Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Seminar 3
WiSe 2011/2012

Theodor Nolte

Implementierungsframework für die Schadsoftwareerkennung auf Android

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Zielvorgabe des SKIMS-Projekts [5] ist die Konzeption, Entwicklung und Analyse einer schichtenübergreifenden kooperativen Sicherheits-Umgebung für mobile Geräte. Hierbei stellt das Erkennen von Schadsoftware einen wichtigen Grundstock dar, um darauffolgend Gegenmaßnahmen ergreifen zu können. Für das Auffinden von Schadsoftware kommen grundsätzlich unterschiedliche Mechanismen in Frage. Ein Vorgehen beispielweise ist die Entropie-Analyse von Datenströmen im Netzwerk. Dieser Ansatz wird von unserem Kommilitonen Benjamin Jochheim untersucht und weiterentwickelt. [3] Jedoch sind auch andere Ansätze möglich, etwa das Überprüfen von Dateien anhand von Virensignaturen, wie es von klassischen Antivirenprogrammen angewendet wird.

Das Szenario mobiler Endgeräte unterscheidet sich signifikant von konventionellen Endbenuter-Systemen:

- Die Ressourcen-Verfügbarkeit sowie die Rechenkapazitäten von mobilen Endgeräten sind beschränkt.
- Die Tatsache, dass mobile Endgeräte als ein *Single Spot* privater Daten verwendet werden, macht sie interessant für Angriffe.
- Unterschiedliche Funk-Schnittstellen ermöglichen neue Angriffsformen: Der *„Äther“* ist ein inhärent allen zugängliches Übertragungsmedium, zugleich sind jedoch Angriffe über dieses Medium lokal begrenzt.

Methoden zur Entdeckung von Schadsoftware haben diesen Besonderheiten Rechnung zu tragen.

Hier hat man es also im Gegensatz zur klassischen PC-Landschaft mit veränderten Rahmenbedingungen zu tun, so dass ein flexibles Rahmenwerk für die Schadsoftwareerkennung wünschenswert wäre. Dies würde es einerseits ermöglichen, sowohl bewährte, auf mobile Umgebungen adaptierte Verfahren einzusetzen, als auch neue Ansätze und Verfahren zur Anwendung einzubringen.

1.2 Inhaltlicher Aufbau der Ausarbeitung

In Abschnitt 2 wird der Ansatz zur Architektur des Frameworks diskutiert. In Abschnitt 3 zeige ich die bisherige Umsetzung für die Android-Umgebung dar. Und in Abschnitt 4 wird nach einer Zusammenfassung noch ein Ausblick gegeben.

2 Framework

2.1 Anforderungen und Abgrenzung

Ziel des Frameworks ist es, unterschiedliche Methoden zur Erkennung von Schadsoftware auf einem mobilen Geräte anzuwenden und somit den Schutz vor solchen schädlichen Programmen zu erhöhen. Das Framework soll nicht den bei gefundenem Schadcode notwendigen Abwehrmechanismus ausführen. Hiermit befassen sich weitere Komponenten im Rahmen des SKIMS-Projektes, mit denen das Framework interagieren können soll.

Insbesondere soll das Framework ein Verfahren zur Erkennung von Schadsoftware ermöglichen, dass sich auf die von Herrn Jochheim vorangetriebene Entropie-Analyse stützt. Aber auch weitere Verfahren sollen mit dem Framework anwendbar sein.

Das Suchen von Schadsoftware sollte nicht vor Grenzen des Systems Halt machen. So müssen Verfahren, die das Dateisystem untersuchen (z.B. Dateiscanner), die Möglichkeit besitzen auf sämtliche Dateien zuzugreifen. Hierfür sind Root-Rechte notwendig. Auch für das Mitschneiden des Netzwerkverkehrs sind Root-Rechte Voraussetzung. Weil von der Implementierung des Framworks ausgegangen wird, dass sie zunächst nicht über einen prototypischen Ansatz hinaus geht, ist als Workaround zugelassen, dass das die Android-Anwendung auf einem gerooteten Gerät ausgeführt wird. Rooten bedeutet, dass das System verändert wird, so dass das Kommando *su* den Android-Anwendungen (Apps) zur verfügung steht.

