Kann Malware in Android-Apps automatisch gefunden werden?

Cöllen, Markus
Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Str. 10, 68163 Mannheim

Zusammenfassung—**Mit der wachsenden Beliebtheit der auf Android basierten Smartphones wächst zunehmend die Anzahl der dazu gehörigen Apps im Google Play Store. Das Tempo indem dort Apps hochgeladen werden steigt rasant und hat eine manuelle Überprüfung dieser schon längst unmöglich gemacht. Durch das einbinden von Smarphones in kommerzielle Geschäfte macht diese auch in Zukunft zu attraktiven Zielen von schädlicher Software. Malware Entwickler kreieren immer komplexere und ausgefallenere Verschleierungsmechanisem um ihre Software für Virenprogramme unauffindbar zu machen. In diesem Paper werden diese Verschleierungsmechanisem erläutert und Ansätze um diese so gut es geht aufzudecken.**

Inhaltsverzeichnis

1. Einleitung

Smartphones bieten aufgrund ihrer vielen Schittstellen zur Außenwelt, wie z.B. WLAN, Bluetooth, NFC, USB usw. viele Angriffspunkte. Ein Hauptgrund, warum Smartphones mit dem Android Betriebssystem so beliebt für Angreifer sind, ist der hohe Verbreitungsgrad dieser Geräte. Dieser liegt im Jahr 2018 bei ca. 86 Prozent und ist somit Sechs mal so hoch wie der Konkurrent IOS mit knapp 16 Prozent Marktanteil [16]. Zudem werden Smartphones zunehmend in kommerzielle Geschäfte eingebunden, dies macht Smartphones auch zukünftig als Ziel für Attacken sehr attraktiv. Da Malware-Autoren immer mehr auf Verschleierungsmechanismen wie Komprimierung, Verschlüsselung oder sich selbst modifizierender Programmcode setzten und diese sich stetig weiter entwickeln, wird es für Antiviren Herstellern zunehmend schwieriger diese als schädliche Software zu erkennen [15].

2. Grundlagen

2.1. Was ist Android?

Das Betriebssystem Android ist eine umfassende Open Source Plattform für Smartphones. Eigentümer des Android Betriebssystem ist die Open Handset Alliance, welches ein Konsortium von 84 Unternehmen ist, Ihr Ziel ist es ëine beschleunigte Innovation im Mobilbereich, um dem Benutzer ein reicheres, kostengünstigeres und besseres Mobilerlebnis zu bieten"[13]. Erstmalig gibt es eine wirklich offene Plattform, welche Hardware und Software von Smartphones trennt. Weil Android unter der Apache Open-Source Lizenz freigegeben ist und das Betriebssystem auf dem Linux-Kernel basiert, können Hersteller das Android Betriebssystem beliebig verwenden und für ihre Produkte anpassen. Als Einsatzgebiete für Android gibt es Smartphones, Tablets, Notebooks aber auch in z.B. Autos kommt das Betriebssystem zum Einsatz.

2.2. Android Architektur

Die Android Architektur basiert auf dem Vier-Schichtenmodell, erstens dem Linux Kernel mit den dazu gehörigen Hardwaretreibern, zweitens der Android Runtime mit der Dalvik Virtual Machine und den Java Core Libraries sowie den Standard-Bibliotheken , drittens dem Application Framework und auf der obersten Schicht den eigentlichen Anwendungen.

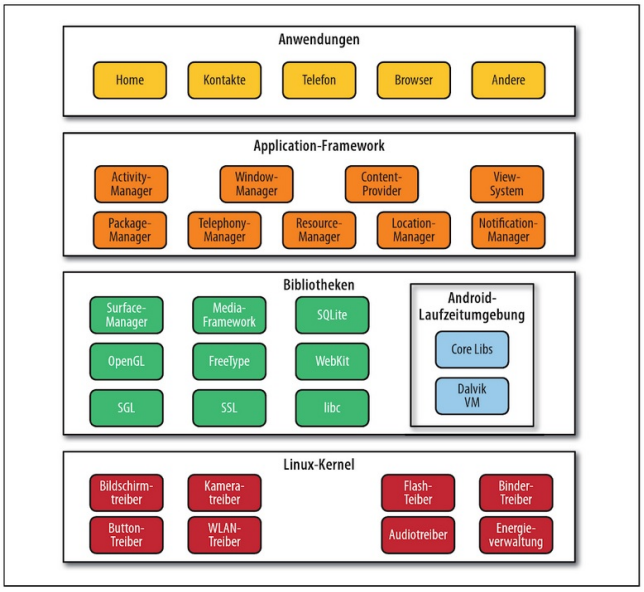

Abbildung 1. Android Architektur [13]

