MISS
Mac-based Identification and Signaling Service

Fabian Schwab (fabian.schwab@uni-ulm.de)

24. September 2014

Kapitel 1

Einleitung

Der *mac-based identification and signaling service* ist ein für Android geschriebener Hintergrunddienst, welcher es ermöglicht mobile oder stationäre WLAN Geräte zu erkennen. Der Service unterscheidet zwischen mobilen Geräten, die hier als *Clients* bezeichnet werden und Stationen den sogenannten *Stations*. Hierbei sind die *Clients* Geräte die sich mit einem WLANs verbinden und *Stations* Geräte, die solch ein Netzwerk aufspannen. Je nach Typ werden verschiedene Informationen gesammelt und zurückgegeben, welche in Kapitel **??** genauer erläutert werden.
Sobald sich eines der gesuchten Geräte in der Nähe befindet und erkannt wird, wird eine Benachrichtigung über dessen Fund ausgelöst.

Kapitel 2

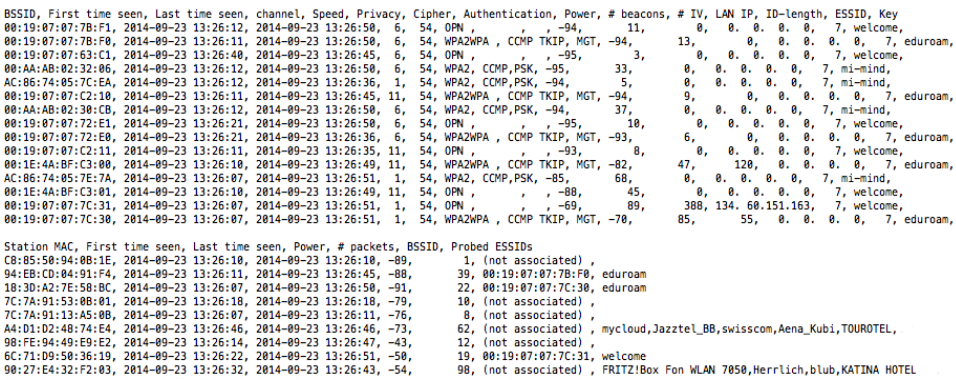
Grundlagen

Um die Funktionsweise des Service genauer verstehen zu können, sollten zunächst einige Grundlagen erklärt werden. In diesem Abschnitt werden die vom Service genutzten Technologien kurz beschrieben. Da dies keine vollständige Beschreibung darstellt, sollten die Technologien in ihren Grundzügen bekannt sein.

2.1 Kabellose Kommunikation

Die Anzahl der Geräte die mithilfe von WLAN kommunizieren nehmen ständig zu. Bei einem Großteil dieser Geräte wird das *WNIC*[1], manchmal trotz limitierter Ressourcen, nicht abgeschaltet. Trotz verschlüsselter Verbindungen werden definiert durch den *IEEE 802.11* Standard einige Daten in Klartext übertragen. Diese Daten werden von MISS genutzt um Geräte in der Nähe zu erkennen und bekannte zu identifizieren.
Durch den *IEEE 802.11* Standard werden auf der Sicherungsschicht des *OSI-Modell* Datagramme, sogenannte *Frames* definiert welche in spezielle Teile unterteilt sind. Jeder *Frame* enthält ein *MAC header* Feld, in dem die Absender MAC-Adresse im Klartext dargestellt ist. Durch belauschen der umliegenden Kommunikation kann so nach einer bestimmten MAC-Adresse gesucht werden. Aber nicht nur Geräte die im Moment Daten austauschen können erkannt werden, sonder auch Geräte die inaktiv sind. Jedes WLAN fähige Gerät senden abhängig von der Implementierung des *WNIC* Treibers periodische Pakete aus. Diese Pakete gehören zu der Gruppe der *Managementframes*.
Bei den *probe request frames* handelt es sich um Pakete die von *Clients* gesendet werden um Informationen über Netzwerke zu sammeln. Durch die Angabe einer *SSID*[2] kann im Kommunikationsbereich des Gerätes nach bestimmten Netzwerken gesucht werden oder durch Angabe der *boradcast SSID* kann nach allen Netzwerken gesucht werden die dieses Paket empfangen und darauf Antworten. Diese Pakete werden periodisch versendet egal ob das Gerät bereits mit einem