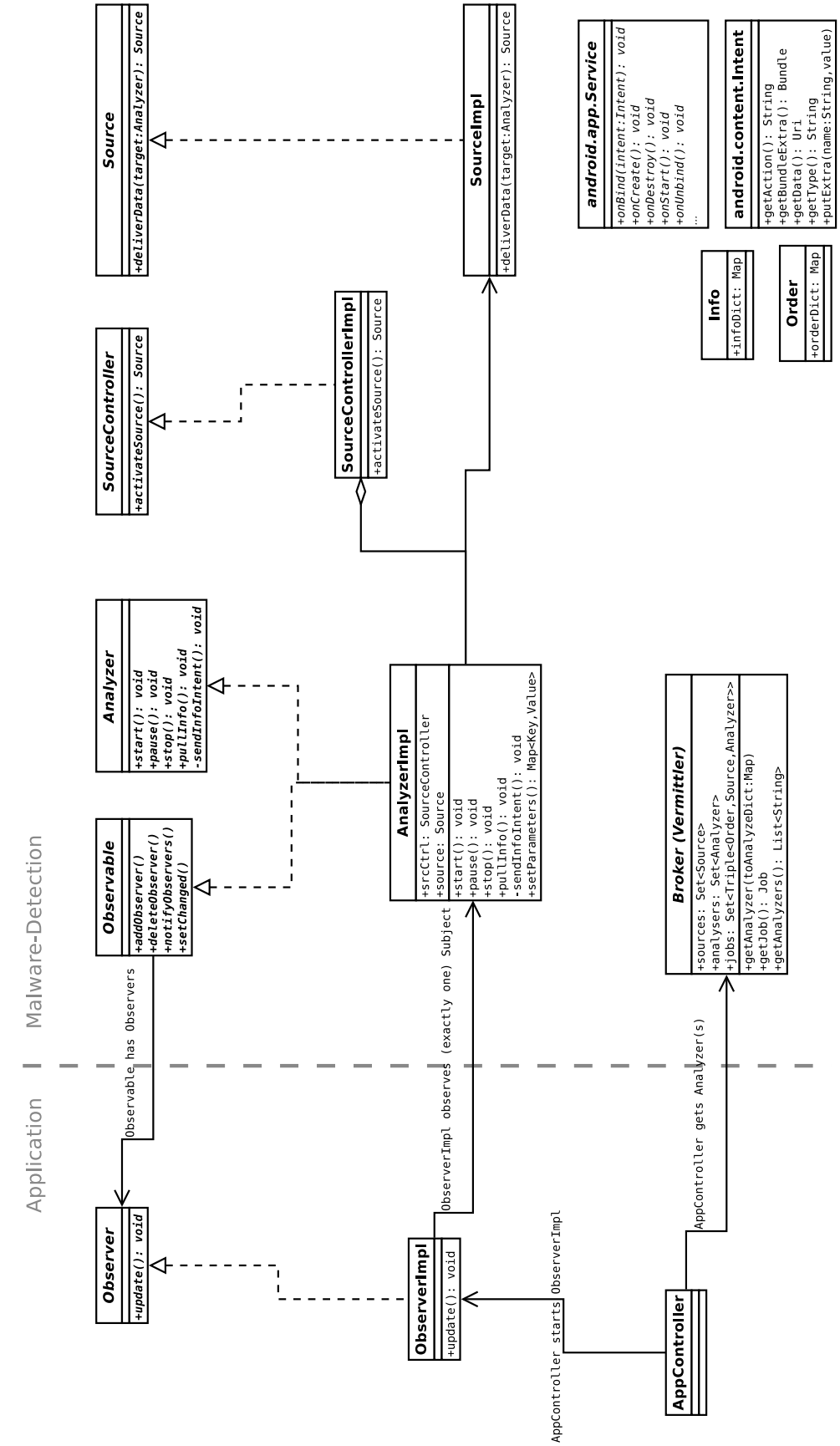
2.2 Erster Entwurf eines Frameworks

Abbildung 1 zeigt das Klassendiagramm des ersten Framework-Entwurfs. Dieser Entwurf ist möglichst generisch ausgelegt, mit dem Ziel auf unterschiedlichen Platformen implementiert werden zu können. Dieses Vorgehen, zunächst mit einem allgemeinen Framework zu beginnen und erst danach zu versuchen, eine Implementierung umzusetzen ist aber problematisch. Denn hierbei werden die Randbedingungen nicht berücksichtigt, die durch die Platform vorgegeben werden, für die zu implementieren ist.

So kann beispielsweise das Observer-Pattern, so wie es im Klassendiagramm mit den Klassen Observer, ObserverImpl, Observable und AnalyzerImpl nicht 1-zu-1 für Android umgesetzt werden. ObserverImpl soll für die Aktualisierung der GUI-Elemente zuständig sein. Die GUI kann in Android ausschließlich durch den Hauptthread der Android-Anwendung (App) aktualisiert werden. Die SourceImpl-Instanz läuft in einem Nebenthread. Weil aber die AnalyzerImpl-Instanz von der SourceImpl-Instanz getriggert die Methode *update()* der ObserverImpl-Instanz aufrufen würde, würde hierbei eine ThreadViolation-Exception geworfen werden und die App würde abstürzen. Nebenthreads in Android können beispielsweise über Intents, die als Broadcast-Messages versendet werden und im Hauptthread empfangen werden, ein Update der GUI bewirken.

Richtig wurde im Entwurf erkannt, das strikt zwischen Quelle und Analyse zu trennen ist, und dass die Quelle zunächst von virtueller Art ist. Erst durch die SourceController-Instanz

Abbildung 1: Klassendiagramm des ersten Entwurfs eines Frameworks zur Erkennung von Schadsoftware



