Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461 ZVR: 948166612 UID: ATU60778947

KLASSIFIKATION MOBILER ANWENDUNGEN ÜBER METADATEN

Version 1.0 vom 29.05.2018
Johannes Feichtner – johannes.feichtner@a-sit.at

Bei der Analyse von Mobilanwendungen wird der Fokus üblicherweise auf eine angenäherte Fassung des originalen Programmcodes gelegt. Die Erhebung sicherheitsrelevanter Eigenschaften gelingt dabei meist nur individuell pro Applikation und erlaubt keinen breiteren Blick darüber, welche Eigenschaften charakteristisch für die Funktionalität von Anwendungen sind. In diesem Projekt wurde daher eine Untersuchung aus einer neuen Perspektive angestrengt, um Aussagen über Anwendungen auf Basis ihrer Metadaten treffen zu können.

Das Ziel des Projekts bestand darin, anwendungsbezogene Informationen, abseits von Programm, etwa GUI-Layouts oder Anwendungstexte zu extrahieren, aggregieren und in weiterer Folge eine Unterscheidungsbasis für einzelne Apps auf Basis dieser Metadaten zu schaffen. Der so entworfene Ansatz kann dabei helfen, einzelne Typen von Anwendungen zu identifizieren und dazu semantisch ähnliche zu finden. Insbesondere im Hinblick auf Malware bzw. Anwendungen mit Sicherheitsrelevanz resultiert dies in einem neuartigen Zugang, um ihre Relation zu anderen Anwendungen zu verstehen und sie entsprechend klassifizieren zu können.

Inhaltsverzeichnis

1. Einleitung

Die Verwendung von Mobilgeräten ist aus unserem Alltag nicht mehr wegzudenken. Mit umfassender Rechenleistung, großem Speichervermögen und Zugriff auf eine Vielzahl von Sensoren ausgestattet, bieten sich für AnwendungsentwicklerInnen eine Vielzahl an Szenarien, in denen diese Kapazitäten genutzt werden können. Als Plattform zwischen diesen Anwendungen und der Hardware haben sich in den letzten Jahren Android und iOS als Marktführer positioniert. Die zugehörigen Produzenten Google und Apple haben jeweils eigene Ökosysteme geschaffen, in denen EntwicklerInnen Anwendungen für KundInnen bereitstellen können.

Beiden Anbietern gemein sind ihre jeweiligen Bestrebungen, Anwendungen auf ihre Sicherheit hin zu untersuchen. Dazu werden alle Applikationen, die über die eigenen Marktplätze „PlayStore“ und „AppStore“ vertrieben werden sollen, vor Veröffentlichung einer automatisierten (und möglicherweise auch manuellen) Inspektion unterzogen. Über den originalen Quellcode der Anwendungen verfügen sie dabei nicht. Welche Analysetechniken schließlich genau zum Einsatz kommen, ist nicht öffentlich dokumentiert. Beide Anbieter sehen sich jedoch nicht nur mit einer stark zunehmenden Zahl an Anwendungen für verschiedenste Einsatzzwecke konfrontiert, sondern auch mit Malware-Applikationen, die vorgeblich harmlos sind. Für alle Überprüfungen können Google und Apple außerdem auf eine, vom EntwicklerInnen bereitgestellte, Beschreibung zurückgreifen, die im Idealfall mit der tatsächlichen Funktionalität übereinstimmt.

Die Untersuchung von Sicherheitsaspekten auf Mobilgeräten hat in den vergangenen Jahren auch in Form wissenschaftlicher Publikationen viel Aufmerksamkeit erfahren. Das Gros der Arbeiten konzentriert sich dabei auf das Android-Betriebssystem, welches durch eine relativ offene Architektur Analysen der Plattform und Anwendungen begünstigt. Im Hintergrund der meisten Untersuchungen steht dabei die Motivation, die Existenz und korrekte Implementierung von Sicherheitsfunktionen nachzuvollziehen. Für diese Analyse wird der Fokus üblicherweise auf eine angenäherte Fassung des originalen Programmcodes gelegt. Von EntwicklerInnen bereitgestellte Metadaten sind zwar vorhanden, werden im Normalfall aber nicht weiter untersucht. Ein möglicher Grund dafür könnte sein, dass es sich dabei um semantisch unstrukturiertes Konvolut an Text handelt, das automatisiert nur schwer interpretierbar ist. Im Gegensatz dazu kann Programmcode mithilfe von „Reverse Engineering“-Techniken strukturiert aufbereitet und untersucht werden. Mit steigender Code-Komplexität, zunehmendem Funktionsumfang und damit inhärent wachsender Anwendungsgröße, steigt jedoch auch der benötigte Ressourceneinsatz dramatisch, um eine zielgerichtete Verhaltensanalyse einer Anwendung vorzunehmen.

1.1. Problemstellung

Bei der individuellen Analyse des Programmcodes einer Mobilanwendung gelingt im Idealfall die Erhebung einzelner sicherheitsrelevanter Eigenschaften. Durch die isolierte Betrachtungsweise ist jedoch kein Kontext verfügbar, mit dem Abläufe innerhalb der Applikationen erklärt werden könnten. Beispielsweise lässt sich ohne kontextueller Information keine Aussage darüber treffen, ob es legitim ist, dass eine Anwendung die lokalen GPS-Koordinaten zu einem Server überträgt. Liegt der Zweck einer Anwendung in der Routennavigation, wäre ein derartiges Verhalten nachvollziehbar. Ohne kontextueller Information könnte es sich jedoch genauso gut um eine nicht gewollte Weitergabe handeln (ein „Leak“ sensibler Daten). Ohne diese Information kann sich die Analyse von Programmcode nur auf ein maximal konservatives Bedrohungsszenario stützen bzw. generalisierte Annahmen über das Verhalten treffen. Die tatsächliche Aussagekraft von Analyseergebnissen wird jedoch mitunter verzerrt, da der Kontext einer Anwendung keine Berücksichtigung findet.

