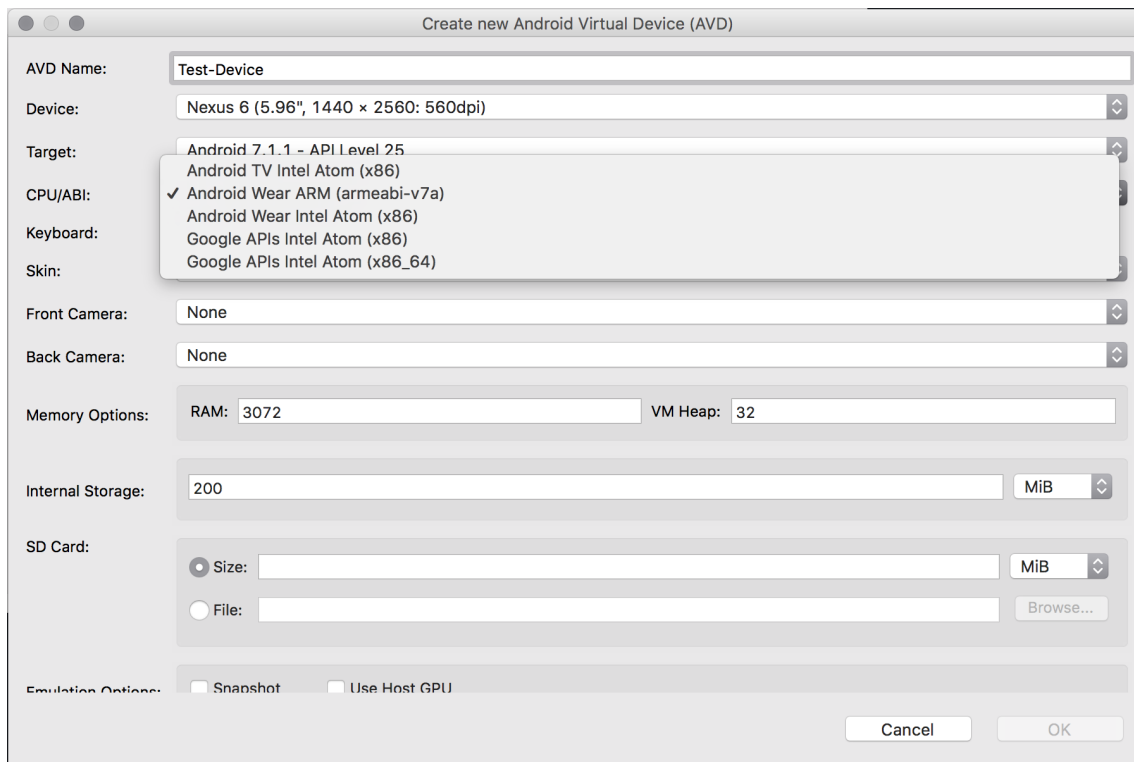


Abbildung 4.9.: Der Konfigurations-Bildschirm des AVD-Managers

```
1 am start -n <package identifier>/.<activity>
```

gestartet werden. Dazu muss jedoch die Start-Aktivität der App in Erfahrung gebracht werden. Dies ist über

```
1 cmd package resolve-activity --brief <package identifier>
```

möglich. Um eine App im Debug-Modus zu starten, wird *am* einfach der Parameter *-D* angehängt.

```
1 am start -n -D <package identifier>/.<activity>
```

Allerdings bleibt anzumerken, dass die Debug-Flag für die meisten Apps nicht gesetzt ist. Um dies, insofern notwendig, zu umgehen, muss *am* als privilegierter User ausgeführt werden. Damit dies möglich ist, muss das Smartphone „gerootet“, also der Root-Account aktiviert, sein. Alternativ kann auch das APK vom Telefon geladen, die Debug-Flag im *AndroidManifest.xml* ergänzen und wieder am Handy installiert werden.

Anschließend kann die App direkt über *jdb*¹⁵ oder eine IDE wie IntelliJ gedebugged werden. Im Detail ist dies zum Beispiel im Blog¹⁶ von Eric Gruber beschreiben.

¹⁵<https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdb.html>

¹⁶<https://blog.netspi.com/attacking-android-applications-with-debuggers/>

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, bestehen jedoch grundsätzlich immer aus der Installation von Python sowie der 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 ist über Farb-Codes 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 *MobSF* unterstütze das Framework die statische und dynamische Analyse von *Android*-Apps sowie die statische Analyse von *iOS*-Apps. Eine Analyse von *Windows-Phone*-Apps ist noch nicht enthalten.

Niedrige False-Positive-Rate *MobSF* beschreibt nicht direkt Schwachstellen, sondern Indikatoren wie beispielsweise die Verwendung von verwundbaren Funktionen. Daher sind die gegebenen Warnungen stets korrekt, jedoch wird womöglich auch in manchen Bereichen nicht alles entdeckt.

Reproduzierbarkeit *MobSF* benutzt zumeist Tools, welche in *MobSF* selbst enthalten sind. Daher sollte dieselbe Version von *MobSF* 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 *MobSF* 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* mit *Pylint* für Python 2.7/3.6 oder *VIM* genutzt.

4.6. Weiterentwicklung MobSF

Ein Kernelement dieser Arbeit ist die Weiterentwicklung des Mobile Security Frameworks *MobSF*. Die Funktionen des Frameworks sind bereits unter Abschnitt 4.4 aufgezeigt. Im Folgenden sind die Änderungen dargestellt, welche an dem Framework vorgenommen und veröffentlicht wurden.

4.6.1. Allgemeine Verbesserungen

Neben Verbesserungen, welche einem genauen Bereich (*Windows-Phone*, *iOS*, *Android*) zuzuordnen sind, gibt es auch einige allgemeine Erweiterungen am *MobSF*. Diese sind ebenfalls 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 4.10 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.

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

Abbildung 4.10.: Struktur des MobSF auf der ersten Ebene

Um hier eine klarere Trennung zu schaffen, wurde die *views.py* in einem ersten Schritt 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 alten Struktur nicht auftauchen.

Dies ist im Detail in der Abbildung 4.11 dargestellt.

Im weiteren Verlauf der Weiterentwicklung und mit der Einführung von Code-Standards (siehe 4.6.1.2) wurde die Struktur weiter verfeinert. So wurden die Datei „android.py“ entsprechend der Funktionalitäten weiter aufgeteilt. So ergibt sich für die statische Analyse von Android-Apps die in 4.12 dargestellte Dateistruktur.

```

StaticAnalyzer/
├── [...]
├── views.py
│   ├── key
│   ├── PDF
│   ├── Java
│   ├── Smali
│   ├── Find
│   ├── ViewSource
│   ├── ManifestView
│   ├── StaticAnalyzer
│   ├── GetHardcodedCertKeystore
│   ├── ReadManifest
│   ├── GetManifest
│   ├── ValidAndroidZip
│   ├── HashGen
│   ├── FileSize
│   ├── GenDownloads
│   ├── zipdir
│   ├── Unzip
│   ├── FormatPermissions
│   ├── CertInfo
│   ├── WinFixJava
│   ├── WinFixPython3
│   ├── Dex2Jar
│   ├── Dex2Smali
│   ├── Jar2Java
│   ├── Strings
│   ├── ManifestData
│   ├── ManifestAnalysis
│   ├── CodeAnalysis
│   ├── StaticAnalyzer_iOS
│   ├── ViewFile
│   ├── readBinXML
│   ├── HandleSqlite
│   ├── iOS_ListFiles
│   ├── BinaryAnalysis
│   └── iOS_Source_Analysis

```

(a) Alte Struktur

```

StaticAnalyzer/
├── [...]
├── views/
│   ├── android.py
│   │   ├── [...]
│   │   ├── GetHardcodedCertKeystore
│   │   ├── ReadManifest
│   │   ├── GetManifest
│   │   ├── ValidAndroidZip
│   │   ├── Dex2Jar
│   │   ├── Dex2Smali
│   │   ├── Jar2Java
│   │   └── [...]
│   ├── ios.py
│   │   ├── StaticAnalyzer_iOS
│   │   ├── ViewFile
│   │   ├── readBinXML
│   │   ├── HandleSqlite
│   │   ├── iOS_ListFiles
│   │   ├── BinaryAnalysis
│   │   └── iOS_Source_Analysis
│   └── windows.py
│       ├── [...]
│       ├── __binskim
│       ├── __binscope
│       └── [...]
└── shared_func.py
    ├── key
    ├── FileSize
    ├── HashGen
    ├── Unzip
    └── PDF

```

(b) Neue Struktur

Abbildung 4.11.: Vergleich der Struktur von *StaticAnalyzer*

```
Mobile-Security-Framework-MobSF/  
├── [...]  
├── StaticAnalyzer/  
│   ├── [...]  
│   └── views/  
│       ├── __init__.py  
│       ├── ios.py  
│       ├── shared_func.py  
│       ├── windows.py  
│       └── android/  
│           ├── __init__.py  
│           ├── binary_analysis.py  
│           ├── cert_analysis.py  
│           ├── code_analysis.py  
│           ├── converter.py  
│           ├── db_interaction.py  
│           ├── dvm_permissions.py  
│           ├── find.py  
│           ├── java.py  
│           ├── manifest_analysis.py  
│           ├── manifest_view.py  
│           ├── smali.py  
│           ├── static_analyzer.py  
│           ├── strings.py  
│           ├── view_source.py  
│           └── win_fixes.py
```

Abbildung 4.12.: Struktur der Dateien für die statische Analyse

4.6.1.2. Code-Standards

Um eine hohe Code-Qualität zu gewährleisten, sollte homogener und gut wartbarer Code geschrieben werden. Um dies über mehrere Entwickler hinweg zu gewährleisten, gibt es sogenannte *Code-Standards*. Diese werden meist von einer zentralen Stelle festgelegt und beschreiben, zum Beispiel wie Variablen benannt werden müssen, wie viele Kommentare nötig sind und wie lang eine Funktion maximal sein darf.