wird einer Analyzer-Instanz eine konkrete Quelle zugeordnet. Dadurch ist es unterschiedlichen Analyzer-Instanzen ermöglicht, aus den selben Quellen (teilweise gefiltert) zuzugreifen. Beispielsweise könnte ein Analyzer A nur HTTP-Netzwerkverkehr analysieren, während ein Analyzer B für die Analyse von UDP-Verbindungen zuständig ist. Beide würden aber von der selben Quelle gespeist werden, einem Prozess, welcher die libpcap verwendet.

3 Implementierung

Die Einarbeitung in die Programmierung von Android-Anwendungen (Apps) ist nicht trivial. Um ein Framework entwerfen zu können, das auch für Android implementiert werden kann und die spezifischen Merkmale der Android-Umgebung berücksichtigt, ist es zunächst erforderlich, ein tieferes Verständnis für das Android-Framework zu erlangen.

Das charakteristische an Android-Anwendungen ist, dass einzelne Entitäten (Activity, Service, BroadcastReceiver, ContentProvier) mittels bestimmter Nachrichten-Objekte, den Intents, Nachrichten austauschen. Alternativ kann auch das Observer-Pattern verwendet werden. Hierbei muss aber darauf geachtet werden, dass GUI-Elemente einer Activity nur vom Hauptthread aktualisiert werden können. Diese hat zur Folge, dass der erste Ansatz des Frameworks nicht auf Android umgesetzt werden kann.

Um praktische Erfahrung diesbezüglich zu erlangen, habe ich mich entschlossen, zunächst eine möglichst einfache Android-Anwendung zu entwickeln, welche lediglich Netzwerk-Verkehr mittels Entropie-Analyse überwacht.

3.1 EntropyAnalyzer

Der *EntropyAnalyzer* ist eine Android-Anwendung (App), die als Prototyp gedacht die praktische Umsetzbarkeit der Entropie-Analyse von Netzwerk-Traffic in Echtzeit auf einem mobilen Gerät zeigt. Sie wurde zusammen mit Benjamin Jochheim entwickelt. Benjamin Jochheim hat die Analyse-Funktion implementiert, während sämtliche anderen Bestandteile der App von mir entwickelt wurden. Die Anwendung soll zeigen, dass ohne Vorwissen Datenströme, die Binärcode enthalten, erkannt werden können. D.h. es sind bei diesem Verfahren keine Virensignaturen oder Ähnliches vonnöten.