[1] Wireless Network Interface Controller
[2] Service Set Identifier



```
BSSID, First time seen, Last time seen, channel, Speed, Privacy, Cipher, Authentication, Power, # beacons, # IV, LAN IP, ID-length, ESSID, Key
00:19:07:07:7B:F1, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, OPN ,       ,   , -94,       11,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:7B:F0, 2014-09-23 13:26:11, 2014-09-23 13:26:50,  6,  54, WPA2WPA , CCMP TKIP, MGT, -94,      13,        0,   0.  0.  0.  0,   7, eduroam,
00:19:07:07:63:C1, 2014-09-23 13:26:40, 2014-09-23 13:26:45,  6,  54, OPN ,       ,   , -95,        3,        0,   0.  0.  0.  0,   7, welcome,
00:AA:AB:02:32:06, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, WPA2, CCMP,PSK, -95,       33,        0,   0.  0.  0.  0,   7, mi-mind,
AC:86:74:05:7C:EA, 2014-09-23 13:26:12, 2014-09-23 13:26:36,  1,  54, WPA2, CCMP,PSK, -94,        5,        0,   0.  0.  0.  0,   7, mi-mind,
00:19:07:07:C2:10, 2014-09-23 13:26:11, 2014-09-23 13:26:45, 11,  54, WPA2WPA , CCMP TKIP, MGT, -94,       9,        0,   0.  0.  0.  0,   7, eduroam,
00:AA:AB:02:30:CB, 2014-09-23 13:26:12, 2014-09-23 13:26:50,  6,  54, WPA2, CCMP,PSK, -94,       37,        0,   0.  0.  0.  0,   7, mi-mind,
00:19:07:07:72:E1, 2014-09-23 13:26:21, 2014-09-23 13:26:50,  6,  54, OPN ,       ,   , -95,       10,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:72:E0, 2014-09-23 13:26:21, 2014-09-23 13:26:36,  6,  54, WPA2WPA , CCMP TKIP, MGT, -93,       6,        0,   0.  0.  0.  0,   7, eduroam,
00:19:07:07:C2:11, 2014-09-23 13:26:11, 2014-09-23 13:26:35, 11,  54, OPN ,       ,   , -93,       8,        0,   0.  0.  0.  0,   7, welcome,
00:1E:4A:BF:C3:00, 2014-09-23 13:26:10, 2014-09-23 13:26:49, 11,  54, WPA2WPA , CCMP TKIP, MGT, -82,      47,      120,   0.  0.  0.  0,   7, eduroam,
AC:86:74:05:7E:7A, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, WPA2, CCMP,PSK, -85,       68,        0,   0.  0.  0.  0,   7, mi-mind,
00:1E:4A:BF:C3:01, 2014-09-23 13:26:10, 2014-09-23 13:26:49, 11,  54, OPN ,       ,   , -88,       45,        0,   0.  0.  0.  0,   7, welcome,
00:19:07:07:7C:31, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, OPN ,       ,   , -69,       89,      388, 134. 60.151.163,   7, welcome,
00:19:07:07:7C:30, 2014-09-23 13:26:07, 2014-09-23 13:26:51,  1,  54, WPA2WPA , CCMP TKIP, MGT, -70,      85,       55,   0.  0.  0.  0,   7, eduroam,

Station MAC, First time seen, Last time seen, Power, # packets, BSSID, Probed ESSIDs
C8:85:50:94:0B:1E, 2014-09-23 13:26:10, 2014-09-23 13:26:10, -89,        1, (not associated) ,
94:EB:CD:04:91:F4, 2014-09-23 13:26:11, 2014-09-23 13:26:45, -88,       39, 00:19:07:07:7B:F0, eduroam
18:3D:A2:7E:58:BC, 2014-09-23 13:26:07, 2014-09-23 13:26:50, -91,       22, 00:19:07:07:7C:30, eduroam
7C:7A:91:53:0B:01, 2014-09-23 13:26:18, 2014-09-23 13:26:18, -79,       10, (not associated) ,
7C:7A:91:13:A5:0B, 2014-09-23 13:26:07, 2014-09-23 13:26:11, -76,        8, (not associated) ,
A4:D1:D2:48:74:E4, 2014-09-23 13:26:46, 2014-09-23 13:26:46, -73,       62, (not associated) , mycloud,Jazztel_BB,swisscom,Aena_Kubi,TOUROTEL,
98:FE:94:49:E9:E2, 2014-09-23 13:26:14, 2014-09-23 13:26:47, -43,       12, (not associated) ,
6C:71:D9:50:36:19, 2014-09-23 13:26:22, 2014-09-23 13:26:51, -50,       19, 00:19:07:07:7C:31, welcome
90:27:E4:32:F2:03, 2014-09-23 13:26:32, 2014-09-23 13:26:43, -54,       98, (not associated) , FRITZ!Box Fon WLAN 7050,Herrlich,blub,KATINA HOTEL
```