Bei der Neu- oder Reimplementierung wurde auf die Verwendung von offiziellen *Code-Standards* geachtet. Insbesondere wurde der *PEP 8* Standard¹⁷ 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 in moderne Entwicklungsumgebungen wie *Atom* Tool wie *Pylint* direkt eingebunden und genutzt werden. Dies hat den Vorteil, dass bereits während des Programmierens Verstöße gegen Standards 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* auf 2000 Code-Zeilen 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 Weiterentwicklung 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 jede auch noch so kleine Änderung in der übergreifenden Datei mit Änderungen einer parallel arbeitenden Partei zusammengeführt werden, was oftmals viel Arbeit bedeutet.

4.6.1.3. strings

Zuerst wurde das *MobSF* um die Fähigkeit erweitert, eine *iOS*-Applikation mit dem *strings*-Programm zu untersuchen. *strings* durchsucht, sofern keine zusätzlichen Parameter übergeben werden, eine binäre Datei auf *ASCII*-Elemente mit mindestens vier Stellen und gibt diese anschließend zurück.

Dies hilft oft bei einer ersten Einschätzung der Anwendung, da zumeist eine grundlegende Funktionsweise und der Zweck der Software abgeleitet werden können. Ebenso können eventuell unbeabsichtigt im Programm vergessene oder eigentlich geheime Strings in einer App aufgedeckt werden, wie zum Beispiel Entwicklerkommentare oder Passwörter. Auch sind je nach Compiler-Einstellungen Funktionsnamen oder Calls in andere Bereiche (z.B. Import-Table bei PE-Files) als String in einem Binary enthalten, was unter Abschnitt

¹⁷<https://www.python.org/dev/peps/pep-0008/>

4.6.2.4 zum Entdecken von verwundbaren Funktionen genutzt wird.

Sowohl *Mac OS X* wie auch *Linux* haben ein integriertes *string*-Kommando, welches jedoch *Windows* fehlt. Um die Multi-Platform-Fähigkeit weiterhin zu gewährleisten, wurde die Funktion in Python-Code abgebildet. Als Vorlage wurde ein bestehender Code von *Stackoverflow*¹⁸ genutzt. Dieser lieferte jedoch eine wesentlich höhere Anzahl von Ergebnissen, da bestimmte Whitespace-Character ebenfalls beachtet wurden (siehe Abbildung 4.13).

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

Abbildung 4.13.: Vergleich der gefundenen Strings mit Python und dem ursprünglichen Strings-Kommando

Da viele der zusätzlich aufgedeckten Strings jedoch nicht bei der Analyse geholfen, sondern eher das Auffinden relevanter Strings erschwert haben, wurde der Originalcode wie in Abbildung 4.14 dargestellt angepasst. Durch die Reduzierung der ausschlaggebenden Zeichen konnten die Ergebnisse optimiert werden, sodass eine effiziente Suche über Strings wieder möglich ist.

4.6.1.4. PDF-Generation

Um eine effiziente Weitergabe der Ergebnisse zu ermöglichen, besitzt *MobSEF* 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. Aufgrund von Leistungsproblemen des Moduls wurde später auf *pdfkit*¹⁹ gewechselt.

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

¹⁸<http://stackoverflow.com/a/17197027>

¹⁹<https://pypi.python.org/pypi/pdfkit>

```
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         for c in f.read():
12             <<<<<< Vor Anpassung
13             if c in string.printable:
14                 =====
15                 if c in (
16                     '0123456789'
17                     'abcdefghijklmnopqrstuvwxyz'
18                     'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
19                     '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
20                 ):
21                     >>>>>> Nach Anpassung
22                     result += c
23                     continue
24                     if len(result) >= min and result[0].isalnum():
25                         yield "'" + result + "'"
26                     result = ""
```

Abbildung 4.14.: Anpassungen am Code der Python-Implementierung von Strings zur Verbesserung der Ergebnisse

```
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 4.15.: `rpc_client.py`

4.6.1.5. 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-Anleitung²⁰, ist in Abbildung 4.15 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.

²⁰<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 4.16.: `rpc_server.py`

Requests

Requests ist ein Python-Modul, welches einfache Anfragen (sogenannte *Requests*) über HTTP(S) ermöglicht. So ist es über ein kurzes Code-Snippet, dargestellt in Abbildung 4.16, möglich, die unter 4.15 aufgezeigt Schnittstelle anzusprechen.

Wird zuerst der Code 4.15 und anschließend der Code 4.16 ausgeführt, wird auf Server-Seite die in Abbildung 4.18 gezeigte Ausgabe erzeugt.

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

Abbildung 4.17.: Ausgabe des RPC-Servers

Auf der Client-Seite erfolgt die in Abbildung 4.17 dargestellte 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 -
```

Abbildung 4.18.: Ausgabe des RPC-Clients

Auf 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 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 Ablauf der Kommunikation ist im Diagramm 4.19 dargestellt.

Durch das Signieren einer bei jedem Funktionsaufruf neu generierten Challenge ist sichergestellt, dass nur der echte Host die Funktion aufruft und keine *Replay-Attacken* möglich sind.

Jedoch gab es bei der Implementierung über *Flask* sowie mit den verschiedenen *Python*-Versionen zwischen Server und Client Probleme beim Datenaustausch. So gibt es in *Python 2* einen dedizierten Variablen-Typ namens *string*, wohingegen *Python 3* Strings in codierten Byte-Objekten speichert. Eine Konvertierung ist möglich und wurde 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 XML-RPC-Protokoll²¹ 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. Dabei ist der Server-Code in Abbildung 4.20, der Client Code in Abbildung 4.21 sowie die Ausgabe in Abbildung 4.22 dargestellt.

Durch den minimalen Eingriff von *xmlrpc* in die Kommunikation konnte die kryptographische Signatur als *base64* codiertes Datum übergeben werden. Die Kern-Funktionen des Clients und Servers sind in den Abbildungen 4.23 und 4.24 dargestellt.

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

²¹<http://xmlrpc.scripting.com/spec.html>

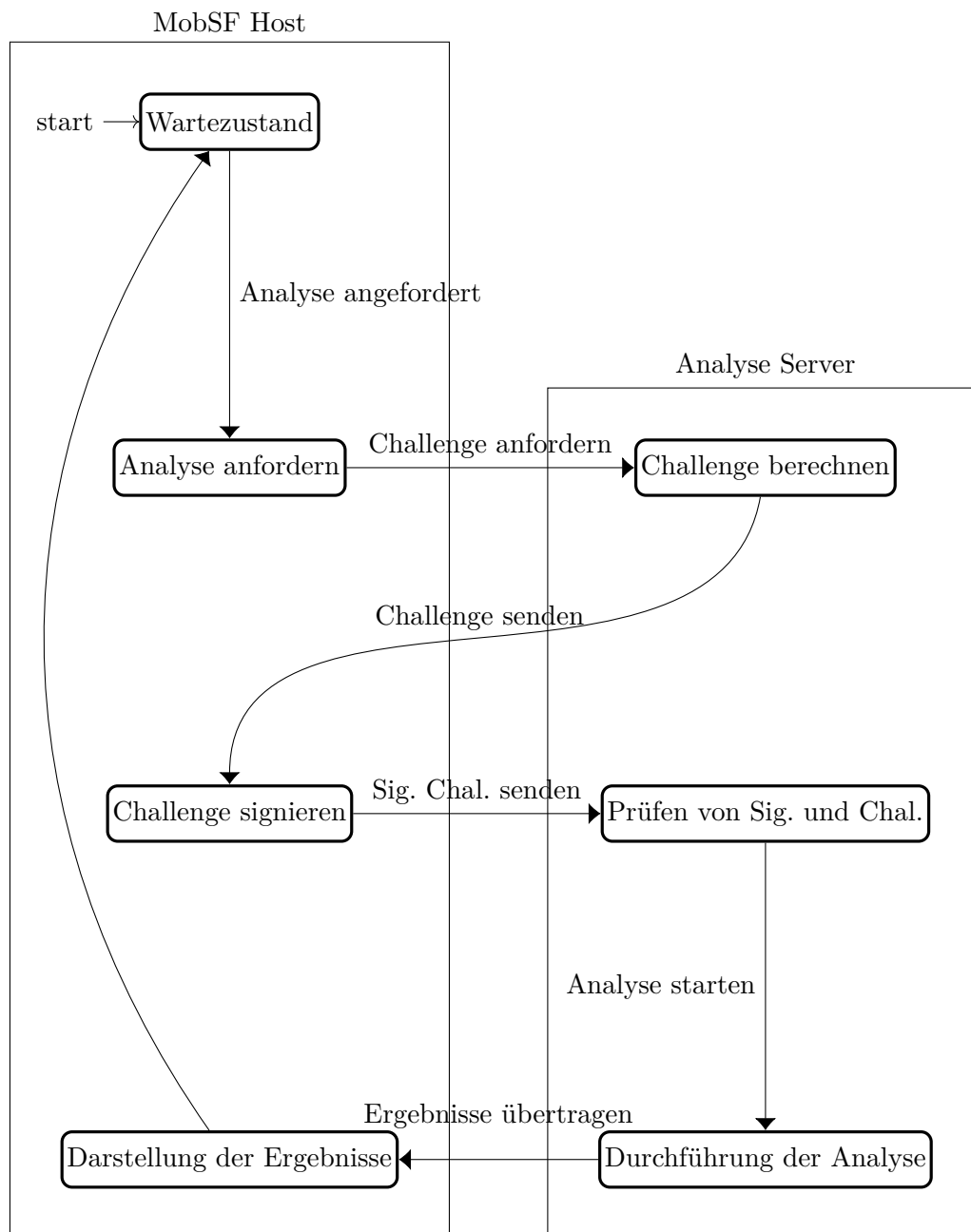


Abbildung 4.19.: Ablauf einer Analyse auf einer separaten VM

```
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")
```