Abbildung 2 bietet einen Überblick über den prinzipiellen Aufbau und der Funktionsweise vom EntropyAnalyzer. Nachdem die Überwachung des Netzwerkverkehrs aktiviert wurde, werden die anfallenden eingehenden Datenpakete mitgeschnitten (Sniffen). In der Abbildung wird beispielsweise eine Datei über das HTTP-Protkoll mit dem in Android integrierten Browser heruntergeladen. Gleichzeitig snifft der *EntropyAnalyzer* den anfallenden HTTP-Traffic mit, welcher an die Analyse-Komponente weitergereicht wird. Die Analyse-Komponente analysiert nun den Traffic und berechnet somit fortwährend den Entropiewert über den Traffic. Diese Ergebnisse werden dann als Nachrichten (Intents) and die GUI weitergeleitet, welche den Entropie-Verlauf mittels eines Graphen im zeitlichen Verlauf anzeigt.

Die Anwendung wird auf der CeBIT 2012 (Halle 26, Stand E 50) präsentiert werden. Hierzu ist es erforderlich, dass die App einen Workflow bietet, der eine Präsentation der Entropie-Analyse unterstützt und dem Präsentierenden nicht durch eine umständliche Bedienung im Wege steht. Abbildung 3 bietet einen Überblick über das Graphical User Interface (GUI) vom EntropyAnalyzer. Die Knöpfe (Buttons), die dem Ablauf eines Präsentations-Workflows entsprechend im jeweiligen Zustand zu drücken sind, sind sowohl in der Abbildung als auch in der App selber farblich (hellgrün) hervorgehoben. Der Workflow ist wie folgt:

Abbildung 2: Überblick der Funktionsweise vom EntropyAnalyzer

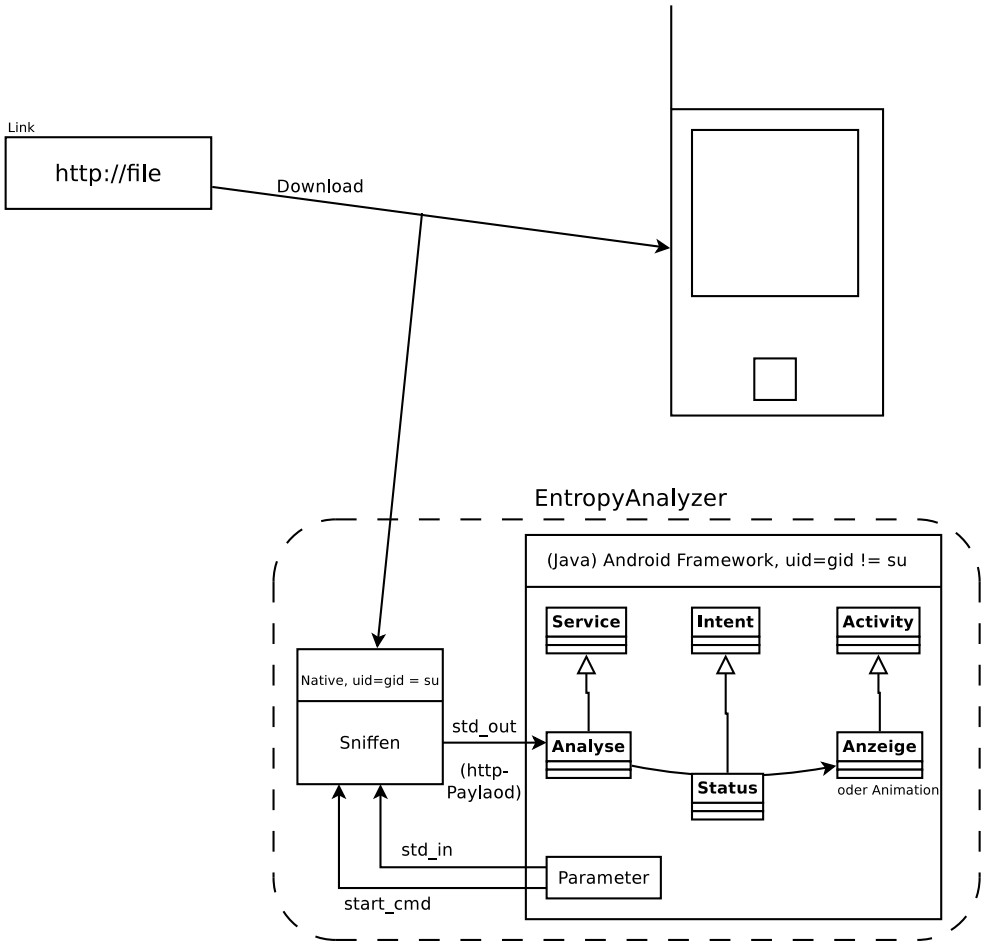
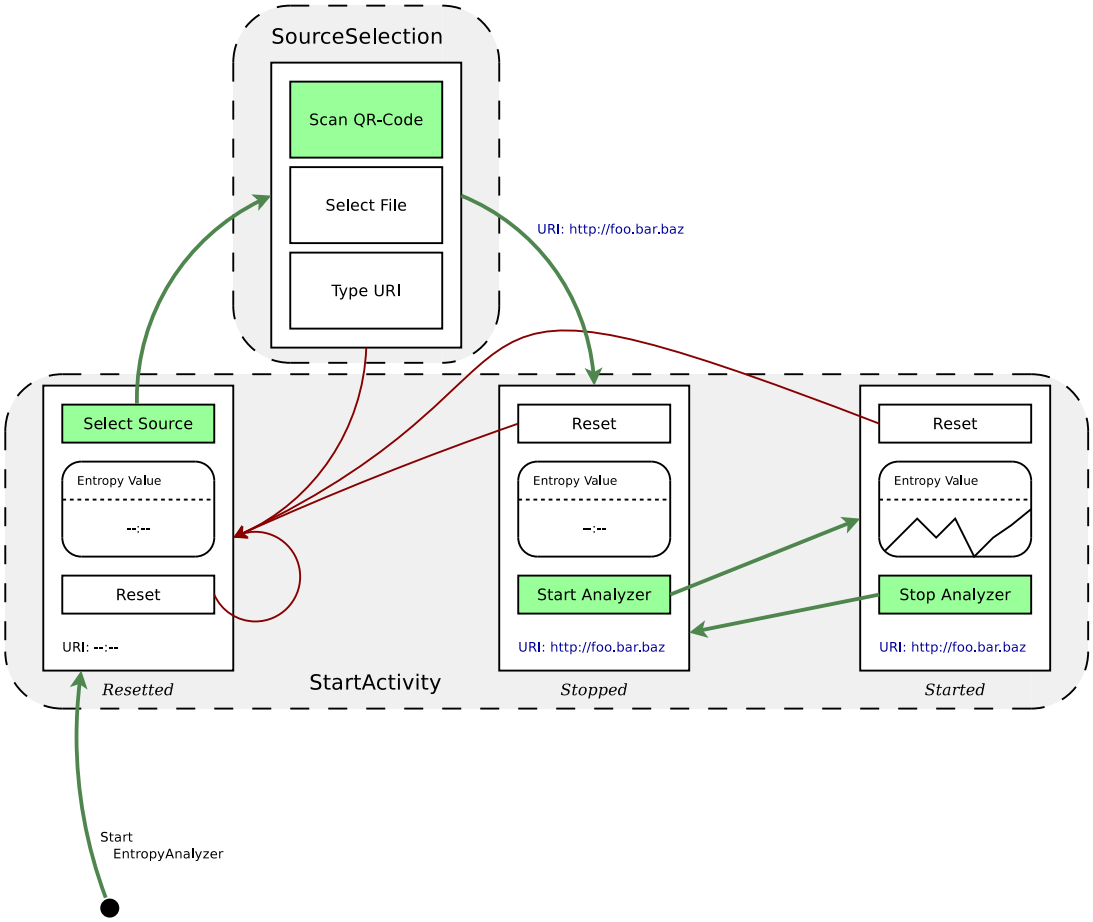


