Masterarbeit

Weiterentwicklung von Methoden und Werkzeugen zum Pentest b. mob. Applikationen

zur Erlangung des akademisches Grades eines
Master of Science

angefertigt von
Dominik Gunther Florian Schlecht
Matrikelnummer: 00032209

Betreuer

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

Fakultät:	Elektrotechnik und Informatik
Studiengang:	Informatik
Schwerpunkt:	Security & Safety
Abgabedatum:	01. April 2016

Ingolstadt, 28. Juni 2016

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, 28. Juni 2016

Inhaltsverzeichnis

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 [**GartnerSales**], siehe Tabelle 1. Mit diesem Fortschritt im Bereich der Verkäufe steigt natürlich auch die Nutzung von Smartphones. Diese können genutzt werden um im Internet zu ßurfen", Medien-Inhalte wiederzugeben oder zu Chatten. Dies sind im Bezug auf Datenschutz und Risiko relativ ungefährliche Anwendungen. Jedoch können auch kritischere Handlungen vollzogen werden, wie zum Beispiel Online-Banking oder das Verwalten von Versicherungs-Verträgen. Das dies gemacht wird, ist eine notwendige Konsequenz aus der Natur des Smartphones und deren User. So ist es einfach praktisch, seinen Kontostand schnell von unterwegs einsehen zu können. Jedoch müssen solche Apps dann so gestaltet sein, dass ein Missbrauch nicht oder nur mit unverhältnismäßig hohem Aufwand möglich ist. Dies ist die Aufgabe der Unternehmen, welche solche Apps bereit stellen. Dies stellt viele vor unerwartet hohe Herausforderungen. Mussten bisher nur lokale oder Web-Anwendungen getestet werden, sind es nun Applikationen auf mobilen Geräten mit einem oft wesentlich höheren Anteil an Web-Services und APIs.

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

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

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

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

2 Arten von Penetrationstests

2.1 Web-Application

2.2 Wireless

2.3 Social-Engineering

2.4 Mobile-Applications

2.5 Blackbox/Whitebox

3 Prozesse zu Penetrationstests

3.1 Vorbereitung

3.1.1 Aufwandsschätzung

3.1.2 Rechtliche Aspekte

BDSG

NDA

Haftungsausschluss

Absicherung gegen §203STGB

3.1.3 Technische Aspekte

Infrastruktur

Tools

3.2 Durchführung

3.2.1 Bewertung von Findings

CVSS

Alternative Modelle

3.2.2 Dokumentation

3.2.3 Reporting

3.3 Nachbereitung

3.3.1 Inhalte

3.3.2 Vorlage

4 Penetrationstests mobiler Anwendungen

4.1 Bestehende Anwendungen

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

4.1.1 All-In-One-Framework: MobSF

MobSF ist das einzige, derzeit öffentlich Verbreitete All-In-One-Framework zur Analyse von Mobilen Applikationen. Es ist eine Plattform zur statischen Analyse von Android und iOS-Apps sowie zur dynamischen Analyse von Android Apps. Es bündelt viele kleinere Anwendungen, welche unter 4.1.2 aufgeführt sind, in einer einfachen Weboberfläche. Es ist Open-Source, in *Python* geschrieben und steht in *GIT* frei zur Verfügung.[1] Die aktuelle Version ist *0.9.2 beta*.
Es unterstützt die statische Analyse von Apps in den Formaten *APK* und *IPA* sowie aus einfach komprimierten Archiven (*zip*). Zusätzlich beinhaltet *MobSF* einen eingebauten API Fuzzer und ist in der Lage, API-spezifische Schwachstellen wie XXE, SSRF oder Path Traversal zu erkennen (TODO Auflisten).