Abbildung 4.20.: XML-RPC Server Code

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

Abbildung 4.21.: XML-RPC Client Code

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

Abbildung 4.22.: Ausgabe des Clients

```
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())
```

Abbildung 4.23.: Client-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!"
```

Abbildung 4.24.: Server-Code

4.6.2. Windows-Apps

Seit Windows 8 und der damit eingeführten *Unified Windows Platform*²² sind moderne Windows-Apps unter Windows sowohl auf einem PC als auch auf einem Handy lauffähig.

Bisher hat das *MobSEF* 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 allererst 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*-Applikationen²³. Dementsprechend ist der Mime-Type zumeist *application/x-silverlight-app*.

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

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

Anschließend liegen allen Dateien der App im Extraktion-Ordner.

APPX

APPX ersetzt ab Windows 8.1 das *XAP*-Format. Auch das *APPX*-Format nutzt einen Kompressionsalgorithmus ähnlich zu *Zip*. Daher können Inhalte über das in Python integrierte Modul *zipfile* extrahiert werden. Dies ist in Abbildung 4.25 dargestellt.

Es wurde folgende Mime-Types für *APPX*-Dateien festgestellt:

- *application/octet-stream*
- *application/vnd.ms-appx*
- *application/x-zip-compressed*

²²<https://docs.microsoft.com/de-de/windows/uwp/get-started/universal-application-platform-guide>

²³<https://www.microsoft.com/silverlight/>

```
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
```

Abbildung 4.25.: Beispiel-Code für das entpacken einer Windows-App

APPXBundle

Das Format *appxbundle* ist ein Zusammenschluss mehrerer *APPX*-Dateien. Entsprechend ist der Mime-Type zum Beispiel *application/zip* und kann ebenfalls mit zum Beispiel dem Python-Modul *zipfile* oder einem gängigen Zip-Extraktions-Programm entpackt werden.

DRM

Leider sind Apps aus dem Windows-Store häufig durch sogenanntes *DRM* (*Digital Rights Management*), hier *Play Ready*, geschützt. Die Apps sind demnach nicht nur durch Zertifikate vor Veränderungen geschützt, sondern zusätzlich sind die App-Dateien mit *AES* im *CTR*-Mod verschlüsselt. Dies ist im *PlayReady*-Header festgelegt, folgend ein Beispiel:

```
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>
```

Zwar ist es möglich, *DRM* geschützte Apps auf einem gerooteten Windows-Phone zu installieren und anschließend über Apps wie dem *ProgramManager*²⁴ als ungeschütztes Archiv an einen PC zu übertragen, jedoch könnte dies als Umgehung eines Kopierschutzes gesehen werden und wurde daher in dieser Arbeit nicht weiter verfolgt.

²⁴<https://forum.xda-developers.com/showthread.php?t=1922454>

4.6.2.2. Virtuelle Maschine zur Analyse von Windows-Apps

Da sowohl für die dynamische Analyse als auch aufgrund bestimmter Tools eine *Windows*-VM notwendig ist, wird der Aufbau für *MobSF* im Folgenden kurz allgemein dargestellt.

Als erstes muss ein Betriebssystem für die Analyse-VM gewählt werden. Der *Windows-Phone*-Simulator steht ab *Windows 8.1 64-Bit* zur Verfügung. Daher sollte *Windows 8.1 64-Bit* oder höher verwendet werden. Hier wurde *Windows 10 64-Bit* verwendet.

Für Tests bezüglich der dynamische Analyse wurde *Visual Studio 2015* Community Edition verwendet. Da diese kostenlos ist, sind bis auf das Betriebssystem sind keine kostenpflichtigen Programme beteiligt.

Für die Kommunikation zwischen *MobSF* und der VM wurde ein Setup-Skript (genauer beschrieben in Abschnitt 4.6.2.3) angefertigt, das die notwendigen Programme herunterlädt und die Installationen anstößt. Das Skript ist im Anhang unter A.2.1 zu finden.

4.6.2.3. Setup-Skript

Die Datei *setup.py* wird über zwei verschiedene Wege aufgerufen, je nachdem ob *MobSF* vollständig auf *Windows* installiert wird oder nur die statische Analyse auf dem *Windows*-System ausgeführt werden soll. Je nach Aufruf werden verschiedene Arbeitsschritte ausgeführt und auch verschiedene *Python*-Versionen verwendet, weshalb manche Funktionen sowohl *Python 2*- wie *3*- kompatibel gestaltet sind. Dies führte im Rahmen dieser Arbeit zu einem erhöhten Implementierungsaufwand, jedoch muss kein Weiterentwicklungsaufwand betrieben werden, wenn der Kern von *MobSF* ebenfalls auf *Python 3* portiert wird.

MobSF auf Windows

Wird *MobSF* vollständig auf *Windows* installiert, wird das Setup-Skript 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 das Config-File 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 Listing 4.26 dargestellt ist.

Nachdem der Download abgeschlossen ist, wird *BinSkim* über *nuget* installiert. Dazu wird *nuget* mit verschiedenen Parametern aufgerufen (siehe Listing 4.27).

Um später eine optimale Ausführung auf sowohl 64- wie 32-Bit Windows-System zu gewährleisten, werden aus den installierten Dateien die Binaries für *BinSkim x86* sowie *x64*


```
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 = urlrequest.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()
```

Abbildung 4.26.: Nuget-Download