Abbildung 2.1: Mit airodump-ng erstellte Log-Datei

Netzwerk verbunden ist oder nicht.
Stations senden sogenannte *beacon frames* aus. Damit geben sie periodisch ihre Präsenz, *SSID* und andere Parameter bekannt.
Im Standard Betriebsmodus eines *WNIC* ist es nicht vorgesehen, dass Pakete, die an andere MAC-Adressen gerichtet sind an höhere Schichten weitergeleitet werden. Diese Pakete werden bereits auf der zweiten Schicht dem *Data Link Layer* verworfen. Damit aber auch diese Pakete weitergeleitet werden, ist es notwendig den Betriebsmodus des *WNIC* auf den sogenannten *Monitor Mode* zu setzten. Dieser Mode wird über den Treiber des *WNIC* eingestellt um muss dort auch implementiert sein.

2.1.1 Airodump-ng

Airodump-ng[3] ist ein Kommandozeilenprogramm welches für das erfassen von *IEEE 802.11 frames* genutzt wird und ein Teil der *Aircrack-ng suit*[4] ist. Mit diesem Programm lassen sich Log-Dateien erstellen welche Informationen über alle erfassten *Stations* und *Clients* enthalten. Eine Auszug von solch einer Log-Datei ist in Abbildung 2.1 zusehen. Die nachfolgenden Auflistungen beschreiben stichpunktartig die erfassbaren Daten für *Stations* und *Clients*.

[3] http://www.aircrack-ng.org/doku.php?id=airodump-ng
[4] http://www.aircrack-ng.org

Erfassbare *Station* Daten	Beschreibung
BSSID	MAC-Adresse der *Station*
First time seen	Datum und Uhrzeit der ersten Kontakts
Last time seen	Datum und Uhrzeit des letzten Kontakts
channel	Kanal auf der die *Station* sendet
Speed	Übertragungsgeschwindigkeit in MBit/s
Privacy	Privatsphäreneinstellungen
Cipher	Art der Verschlüsselung
Authentication	Genutztes Authentifizierungsprotokoll
Power	Signalstärke
# beacons	Anzahl der Empfangenen *beacon frames*
# IV	Anzahl der Erkannten Initialisierungsvektoren
LAN IP	IP Adresse
ID-length	Länge der ESSID
ESSID	Netzwerkname
Key	Netzwerkschlüssel falls bekannt (genutzt mit aircrack-ng)

Erfassbare *Client* Daten	Beschreibung
Station MAC	MAC-Adresse *Clients*
First time seen	Datum und Uhrzeit der ersten Kontakts
Last time seen	Datum und Uhrzeit der letzten Kontakts
Power	Signalstärke
# packets	Anzahl der Empfangenen *frames*
BSSID	BSSID mit dem der *Client* verbunden ist
Probed ESSIDs	ESSID die der *Client* sucht