Der Linux-Kernel dient als eine Abstraktion zwischen Software und Hardware und somit kann Android auf verschiedenen Geräten eingesetzt werden. Aufgrund der Tatsache, dass Linux über ein etabliertes Sicherheitskonzept verfügt [20], greift Android auf viele Schutzmechanismen des Linux Kernels zurück oder benutzt diese als ihre Basis[19]. Zudem verwendet Android viele von Linux mitgebrachten Funktionen wie z.B. die Unterstützung für Speicherverwaltung, Power-Management und Netzwerkzugriff. Wie man in Abbildung 1 sehen kann, gibt es zudem noch viele von Android mitgebrachte Bibliotheken wie SQLite (eine SQL Datenbank) oder OpenGL (eine 3-D-Grafikbibliothek). Bis zu den Android Versionen 4.x war die Dalvik VM ein Hauptbestandteil des Android Betriebssystem. Hierdurch wurde die geschriebenen Apps erst in Bytecode formatiert wenn sie auch wirklich gebraucht werden. Ab Android 5.x wurde dies durch den Ahead-of-time-Compiler ersetzt. Dieser übersetzt die Apps schon beim installieren in Bytecode-Format. Hierdurch wird sich bessere Performance und eine Beschleunigung von Apps erhofft ohne dabei Flexibilität verzichten zu müssen[3]. Die Schicht Application-Framework stellt einem Entwickler die unterschiedlichsten Dienste zur Verfügung. Hierzu zählen viele Dienste, welche für eine Applikation die Infrastruktur zur Verfügung stellt, wie die Positionsermittlung, Sensoren, WLAN oder Telefoniefunktionen. Diese Schicht ist für die Programmierung von Android Apps die wichtigste und daher auch am gründlichsten Dokumentiert. Die oberste Schichte sind die Anwendungen. Diese sind teilweise schon vorinstalliert oder können heruntergeladen und installiert werden.

2.3. Android Update Problematik

Wie man in Abbildung 2 sehen kann verdrängen die neueren Android Versionen die älteren. Allerdings ist Android Oreo, mit gerade mal 4,6 Prozent, eine der am wenigsten vertretenen Version, obwohl sie die neueste und auch schon seit dem 27. August 2017 auf dem Markt ist.

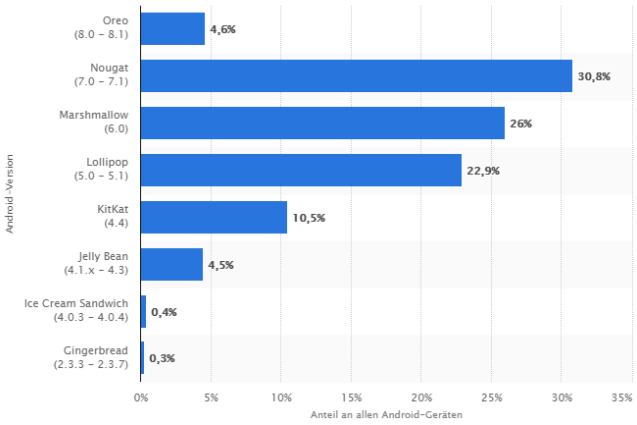

Abbildung 2. Verteilung der Android-Versionen im Jahr 2018 [1]

Dies liegt daran, dass nicht jede Android Version mit jedem Handy kompatibel ist. Die Smartphone Hersteller müssen die Versionen an ihre Geräte und die jeweilige Hardware anpassen. Diese umfassen Anpassungen an WLAN, Kamera, Bluetooth, GPS usw. Anschließend müssen die erstellten Anpassungen noch umfangreich getestet werden, was auch noch viel Zeit in Anspruch nimmt. Weil die Hersteller oft neue Produkte verkaufen wollen und das Updaten älterer Smartphones zu teuer und zeitaufwendig ist, bleibt den Nutzern oft nichts anderes übrig, als neue Smartphones zu kaufen. Andernfalls behält man ein Smartphone mit einer veralteten Version und bekannten Sicherheitslücken. [19].