```
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 )
```

Abbildung 4.27.: Installation von BinSkim über Nuget

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. Da dies nicht eindeutig von Microsoft dokumentiert ist, kann der Quellcode dem Appendix unter [A.3.1](#) entnommen werden.

Um sicherzustellen, dass das Setup-Skript 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-Skript direkt mit *Python 3* ausgeführt. Dabei wird ähnlich zur Installation auf Windows zuerst das Config-File initialisiert und anschließend werden die Ordner erstellt. Auch werden *nuget*, *BinSkim* und *BinScope* installiert.

Die Ordnerstruktur wurde dabei anfangs wie folgt aufgebaut:

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

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

Die *config.txt* enthält Inhalte, welche zentral abgelegt werden und für verschiedene Skripte eine wichtige Rolle spielen. Ein Beispiel wäre der Pfad zum Verzeichnis, in welchem die Tools gespeichert werden. Der Download-Ordner enthält die durch das Setup-Skript heruntergeladenen Binaries, der Tools-Ordner die installierten Tools.

Zusätzlich zur lokalen Installation wird ein XMLRPC-Server unter Tools installiert und konfiguriert. Die genaue Funktionsweise dieses Services ist unter [4.6.1.5](#) beschrieben.

Von Ajin Abraham, einem Contributor zu *MobSF*, wurde nach der Implementierung ein ergänzendes Video zur Installation erstellt und auf Youtube veröffentlicht²⁵.

²⁵<https://www.youtube.com/watch?v=17ilENuMj58>

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 getestet²⁶. Das Tool analysiert Compiler-Flags und verschiedenste andere statisch feststellbare Eigenschaften, bewertet diese und gibt die Ergebnisse im SARIF-Format²⁷ 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.

Files

Ein einfacher, aber oft sehr hilfreicher erster Eindruck entsteht durch die Auflistung der enthaltenen Files. In *MobSF* existiert daher eine Methode, welche *ZIP*-Archive entpackt und Pfad sowie Name der entpackten Dateien zurückliefert. Über diese Methode wurden auch hier die enthaltenen Files erfasst und in der Oberfläche dargestellt.

Bad Functions

Ebenso wie bei *iOS* wurde auch hier eine Extraktion der Strings aus dem Binary implementiert. Genutzt wurde dafür die unter 4.6.1.3 entwickelte Implementierung des Strings-Kommandos in Python.

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

²⁶<https://github.com/Microsoft/binskim/releases>

²⁷<https://github.com/sarif-standard/sarif-spec/>

4.6.2.5. Zusätzlich eingebundene Tools zur statischen Analyse

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

BinSkim

BinSkim ist ein Tool des *Microsoft SDL (Secure Development Lifecycle)* und prüft *Windows*-Applikationen bezüglich der Konfiguration des Compilers/Linkers sowie andere Security-Relevante Merkmale.[9]

Dabei können folgende Fehlkonfigurationen erfasst werden:

LoadImageAboveFourGigabyteAddress 64-bit Images sollten eine *base address* über der 4GB Grenze nutzen, um den Kompatibilitätsmodus von *ASLR* nicht zu aktivieren. Dieser würde zu einem kleineren Raum für *ASLR* und somit zu niedrigerer Sicherheit führen.

DoNotIncorporateVulnerableDependencies Binaries sollten keine Abhängigkeiten zu Code mit bekannten Schwachstellen haben.

DoNotShipVulnerableBinaries Es sollten keine Libraries enthalten sein, welche bekannte Schwachstellen besitzen.

BuildWithSecureTools Applikationen sollten mit den aktuellsten Compiler- und Tool-Versionen kompiliert werden, um alle Sicherheitsfunktionen nutzen zu können.

EnableCriticalCompilerWarnings Binaries sollten mit einem Warn-Level kompiliert werden, welcher alle Sicherheit-relevanten Checks angibt.

EnableControlFlowGuard Binaries sollten das *compiler control guard feature (CFG)* zur Buildzeit aktivieren, um Angreifer davon abzuhalten, den Programmablauf zu verändern.

EnableAddressSpaceLayoutRandomization Binaries sollten als *DYNAMICBASE* gelinkt sein, um an *ASLR* teilnehmen zu können.

DoNotMarkImportsSectionAsExecutable PE-Import-Sektionen sollten nie gleichzeitig als schreibbar und ausführbar markiert sein.

EnableStackProtection Binaries sollten *stack protection* (zum Beispiel *Stack-Canaries* über die */GS*-Flag) nutzen, um das Ausnutzen von *stack buffer overflows* zu erschweren.

DoNotModifyStackProtectionCookie Applikations-Code sollte keinen Einfluss auf die Stack-Absicherung haben.

InitializeStackProtection Binaries sollten die Stack-Absicherung korrekt initialisieren, um das Ausnutzen von *stack buffer overflows* zu erschweren.

DoNotDisableStackProtectionForFunctions Applikations-Code sollte keine Stack-Absicherung für einzelne Funktionen abschalten.

EnableHighEntropyVirtualAddresses Binaries sollten als kompatibel mit *high entropy ASLR* markiert werden.

MarkImageAsNXCompatible Binaries sollten das *NX*-Bit gesetzt haben, um ungewollte Ausführung von Daten als Code zu verhindern.

EnableSafeSEH X86 Binaries sollten *SafeSEH* aktivieren, um das Ausnutzen von Memory-Schwachstellen zu erschweren.

DoNotMarkWritableSectionsAsShared Code- und Data-Sections sollten nicht sowohl als *shared* und *writable* markiert sein.

DoNotMarkWritableSectionsAsExecutable PE-Sections sollten nicht sowohl *writable* wie auch *executable* sein.

SignSecurely Images sollten über sichere kryptographische Signaturen vom einen vertrauenswürdigen Author geschützt sein.

Der originale Text sowie längere Beschreibungen sind dem Output des Kommandos

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

zu entnehmen.

BinScope

BinScope ist ebenfalls ein Programm des *Microsoft Secure Development Lifecycle* und untersucht ebenfalls Binaries auf fehlerhafte Konfigurationen. Es ist der Vorgänger zu *BinSkim* und hat daher ähnliche Fehlerkategorien, welche im Folgenden dargestellt sind. Leider sind die Checks nicht dokumentiert, sodass der Autor aus den Namen und den bisher beobachteten Ergebnissen auf deren Funktionsweise schließt.

Im Gegensatz zu *BinSkim* braucht *BinScope* weniger Konfigurationsaufwand und ist daher, obwohl es der Vorgänger von *BinSkim* ist, durchaus eine sinnvolle Ergänzung.

ATLVersionCheck stellt sicher, dass die ATL-Header, welche für den Build des Binaries verwendet wurden, in Ordnung sind. Diese Regel kommt nur bei Dateien vom COM-Type zum Einsatz.

ATLVulnCheck prüft, ob Klassen, welche *IPersistStreamInit* implementieren, potenziell verwundbare Eigenschaften aufweisen. Diese Regel kommt nur bei Dateien vom COM-Type zum Einsatz.

AppContainerCheck prüft, ob die App mit der AppContainer-Flag kompiliert wurde. Die Flag ermöglicht eine isolierte Umgebung zur Ausführung der App²⁸.

CompilerVersionCheck prüft, ob für die App eine aktuelle Kompilerversion genutzt wurde (mindestens 14.00.50727).

²⁸[https://msdn.microsoft.com/en-us/library/windows/desktop/mt595898\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/mt595898(v=vs.85).aspx)

DBCheck prüft, ob das Binary mit der *Dynamic Base*-Flag kompiliert wurde, also an *ASLR* teilnehmen kann.

GSCheck verifiziert, dass die */GS-Flag* beim Kompiliervorgang verwendet wurde, welche den *Stack* vor *Memory-Corruption* schützt (z.B. über *Stack-Canaries*).

DefaultGSCookieCheck ist ein Check zur Konfiguration von GS, welcher die Art der verwendeten Cookies überprüft.

GSFriendlyInitCheck ist ein Check zur Konfiguration von GS, welcher prüft, ob das Binary entsprechende GS-freundliche Einstiegspunkte bietet.

GSFunctionSafeBuffersCheck ist ein weiterer Check zur Konfiguration von GS, welcher prüft, ob auch die Funktions-Buffer durch GS abgesichert werden.

ExecutableImportsCheck prüft, ob die Import-Section als ausführbar markiert ist.

FunctionPointersCheck prüft auf *global function pointers*. Durch das Überschreiben von statischen Buffern können eventuel auch *global function pointer* überschrieben werden, was eine Schwachstelle darstellen kann.

HighEntropyVACheck prüft, ob für ASLR eine hohe Entropie zur Verfügung steht.

NXCheck prüft, ob das NX-Flag (stehend für „no execute“) gesetzt ist. Durch diese Maßnahme werden Exploits erschwert, da bestimmte Speicherbereiche als nicht ausführbar gekennzeichnet werden.

RSA32Check prüft auf 32-Bit RSA-Keys, für welche eine Primfaktorzerlegung leicht möglich wäre.

SafeSEHCheck prüft, ob SafeSEH aktiviert wurde. Die Verwendung von SafeSEH verbessert das Fehlerhandling und erschwert die Entwicklung von Exploits.

SharedSectionCheck prüft, ob Sections sowohl als *shared* als auch *writable* markiert sind.

VB6Check prüft, ob VB6-Code zum Einsatz kommt.

WXCheck prüft, ob die WX-Flag verwendet wurde. Durch diese Flag müssen Linker-Warnungen wie Fehler behandelt werden, wodurch die Sicherheit erhöht wird.

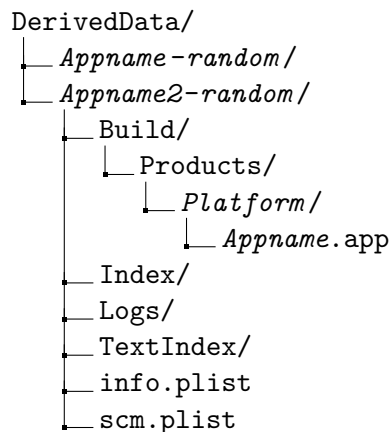


Abbildung 4.28.: Struktur des DerivedData-Ordners

4.6.3. iOS-Apps

Ein weiteres wichtiges Betriebssystem ist das auf *IPhones* verwendete *iOS*. Es hat einen Marktanteil von ca. 20%^[10] und wird auch von Unternehmen immer häufiger genutzt. Umso wichtiger ist es, dass angebotene Apps entsprechend auf deren Sicherheit geprüft werden können. Daher sind im Folgenden durchgeführte Erweiterungen zu *MobSF* aufgezeigt.

4.6.3.1. IPA- und APP-Format

Um eine *iOS*-App zu analysieren, muss diese in einem passenden Format vorliegen. Geläufig und bereits von *MobSF* unterstützt ist das *IPA*-Format. Nativ werden *iOS*-Apps jedoch als *.app*-Dateien abgelegt. 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/username/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 4.28 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 4.29). 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

Eine wichtige Information über *iOS*-Apps ist, welche Berechtigungen diese erfordern. In der aktuelle 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.

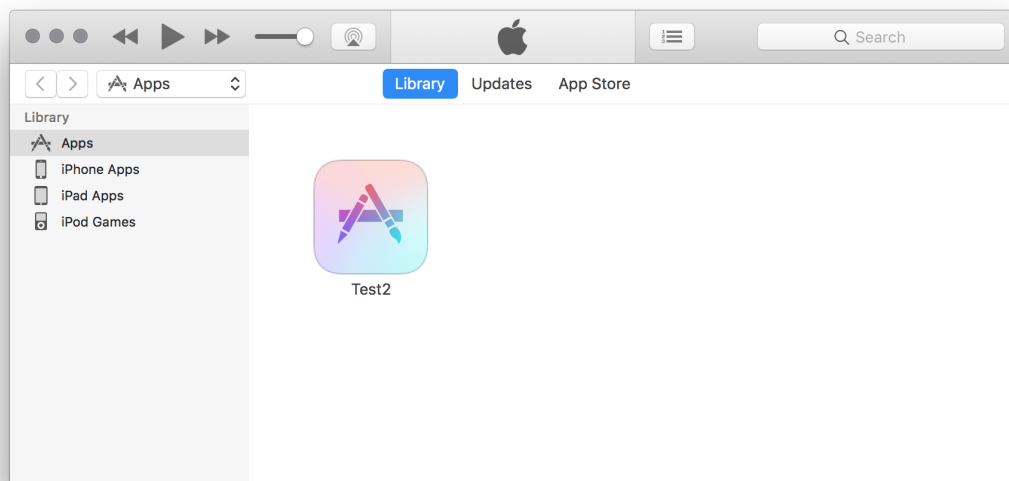


Abbildung 4.29.: iTunes-App-Fenster

NSAppleMusicUsageDescription beschreibt, dass die App den Zugriff auf die Musik-Bibliothek anfordern kann.

NSBluetoothPeripheralUsageDescription beschreibt, dass die App die Berechtigung anfordern kann, auf das Bluetooth-Interface zuzugreifen.

NSCalendarsUsageDescription beschreibt, dass die App den Zugriff auf den Kalender anfordern kann. Dadurch können alle Termine gelesen und verändert sowie gelöscht werden.

NSCameraUsageDescription beschreibt, dass die App den Zugriff auf die Kamera anfordern kann. Dadurch können Fotos und Videos aufgenommen werden.

NSContactsUsageDescription beschreibt, dass die App den Zugriff auf die Kontakte des User anfordern kann. Dadurch können alle Einträge im Kontaktbuch gelesen, verändert und gelöscht werden.

NSHealthShareUsageDescription beschreibt, dass die App den Lese-Zugriff auf die durch das Gerät erfassten Gesundheitsdaten anfordern kann. Diese können zum Beispiel zurückgelegte Strecken oder die Herzfrequenz enthalten.

NSHealthUpdateUsageDescription beschreibt, dass die App den Schreib-Zugriff auf die Gesundheitsdaten des Users anfordern darf.

NSHomeKitUsageDescription beschreibt, dass die App den Zugriff auf das Home-Kit des Users anfragen kann. Weitere Details zum Home-Kit sind auf der Apple-Webseite²⁹ zu finden.

²⁹<http://www.apple.com/de/ios/home/>

CLLocationAlwaysUsageDescription beschreibt, dass die App das Recht anfragen kann, zu jeder Zeit den GPS-Standort des Users erfassen.

CLLocationWhenInUseUsageDescription beschreibt, dass die App das Recht anfragen kann, den GPS-Standort auszulesen, solange die App im Vordergrund aktiv ist.

NSMicrophoneUsageDescription beschreibt, dass die App das Recht anfragen kann, auf das Mikrophon zuzugreifen.

NSMotionUsageDescription beschreibt, dass die App den Zugriff auf den Motion-Sensor anfragen kann. Dadurch können Bewegungen erfasst werden.

NSPhotoLibraryUsageDescription beschreibt, dass die App den Zugriff auf die Photo-Bibliothek anfordern kann.

NSRemindersUsageDescription beschreibt, dass die App den Zugriff auf die Reminder des Users anfordern kann. Dadurch können Reminder ausgelesen, erstellt, verändert oder gelöscht werden.

NSVideoSubscriberAccountUsageDescription beschreibt, dass die App den Zugriff auf den TV-Account des Users anfragen kann.

Diese, sowie weitere Einträge für die *Info.plist* sind der Apple-Dokumentation zu entnehmen³⁰.

In der Auflistung wurde die Formulierung „anfragen kann“ genutzt, da der Eintrag in der *info.plist* den App nicht automatisch die Rechte gibt, sondern der User diese bestätigen muss, sobald die App diese wirklich anfordert. Jedoch können zur Laufzeit keine Rechte angefordert werden, welche nicht vorher in der *info.plist* festgelegt wurden.

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!")
```