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

Dex2Jar (d2j): https://github.com/pxb1988/dex2jar

Dex2Smali: https://github.com/JesusFreke/smali

Jar2Java: asasd

enjarify: https://github.com/google/enjarify

[1] https://github.com/ajinabraham/Mobile-Security-Framework-MobSF

procyon : https://bitbucket.org/mstrobel/procyon/overview

Für iOS-Apps:

otool: -L -hv -Iv

[-L] Test

•

4.2 Aktuelle Situation und Vergleich

4.2.1 IOS

Emulation

Die Emulation von iOS-Geräten ist derzeit mit der Verwendung von XCode möglich. XCode wiederum ist nur unter Mac OSX erhältlich. Da Max OSX laut EULA nur auf Äpple-branded computers"verwendet werden darf [**AppleEULA**], ist die Simulation von iOS-Geräten nur unter Apple-Hardware möglich. Nach der Installation über den in Mac OSX enthaltenen App-Store kann ein virtualisierter IPhone über die Schritte XCode, Open Developer Tools, Simulator gestartet werden.

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 https://opensource.org/licenses/UoI-NCSA.php Lizenz, welche die Vervielfältigung und Veränderung des Quellcodes unter 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 unter http://lldb.llvm.org/lldb-gdb.html 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.
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 offen 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 dem Artikel https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/gdb_to_lldb_transition_guide/document/lldb-terminal-workflow-tutorial.html von Apple, ist es