2.2 Android

Bevor der eigentliche Hintergrunddienst installiert werden kann und funktionsfähig ist, müssen einige Vorbereitungen getroffen werden. Da Google in seinem offenen Betriebssystem Android keine Möglichkeit bietet einen *Monitor Mode* zu aktivieren, implementieren die meisten Hersteller diesen auch nicht für ihre WLAN Module.
Aktuell gibt es kein Android Gerät welches einen *WNIC* besitzt, für den es einem vom Hersteller implementierten *Monitor Mode* gibt. Für bestimmte Chipsätze des Herstellers Broadcom gibt es eine kleine Gruppe von Programmierer, die für die Chipsätze *BCM4330* und *BCM4329* eine neue Firmware geschrieben haben um eben diesen Modus zu aktivieren. Momentan werden folgende Geräte mit diesem Chipsatz unterstützt:

- Samsung Galaxy GS1 mit Cyanogen 7
- Samsung Galaxy GS2 mit Cyanogen 9 oder Cynaogen 10

- HTC Nexus One mit Cyanogen 7
- Asus Nexus 7 mit Cyanogen 7

2.2.1 Cyanogen

Alle zuvor ausgelisteten Geräte nutzen Cyanogen[5]. Cyanogen ist eine erweiterte *open source* Firmware Distribution für Smartphones und Tablets welche auf den Android Betriebssystem basiert. Diese bietet Eigenschaften und Erweiterungen die es in der offiziellen oder in der des Herstellers ausgelieferten Firmware nicht gibt.
Um Cyanogen auf einem Unterstützten Gerät zu installieren, muss es zuvor *gerootet* werden. Diese bedeutet, das man alle Rechte erlangt was normalerweise auch aus Sicherheitsgründen so nicht vorgesehen ist.

2.2.2 bcmon.apk

Nachdem das Gerät *gerootet* wurde und Cyanogen installiert wurde. Wird das kleine Programm *bcmon*[6] installiert. Mithilfe dieses Programms, welches die neue Firmware für den *WNIC* enthält, kann der *Monitor Mode* aktiviert werden. Anschließend ist das Gerät für die Verwendung des MISS Hintergrunddienstes vorbeireitet.

[5] http://www.cyanogenmod.org
[6] http://bcmon.blogspot.de

Kapitel 3

Architektur und Implementierung

In diesem Kapitel wird zunächst die grundlegende Architektur des verwendeten Dienstes erläutert und anschließend auf die Implementierung eingegangen. Dieses und nachfolgende Kapitel setzten voraus, dass Grundlagen der Programmierung in Java und Android bekannt sind. Sind diese nicht bekan nt, wird an dieser Stelle auf die Android-Entwicklerseite[1] verwiesen.

3.1 Architektur

3.1.1 Android IntentService

Ein Service ist eine im Hintergrund laufende Komponente welche keine direkte Interaktion mit einem Nutzer besitzt. Da ein Service kein Benutzeroberfläche benötigt, ist ein dieser auch nicht an den Lebenszyklus einer *activity* gebunden. Im Allgemeinen werden Services genutzt um wiederkehrende und potentiell lange andauernde Aufgaben zu erledigen wie Beispielsweise das herunterladen von Inhalten aus dem Internet oder das aktualisieren von Daten. Ebenfalls werden Services mit einer höheren Priorität als sich im Hintergrund befindlichen Anwendungen ausgeführt und daher ist es unwahrscheinlicher, dass sie vom Betriebssystem abgeschaltet werden.
Zusätzlich können Services unter Android so Konfiguriert werden, dass sie neu gestartet werden sollte das Betriebssystem sie beenden.
Da der in diesem Projekt genutzte Service nicht immer aktiv sein soll, wird hier auf eine besondere Art eines Service zurückgegriffen. Bei dieser Art von Hintergrunddienst handelt es sich um einen in *intentService* welcher von der Klasse *Service* erbt. Der Lebenszyklus eines *intentService* ist in Abbildung 3.1 abgebildet. Hierbei wird ersichtlich, dass diese Art von Service nicht immer im Hintergrund aktiv ist. Nur wenn ein Client an der Service gebunden ist wird, ist