2.4. Application Sandbox

Linux arbeitet als ein Mehrbenutzer-Betriebssystem. Dies bedeutet, dass es mehrere Nutzer geben kann und das Betriebssystem verhindert, dass Daten eines Nutzers von einem anderen Nutzer eingesehen, geändert oder gelöscht werden können. Diese Nutzermanagement wird beim Application Sandboxing verwendet. Hierbei erhält jede Applikation eine eigene Nutzer-ID und führt diese in einem separaten Prozess aus. Somit gewährleistet Android, dass eine Applikation anderen Applikationen oder dem Betriebssystem keinen Schaden zufügen kann. Sollte eine Applikation aber durch das Ausnutzen einer Schwachstelle oder einer Sicherheitslücke an Root-Rechte kommen, kann die Applikation dieses Application Sandboxing einfach umgehen. Wie schon in Abschnitt 2.3 beschrieben, gibt es wegen den stark verzögerten Android Updates oft Sicherheitslücken, welche über längere Zeit bekannt sind und leicht ausgenutzt werden können um Root-Rechte zu erlangen [13].

2.5. Permission Model

Standardmäßig hat eine Applikation keinen Zugriff auf Daten außerhalb der Sandbox. Sollte eine Applikation Systemressourcen außerhalb dieser verwenden wollen, muss erst eine Zugriffsanfrage gestellt werden. Geschützte Ressourcen sind z.B. Kamera, SMS, Bluetooth usw. [17]. Diese Rechte können bei einer Anfrage allerdings nur angenommen oder abgelehnt werden, bestimmte Einschränkungen können also nicht vorgenommen werden.

Hier sollte der Nutzer sich Gedanken darüber machen, welche Applikationen welche Systemressourcen wirklich benötigen. Sollte ein Spiel wie z.B. Snake Rechte anfordern SMS zu verschicken, könnte dem Nutzer klar werden, dass etwas mit der Applikation nicht stimmt.

2.6. Sicherheit des Google Play Store

Anders als bei IOS, bei welchem die Nutzer an den Apple Store gebunden sind, können die Android Benutzer selbst entscheiden welchen Store sie verwenden. Der Google Play Store ist mit 82 Milliarden Downloads [2] mit Abstand die größte Plattform. Er wurde am 28. August 2008 unter dem Namen Android Market veröffentlicht. Dieser bietet den Nutzern einfaches herunterladen und installieren von mobilen Anwendungen, sogenannten Applikationen. Im Google Play Store sind inzwischen über 3,7 Millionen Anwendungen bereit zum Herunterladen [14] und jeden Monat kommen ca. 30.000 neue Applikationen hinzu [5]. Durch den rasanten Wachstum steigt auch die Anzahl von schädlicher Software, sogenannter Malware. Der Anteil von bösartigen Applikationen ist von 2011 bis 2013 um 388% gewachsen [18]. Da nicht alle Anwendungen von Mitarbeitern geprüft werden können, hat Google das Programm Bouncer ins Leben gerufen (siehe Kapitel **??**). Android bietet zudem die Möglichkeit per Remote-Verbindung Applikationen zu installieren und zu löschen. Hierfür braucht man lediglich Zugriff auf den Google Play Account. Falls ein Angreifer an diese Daten kommen sollte, könnte er ohne Erlaubnis des Nutzers, Applikationen aus dem Google Play Store auf dem Smartphone installieren.

2.7. Malware

Bei dem Begriff Malware handelt es sich um ein Kunstwort, welches sich aus malicous und software zusammensetzt und bezeichnet Programme, welche unerwünschte oder auch schädliche Funktionen ausführen. Maleware kann man prinzipiell in drei unterschiedliche Hauptkathegorien unterteillen, Viren, Würmer und Trojaner. Ein Virus ist eine Software, welche sich selbst vervielfälltigen kann. Durch das Ausführen einer Infizierten Anwendung wird das Programm gestartet und schleust sich in anderen Programmen oder Dokumenten ein. Die sogenannten Würmer haben ähnliche Eigenschaften wie Vieren, der Hauptunterschied liegt darin, dass sie sich über das Intra- order Internet vermehren können und oft keine Interaktion mit den betroffenen Benutzern benötigen. Die dritte und gefährlichste Kathegorie ist das sogenannte Trojanische Pferd. Dieses täuscht dem Nutzer ein nützliches Programm vor, führt allerdings ohne Wissen des Benutzers im Hintergrund schädliche Funktionen und Routinen durch. Allerdings enthält die meiste Malware mehr als nur eine schädliche Funktion, was eine eindeutige Klassifikation fast unmöglich macht. Weitere Arten sind zudem Spyware, Exploits, Backdoors und Rootkits. Jede dieser Arten kann von ihren Maleware-Authoren druch Verschleierungsmechanismen wie Komprimierung, Verschlüsselung oder sich selbst modifizierender Programmcode verschleiert werden, was das Auffinden durch Vierenprogramme um einiges erschwert. [15] Die erfolgreichsten und am meisten genutzten Arten sind dabei Code Obfuscation, Anti-Analyse, Laufzeitpacker, Polymorphie und Metamorphie.