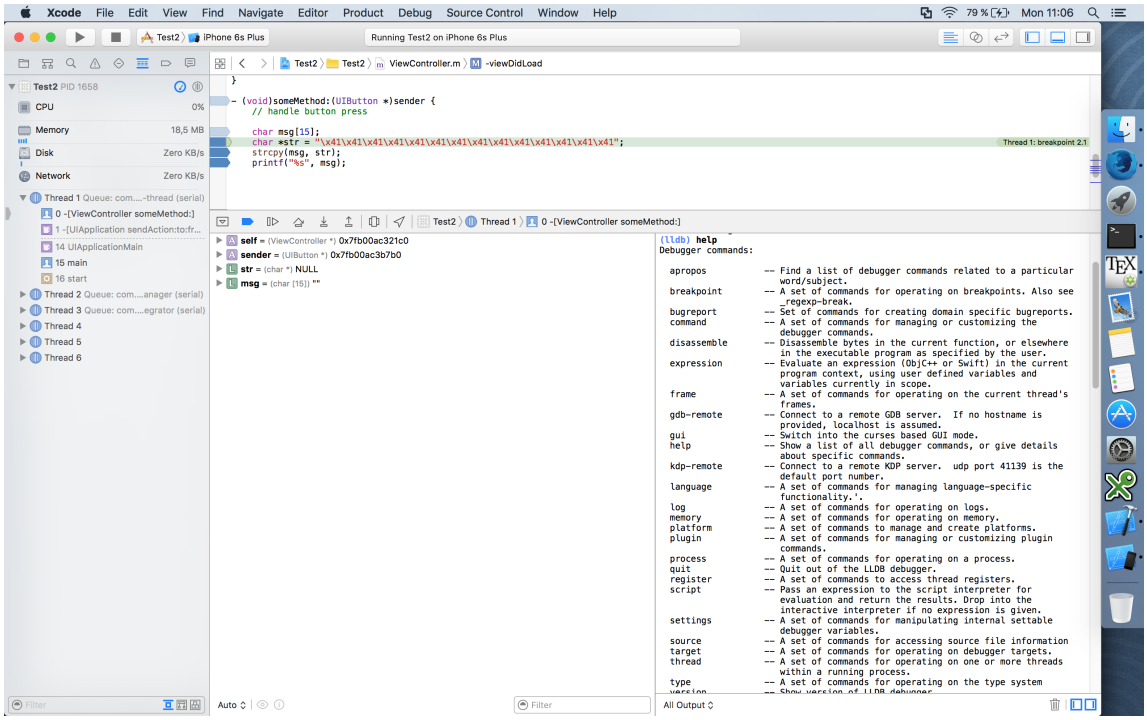

Abbildung 4.1: LLDB in XCode

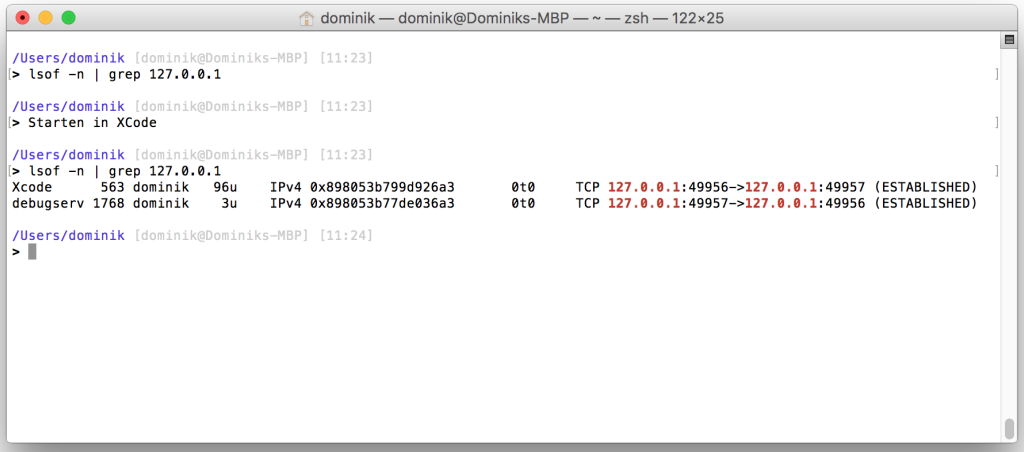

Abbildung 4.2: Vergleich der offenen Pipes vor und nach der Ausführung der Applikation in XCode

möglich, mit LLDB eine App auch als SStandalone Debugger", also ohne XCode, zu verwenden. Dies ist in Abbildung 4.3 aufgezeigt.