[1] http://developer.android.com

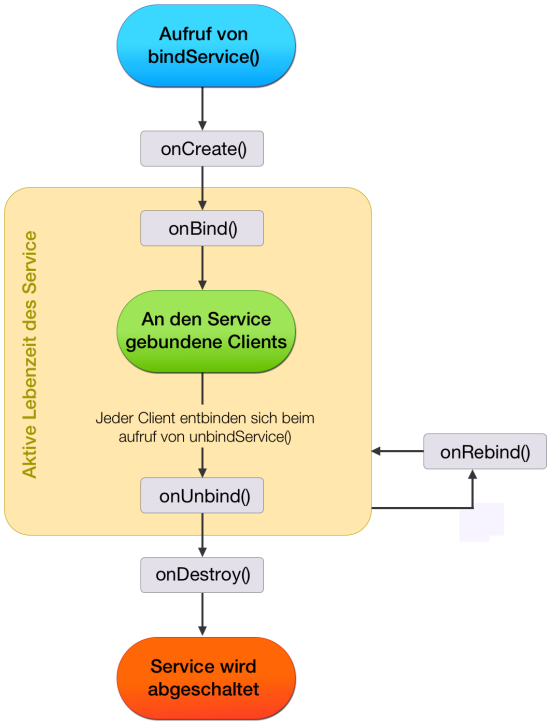


Abbildung 3.1: Lebenszyklus eines *intentService*

dieser aktiv. Im Gegensatz zu seiner Basisklasse, wobei ein Aufruf von *stopService()* genügt um diesen zu beenden, wird ein *intentService* erst beendet, wenn sich alle Clients entbunden haben.

3.1.2 Service Logik

Der Aufbau des Service wie er Implementiert werden soll ist in Abbildung 3.2 ersichtlich. Bindet sich eine Anwendung am Service wird dieser, falls er noch nicht aktiv ist, gestartet. Der Service verfügt über einen eigenen Thread, welcher die eigentliche Arbeit übernimmt. Dies ermöglicht ein sofortiges abarbeiten weiterer Anfragen von bereits gebundenen Anwendungen oder wenn sich weitere Anwendungen an den Service binden möchten.
Aus Effizienzgründen wird der Thread nur gestartet wenn Geräte gesucht werden. Diese werden zu suchenden Geräte werden an den Service übermittelt, der daraufhin den Thread nach bedarf startet oder stoppt.
Die Aufgabe des Threads ist es alle Geräte in der Nähe zu erfassen. Dabei sollen alle zu erfassenden Daten, wie in Abschnitt 2.1.1 gezeigt, erfasst werden. Anschließend werden die gefunden Geräte mit den gesuchten Verglichen. Befindet sich ein gesuchtes unter den gefunden so benachrichtigt der Thread den Service.

Der Service benachrichtigt die entsprechende Anwendung über den Fund des angefragten Gerätes.
Je nach Implementierung der Zielanwendung entscheidet diese über das weitere vorgehen. In der Regel wird davon ausgegangen das die Anwendung eine Aktion auslöst und das gesuchte Gerät nicht mehr benötigt wird und es dem Service mitteilt. Durch eine Nachricht an den Service wird das zuvor gescuhte Gerät entfernt.
Der Service überprüft bei jedem Empfang einer Nachricht ob der Arbeiter-Thread gestartet werden muss oder nicht. Das entscheiden Kriterium ist, ob sich gesuchte Geräte im Service befinden. Antwortet eine Anwendung auf den Fund eines seiner Gesuchten Geräte nicht, wird davon ausgegangen das die Anwendung unerwartet beendet wurde. Der Service entfernt die Anwendung und all ihrer Geräte welche im Zusammenhang mit ihr stehen.
Befinden sich keine gesuchten Geräte im Dienst, beendet dieser den Arbeiter-Thread. Haben sich alle Anwendungen ordnungsgemäß vom Service abgemeldet oder wurden durch eine ausbleibende Antwort entfernt, beendet sich der Service selbst da er nicht mehr benötigt wird.