Code Obfuscation bezeichnet dabei die Transformation des Quellcodes mit dem Ziel, die Ermittlung der Semantik und der Funktionalität zu erschweren ohne dabei die eigentliche Funktionalität des Programmes zu verändern [8]. Hierzu werden zum Beispiel leere Berechnungen eingebaut, Funktionen in viele kleine Teile aufgespalten oder Befehlsabfolgen kompliziert Reorganisiert.

Anti-Analyse beschreibt hingegen das torpedieren wichtiger Werkzeuge wie Disassembler, Debugger und Emulatoren welche für das erstellen von Control Flow-Graphen 3.2.1 gebraucht werden.

Laufzeitpacker sind Programme, welche den Programmcode komprimieren. Dies kann man sich ähnlich wie bei einem ZIP-Archiv vorstellen. Dies hat zur Folge, dass bei Ansicht des Programmcodes nur der Code des Packers sichtbar ist und nicht der Code der eigentlich ausgeführt wird.[6] Die Entschlüsselung der sogenannten Payload geschieht durch einen Decrypter zur Laufzeit.

Polymorphie hat die Bedeutung Vielgestaltigkeit. Sollte das Programm nun von einem Laufzeitpacker verschlüsselt worden sein aber nach jeder Replikation seinen Schlüssel zur Ver- und Entschlüsselung ändern, nennt man dies ein polymorphes Programm. Dieses Verfahren wird in Abbildung 3 gezeigt.

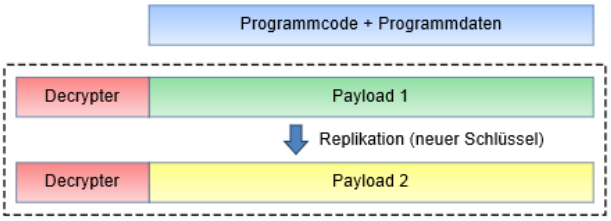

Abbildung 3. Polymorphie des Programmcodes [10]

Metamorphie hat die Bedeutung Verwandlung. Sollte nun nicht nur, wie in Abbildung 3 gezeigt ist, die Payload bei jeder Replikation geändert werden, sondern auch der Decrypter, spricht man von einem Methamorphosen Programm. Dies hat zur folge, dass der gesamte Quellcode nach jeder Replikation verändert ist. Trotz der gesammten Veränderung des Quellcodes funktioniert jede Version der Malware auf die selbse Art und Weise. Dies hat zur folge, dass signaturbasierte Malwareerkennung, welche in Kapitel 4.1 beschrieben wird, absolut keine Chance hat, neue Versionen der Malware zu erkennen.[10]

Malware stellt wegen all dieser Verschleierungsmechanismen eine immer größer werdende Bedrohung dar und durch den rasanten Wachstum des Smartphone-Marktes ist eine manuelle Auswertung mittlerweile unmöglich geworden. Obwohl es sich bei vielen neuen Arten um verschiedene Varianten bereits bekannter Malware handelt, müssen Analysten erst jedes Sample erneut analysieren um dies feststellen zu können [21]. Bei der Analyse kann grundsätzlich in zwei Arten unterschieden werden: Statische und Dynamische Analyse (siehe Kapitel 3.2 und 3.1).

3. Ansätze zur Merkmalsgenerierung

3.1. Dynamische Analyse

Bei der dynamischen Analyse wird das Programm während der Laufzeit untersucht. Der Ablauf besteht dabei aus drei Teilen: der Ausführung des Programms, der Protokollierung des Ablaufes und der Ergebnisse und das Analysieren dieser [9]. Ein Beispiel der dynamischen Analysemethoden ist das Testen. Ein bedeutender Nachteil ist es, dass nicht immer alle Eingabeparameter überprüft werden können da diese exponentiell mit der Anzahl der Parameter steigt. Daher werden oft nur Rand- und Grenzfälle geprüft. Weitere Probleme können bei der Netzkommunikation, der Erstellung von Zufallszahlen wie auch bei Nutzereingaben auftreten. Wegen all dieser Eigenschaften eines Programmes ist eine Reproduzierbarkeit der Ergebnisse nicht gegeben. Zudem gibt es die Möglichkeit für Malware während Laufzeit zu erkennen, ob sie überprüft wird und kann sich gegebenenfalls zu diesem Zeitpunkt harmlos verhalten. Viele dieser Nachteile lassen sich durch eine statische Analyse vermeiden welche im Abschnitt 3.2 vorgestellt wird.