Senden EntwicklerInnen neue oder aktualisierte Mobilanwendungen zu den Vertriebsplattformen von Google und Apple, müssen jene die Entscheidung treffen, ob eine Anwendung harmlos ist oder nicht. Eine differenzierte Beurteilung unter Einbeziehung möglicher Verwendungsszenarien ist dabei aber nicht ohne weiteres möglich. Die zusätzlich zum Programmcode vorhandenen Metadaten ebenfalls in diese Auswertung aufzunehmen, würde aber eventuell eine Vergleichsbasis mit anderen Anwendungen schaffen und so zu zuverlässigeren Prüfergebnissen führen. Damit das gelingt, bedarf es aber einer Aufbereitung und Möglichkeit zur strukturierten Auswertung von Metadaten.

Die grundsätzliche Problemstellung lässt sich auch exemplarisch im Hinblick auf „Kategorien“ gut nachvollziehen: Ein pragmatischer Ansatz wäre es, Anwendungen in Kategorien einzuteilen, die ihrer Funktionalität entsprechen. Im Android „PlayStore“ und iOS „AppStore“ wird eine solche Zuteilung bereits vorgenommen, jedoch sind die Kategorien vergleichsweise abstrakt, wie z.B. „Tools“ oder „Entertainment“. Unter welcher Kategorie eine Anwendung schließlich angeboten wird, entscheiden jedoch EntwicklerInnen durch freie Auswahl. Da die Auswahl möglicher Kategorien aber fix vorgegeben ist und die Zugehörigkeit nur relativ grob erhoben wird, kann eine derartige Einteilung genauere Unterschiede zwischen einzelnen Anwendungen nicht abbilden. Je abstrakter eine Kategorie also formuliert ist, desto weniger eignet sie sich, wenn Anwendungen klassifiziert werden und dabei einer Kategorie zugewiesen werden sollen. Auch eine alternativ durchführbare manuelle, genauere Definition von Kategorien wäre jedoch potentiell fehlerhaft, unvollständig und könnte a priori kein gleichwertiges Maß an Unterscheidbarkeit bieten.

1.2. Zielsetzung

Als Schlussfolgerung aus der zuvor angeführten Problemstellung lässt sich ableiten, dass aus Metadaten von Mobilanwendungen potentiell Informationen über die Funktionsweise, den Zweck bzw. die Zugehörigkeit von Anwendungen enthalten können. Die Tatsache, dass diese Daten nicht wie Programmcode in strukturierter Form vorliegen, verhinderte bislang eine sinnvolle Auswertung.

Die zentralen Fragestellungen, die in diesem Bericht erörtert werden sollen, sind demnach wie folgt:

- Welche Metadaten von Mobilanwendungen haben deskriptiven Charakter und liefern präzise Informationen über Zweck und Funktionalität von Anwendungen?
- Wie können diese Daten praxistauglich extrahiert, aggregiert und verarbeitet werden?
  Wie lassen sich Applikationen anhand ihrer Metadaten voneinander unterscheiden?
- Können anhand von Metadaten auch ähnliche Applikationen gefunden werden?
  Diese Fragestellung zielt nicht nur auf das zuvor gebrachte Beispiel mit „Kategorien“ ab, sondern schließt auch die folgenden Überlegungen ein:
  - Wenn die Metadaten von Anwendungen ähnlich sind, könnte auch ihre Implementierung ähnlich sein. Auf eine Korrelation von Metadaten und Programmcode wird in diesem Projekt jedoch nicht weiter eingegangen.
  - Applikationen, die hinsichtlich ihrer Metadaten ähnlich sind, könnten jedoch auch einem gemeinsamen Typus angehören, in ihrer Implementierung aber unterschiedlich sein. Ein zugespitztes Beispiel, wo diese Eigenschaft vom Urheber sogar gewünscht wird, ist Malware. In der Praxis zeigt sich, dass Schadcode oft aus funktionaler Sicht zueinander sehr ähnlich ist, jedoch das gleiche anders umgesetzt wurde. Sofern die Metadaten zweier Varianten dennoch gleich wären, ließe sich dennoch eine Ähnlichkeit der Anwendungen feststellen.
- Können aus Metadaten auch Indizien für Anomalien und potentiell sicherheitsrelevante Probleme abgeleitet werden?

Die Analyse von Metadaten wirft somit einen Blick auf Mobilanwendungen, der von der jeweiligen Implementierung entkoppelt ist. Eine Korrelation zwischen beiden wäre zwar naheliegend, muss aber nicht grundsätzlich gegeben sein. Im Idealfall können die Daten so aufbereitet werden, dass sie mit Quellcode bei der Programmanalyse verknüpft werden können und kontextuelle Zusatzinformationen liefern, die letztlich dazu beitragen könnten, die Präzision einer Analyse zu verbessern. Auch ohne diesen Mehrwert zu schaffen, könnte die Beantwortung der angeführten Fragestellungen jedoch dabei helfen, Mobilanwendungen über ihre Metadaten in Relation zu setzen und daraus Schlüsse zu ziehen, die der Applikationsanalyse insgesamt zuträglich ist.

In diesem Projekt wird den angeführten Fragestellungen Rechnung getragen, indem zunächst ein geeignetes Set an Metadaten zu realen Anwendungen akquiriert wird. Ausgehend davon werden die verfügbaren Daten hinsichtlich ihrer Aussagekraft evaluiert. Unter Anwendung von Methoden zur „Knowledge Discovery“ in unstrukturierten Daten wird daraufhin angestrebt, eine Homogenität zu erzeugen, die weitere Verarbeitung ermöglicht. Anschließend an diese Datenaufarbeitung wird unter Anwendung geeigneter Techniken ein semantisches Modell abgeleitet, das die Daten sinnvoll in Relation zueinander setzt und entsprechende Abfragen ermöglicht.

Der verbleibende Teil dieses Dokuments untersucht die angeführten Fragestellungen und verwendet die dabei festgestellten Erkenntnisse zur Entwicklung eines Ansatzes zur Klassifikation von Metadaten in Mobilanwendungen.

2. Datenerfassung

Ein erster Startpunkt, um Metadaten von Mobilanwendungen zu erfassen, sind die jeweiligen Vertriebsplätze von Google und Apple. Elemente wie der Name der EntwicklerInnen, die Anzahl der Installationen, die Zuordnung zu einer Kategorie, Kommentare und Bewertungen sind von der Implementierung unabhängige Daten, die sich als Ausgangspunkt für eine erste statistische Analyse zu eignen scheinen. Durch den aber nur indirekt vorhandenen Bezug zur tatsächlichen Implementierung, sind sie aber bestenfalls zusätzliche Indikatoren im Hinblick auf die zuvor angeführten Fragestellungen. Praktisch heißt das, dass sie nicht hinreichend den Anspruch erfüllen, Zweck bzw. Funktionalität von Anwendungen zu beschreiben.