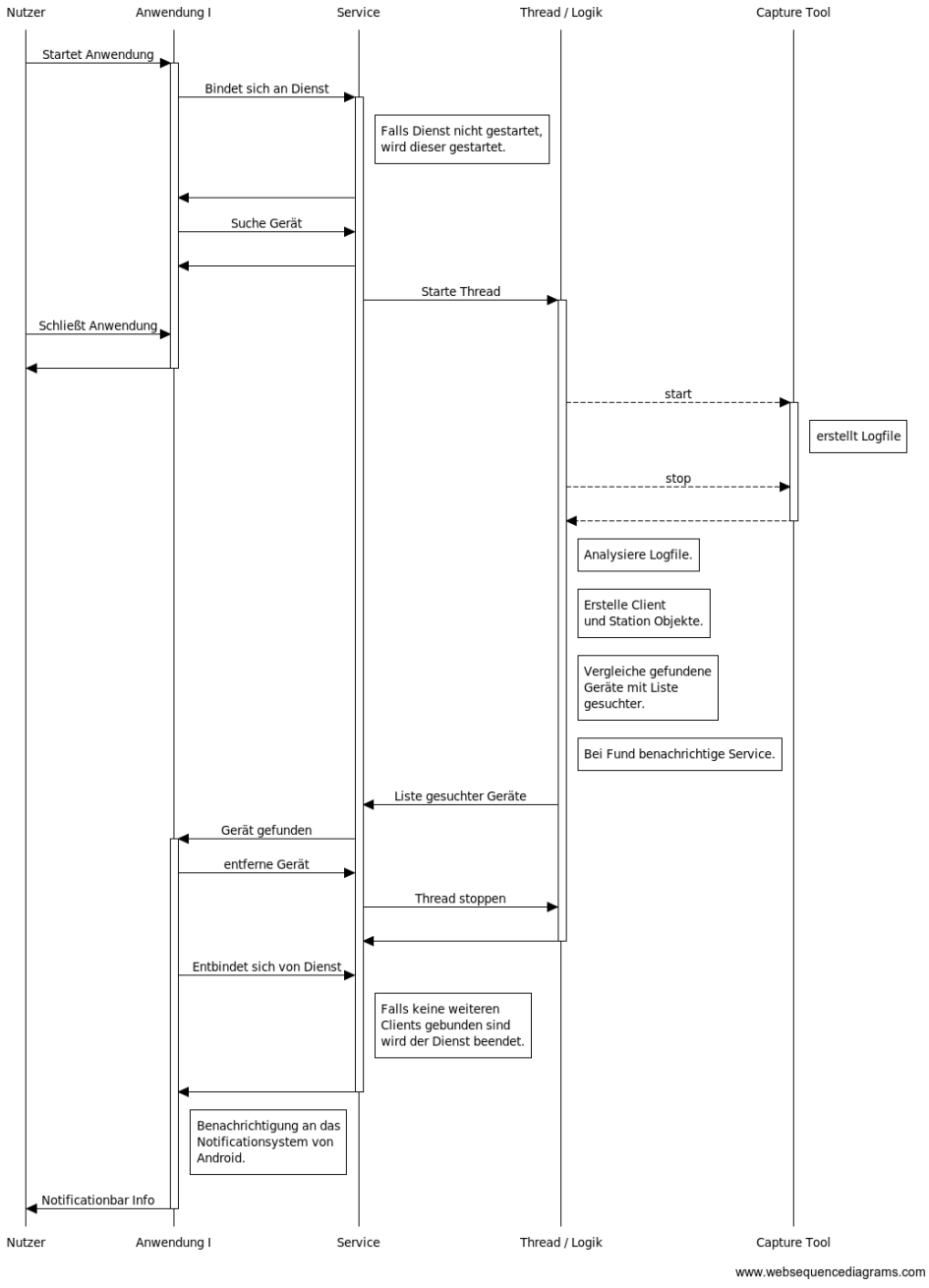


Abbildung 3.2: Anwendungsfall und Lebenszyklus des MISS als UML-Sequenzdiagramm.