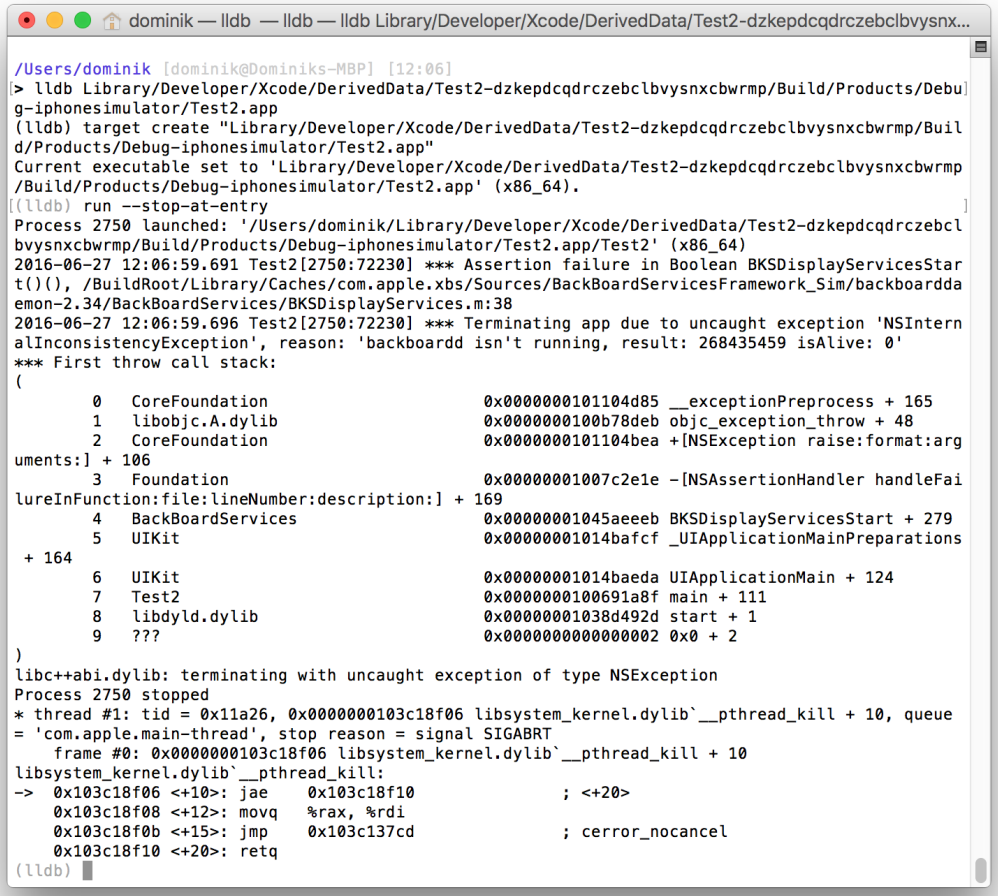

Abbildung 4.3: LLDB als Standalone Debugger

Um zu verifizieren, dass die App auf einem simulierten IPhone ausgeführt wird, können entweder die geöffneten Prozesse (siehe Abbildung 4.2.1) oder die geladenen Bibliotheken der Programme (siehe Abbildung 4.4) verglichen werden.
Bei Methoden zeigen, dass die App auf dem simulierten IPhone gestartet wird. Bei den Prozesse ist zu beobachten, dass vor Start von LLDB nur der Hintergrund-Service ausgeführt wurde. Nach dem Start von LLDB dagegen, läuft der gesamte Simulator.
Auch der Vergleich der geladenen Libaries legt nahe, dass die LLDB Standalone und XCode in der gleichen Umgebung ausgeführt werden. Die Adressen im RAM variieren aufgrund von ASLR zwar, aber es werden die Bibliotheken verwendet (am Pfad zu erkennen).

Memory Corruption Bad Functions

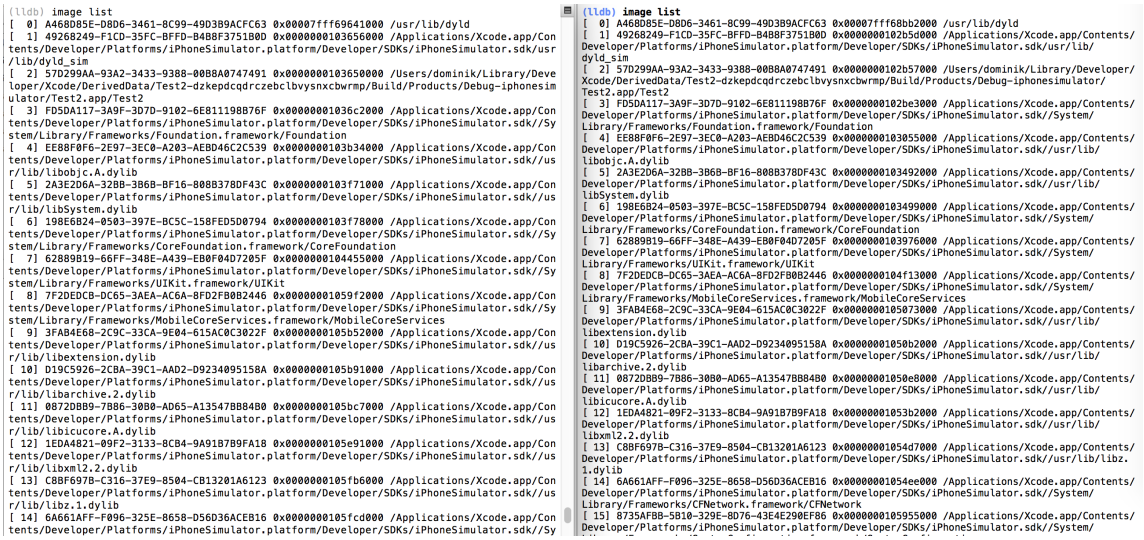

Abbildung 4.4: LLDB als Standalone Debugger

Ungesicherte Verbindungen

https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/Coco
unter NSAppTransportSecurity

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

```
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.
```

4.2.2 Windows-Phone

Emulation vs. Hardware

VS

Debugging

VS

4.2.3 Android

Android ist ein Ursprünglich 2003 von der Android, Inc. entwickeltes mobiles Betriebssystem. 2005 wurde es durch Google übernommen und wird seit dem weiterentwickelt. 2015 liegt es bei einem Marktanteil von TODO %. Aufgrund der Quelloffenheit des Systems wird von