Aus diesem Grund fokussieren wir uns bei der weiteren „Feature Selection“ auf Attribute, die in direktem Zusammenhang mit der Implementierung stehen. Bei der Suche nach weiteren Metadaten konnten verschiedene „Features“ (und Gruppierungen) identifiziert werden, die entweder numerischen oder kategorischen Beschreibungscharakter aufweisen:

- **EntwicklerInnenbezogene Eigenschaften**
  Neben einer verhältnismäßig groben Kategorisierung sind EntwicklerInnen auch angehalten, Beschreibungen über Inhalt und Zweck in verschiedenen Sprachen bereitzustellen. Im Idealfall korreliert diese Angabe mit der Implementierung. Wäre ein Vergleich der Beschreibungen untereinander gegeben, könnten Anwendungen gefunden werden, die gleiche Funktionalität durch unterschiedliche Implementierungen abbilden. Da es sich bei der Beschreibung um Fließtext handelt, liegt es nahe, sich zur Verarbeitung bei Methoden zu bedienen, die auch sonst für „Natural Language Processing“ (NLP) zum Einsatz kommen und deren Ziel „Knowledge Discovery“ ist.

  Weitere entwicklerspezifische Eigenschaften in Mobilanwendungen finden sich auch in den Anwendungen selbst, in Form von Zertifikaten, wieder. Praktisch heißt das, dass anhand der Zertifikate beispielsweise nach Anwendungen der gleichen EntwicklerInnen gesucht werden könnte. Beispielsweise sind sämtliche Anwendungen von Google mit dem gleichen Zertifikat signiert und können daher gewissen EntwicklerInnen zugewiesen werden. Dies ist vor allem auch dann relevant, wenn eine Relation nicht bereits aus dem Zweck hervorgeht. Etwa verwendet Google zur Signatur von Anwendungen das gleiche Zertifikat für die Hauptanwendung selbst aber auch den „Messenger“ und „Instagram“. Eine inhaltliche Überschneidung z.B. in Form von mehrfach vorkommenden Codefragmenten in den jeweiligen Applikationen lässt sich dadurch besser erklären und auf einen gemeinsamen Ursprung zurückführen.

- **Eigenschaften von Malware**
  Kategorische Metadaten werden auch von externer Seite bezogen, indem Anwendungen mithilfe gängiger Systeme zur Erkennung von Malware geprüft werden. Das Ergebnis ist für gewöhnlich ein binärer Indikator, der darüber Auskunft gibt, ob eine Anwendung Malware ist oder nicht, sowie die Familie des Schadcodes benennt. Je nach Zuverlässigkeit dieser Angaben, ließen sich somit schädliche von harmlosen Anwendungen untersuchen. Im Hinblick auf die hier verfolgten Zielsetzung wäre es somit in jedem Fall möglich, eine Binärklassifikation zwischen guten und schlechten Applikationen vorzunehmen, sowie weitere Malware einer gewissen Familie / eines Typs zu finden.

- **Intrinsische Applikationseigenschaften**
  Hierunter werden jene Angaben verstanden, die sich auf ein konkretes Release einer Mobilanwendung beziehen, in direkter Relation zum Programmcode stehen, jedoch nicht Teil desselbigen sind. Eine unmittelbare Relation zum Code lässt sich feststellen, wenn die Eigenschaften zur Ausführung von Anwendungen benötigt werden:

- ***Berechtigungen***
  Jede Mobilanwendung für Android inkludiert in der Datei „AndroidManifest.xml“ eine Auflistung an Berechtigungen, die von BenutzerInnen gewährt werden müssen, um ausgeführt werden zu können.

- ***GUI Layouts***
  Die bei Android-Anwendungen dargestellte Benutzeroberfläche ist intern in Form von Dateien im XML-Format abgebildet. Je nach XML-Tag und definierten Attributen werden unterschiedliche Oberflächenelemente (Buttons, Menüleiste, Eingabefelder) angezeigt. Die Überlegung hierbei ist, dass auch anhand dieser Metadaten eine Klassifikation möglich sein könnte, da das graphische Layout im Normalfall dem Zweck der Anwendung entspricht. Empirisch ließ sich beispielsweise feststellen, dass das Gros der Anwendungen für Mobile Banking graphisch ähnlich strukturiert ist und zueinander ähnliche Eingabemöglichkeiten bietet. Dies ist nicht weiter verwunderlich, da unabhängig vom Anbieter etwa für Überweisungen stets die gleichen Eingabedaten benötigt werden.

- ***GUI Strings***
  Textuelle Beschreibungen von GUI-Elementen werden in Android-Anwendungen gleich wie GUI Layouts in XML-Dateien abgelegt. Nach Sprache getrennt finden sich darin alle Elemente, die während der Ausführung der Anwendung auch angezeigt werden. Relevant für die Klassifikation könnten GUI Strings insofern sein, da sie dem Zweck der Anwendung nach unterschiedlich sind. Sind z.B. die Zeichenketten „Neue Überweisung“ und „Login“ hinterlegt, kann mit hoher Wahrscheinlichkeit angenommen werden, dass es sich um eine Anwendung aus dem Bereich Mobile Banking handelt. Insbesondere im Hinblick auf die zuvor angeführten Fragestellungen könnten sich so Anwendungen finden lassen, deren Texte semantisch verwandt sind, die aber intern unterschiedlich implementiert sind.

2.1. Akquisition

Anhand der vorgenommenen Aufstellung zeigt sich, dass für die Analyse von Metadaten auch die zugehörigen Mobilanwendungen selbst vonnöten sind. Die angeführten „Intrinsischen Applikationseigenschaften“ sind in den offiziellen Vertriebskanälen nicht oder nur in sehr limitierter Form vorhanden, da die Anbieter sie nicht aus den Anwendungspaketen extrahieren. Praktisch bedeutet das, dass für das konkrete Vorhaben sowohl Metadaten aus den offiziellen Vertriebsstellen benötigt werden, wie auch das zugehörige Programmpaket.