3.2 Implementierung

Für die Implementierung der voran gezeigten Architektur wurde das Eclipse ADT[2] verwendet. Als Zielgerät wurde das Samsung Galaxy GS1 mit Cyanogen 7 gewählt. Hierbei war zu beachten, das bei der Programmierung nur Eigenschaften bis API Level 10 zu Nutzen sind. Dies liegt an der geringen Android Version 2.3.3 welche dem Cyanogen 7 Mod zu Grunde liegt.
Da im späteren Betrieb des Service Skripte benötigt werden, wurden diese zur besseren Kapselung und Entwicklung als *.sh* Datei in *assets* gespeichert. Dies hat den entscheidenden Vorteil, dass die Skripte nicht vom Nutzer separat nach der Installation im Programmverzeichnis abgelegt werden müssen sondern programmatisch generiert werden können.
Folgende Skripts werden bei der Installation erstellt:

- removeCaptureFiles.sh
- startCapture.sh
- stopCapture.sh

Alle Skripte werden ausschließlich von später gezeigten Arbeiter-Thread genutzt. Wobei *removeCaptureFiles.sh* lediglich zum entfernen der anfallenden Log-Datei genutzt wird.
Das Starten und Stoppen von *airodump-ng* wird durch die beiden anderen Skripte veranlasst, wobei *startCapture.sh* wie in Listing 3.1 einige Parameter enthält. Unter anderem wird hier das Ausgabeformat festgelegt und der *WNIC* Name angegeben. Des weiteren werden die Umgebungsvariablen ergänzt um gewisse Bibliotheken für *airodump-ng* bereitzuhalten um eine fehlerfreie Ausführung zu gewähren.

Listing 3.1: Airodump-ng Parameter

```
1 export PATH=
2  $PATH:/data/data/com.bcmon.bcmon/files/tools
3 export LD_LIBRARY_PATH=
4  $LD_LIBRARY_PATH:/data/data/com.bcmon.bcmon/files/libs
5 export LD_PRELOAD=
6  /data/data/com.bcmon.bcmon/files/libs/
7  libfake_driver.so
8 airodump-ng -w /datadata/de.uulm.miss/files/capture
9  --output-format csv -w capture wlan0 2>&1
```

In der nachfolgenden Auflistung werden alle Klassen und eine dazugehörige Beschreibung des MISS aufgeführt.

[2] http://developer.android.com/sdk/index.html

Klassenname	Beschreibung
MainActivity.java	Wird nur bei der Installation geöffnet und erzeugt alle nötigen Skripte.
MISService.java	Nimmt Anwendungsanfrage entgegen und erzeugt und kontrolliert den Arbeiter-Thread.
ScanLogic.java	Arbeiter-Thread der mithilfe der Skripte alle Geräte findet und mit zu suchenden Vergleicht.
FileParser.java	Erstellt Client und Station Objekte anhand des Logfiles.
Client.java	Objekt welche alle für Clients erfassbaren Daten enthält.
Station.java	Objekt welche alle für Stations erfassbaren Daten enthält.
ScanOrder.java	Enthält die Liste von gesuchten Stations und Clients.

Auf eine ausführliche Beschreibung der Funktionen und Parameter wurde an dieser Stelle verzichtet, da diese aus dem Quellcode entnommen werden kann. Um diesen Service zu Nutzen kann die beiliegende Anwendung, *PAR* welche in Abschnitt 3.2.2 erläutert wird, genutzt werden.
Die Basisfunktionen die der Service nach außen hin bietet werden im nächsten Kapitel 3.2.1 erklärt.

3.2.1 Nutzung des Service

Für die Nutzung des Service müssen folgende Punkte erfüllt sein:

- Ein unterstützter Chipsatz (BCM4330 bzw. BCM4329) wurde verbaut.
- Das Gerät ermöglicht *root* zugriff
- Eine kompatible Cyanogen Firmware ist aufgesetzt
- Die bcmon Anwendung ist installiert.
- MISS installiert und *root* Rechte bewilligt.
- Aktivierter *Monitor Mode* via bcmon

3.2.2 Anwendungsprogramm PAR