Abbildung 3: GUI mit ihren unterschiedlichen Views vom EntropyAnalyzer



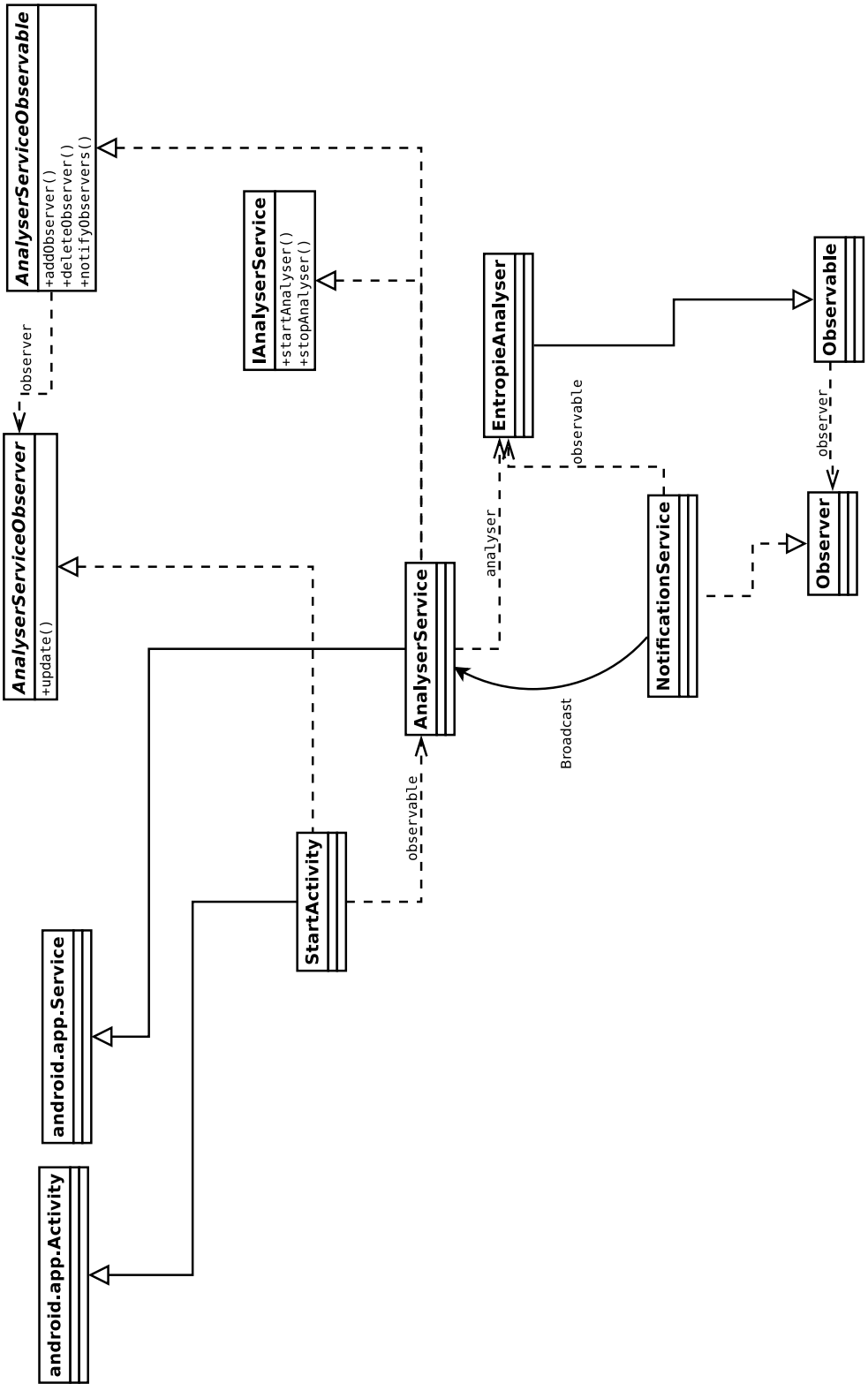
- Zu Beginn wird die Hauptansicht (*StartActivity*) im Zustand *Resetted* angezeigt.
- Als nächstes wird die Quelle, d.h. der Uniform Resource Identifier (URI), festgelegt (*SourceSelection*). Dazu gibt es drei Möglichkeiten: Erstens kann der *EntropyAnalyzer* einen QR-Code einscannen und die dort kodierte URI auslesen. Dann ist der *EntropyAnalyzer* auch befähigt, die Entropie-Werte lokaler Dateien zu untersuchen; so gibt es auch die Möglichkeit eine Datei auszuwählen. Als dritte Möglichkeit kann die URI auch von Hand eingegeben werden.
- Nachdem die Quelle ausgewählt worden ist, ist die URI in der Hauptansicht (*StartActivity*) als ausführbarer Link angezeigt. Wird dieser gedrückt, so öffnet sich der Browser, welcher versucht, die Seite bzw. Ressource zu laden, auf welche die URI verweist.
- Es wird wieder die Hauptansicht angezeigt im Zustand *Stopped*. Mit dem Knopf *Start Analyzer* wird die Entropie-Analyse im Hintergrund gestartet. Wurde eine lokale Datei ausgewählt, so wird der Entropie-Wert von genau dieser Datei berechnet. Wenn die URI das Schema „http'ënthält, so wird sämtlicher eingehender Traffic analysiert.
- Nachdem die Analyse gestartet wurde (Zustand *Started*), ermöglicht nun der untere Knopf der Hauptansicht, diese anzuhalten. Erfolgt dies, so ist die Hauptansicht wieder im Zustand *Stopped*.
- Aus allen Zuständen der Hauptansicht wird auch der Knopf *Reset* angeboten, welcher eine Transition in den Zustand *Resetted* ermöglicht.