Ein systematischer Download von Metadaten und Anwendungen von Apples „AppStore“ oder Googles „PlayStore“ ist leider nicht ohne weiteres möglich. Beide Anbieter stellen durch Schutzmaßnahmen sicher, dass „Crawler“ nur eingeschränkt funktionieren und z.B. in einem gewissen Zeitraum nur eine bestimmte Anzahl an Anwendungen heruntergeladen werden kann. Eine großflächige Aggregation von Anwendungen wird dadurch ungemein erschwert. Im Rahmen einer thematisch verwandten wissenschaftlichen Publikation namens „PlayDrone“ [1] wurde jedoch eine Methodik entwickelt, um Googles „PlayStore“ systematisch nach Anwendungen zu durchsuchen und Metadaten gleich wie Anwendungspakete zu „crawlen“. Auch in einer alternativen Quelle fanden sich Anwendungen aus dem „PlayStore“, wenngleich ohne Metadaten [2]. Die Erstellung eines eigenen „Crawlers“ wäre somit redundant und kann an dieser Stelle entfallen.

Für die weitere Analyse wird auf das von „PlayDrone“ erstellte Archiv zurückgegriffen. In der Sammlung finden sich 1,1 Mio. Android-Anwendungen, mitsamt ihren Metadaten aus dem „PlayStore“. Aus Speichergründen wählen wir für die nachfolgende Untersuchung ein Subset von 40.000 Anwendungen aus. Aus jeder der 42 Kategorien werden dabei zu gleichen Teilen Anwendungen bezogen und mit den Metadaten im JSON-Format abgelegt. Letzteres ermöglicht eine vergleichsweise einfache Weiterarbeitung und Extraktion einzelner Felder wie etwa der Anwendungsbeschreibung oder den Berechtigungen.

Ein separates Set an Anwendungen, die als Malware bekannt sind, wurde nicht akquiriert, da es dazu keine separat aufgeführten Metadaten vorliegen und ein Vergleich mit dem „PlayDrone“-Set daher nur in Bezug auf die „Intrinsischen Applikationseigenschaften“ möglich gewesen wäre.

3. Klassifikation von Metadaten

Die Heterogenität der potentiell relevanten Metadaten legt nahe, sie getrennt voneinander zu betrachten und auch individuell die Aussagekraft der einzelnen Datentypen zu bewerten. Nachfolgend wird auf die im vorangehenden Abschnitt aufgelisteten Typen von Metadaten der Reihe nach eingegangen und eine Klassifikation von Anwendungen anhand ihrer Metadaten angestrebt.

3.1. Eigenschaften von Malware

Zur initialen Feststellung, wie viele der Anwendungen Charakteristiken von Malware erfüllen, wurden sie automatisiert mit VirusTotal[1] geprüft. Mithilfe dieses Dienstes ist es möglich, die Anwendungen zeitgleich einer Vielzahl an (kommerziellen) Systemen zur Erkennung von Malware zur Verfügung zu stellen. Dienste von McAfee, AVG, Eset, TrendMicro, etc. liefern daraufhin ein binäres Ergebnis zurück, das aussagt, ob eine Anwendung für Malware gehalten wird oder nicht. Mit den Daten von „PlayDrone“ wurden dabei 2% bzw. 812 Anwendungen von mindestens einem Prüfdienst als Malware identifiziert. Eine manuelle Verifikation dieser Ergebnisse hat gezeigt, dass es sich dabei vor allem um heuristische Warnungen handelt, die im Ergebnis als Malware tituliert werden. Konkrete, gewissen Malware-Familien zugehörige Resultate fanden sich jedoch nicht.

Im Gesamtergebnis können wir durch diese Prüfung eine Art „Clustering“ in harmlose und potentiell gefährliche Anwendungen vornehmen. Da die Ergebnisse der Prüfungen aber nicht mehr als heuristische Warnungen ausgeben, fehlen konkrete Anhaltspunkte, um Anwendungen tatsächlich zuverlässig einem Cluster zuordnen zu können.

3.2. Intrinsische Applikationseigenschaften

Für eine Klassifikation mithilfe von Daten, die in den Anwendungspaketen enthalten sind, wurden alle Anwendungen mithilfe von apktool[2] entpackt und dekodiert. Dies ermöglicht die Verarbeitung der zuvor angeführten Eigenschaften.

3.2.1. Berechtigungen

Sofern in der Anwendung die Datei „AndroidManifest.xml“ enthalten war, wurden daraus Berechtigungen extrahiert und aufbereitet. Eine statistische Auswertung zeigt folgende Berechtigungen als die häufigsten in den untersuchten Anwendungen:

android.permission.INTERNET	96,07%
android.permission.ACCESS_NETWORK_STATE	91,15%
android.permission.READ_EXTERNAL_STORAGE	54,50%
android.permission.WRITE_EXTERNAL_STORAGE	54,12%
android.permission.READ_PHONE_STATE	39,81%

Es fällt außerdem auf, das seine Vielzahl sog. „Custom Permissions“ zum Einsatz kommen[3], die von Entwicklern selbst definiert wurden und deren Einsatzzweck darauf abzielt, mehrere, zusammenhängende Anwendungen jeweils zu beschränken. Da diese Berechtigungen vergleichsweise nur sehr selten vorkommen, ist ihre „Aussagekraft“ ungemein höher, als z.B. bei der „INTERNET“-Berechtigungen, die bei 40.000 getesteten Anwendungen in 96,07% vorkam. Praktisch können anhand der benutzerdefinierten Berechtigungen außerdem verwandte Anwendungen gefunden werden, da hinter ihrem Einsatz ein gemeinsamer Entwicklungsprozess stehen dürfte. Konkret werden dabei von EntwicklerInnen mehrere Anwendungen fabriziert und mithilfe dieser Berechtigungen in ihrer Interprozesskommunikation eingeschränkt.