3.2. Statische Analyse

Bei der statischen Analyse wird versucht durch reverse engineering Code aus der Maleware zu extrahieren um somit auf das Verhalten des Programmes schließen zu können. Dieser extrahierte Code kann anschließend eingehend untersucht werden. [21]. Diese Analysemethode wird also im Gegensatz zu dynamischen Analyse nicht zur Laufzeit des Programms angewandt und somit muss die App auch nicht ausgeführt werden. Da die korrekte Abbildung der Struktur des Quellcodes eine komplexe Aufgabe ist, wird hierzu oft ein Compiler verwendet. Dieser erzeugt aus dem Quellcode mit der Hilfe eines Lexers einen Tokenstream, also eine Folge aus den kleinsten sinntragenden Einheiten des Programms wie Literalen, Bezeichnern und Schlüsselwörtern, ein Beispiel hierfür wäre eine if-else Anweisung. Anschließend erzeugt ein Parser aus dem Tokenstream einen abstrakten Syntaxbaum, welcher die syntaktischen Zusammenhänge der einzelnen Tokens darstellt. Durch eine anschließende semantische Analyse wird aus dem Syntaxbaum ein Graph, auf welchen beispielsweise eine Kontrollflussanalyse3.2.1 oder Datenflussanalyse 3.2.2 durchgeführt werden kann.

3.2.1. Kontrollflussanalyse. Bei der Kontrollflussanalyse wird untersucht, welcher Block im zu untersuchendem Programm die Kontrolle an welchen Block übergibt, und welche Funktionen dabei erreichbar sind. Zudem wird überprüft welche Werte Parameter einer Funktion möglicherweise zugewiesen bekommen. Um die Ergebnisse zu Analysieren wird ein Kontrollflussgraph erstellt. Dieser besteht aus einer Menge Knoten welche die Grundblöcke des Programmes darstellen. Zudem gibt es noch eine Menge von gerichteten Kanten, welche einen Übergang von einem Block zu einem anderen darstellen. Diese zeigen wie die Kontrolle von einem Block zu einem anderen übergeht. [23] Die Abbildung 4 zeigt ein Beispiel eines solchen Kontrollflussgraphen.

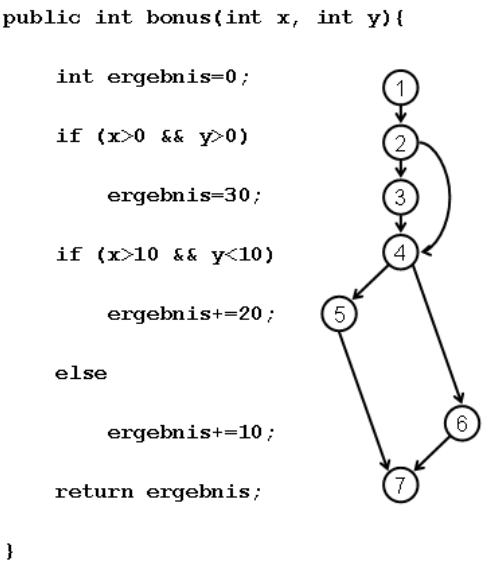

Abbildung 4. Kontrollflussgraph [7]

Wie man in Abbildung 4 sehen kann, wird für jede Anweisung ein Block erstellt. int ergebnis=0; ist die erste Anweisung und somit auch Block 1. Block 2 ist die if Bedingung und je nachdem ob sie erfüllt wird geht sie zu Block 3, ergebnis=30; oder zur nächsten if Bedingung über. Bei dieser wird entweder der if Block oder der else Block ausgeführt, daher kommt auch die Verzweigung bei Block 4. return ergebnis; wird immer ausgeführt, daher laufen Block 5 und Block 6 auch in Block 7 wieder zusammen. Diese Kontrollflussgraphen werden auch zum überprüfen der Überdeckung verwendet.