³⁰<https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html>

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

Abbildung 4.30.: Auslesen von Ausnahmen bezüglich der TLS-Konfiguration aus der Info.plist

In *MobSEF* wird auf diese Art die *info.plist*-Datei verarbeitet und die Ergebnisse in der Web-Oberfläche dargestellt. Das Ergebnis für eine App mit Bluetooth-Berechtigung ist in Grafik 4.31 dargestellt.

Auf einige Rechte wie den Zugriff auf Kontakte, Kalender, Reminder, Kamera und Mikrophon sollte bei Analyse besonders geachtet werden.

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 2818³¹ nutzen, um Verbindungen zu Webseiten oder APIs aufzubauen.

Daher wurde *MobSEF* um ein Feature ergänzt, welches die *Info.plist* auf Ausnahmen überprüft. Der Code ist unter Abbildung 4.30 dargestellt.

Werden Ausnahmen gefunden, werden diese in der HTML-Oberfläche dargestellt. Ein Beispiel ist in 4.31 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 könnte. Um dieses Ziel zu erreichen, muss zuerst eine Automatisierung des Simulators möglich sein. Diese Automatisierung kann über das in *Xcode* enthaltene Tool „xcrun“ realisiert werden.

³¹<https://tools.ietf.org/html/rfc2818>

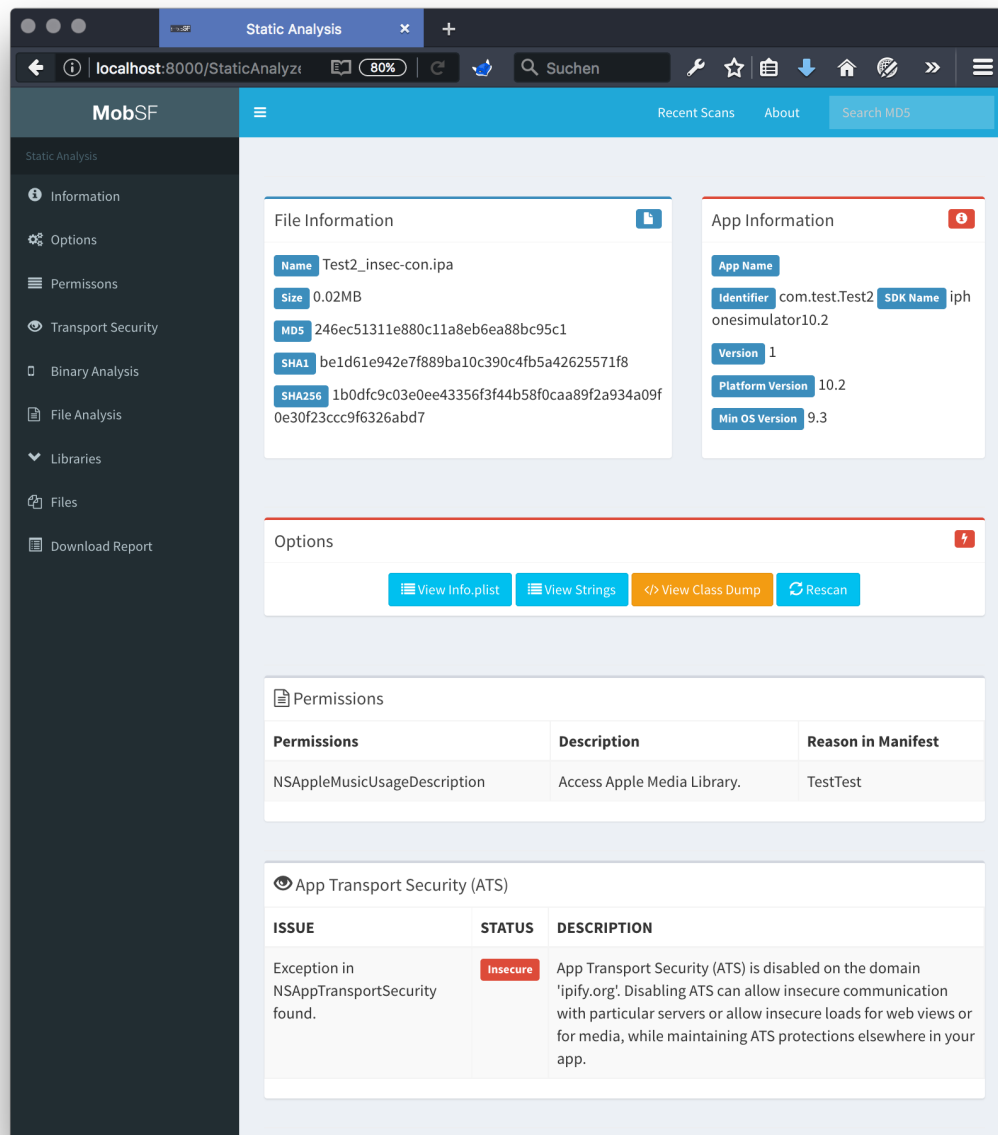


Abbildung 4.31.: Ergebnis für eine App mit Bluetooth-Berechtigung und einer Ausnahme der *ATS*

Über die Befehle von *xcrun* kann folgender Ablauf für die dynamische Analyse realisiert werden.

1. Gerät starten (Parameter *boot*)
2. App installieren (Parameter *install*)
3. App ausführen (Parameter *launch*)
4. Analyse durchführen
5. App schließen (Parameter *termiate*)
6. App deinstallieren (Parameter *uninstall*)
7. Gerät herunterfahren/ggf. löschen (Parameter *shutdown* oder *delete*)

Somit sollte für erste Tests ein geeigneter Ablauf zur Verfügung stehen. Alternativ hätte ein Tool von *Facebook*³² genutzt werden können, welches wohl einen ähnlichen Funktionsumfang besitzt. Da das Ziel jedoch eine Eingliederung in MobSF war, wurde die Eigenimplementierung über *xcrun* gewählt.