[1] https://www.virustotal.com
[2] https://ibotpeaches.github.io/Apktool/
[3] https://developer.android.com/guide/topics/permissions/defining

Um Vergleichbarkeit zwischen Anwendungen trotz einer Vielzahl an Berechtigungen zu ermöglichen, ist eine Reduktion unumgänglich. Die Liste mit Berechtigungen ähnelt grundsätzlich einer sehr dünnbesetzten hochdimensionalen Matrix. In solchen Fällen würde sich z.B. „Feature Hashing“ [3] als effektive Strategie dazu eignen, um die Dimensionen zu reduzieren. Die Matrizen der Berechtigungen einzelner Applikationen werden dabei komprimiert und auf jene „Berechtigungs-Features“ reduziert, die tatsächlich gesetzt sind. Die resultierenden Hash-Werte können auch miteinander verglichen werden.

3.2.2. GUI Layouts

Um graphische Elemente aus Android-Anwendungen zu extrahieren, müssen die Dateien des Ordners „*res/layout*“ aus der Applikation extrahiert und dekodiert werden. Analog zur Vorgehensweise bei der Analyse der Berechtigungen, kann dafür *apktool* eingesetzt werden.

Die einzelnen XML-Dateien beschreiben in hierarchischer Gliederung verschiedene UI-Elemente. Die Elemente selbst sind dabei als XML-Tag beschrieben, die konkrete Ausgestaltung (Größe, Farbe, Form, etc.) anhand von Attributen. Zur besseren Veranschaulichung sei ein Beispiel gegeben, das zwei Felder für Texte („TextView“) über ein „LinearLayout“-Element nacheinander entweder in zwei Zeilen oder einer Spalte anzeigt.

```
<LinearLayout android:gravity="center_vertical" android:orientation="horizontal"
    android:id="@id/date_time_suggestion" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:minHeight="44.0dip"
    xmlns:android="http://schemas.android.com/apk/res/android">
  <TextView android:textSize="18.0sp" android:ellipsize="end"
    android:id="@id/date_time_suggestion_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_marginLeft="10.0dip"
    android:layout_marginRight="10.0dip" android:singleLine="true" />
  <TextView android:textSize="18.0sp" android:textColor="#ff8b8b8b"
    android:ellipsize="end" android:gravity="end"
    android:id="@id/date_time_suggestion_label" android:layout_width="0.0dip"
    android:layout_height="wrap_content" android:layout_marginRight="10.0dip"
    android:singleLine="true" android:layout_weight="1.0" />
</LinearLayout>
```

Angesichts der hierarchischen Gliederung wurde im Hinblick auf die gewünschte Klassifikation zunächst nach einer Methodik gesucht, um die vorliegende Struktur als Baum abzubilden. In mehreren Tests hat sich gezeigt, dass die Attribute entfernt werden können, da sie keine semantisch wertvollen Informationen über UI-Elemente beinhalten. Der oben angeführte Code etwa ist einer Anwendung für Mobile Banking entnommen und kommt in der gleichen Applikation mehrfach vor: einmal für Smartphones und einmal in einer Variante für Tablets. Ohne Attribute verbleiben in diesem Beispiel jedoch nur mehr die Elemente „LinearLayout“ und zweimal „TextView“ als Metadaten. Durch die generische Terminologie und die Tatsache, dass es sich dabei um Elemente handelt, die praktisch in jeder Android-Anwendung zum Einsatz kommen, ist ihre Aussagekraft vergleichsweise sehr eingeschränkt.

Als brauchbarere Abhilfe wurde versucht, alle XML-Tags der verschiedenen UI-Elemente einer Anwendung zu extrahieren, doppelt vorhandene Tags zu entfernen und in zusammengehängter Form einen einfachen Machine Learning Classifier (Naïve Bayes) zu trainieren. Hierfür wurden beliebige 100 Applikationen aus dem zuvor gecrawlten Datensatz ausgewählt. Beim Trainieren der GUI Layouts wurde der Name der jeweiligen Anwendung mitgelernt, beim nachfolgenden Testen ohne die entsprechenden Namen der gleiche Datensatz mithilfe des gelernten Modells evaluiert. Als Ergebnis hat sich gezeigt, dass der Classifier nur eine Konfidenz von 12% erreicht.

Als plausibelste Erklärung für diese schlechte Aussagekraft kann angenommen werden, dass es am generischen Wesen der gelernten Tags liegt. Dazu kommt, dass die hierarchische Beziehung (z.B. zwischen „LinearLayout“ und „TextView“) nicht berücksichtigt wird und XML-Tags als voneinander unabhängig betrachtet wurden. Auch ein erneuerter Versuch mit 500 Anwendungen hat nur eine unwesentliche Veränderung ergeben.

3.2.3. GUI Strings

Als weiteres Metadaten-Merkmal wurden in einer Mobilanwendung vorkommende GUI Strings für Klassifikationszwecke aufbereitet. Ähnlich wie UI-Elemente sind Zeichenketten, die in Anwendungen als Text angezeigt werden, in XML-Dateien abgelegt. Jene finden sich bei Android-Anwendungen in erster Linie in der Datei „*res/values/strings.xml*“. Applikationen, die mehrsprachig bereitgestellt werden, können diese unterschiedlichen Versionen in weiteren XML-Dateien ablegen, die dann anhand des Suffix des „*values“*-Ordners unterschieden werden (z.B. „*values-it“* für die italienische Version).

Als „Feature“ für die nachfolgende Klassifikation wurden die Textwerte aus der Datei „*res/values/strings.xml*“ extrahiert und aufbereitet:

- Am Beginn und Ende jeder Zeichenkette wurden Punktation, Leerzeichen und Zeilenumbrüche entfernt.
- Alle nicht ASCII-konformen Zeichen wurden entfernt. Dies betrifft neben Sonderzeichen auch alle Multibyte-Zeichen, die nur mit erweiterten Zeichensätzen (z.B. UTF-8 oder Unicode) interpretierbar sind. Praktisch heißt das, dass alle nicht römischen Buchstaben nicht weiter berücksichtigt wurden. Dahinter steht weniger eine technische Notwendigkeit, als die Absicht, die Ergebnisse der Klassifikation bestmöglich vergleichen und interpretieren können zu wollen.
- Jede Zeichenkette wurde zu Kleinbuchstaben transformiert, um zu verhindern, dass unterschiedliche Formen der Groß- und Kleinschreibung von Wörtern als unterschiedliche Features klassifiziert werden könnten.