3.2.2. Datenflussanalyse. Bei der Datenflussanalyse wird untersucht, welche Teiles des Programmes welche Daten weitergeben und welche Abhängigkeiten daraus resultieren. Als Grundlage dieser Methode wird ein Kontrollflussgraph aus Kapitel 3.2.1 verwendet. Die Datenflussanalyse untersucht, wie sich Daten durch die einzelnen Blöcke verändern. Enthält ein Block beispielsweise den Code x=1, so verändert sich der Wert von x nach dem Block auf 1. Diese gesamten Änderungen jedes Blocks werden überwacht und aufgezeichnet. [23]

3.3. Gegenüberstellung von statischer und dynamischer Analyse

Durch eine dynamische Analyse können durch die Ausführung des Programmes konkrete Fehler gefunden werden, allerdings kann durch diese Methode nicht die Fehlerfreiheit des Programmes bewiesen werden, da das Programm meistens nicht mit allen Parametern gestartet und durchgeführt werden kann. Bei der statischen Analyse kann das Ausführen des Programmes mit allen möglichen Parametern betrachtet werden. Somit kann bewiesen werden, dass bestimmte Fehlerarten nicht auftreten können. Allerdings können Fehler auftreten welche durch das Fehlen entfernter Zusammenhänge entstehen. Zudem ist das Untersuchen mit dem statischen Ansatz Ungefährlich, da das Programm nicht laufen muss. Beide Analysemethoden haben Vor- und Nachteile, ergänzen sich allerdings

gegenseitig gut, daher sollten beide Methoden zusammen verwendet werden. [11]

4. Erkennungsmechanismen

4.1. signaturbasierte Malwareerkennung

Die am meisten Vertretene Methode der Malwareerkennung ist die auf Signatur basierte Erkennung von Malware. Diese Methode basiert auf der statischen Analyse 3.2 und extrahiert aus einem potentiellen Malware-Programm eine eindeutige Byte-Folge , welche als eindeutige Identifikation und Signatur dient, und vergleicht diese mit allen verfügbaren Signaturen in der Erkennungssoftware Datenbank. Wird diese Signatur in einem anderen Programm gefunden, kann mit einer hohen Sicherheit davon ausgegangen werden, dass das Programm schädliche Komponenten enthält. Diese Methode ist unverzichtbar bei jeder Antivirus Software, denn durch diese Verfahren wird ein sehr großer Teil der Schädlichen Software gefunden und gefiltert. Der erste große Nachteil dabei ist, dass nur bereits bekannte Malwarearten gefunden werden können. Gegen Polymoprhie oder Metamorphie hat dieses Verfahren absolut keine Chance. Zudem ist das Gerät in der gesamten Zeit in der die Signaturen verglichen werden gegen Angriffe ungeschützt. [15]

4.2. Virenschutz durch Machine Learning

Glaubt man den Firmen , welche Machine Learning für ihr Virenerkennung einsetzten, so hat die traditionelle Malwareerkennung ein Ende gefunden. Sobald man sich mit Machine Learning auseinander setzt wird man auch auf den Begriff Künstliche Intelligenz stoßen. Dieser ist ein Teil des Forschnungsgebietets der Informatik. Allerdings ist Intelligenz nur mangelhaft definiert und somit lässt sich dieses Forschnungsgebietet nur so einschränken, dass es sich nicht um eine Abarbeitung festgelegter Befehlssequenzen handelt. Die Berücksichtigung weitere Merkmale wie zum Beispiel Verarbeitung von Sprache, Visuelle Erkennung, Kreativität, Schlussfolgern und allgemeine Lernfähigkeit hängen von der Sicht des Benutzers ab. Das Lernen in Machine Learning impliziert dabei, dass Informationen Rückmeldung von Außerhalb dazu führen, dass eine Aufgabe besser gelöst werden kann. Dabei spielen folgende drei Kernaufgaben des Machine Learnings eine große Rolle. Erstens, eine Vorhersage treffen, zweitens Ursache finden und verstehen und drittens Muster in meistens nicht sortierten Daten erkennen. Umso mehr Daten in der sogenannten Lernphase eingespielt werden, desto sicherer und höher ist die Erkennungsrate. Virenschutzprogramme mit Machine Learning verwenden zwei Ansetzte, die statische 3.2 und die dynamische Analyse 3.1. In der statischen Analyse wird der Inhalt, also der Quellcode eines Programmes bewertet, ähnlich wie bei der signaturbasierten Malwareerkennung. Bei der dynamischen Analyse wertet das Virenprogramm während der Laufzeit Informationen über Prozesse, Interprozesskommunikation, Dateisystemzugriffe, Muster in der Netzwerkkommunikation und Aktionen in der Registry aus. Der Virenschutz durch Machine Learning hat allerdings die selben Probleme wie andere Malwareerkennungsmethoden. Die Malware entzieht sich der Entdeckung durch verschiedene Verschleierungsmethoden, welche in Abschnitt 2.7 beschrieben wurden. Zudem können Neuronale Netze eine Aussage nur mit einer gewissen wahrscheinlichkeit ermöglichen, daher kann keine 100 %ige Erkennung garantiert werden. [6]