Ein Kerninhalt der dynamischen Analyse ist zumeist der Netzwerkverkehr. Eine simple Aufzeichnung über *TCPDump* oder *Wireshark* kann natürlich leicht vollzogen werden. Um dies jedoch zu verifizieren, wurde eine App mit dem in Abbildung 4.32 gezeigten Code angelegt.

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

Abbildung 4.32.: Aufruf von <https://api.ipify.org>

Die *Wireshark*-Ausgabe verifiziert, dass die Verbindung aufgezeichnet werden kann, wie in Grafik 4.33 zu sehen ist.

Aufgrund der strengen Vorkonfiguration bezüglich der Transport-Sicherung (siehe Abschnitt 4.3.1.4), ist mit mehr TLS-Verkehr als bei gewöhnlichen Anwendungen zu rechnen. Umso wichtiger ist es, diese unterbrechen und den Inhalt entschlüsseln zu können. Ein passendes Tool hierfür ist *MITMProxy*, welches als lokaler Proxy-Server alle TLS-Verbindungen unterbricht. Zum Client hin wird ein selbst generiertes Zertifikat angeboten, während zum Webserver eine valide TLS-Verbindung aufgebaut wird. Dies hat natürlich den Nachteil, dass am emulierten Gerät dem selbst generierten Zertifikat vertraut werden muss. Auch zu prüfen bleiben die Möglichkeiten zur Einstellung des Proxy-Servers auf dem emulierten Gerät.

³²<https://github.com/facebook/FBSimulatorControl>

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=120479
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	105.728517	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 4.33.: Verbindung zu https://api.ipify.org aus dem iOS-Simulator

Zuerst wurde die Möglichkeit geprüft, das Zertifikat auf emulierten Geräten in den Trust-Store aufzunehmen. Da eine händische Aufnahme weder möglich, noch für die Automatisierung vorteilhaft wäre, sollte ein passendes Tool genutzt werden. *ADVTOOLS*³³ ermöglicht es, eine beliebige CA in den Trust-Store der emulierten Geräte aufzunehmen, und bietet damit die passende Funktionalität.

Als nächstes ist die Konfiguration von Proxy-Servern auf den emulierten Geräte zu realisieren. Auch dies ist leider nicht direkt über den Simulator realisierbar, jedoch nutzt der Simulator den Proxy-Server des *Mac OS X*-Host-Systems. Dieser kann über die System-Einstellungen konfiguriert werden (siehe Grafik 4.34).

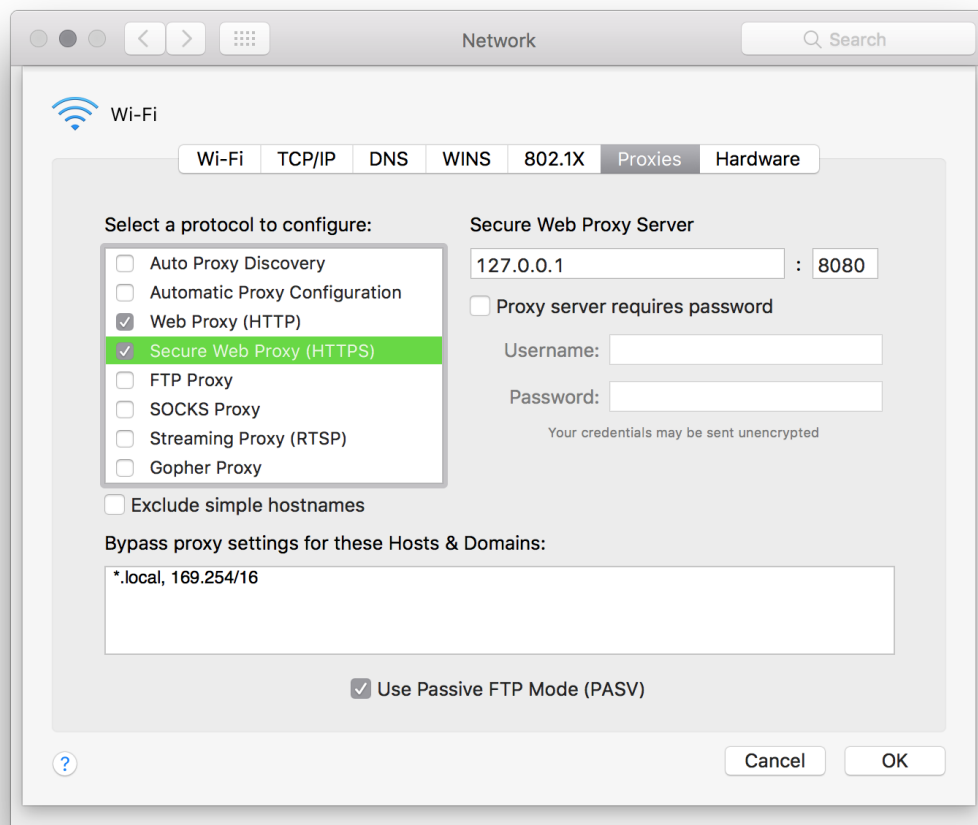
Somit ist nun eine TLS-Unterbrechung über *MITMProxy* denkbar. *MITMProxy* bietet eine *NCurses*-Oberfläche an, um den durchlaufenden Netzwerkverkehr zu analysieren. Diese ist in Grafik 4.35 dargestellt.

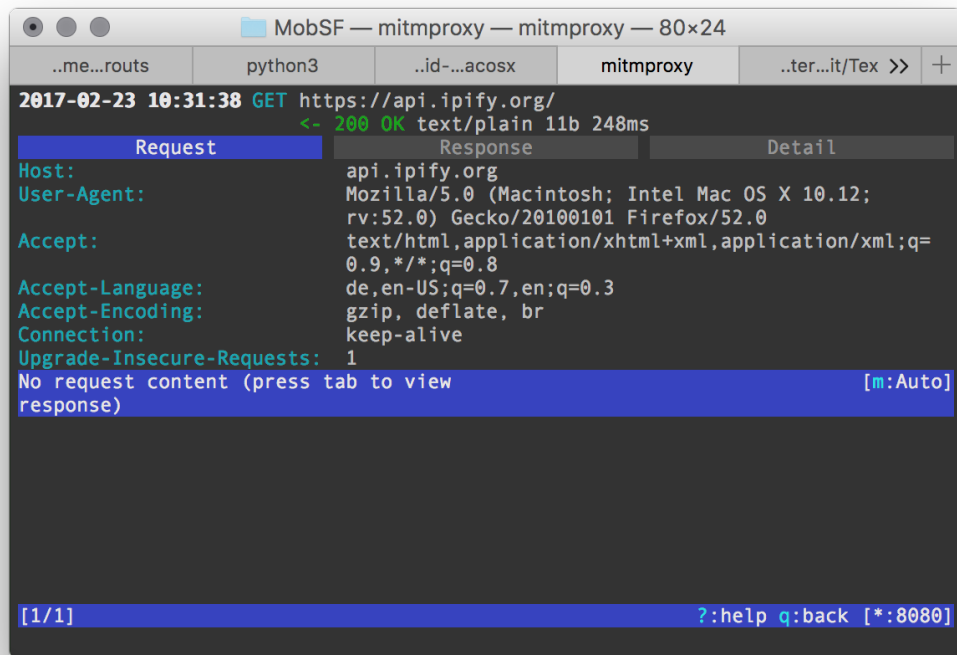
Diese Oberfläche ist jedoch für eine automatisierte Auswertung ungeeignet. Als zusätzliches Feature bietet *MITMProxy* seine Funktionalität auch als Python-Modul an. Über diese wurde ein Server implementiert, welcher den Netzwerkverkehr aufzeichnet und in einem geeigneten Format ablegt. Der Server ist im Anhang unter A.2.2 dargestellt.

Insoweit ist eine dynamische Analyse möglich. Der entsprechende Code, um alle Schritte zusammen zu führen, ist ebenfalls im Anhang unter A.2.1 zu finden. Da jedoch, wie in 4.3.1.3 beschrieben, nur entsprechend kompilierte Apps oder solche, für welche der Sourcecode zur Verfügung steht, getestet werden können, ist der Use-Case für *MobSF* relativ gering. Daher wird der Sourcecode zwar veröffentlicht, jedoch nicht in *MobSF* integriert.

Im Laufe des Jahres soll eine Implementierung der dynamischen Analyse auf Basis eines physikalischen Geräts mit Jailbreak erfolgen. Dies ist jedoch nicht mehr Teil dieser Arbeit.

³³<https://github.com/ADVTOOLS/ADVTrustStore>

Abbildung 4.34.: Einstellung des Proxy-Servers unter *Mac OS X*

Abbildung 4.35.: Oberfläche von *MITMProxy*

```
1 char to[10];  
2 char from[] = "AAAAAAAAAAAA";  
3 strcpy(to, from);
```

Abbildung 4.36.: Buffer-Overflow über strcpy in der Windows-Phone-Test-App

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

Um die eingebauten Features bezüglich der statischen Analyse von *Windows-Phone*-Apps zu testen, wurde eine minimalistische App geschrieben.

Diese enthält ein Code-Segment mit der *strcpy*-Funktion, welches unter 4.36 dargestellt ist. Ebenso wurde die Option *CFG* (*Control Flow Guard*) für den Test deaktiviert. Der vollständige Quellcode ist dem Datenträger unter

```
1 /Apps/iOS/vuln_app_wp.zip
```

angehängt. Die Inhalte des Datenträgers sind ebenfalls dem zu dieser Arbeit zugehörigen Repository in Github veröffentlicht³⁴.