In weiterer Folge wurde mit ähnlichen Rahmenbedingungen wie mit den „GUI Layouts“ ein Machine Learning Classifier (Naïve Bayes) mit einem Set an 500 beliebig aus verschiedenen Kategorien gewählten Applikationen trainiert. Die Zuverlässigkeit des Classifiers beim Testen des trainierten Modells lag bei 34%. Da das gewählte Trainingsset bewusst aus Anwendungen verschiedener Kategorien zusammengestellt wurde, hätte angenommen werden können, dass es daher signifikantere Unterschiede geben sollte.

Für ein besseres Verständnis des Ergebnisses wurden die Zeichenketten manuell mithilfe des NLP-Frameworks *nltk*[4] in einzelne Wörter „tokenisiert“ und ihrer Häufigkeit des Auftretens nach gruppiert. Dabei hat sich folgende Erkenntnisse ergeben:

- Im Gros der Anwendungen finden sich generische Phrasen und Wörter wie etwa „Username“, „Password“, „Log in“, „Minute“, „Share“, „Set month“, „Loading video“, die Anwendungen nicht als hinreichend individuell identifizieren können.
- In der strings.xml finden sich neben tatsächlich vom Entwickler eingefügten Wörtern auch Zeichenketten, die durch verwendete Bibliotheken Dritter eingefügt wurden. Beispielsweise enthielten 29% der 500 untersuchten Anwendungen die Phrasen „Google Play services“ und „Network Error“, die aus „Google Play“-Library stammen.
- Die natürliche Sprache, in der die Zeichenketten in „*res/values/strings.xml*“ formuliert werden, ist nicht vorgeschrieben. Praktisch können in dieser Datei somit Ausdrücke in beliebigen Sprachen vorkommen. Im Zuge der Klassifikation ist eine Korrelation somit nicht erkennbar. Eine „Feature Selection“ mit Zeichenketten bestimmter Sprachen, z.B. nur deutschsprachiger Wörter der XML-Elemente im Ordner „*values-de*“ ist in der Praxis ebenfalls nur bedingt tauglich, da nicht alle Anwendungen notwendigerweise eine Übersetzung in der gewählten Sprache bereitstellen.

Im Zuge dieser Klassifikation hat sich auch ein grundsätzliches Problem am gewählten Classifier aufgezeigt. Beim Training eines Naïve Bayes Classifiers findet keine Gewichtung einzelner Wörter entsprechend der Häufigkeit ihres Auftretens statt (sog. „Tf-idf-Maß“). Würden seltener vorkommende Zeichenketten stärker gewichtet, könnten GUI Strings mithilfe eines geeigneten Classifiers auch eine brauchbarere Ergebnisse bei Klassifikation vorweisen.

[4] http://www.nltk.org/index.html

3.3. Entwicklerbezogene Eigenschaften

Die von EntwicklerInnen einer Mobilanwendung verfasste Beschreibung über die Funktionalität einer Anwendung ist als unstrukturierter Fließtext in natürlicher Sprache verfasst. Hinsichtlich der Länge und Qualität gibt es für EntwicklerInnen keine Einschränkungen oder Voraussetzungen. Im Idealfall ist jedoch beschrieben, welche Funktionalität die Anwendung bietet und wie jene realisiert wurde.

Die trivialste Analyse dieses Texts könnte sich auf einem statistischen Ansatz basieren, der die Häufigkeit des Vorkommens („Word frequency“) einzelner Wörter in der Beschreibung aufsummiert und vergleicht. Werden diese Angaben folglich als Features für einen Klassifikationsalgorithmus wie Naïve Bayes herangezogen, ermöglicht dies einfache Textkategorisierungen. Prominente Beispiele, wo ein vergleichbarer Ansatz zur Anwendung kommt, sind außerdem Spamfilter und „Sentiment Analysis“. Für gewöhnlich werden Wörter dabei in ein Modell überführt, das sie mit ihren Auftrittshäufigkeiten in einem Vektorraum abbildet. Charakteristisch dabei ist, dass Wörter einzelnen Sätzen entnommen und als sog. „Bag-of-Word“ zur Klassifikation herangezogen werden.

Mit Fortschritten im Bereich des maschinellen Lernens sind in den letzten Jahren auch für „Natural Language Processing“ (NLP) neue Techniken entwickelt worden, die Zusammenhänge zwischen Wörtern nicht nur hinsichtlich ihrer Quantität, sondern auch im semantischen Kontext vergleichen können. In Anlehnung an die Idee zu „Latent Semantic Analysis“ wurde von Teufl et al. [4] ein Konzept entworfen, um semantische Beziehungen in beliebigen Eingabedaten zu modellieren. Neuere wissenschaftliche Ansätze, die darauf abzielen, ein semantisches Verständnis von Wörtern abzubilden, sind word2vec [5] und GloVe [6]. Wörter werden dabei als Vektoren in einen hochdimensionalen Raum transformiert, der die semantischen Beziehungen zwischen ihnen repräsentiert. Vektoren, die nahe beisammen liegen, weisen dabei auf ähnliche semantische Bedeutungen hin. Diese als „Word embeddings“ bekannten Wortbeschreibungen zeigen für einzelne Wörter auf, mit welchen sie verwandt sind, da sie gemeinsam oder in Nähe zueinander vorkommen. Da gewisse sprachliche Konstrukte wie Präpositionen oder Artikel verhältnismäßig oft vorkommen, jedoch per se deshalb nicht mehr Informationen liefern, würde eine, sich rein auf Häufigkeit von Wörter beschränkende, Technik die Qualität der Klassifikation reduzieren. Neuere Methoden wie word2vec sind sich diesem Umstand bewusst und setzen ein sog. „Sub-sampling“ häufiger Wörter ein. Häufig vorkommende Wörter werden dabei verhältnismäßig geringer gewichtet oder gar entfernt, seltene dafür umso höher priorisiert. Wie nachfolgend noch illustriert, kann jedoch durch manuelles „Feature Engineering“ eine bewusste Steuerung vorgenommen werden.