4.3. Integritätsprüfung

Damit sich ein Virus verbreiten kann, muss er sich in andere Daten hineinkopieren und dabei Änderungen an dieser vornehmen. Ein Virenschutz kann diese Änderungen erkennen und somit auf eine Infizierung schließen. Dabei bildet der Virenschutz eine einmalige Checksumme über jede Datei des Systems und speichert diese in einer Datenbank. Sollte nun ein Virus eine Datei verändern ändert sich die Checksumme dieser Datei. Wir nun dieser unterschied einer Checksumme festgestellt kann mit einer hohen Sicherheit auf ein Virus geschlossen werden. Mit dieser Technik ist es möglich die Anwesenheit auch bisher unbekannte Malware festzustellen. Das große Problem ist dabei, dass nicht nur Malware, sondern auch der Nutzer, legale Programme oder auch Updates Daten auf dem System verändern können. Zudem wird die Malware erst bei der erneuten Prüfung einer Checksumme erkannt und bis zu diesem Zeitpunkt könnte die Schadfunktion schon ausgeführt worden sein. Somit liegt es in der Hand des Entwicklers und des Anwenders zu entscheiden welche Änderungen legal und welche es nicht sind. [15]

4.4. Behavior Blocker

Beim Behavior-Blocking werden Dateiaktivitäten und deren Verhalten überwacht. Zuerst wird festgelegt welche Aktionen und Änderungen an Dateien erlaubt und welche verboten sind. Anschließen kann so überwacht werden, welches Programm gegen diese fest gelegten Regeln verstößt. Obwohl dies komplett in Echtzeit geschieht, kann bei einer Verhaltensanomalie nicht festgestellt werden, ob es sich tatsächlich um Malware handelt oder nur um eine ungewöhnliche, aber zulässige Aktion.[15] Vorteil dabei ist, dass unerlaubte Zugriffe sofort blockiert werden und somit keinen Schaden angerichtet werden kann. Nachteil dabei ist, dass viele nützliche Programme ebenfalls blockiert werden. Die Auswahl liegt also beim Nutzer, welche Aktionen er zulässt und welche nicht. [22]

4.5. Heuristische Analysis

Die heuristische Analyse ähnelt sehr dem signaturbasiertem Ansatz, aber im Gegensatz wird nicht nach einzigartigen Signaturen gesucht, sondern es werden Funktion- und Prozessaufrufen des Betriebssystem auf bestimmte Verhaltensmuster untersucht. Dabei gibt es zwei verschieden Ansätze der heuristische Analyse, die statische und die dynamische Analyse. Bei der statischen Heuristik wird wie auch bei dem signaturbasiertem Ansatz nach Byte-Folgen im Quellcode gesucht, allerdings ist hier der entscheidende Unterschied, dass hier nach bestimmten Abfolgen von Instruktionen gesucht wird. Die Byte-Folge

setzt sich somit aus bestimmten Instruktionen zusammen und diese beschreibt dann das Verhalten des Programmes. Zudem wird untersucht, ob verschlüsselte Malware bestimmte Systemfunktionen aufruft, um den eigenen Programmcode zu entschlüsseln. Durch diese Technik wird, wie alle bei jeder signaturbasierten Analyse, nur bereits bekannte oder verwandte Malware gefunden. Auch bei verschleierter Malware, wie bei polymorpher oder metamorpher Malware 2.7 versagt die statische heuristische Analyse. Bei der dynamischen Heuristik wird das zu untersuchende Programm in einer virtuellen Umgebung gestartet. Dadurch können z.B. z.B. alle Interrupt- Aufrufe, die das verdächtige Programm an das Betriebssystem schickt überwacht und erfasst werden. Es wird erlaubt, dass sich das Programm frei im emulierten Raum bewegen kann. Somit wird spätesten bei der kommunikation mit dem Betriebssystem klar, welche Absichten das Programm hat. Zudem können alle Schritte des Programmes überwacht werden, wie z.B. die Entschlüsselungsschritte. Zudem kann der entschlüsselte Quellcode des Programmes betrachtet werden. Durch dieses Verfahren kann somit auch noch unbekannte Malware gefunden werden. Da alles in einer simulierten Umgebung läuft, kann das infizierte Programm auch keine Schäden anrichten. Ein großer Nachteil dabei ist allerdings, dass diese Methode sehr Zeit intensiv ist. [15]