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

Listing 4.1: Gestartete Prozesse nach LLDB Aufruf

vielen Herstellern auf verschiedensten Plattformen genutzt. Jedoch bringt die weitführende Fragmentierung des Betriebssystems auch Nachteile mit sich. So sind in 2015 nur TODO % der Android-Devices auf einer aktuellen Version.[**Drake2014**]

Android-Studio und SDK

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

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

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

Compatibility Testing Suite

[**Drake2014**] Seite 18

Emulation vs. Hardware

Android SDK

Debugging

Android Debug Bridge[**androidDebugBridge**]

Logcat

Android Debug Bridge[**androidDebugBridge**]

4.3 Anforderungen und Abgleich mit MobSF

- Automatisierung
- Blackbox/Whitebox
- Reporting
- False-Positive-Rate

4.4 Weiterentwicklung MobSF

4.4.1 Struktur

```
StaticAnalyzer/
├── __init__.py
├── admin.py
├── dvm_permissions.py
├── models.py
├── tests.py
├── views.py
└── tools/
    ├── __init__.py
    ├── AXMLPrinter2.jar
    ├── baksmali.jar
    ├── CertPrint.jar
    ├── cfr_0_115.jar
    ├── jd-core.jar
    ├── procyon-decompiler-0.5.30.jar
    ├── strings_from_apk.jar
    ├── d2j2/
    │   ├── d2j-baksmali.bat
    │   ├── [...]
    │   └── lib/
    │       ├── antlr-runtime-3.5.jar
    │       └── [...]
    ├── enjarify/
    │   └── [...]
    └── mac/
        └── class-dump-z
```

4.5 Laboraufbau

4.6 Entwicklung der Umgebung

4.6.1 Aufbau

4.6.2 Schnittstellen

4.6.3 RPC-Service

Um eine Kommunikation zwischen den verschiedenen Virtuellen Maschienen zu ermöglichen, wurde ein minimaler RPC-Server in Python 3.5 entwickelt. Verwendet wurden hierzu die Bibliotheken *Flask* und *Requests*.

```
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 4.5: rpc_client.py

```
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 4.6: rpc_server.py

Flask

Flask ist ein schneller, minimaler Webserver. Mit nur sehr wenig Code es möglich, eine Schnittstelle bereit zu stellen. Ein Code mit zwei akzeptierenden Funktionen, basierend auf der Schnellstart-Anleitung[2], ist in Abbildung 4.6.3 dargestellt.
Es wird eine minimale Anwendung erstellt, welche auf dem Pfad "/"lokal das print-Statement ausführt und den Text 'Hello World!' zurück gibt. Wir die Anwendung unter dem Pfad "/second_command/ängesprochen, wird ein anderes print-Statement ausgeführt und ein anderer Wert zurück gegeben. Auf diese Weise können schnell API-Funktionen auf verschiedene Pfade gelegt und angesprochen werden.

Requests

Requests ist eine Python-Bibliothek, welche einfache Anfragen (sogenannte Requests, daher der Name) an Web-Server ermöglicht. So ist es über ein kurzes Code-Snippet, dargestellt in Abbildung 4.6.3, möglich die unter 4.6.3 aufgezeigt Schnittstelle anzusprechen.
Wird zuerst der Code 4.6.3 und anschließend der Code 4.6.3 ausgeführt, wird auf Server-Seite folgende Ausgabe erzeugt:

[2] http://flask.pocoo.org/docs/0.10/quickstart/

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

Auf der Client-Seite erfolgt folgende Ausgabe:

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

Aufgrund dieser Basis wurde das RPC-Tool implementiert.

4.6.4 Technisches Detail 1

4.6.5 Technisches Detail 2

4.7 Abgleich mit Anforderungen

5 Anwendung der Umgebung

5.1 Pentest Anwendung 1

5.2 Pentest Anwendung 2

6 Fazit