Für die hier vorgenommene Analyse von Beschreibungen von Anwendungen sind diese Techniken deshalb interessant, da sich so potentiell Anwendungen finden ließen, die von der Beschreibung und womöglich auch ihrer Implementierung her Ähnlichkeiten aufweisen. Der Vorteil gegenüber einer reinen Textkategorisierung liegt also darin, dass keine Kategorien manuell definiert oder automatisiert abgeleitet werden müssen. Eine Unterteilung in Kategorien ist mithilfe von Clustering-Techniken natürlich dennoch möglich. Durch die Betrachtung semantischer Informationen, anstelle des reinen Fokus auf Häufigkeit, im Text steigt zudem die „Qualität“ bzw. Aussagekraft der Vektoren und dadurch implizit des gesamten Modells. Vergleichbare Ansätze zur Auswertung semantischer Informationen in Anwendungen [7, 8] zielen darauf ab, Anwendungen zu finden, die in schädlicher Absicht modifiziert und neu gepackt wieder verteilt wurden. Wohingegen der Fokus dabei auf der Erkennung minimaler Unterschiede zwischen Anwendungen liegt, beabsichtigten wir bei der Klassifikation von Beschreibungen ein globaleres Verständnis der Texte zu erhalten, um sie in weiterer Folge miteinander vergleichen zu können.

Angesichts der eingangs vorgestellten Zielsetzungen würde es die Verwendung eines Klassifikationsalgorithmus, der semantische Informationen im Text auswertet, auch ermöglichen, zueinander, von der Beschreib her ähnliche Anwendungen zu finden. Potentiell handelt es sich dabei den gleichen Zweck, der aber von der Implementierung unterschiedlich umgesetzt wurde. Wird in weiterer Folge ein Clustering-Algorithmus auf die als ähnlich klassifizierten Applikationen angewandt, könnte dies Rückschlüsse auf eine gemeinsame Programmkategorie erlauben. Hinsichtlich der von Google und Apple gewählten monolithischen Definitionen von Kategorien würde sich somit eine deutlich flexiblere und realitätsgetreuere Unterteilung von Applikationen ermöglichen.

3.3.1. Klassifikation von Anwendungsbeschreibungen über word2vec

Eine vergleichsweise einfach zu verwendende Implementierung des Algorithmus von Mikolov et al. wird in einem quelloffenen Framework namens gensim[5] bereitgestellt. Für die nachfolgende Analyse wird hierauf zurückgriffen.

Bevor ein semantisches Modell mithilfe von gensim trainiert werden kann, müssen die von „PlayDrone“ erhaltenen Metadaten für ihre Verwendung hin aufbereitet werden. Nach ersten Experimenten mit den vorliegenden Daten hat sich gezeigt, dass in den Texten auch Symbole, Punktationen, URLs und sonstige Sonderzeichen vorkommen, die nicht als gewöhnliche Wörter gesehen werden können. Analog zu der in Abschnitt 3.2.3 beschriebenen Vorgehensweise der Aufbereitung von Strings wurden auch bei den Anwendungsbeschreibungen zunächst Leerzeichen, Punktationen, Zeilenumbrüche etc. entfernt. Darüber hinaus wurden URLs extrahiert und verworfen, da ihnen naturgemäß kein Charakter zuteilwerden kann.

Die aufbereiteten Features wurden daraufhin „tokenisiert“ d.h. in Wörter unterteilt und zusammen mit der Bezeichnung der Anwendung (als sog. „Tag“) als Vektoren an word2vec übergeben. Als Resultat ergab sich ein semantisches Modell, das zueinander ähnliche Wörter in Beschreibungen sowie Applikationen finden kann.

Zur besseren Veranschaulichung wird nachfolgend an einigen Beispielen die praktische Tauglichkeit des Modells vorgezeigt. Die Ergebnisse zeigen neben den Auffindungen jeweils die Wahrscheinlichkeit der Ähnlichkeit bzw. ein Maß für die semantische Korrelation der Ausdrücke.

Suche nach Verwandtheit von Ausdruck ‚wien‘ in Beschreibungen von 40.000 Applikationen:

```
[('wien', 1.0),
 ('schwechat', 0.6495869159698486),
 ('graz', 0.6441542506217957),
 ('eberswalde', 0.6405308246612549),
 ('genf', 0.6354057788848877),
 ('biesenthal', 0.629092812538147),
 ('salzburg', 0.6271940469741821),
 ('venedig', 0.6236460208892822),
 ('mödling', 0.6234492063522339),
 ('niederösterreich', 0.6224139928817749)]
```

Wie in der Aufstellung gut erkennbar ist, findet eine Suche nach dem Terminus „wien“ semantisch verwandte Objekte in Bezug auf ihre geographische Nähe, als Hauptstadt oder Bundesland.

Suche nach Anwendungen mit Ausdruck „wien“ + „karte“ in den Beschreibungen:

```
[('wien.liman_restaurant.app', 0.7743138670921326),
 ('com.mdv.VRTCompanion', 0.7399436235427856),
 ('com.mdv.BSVAGCompanion', 0.7392444014549255),
 ('ma25rentcalc.s4w.at', 0.7338329553604126),
 ('de.park', 0.7313312292098999),
 ('com.mdv.RheinbahnCompanion', 0.7291196584701538),
 ('de.applicate.android.centralberlintogo', 0.728323221206665),
 ('com.mdv.DINGCompanion', 0.7269071340560913),
 ('de.inmediasres.mythischeorte', 0.7251682877540588),
 ('com.wetterdo', 0.7226649522781372)]
```

Eine kombinierte Suche nach zwei Vokabeln führt intern zu zwei Vektoren. Die Nachbarn beider Vektoren werden folglich mit Angabe ihres implizierten Grad an Ähnlichkeit ausgegeben. Die Ergebnisse zeigen zudem die Effekte von Polysemie in natürlicher Sprache auf. „Karte“ kann sowohl im geographischen Kontext verstanden werden, als auch im Sinne einer „Speisekarte“. Die erstgefundene Anwendung verweist dabei auf die Mobilapplikation eines Restaurants, die verbleibenden Treffer indizieren Anwendungen zur Navigation oder Auffindung von Orten in Wien.