Der *EntropyAnalyzer* gibt die aktuelle Bedrohungslage an die SKIMS-App weiter. Die Bedrohungslage wird aus den berechneten Entropie-Werten abgeleitet. Ab einem einstellbaren Schwellwert ist der Datenstrom (Traffic/File), der analysiert wird, mit hoher wahrscheinlichkeit Binary-Code. In diesem Fall wird eine entsprechende Nachricht (Broadcast-Intent) an die SKIMS-App gesendet. Die SKIMS-App wird von der Firma *Escrypt* [2] entwickelt und repräsentiert einen Dienst, der die aktuelle Bedrohungslagen auf dem Android-Gerät von verschiedenen anderen Apps kumuliert und zentral anzeigt. Neben dem *EntropyAnalyzer* gibt es zur Zeit auch einen FTP-Honypot (vom DFN-CERT [1] entwickelt) und eine Beispiel-Anwendung, die bei einer W-LAN Verbindung des Smartphones über einen bestimmten Access-Point (dem „bösen AP'") einen Alarm auslöst.

3.2 Implementierung vom EntropyAnalyzer

Abbildung 4 zeigt das Klassendiagramm vom EntropyAnalyzer. Auch hier wird wie im zuvor beschriebenen Entwurf des Framworks das Observer-Pattern verwendet. Von einem Nebenthread wird aus der EntropyAnalyzer-Instanz heraus die *update()*-Methode im Notifications-Service aufgerufen. In dieser Methode wird ein Intent-Broadcast versendet, worauf diese Nachricht vom Hauptthread durch die AnalyzerService-Instanz empfangen wird, welche wiederum die GUI der StartActivity aktualisiert (wiederum über das Observer-Pattern).