Die App wurde anschließend durch *MobSF* analysiert. Dabei wurden mehrere unsichere Funktionen, unter anderem die absichtlich eingebaute *strcpy*-Funktion gefunden. Ebenso wurde detektiert, dass *CFG* nicht aktiviert ist. Der vollständige Bericht, inklusive der String-Analyse, ist auf dem Datenträger unter

```
1 /Apps/iOS/vuln_app_wp_report.pdf
```

abgelegt.

³⁴https://github.com/DominikSchlecht/Pen-Test_bei_mob_Anwendungen

4.8.2. Test Anwendung 2 - iOS

Zur Entwicklung der Test-Applikation wurde *Xcode 8* verwendet. Die App hat eine definierte Ausnahme in der *ATS* und benötigt die *NSAppleMusicUsageDescription*-Berechtigung. Ebenso benutzt sie, ähnlich zur *Windows-Phone*-App, *strcpy* zum Kopieren eines *char*-Arrays. Die vollständigen Source-Dateien sind auf dem beigelegten Datenträger unter

```
1 /Apps/iOS/vuln_app_ios.zip
```

zu finden.

Auch hier findet *MobSF* die eingebauten Anomalien. So werden die Ausnahme der *ATS*, sowie die *strcpy*-Funktion gefunden und als unsicher dargestellt. Auch die *NSAppleMusicUsageDescription*-Berechtigung wurde detektiert und mit der Begründung dargestellt. Unter Grafik 4.37 sind die Ergebnisse im Überblick dargestellt; der vollständige Bericht ist auf dem Datenträger unter

```
1 /Apps/iOS/vuln_app_ios_report.pdf
```

abgelegt.

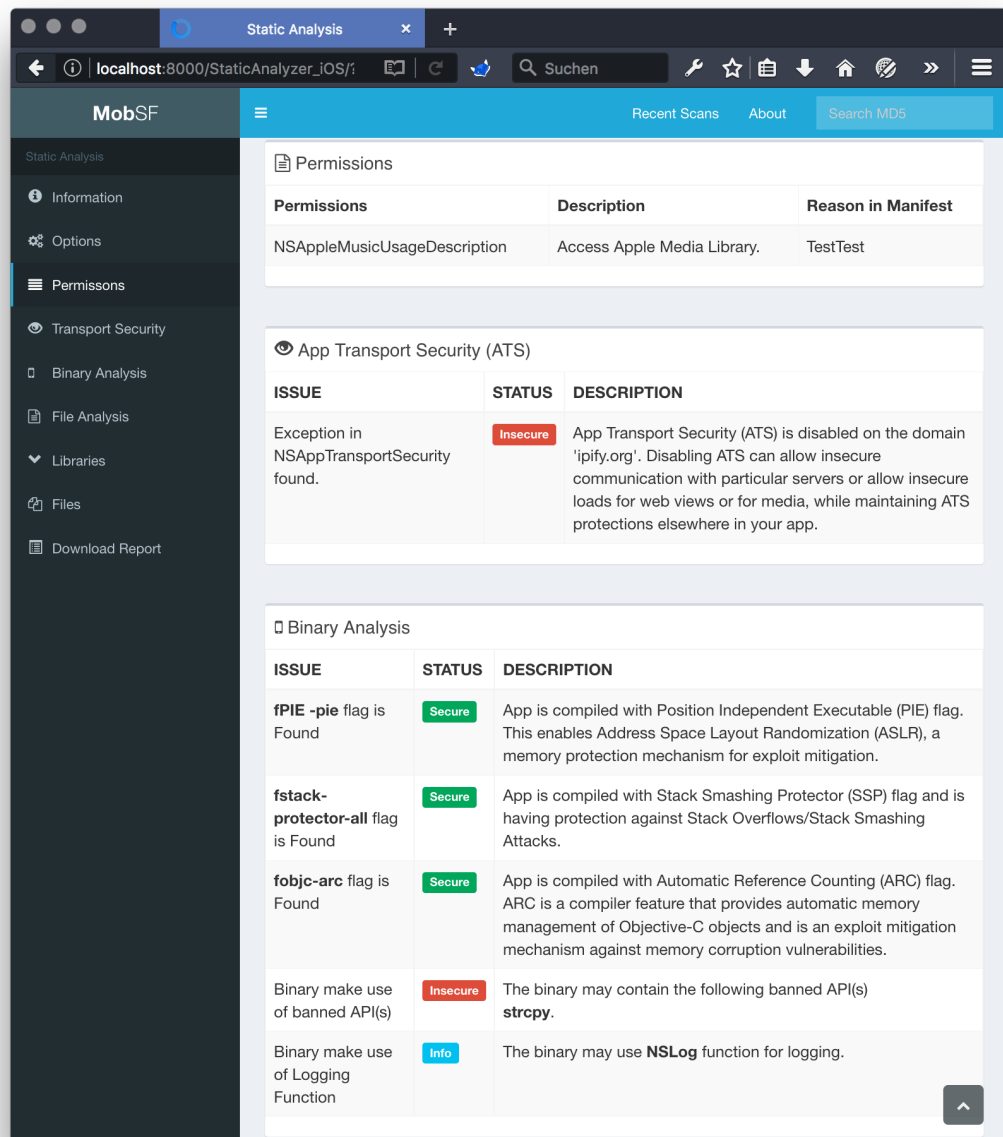


Abbildung 4.37.: Ergebnis des MobSF für die iOS-App

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.

Dabei wurden bestehende Praktiken um die Anwendung auf mobile Applikationen erweitert. Zudem wurden für aufwändige Prozessschritte Tools entwickelt, welche die Effizienz sowohl auf Seiten der Pen-Tester als auch des Unternehmens erhöhen können. Diese Schritte sind die Fragebögen zur Aufwandsschätzung und des Kickoffs, sowie die Erstellung des Pen-Test-Reports. Die Inhalte für diese Schritte können effizient und komfortabel in einer Weboberfläche eingepflegt und als PDF-Dokument exportiert werden. In Hinsicht auf mobile Anwendungen wurde anschließend auf Tools und Frameworks eingegangen und das *Mobile Security Framework (MobSF)* so weiterentwickelt, dass dieses alle gesetzten Anforderungen erfüllt. Dabei wurden über 3700 Zeilen an Code hinzugefügt oder überarbeitet, um eine genauere Analyse von *iOS*-App und eine statische Analyse von *Windows-Phone*-Apps zu ermöglichen.

Abschließend wurde das Framework anhand von zwei Beispielen vorgestellt. Dabei wurden eine *iOS*-App sowie eine *Windows-Phone*-App analysiert und die Ergebnisse dargestellt. Alle entwickelten Werkzeuge werden als Open-Source-Produkte auf Github veröffentlicht.

Somit stellt diese Arbeit neu entwickelte oder weiterentwickelte Werkzeuge für den Test von mobilen Applikationen kostenfrei bereit. Diese können Unternehmen sowie private Entwickler nutzen, um auf die stark Ansteigende Nachfrage für mobilen Applikationen zu reagieren und die damit verbundenen sicherheitstechnischen Herausforderungen effizienter zu bewältigen.

A. Quellcode

A.1. Pentest-Helper

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 ein 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

A.1.2. Docker

```
1 FROM ubuntu:16.10
2 RUN apt-get update
3
4 # Set bash
5 RUN rm /bin/sh && ln -s /bin/bash /bin/sh
6
7 # Supervisor
8 RUN DEBIAN_FRONTEND=noninteractive apt-get -yq install
   supervisor
9 run mkdir -p /var/log/supervisor
10
11 RUN apt-get update && apt-get upgrade -y
12
13 ## Python 3.6
14 RUN apt-get install -y \
15     python3.6 \
16     python3.6-dev \
17     python3-pip \
18     supervisor \
19     && apt-get autoremove \
20     && apt-get clean
21
22 RUN python3.6 -m pip install pip --upgrade
23
24 # Folders
25 run mkdir -p /var/log/supervisor
26
27 RUN mkdir -p /opt/pentest-helper
28 RUN mkdir -p /opt/pentest-helper/compile/report/bilder
29 RUN mkdir -p /opt/pentest-helper/compile/report/configs
30 RUN mkdir -p /opt/pentest-helper/compile/report/screenshots
31 RUN mkdir -p /opt/pentest-helper/download
32 RUN mkdir -p /opt/pentest-helper/static
33 RUN mkdir -p /opt/pentest-helper/templates/fragebogen
34 RUN mkdir -p /opt/pentest-helper/templates/report
35 RUN mkdir -p /opt/pentest-helper/templates_tex/report
36 RUN mkdir -p /opt/pentest-helper/tex/download
37 RUN mkdir -p /opt/pentest-helper/tex/report
38 RUN mkdir -p /opt/pentest-helper/uploads
39
40 # Requirements
```

```
41 COPY requirements.txt /opt/pentest-helper/requirements.txt
42 RUN python3.6 -m pip install -r /opt/pentest-helper/requirements
   .txt
43
44 # Copy Code-Files
45 COPY compile/report/bilder/* /opt/pentest-helper/compile/report/
   bilder/
46 COPY compile/report/configs/* /opt/pentest-helper/compile/report
   /configs/
47 COPY static/* /opt/pentest-helper/static/
48 COPY templates/fragebogen/* /opt/pentest-helper/templates/
   fragebogen/
49 COPY templates/report/* /opt/pentest-helper/templates/report/
50 COPY templates/* /opt/pentest-helper/templates/
51 COPY templates_tex/report/* /opt/pentest-helper/templates_tex/
   report/
52 COPY templates_tex/* /opt/pentest-helper/templates_tex/
53 COPY tex/* /opt/pentest-helper/tex/
54 COPY clean.sh /opt/pentest-helper/clean.sh
55 COPY pentest_helper.py /opt/pentest-helper/pentest_helper.py
56 COPY pentest_templates.py /opt/pentest-helper/pentest_templates.
   py
57 COPY run_docker.py /opt/pentest-helper/run.py
58
59 #CMD ["/usr/bin/python3.6", "/opt/pentest-helper/run.py"]
60
61 # Supervisor
62 # COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
63 # CMD ["/usr/bin/supervisord"]
64
65 EXPOSE 5000
```