[5] https://radimrehurek.com/gensim/index.html

Suche nach „facebook“ in den Beschreibungen von 40.000 Anwendungen:

```
[('facebook', 0.9999999403953552),
 ('twitter', 0.9196221828460693),
 ('instagram', 0.8319854140281677),
 ('social', 0.7319374084472656),
 ('pinterest', 0.7220771312713623),
 ('tumblr', 0.6985977292060852),
 ('whatsapp', 0.693464994430542),
 ('email', 0.6870206594467163),
 ('share', 0.6764687895774841),
 ('linkedin', 0.6573704481124878)]
```

Bei der Abfrage des Modells nach semantischen Ähnlichkeiten zum Terminus „facebook“, ist die Ausgabe eine Liste mit ähnlichen Produkten. Es ist dabei klar erkennbar, dass die Rolle dieses Suchworts beim Training dahingehend berücksichtigt wurde, um auch Analogien finden zu können. In Bezug auf die eingangs formulierten Fragestellungen ist das ein klares Indiz dafür, dass dieser Ansatz das Potential hat, auch einzelne Kategorien von Anwendungen zu identifizieren, ohne, dass hierfür gesondert ein Clustering vorgenommen wird.

Suche nach „secure“ + „messenger“ in den Beschreibungen von 40.000 Anwendungen:

```
[('com.securdata.android', 0.7514603734016418),
 ('com.mindprotectionkit.freechat', 0.7437781691551208),
 ('org.telegram.wXeonMessenger3', 0.7083243131637573),
 ('org.jastrzab.serwer', 0.7008682489395142),
 ('com.test.found', 0.6977579593658447),
 ('it.klaim', 0.6877865791320801),
 ('it.geeklogica.imgeek', 0.6877852082252502),
 ('ryanjoh.talktalk', 0.6831209063529968),
 ('com.martin.natterbox', 0.6830791234970093),
 ('com.kfmes.jateroid', 0.6820418834686279)]
```

Bei einer kombinierten Suche nach “secure” und “messenger” zeigen die Ergebnisse mit vergleichsweise hoher Wahrscheinlichkeit eine Relation zu Produkten, die diese Funktionalität tatsächlich implementieren. Gleichzeitig fällt auf, dass sich in dieser Auflistung mit Ausnahme von „Telegram“ kein verbreitet populäres Derivat eines „Secure Messenger“, wie etwa „Signal“, findet. Angesichts der Auswahl eines beschränkten Sets von nur 40.000 Anwendungen für diesen Test, drängt sich die Annahme auf, dass gewisse Kandidaten schlichtweg nicht enthalten sind oder ihre Beschreibungen (und mitunter Funktionalität) zum Zeitpunkt der Anwendungserfassung durch „PlayDrone“ noch nicht im Kontext von „Secure Messenger“ standen.

4. Fazit

Im Zuge dieses Projekts wurde ein Konzept erarbeitet, um qualitative Metadaten von Mobilanwendungen zu identifizieren und zu klassifizieren. Da Metadaten oft auch Indikatoren für den Zweck und die Funktionalität von Anwendungen sind, können sie dabei helfen, ein kontextuelles Verständnis davon zu bekommen, wie Anwendungen umgesetzt wurden. Nicht zuletzt eignen sich Metadaten auch als Anhaltspunkt für die Erkennung von Malware.

Um aus Metadaten eine sinnvolle Aussage ableiten zu können, ist es essentiell, eine geeignete Strategie der Aufbereitung und Verarbeitung zu verfolgen. In diesem Bericht wurden die wesentlichen Daten aufgeschlüsselt und infolgedessen eruiert, inwieweit sie sich als Input für einen Klassifikationsalgorithmus eignen würden. Mithilfe praktisch durchgeführter Versuche an realen Android-Anwendungen konnte ein realistischer Eindruck davon gewonnen werden, aus welchen Daten sich tatsächlich brauchbare Schlüsse ziehen lassen und welche anderen einen zu generischen Charakter haben. So wurde beispielsweise identifiziert, dass eine Layout-Beschreibung im XML-Format zu wenig individuellen Informationsgehalt aufweist, um darauf aufbauend, Anwendungen zu klassifizieren. Hingegen konnte nach Anwendung fortschrittlicher Methoden aus dem Bereich „Natural Language Processing“ bzw. „Machine Learning“ gezeigt werden, dass sich aus den Beschreibungen von Anwendungen ein semantisches Modell ableiten lässt, das Auskunft geben kann über den Kontext von Anwendungen. Diese Informationen können insbesondere auch

in Situationen hilfreich sein, wenn es darum geht, Anwendungen dynamisch in Kategorien einzuordnen und Querverbindungen mit anderen herzustellen. Im Hinblick auf vermehrt auftretende Fälle von Malware in Form von Mobilanwendungen findet sich hier ein tauglicher Ansatz, um ähnliche Kandidaten aufzudecken und ggf. dann weiter zu analysieren.

Die Erkenntnisse aus diesem Projekt beschränken sich rein auf die Metadaten und klammern dabei bewusst aus, dass die tatsächliche Implementierung einer Anwendung nicht notwendigerweise mit der Beschreibung in den jeweiligen „AppStores“ korrespondieren muss. Eine Einbeziehung des Quellcodes von Anwendungen wäre also eine sinnvolle wechselseitige Ergänzung, um besser zu verstehen, welchen Zweck, welche Funktionalität Applikationen implementieren und in welcher semantischen Ähnlichkeitsbeziehung sie zu anderen stehen.

Referenzen

[1] N. Viennot, E. Garcia und J. Nieh, „A Measurement Study of Google Play,“ *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2014),* 2014.

[2] L. Li, „AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community,“ 2017.

[3] K. Weinberger, A. Dasgupta und J. Langford, „Feature hashing for large scale multitask learning,“ *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09),* 2009.

[4] P. Teufl, H. Leitold und R. Posch, „Semantic Pattern Transformation: Applying Knowledge Discovery Processes in Heterogeneous Domains,“ *13th International Conference on Knowledge Management and Knowledge Technologies, I-KNOW '13,* 2013.

[5] T. Mikolov, I. Sutskever und K. Chen, „Distributed Representations of Words and Phrases and their Compositionality,“ *Neural Information Processing Systems NIPS,* 2013.

[6] J. Pennington, R. Socher und C. Manning, „Glove: Global Vectors for Word Representation,“ *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP,* 2014.

[7] J. Crussell, C. Gibler und H. Chen, „Andarwin: Scalable detection of semanitcally similar Android Applications,“ *ESORICS,* 2013.

[8] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. M. Al-Kofahi und T. N. Nguyen, „Accurate and efficient structural characteristic feature extraction for clone detection,“ *Fundamental Approaches to Software Engineering,* 2009.