Abbildung 4: Klassendiagramm vom EntropyAnalyzer



Das Sniffen ist durch Funktionen der C-Bibliothek *libpcap* implementiert. Es handelt sich also um ein C-Programm (*http_capture*), welches als Linux-User *root* (uid = gid = 0) ausgeführt wird. Weil aber jede Android-App in einer Sandbox mit einer individuellen User- und Group-ID ausgeführt wird, kann *http_capture* nicht über den vom Android-Framework vorgesehenen Weg ausgeführt werden, da es nicht mit den erforderlichen Rootrechten laufen würde. Stattdessen wird *http_capture* als ein eigener Prozess mit Hilfe des Kommandos *su* mit Rootrechten ausgeführt. Dazu wurde das Android-Gerät zuvor *gerootet*, d.h. das Kommando *su* installiert, welches normalerweise nicht verfügbar ist. Die genaue Funktionsweise habe ich in [4] beschrieben.

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Der erste Entwurf des Frameworks hat sich als für Android nicht implementierbar herausgestellt. Insbesondere ist der Weg, zunächst ein allgemeines Framework zu entwerfen und anschließend dieses umzusetzen zu versuchen, nicht der richtige. Hierbei geht der Entwurf an der Wirklichkeit, hier den Randbedingungen der Android-Platform, vorbei.

Stattdessen habe ich nun mit der Entwicklung der Android-Anwendung *EntropyAnalyzer*, in der die Erkennung von Schadsoftware mittels genau eines Verfahrens (der Entropie-Analyse von HTTP-Datenströmen) realisiert ist, einen realitätsbezogenen Weg eingeschlagen. Diesem Weg folgend werde ich die Software parallel zu einer Diskussion die auf das Design der Software gerichtet ist, erweitern, was uns zum Ausblick führt.

4.2 Ausblick

Als nächstes sollte systematisch untersucht werden, welche Verfahren zur Erkennung von Schadsoftware existieren und welche davon auf mobilen Geräten auch praktisch einsetzbar sind. Insbesondere möchte ich hierbei ermitteln, ob das Prinzip der Trennung von Analyse und Quelle in den unterschiedlichen Verfahren auch anwendbar ist.

Einen neuen Entwurf für ein Framework werde ich iterativ und parallel zur Implementierung entwickeln (und eben nicht einfach entwerfen):

- Im Anschluss der Entwicklung der App *EntropyAnalyzer* werde ich zunächst ein für Android zugeschnittenes Framework entwerfen und somit die Android-spezifischen Rahmenbedingungen berücksichtigen. Parallel zum Entwurf werde ich auch die Implementierung als Android-Anwendung umsetzen, so dass stets ein realer Bezug vorhanden ist. Hiermit soll vorgebeugt werden, dass der neue Framework-Entwurf nicht umsetzbar ist.

- Darauf folgend habe ich vor, ein weiteres Analyseverfahren in einer App umzusetzen und anschließend mit der Entropie-Analyse in einer weiteren App zu kombinieren. Das wird mir schließlich zu einem auf die Android-Umgebung zugeschnittenen Framework verhelfen.

Ob anschließend von mir noch ein Vorschlag für ein plattformunabhängiges aber grundsätzlich implementierbares Framwork für die Erkennung von Schadsoftware vorgeschlagen wird, wird sich dann zeigen. Es kann auch möglich sein, dass gute Argumente gegen ein solches (umsetzbares!) Framework sprechen.

Literatur

[1] DFN-CERT. DFN-CERT: Forschung, Beratung, Dienstleistungen für IT-Sicherheit - DFN-CERT. http://www.dfn-cert.de/, 29. Februar 2011.

[2] Escrypt. ESCRYPT - Embedded Security. https://www.escrypt.com/, 29. Februar 2011.

[3] Benjamin Jochheim, Thomas C. Schmidt, and Matthias Wählisch. A Signature-free approach to malicious code detection by applying entropy analysis to network streams. In *Proc. of the TERENA Networking Conference*, page Poster, Amsterdam, Mai 2011. Terena. Student Poster Award.

[4] Theodor Nolte. Execute Native Code as root. http://141.22.27.191/trac/wiki/Android, 28. Februar 2011.

[5] Thomas C. Schmidt. SKIMS - A Cooperative Autonomous Immune System for Mobile Devices. http://www.realmv6.org/skims.html, 31. August 2011.