Listing A.4: Dockerfile für Entwicklungsversion

A.2. iOS-Simulator

A.2.1. Control-Script

```
1 import io
2 import os
3 import shutil
4 import signal
5 import subprocess
6 import xml.etree.ElementTree as ET
7
8
9 from time import sleep
10
11 def read_bin_xml(FILE):
12     """Convert plist binary to readable xml"""
```

```
13     try:
14         args = ['plutil', '-convert', 'xml1', FILE]
15         dat = subprocess.check_output(args)
16         with io.open(FILE, mode='r', encoding="utf8", errors="
            ignore") as f:
17             dat = f.read()
18         return dat
19     except:
20         print("[ERROR] Converting Binary XML to Readable XML")
21
22 ipa_path = "/Users/dominik/Masterarbeit/MobSF/Testfiles/
    Test2_sim.ipa"
23
24 # Chain together path
25 working_path = os.path.join(os.getcwd(), "tmp")
26
27 # Creat if not existing
28 if not os.path.exists(working_path):
29     os.makedirs(working_path)
30
31 # Extract to path
32 subprocess.call(
33     [
34         "unzip", ipa_path,
35         "-d", working_path
36     ]
37 )
38
39 # Look for file
40 import fnmatch
41 import plistlib
42
43 plist_matches = []
44 app = ""
45 for root, dirnames, filenames in os.walk(working_path):
46     for filename in fnmatch.filter(filenames, 'Info.plist'):
47         plist_matches.append(os.path.join(root, filename))
48     for dirname in dirnames:
49         if dirname.endswith(".app"):
50             app = os.path.join(root, dirname)
51
52 # Read XML
53 bundle_identifier = ""
54 for elem in plist_matches:
55     xml = plistlib.readPlistFromString(read_bin_xml(elem))
56     if 'CFBundleIdentifier' in xml:
57         bundle_identifier = xml['CFBundleIdentifier']
58         break
59
60 subprocess.call(["open", "/Applications/Xcode.app/Contents/
    Developer/Applications/Simulator.app"])
61
```

```
62 raw_input("Press Enter when simulator is started...")
63
64 subprocess.call(["xcrun", "simctl", "install", "booted", app])
65 subprocess.call(["xcrun", "simctl", "launch", "booted",
    bundle_idenfier])
66
67 sleep(5)
68
69 raw_input("Press Enter to start testing...")
70
71 # Start mitmproxy
72 mitm = subprocess.Popen(["python", "mitm_test.py", "--file", "
    test"], preexec_fn=os.setsid)
73
74 raw_input("Press Enter when done testing...")
75
76 # Stop mitmproxy
77 os.killpg(os.getpgid(mitm.pid), signal.SIGTERM)
78
79 subprocess.call(["xcrun", "simctl", "terminate", "booted",
    bundle_idenfier])
80 sleep(2)
81 subprocess.call(["xcrun", "simctl", "uninstall", "booted",
    bundle_idenfier])
82
83
84
85 ## Cleanup
86 # Remove after everything is done
87 shutil.rmtree(working_path)
```

Listing A.5: control.py

A.2.2. MITM-Server

```
1 #!/usr/bin/env python
2 """
3     Simple script to start mitmproxy up and write req, resp and
4     log to a specified file line by line
5 """
6 import argparse
7 import json
8 import signal
9 import sys
10
11 from mitmproxy import flow, controller, options
12 from mitmproxy.proxy import ProxyServer, ProxyConfig
13
14 class MyMaster(flow.FlowMaster):
15     log_file = ""
```

```
16     def run(self):
17         try:
18             flow.FlowMaster.run(self)
19         except KeyboardInterrupt:
20             self.shutdown()
21
22     @controller.handler
23     def request(self, f):
24         req_dic = { 'type' : 'request' }
25         req_dic['content'] = f.request.content
26         req_dic['headers'] = f.request.headers
27         req_dic['cookies'] = f.request.cookies
28         self.log_file.write(json.dumps(str(req_dic)) + "\n")
29         self.log_file.flush()
30
31
32     @controller.handler
33     def response(self, f):
34         resp_dic = { 'type' : 'response' }
35         resp_dic['content'] = f.response.content
36         resp_dic['headers'] = f.response.headers
37         resp_dic['cookies'] = f.response.cookies
38         self.log_file.write(json.dumps(str(resp_dic)) + "\n")
39         self.log_file.flush()
40
41     @controller.handler
42     def error(self, f):
43         print("error", f)
44         print(dir(f))
45
46     @controller.handler
47     def log(self, l):
48         log_dic = { 'type' : 'log' }
49         log_dic['level'] = l.level
50         log_dic['msg'] = l.msg
51         log_dic['reply'] = l.reply
52         self.log_file.write(json.dumps(str(log_dic)) + "\n")
53         self.log_file.flush()
54
55     def set_log_file(self, log_file):
56         self.log_file = log_file
57
58
59 def start_mitm_proxy(log_file):
60     opts = options.Options(cadir="~/mitmproxy/")
61     config = ProxyConfig(opts)
62     state = flow.State()
63     server = ProxyServer(config)
64
65     log_file = open(log_file, 'w')
66
67     m = MyMaster(opts, server, state)
```

```
68     m.set_log_file(log_file)
69     m.run()
70     log_file.close()
71
72 if __name__ == "__main__":
73     parser = argparse.ArgumentParser(description='Capture
74         mitmproxy traffic to file.')
75     parser.add_argument('--file', metavar='file', help='the file
76         to write to', required=True)
77     args = parser.parse_args()
78     start_mitm_proxy(args.file)
```

Listing A.6: MITM-Server

A.3. MobSF

A.3.1. BinScope-Installer

```
1 def tools_binscope():
2     """Download and install Binscope for MobSF"""
3
4     mobsf_subdir_tools = CONFIG['MobSF']['tools']
5     binscope_path = mobsf_subdir_tools + 'BinScope'
6
7     # Download the right version for os
8     if platform.machine().endswith('64'):
9         binscope_url = CONFIG['binscope']['url_x64']
10        binscope_installer_path = binscope_path + "\\
11            BinScope_x64.msi"
12    else:
13        binscope_url = CONFIG['binscope']['url_x86']
14        binscope_installer_path = binscope_path + "\\
15            BinScope_x86.msi"
16
17    if not os.path.exists(binscope_path):
18        os.makedirs(binscope_path)
19
20    binscope_installer_file = open(binscope_installer_path, "wb"
21        )
22
23    # Downloading File
24    print("[*] Downloading BinScope..")
25    binscope_installer = urlrequest.urlopen(binscope_url)
26
27    # Save content
28    print("[*] Saving to File {}".format(binscope_installer_path
29        ))
30
31    # Write content to file
```

```
28     binscope_installer_file.write(bytes(binscope_installer.read
    29         ()))
30     # Aaaand close
31     binscope_installer_file.close()
32
33     # Execute the installer
34     print("[*] Installing BinScope to {}".format(binscope_path))
35     os.system(
36         'msiexec' + ' ' +
37         'INSTALLLOCATION="' + binscope_path + '"' +
38         '/i "' + binscope_installer_path + '"' +
39         '/passive'
40     )
41
42     CONFIG['binscope']['file'] = binscope_path + "\\Binscope.exe"
43
44     # Write to config
45     with open(os.path.join(CONFIG_PATH, CONFIG_FILE), 'w') as
46         configfile:
47         CONFIG.write(configfile) # pylint: disable-msg=E1101
```

Listing A.7: BinScope-Installer

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 (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] *Github-Repository von BinSkim*. URL: <https://github.com/microsoft/binskim> (besucht am 26.02.2017).
- [10] *Kantar Worldpanel über die Marktanteile von mobilen Betriebssystemen*. URL: <https://www.kantarworldpanel.com/global/News/Android-Share-Growth-is-Highest-in-EU5-in-Over-Two-Years> (besucht am 26.02.2017).
- [11] *OpenStack über DREAD*. URL: <https://wiki.openstack.org/wiki/Security/OSSA-Metrics#DREAD> (besucht am 21.02.2017).
- [12] *OWASP Top 10*. URL: https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project#The_OWASP_Foundation (besucht am 21.02.2017).
- [13] *OWASP über DREAD*. URL: https://www.owasp.org/index.php/Threat_Risk_Modeling (besucht am 21.02.2017).
- [14] *Statistika zu der Verteilung von Android-Versionen*. URL: <https://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/> (besucht am 09.03.2017).