4.6. Taint Analysis

Die Taint-Analyse ist die Analyse und Erkennung von Datenflüssen, von privaten Datenquellen zu öffentlichen Datensenken. Viele der heutigen Malware verwendet einfache Mechanismen in einer Komponente um unerwünschte private Daten, den sogenannten Leaks, weiterzugeben. Diese sind mit relativ geringem Aufwand aufzudecken. Werden diese privaten Daten jedoch über mehrere Komponenten und Applikationen hinweg übergeben, ist es nicht mehr so einfach diese zu erkennen. Die Weitergabe von privaten Daten zwischen Applikationen ist, etwa im klassischen Sinne, als Rechteausweitung (privilege escalation) bekannt. Durch die Weitergabe der Daten ist keine direkte Verbindung zwischen privater Datenquelle und öffentlicher Senke zu erkennen. Für eine Erkennung eines solchen Leaks müssen intelligente Datenfluss- und Kommunikationstools verwendet werden. Zuerst müssen jedoch die Quellen und Senken richtig klassifiziert werden. Als Quelle wird der Ursprung privater Daten wie z.B. Standort des Gerätes oder Telefonbuchdaten bezeichnet. Allerdings sind nicht alle Daten private Daten. Öffentlich zugängliche Daten werden als öffentliche Quelle bezeichnet. Auch Senken werden mit diesem Prinzip klassifiziert. Jede Applikation hat auf Android ihren eigenen Speicherbereich. Endet eine Senke in einem solchen Speicherbereich wird diese als private Senke bezeichnet. Endet sie dagegen in einem nicht vertrauenswürdige Speicherbereiche, wie z.B. in einer SD-Karte oder in einem Netzwerk-Socket, also einem Speicherbereich außerhalb des Einflusses der Applikation, wird diese als öffentliche Senke bezeichnet. Ein Datenfluss ist nur dann tainted, also beschmutzt, wenn der Fluss in einer privaten Quelle startet und in einer öffentlichen Senke endet. [4] Wichtig dabei ist zu erkennen, welche Aliase eine Variable annimmt. Dies Zeigt das folgende Codebeispiel 5.

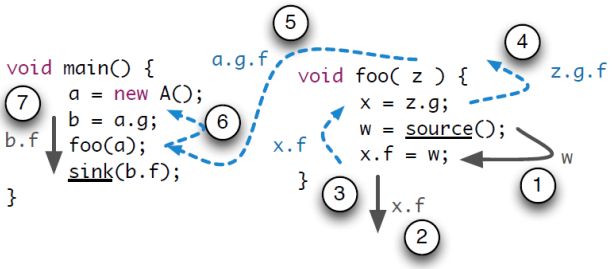

Abbildung 5. Taint Analysis [12]

Hier wird zuerst Untersucht, ob es eine Verbindung zwischen der private Quelle und der öffentliche Senke gibt. Als erstes wird jede Zeile nacheinander analysiert. Bei Nummer 1 wird dann eine private Datenquelle erkannt und somit beginnt die backwards oder auch rückwärts Analyse. Ab dort werden die Statements in der umgekehrten Reihenfolge analysiert und es wird erkannt, dass z.g.f, a.g.f und b.f Aliase von x.f sind. Somit wird erkannt, dass eine Verbindung zwischen Quelle und Sinke besteht und wird deshalb als tainted markiert.

5. Fazit

Durch den rasanten Wachstum des Smartphone-Marktes und die Einbindung von Smartphones in die kommerziellen Nutzung wird auch in der Zukunft dies ein attraktives Ziel für Angreifer bleiben. Daraus kann man schließen, dass immer komplexere und raffiniertere Verschleierungsmechanismen entwickelt werden. Da allerdings auch die Entwickler der Erkennungssoftware stets ihr Produkte weiterentwickeln, gibt es zum heutigen Entwicklungsstand keine Malware die absolut nicht erkennbar ist. Allerdings gibt es noch kein Erkennungsprogramm, welches in der Lage ist, alle der verschiedenen Verschleierungsmechanismen zu erkennen. Um ein möglichst sicheres Ergebnis zu erlangen müssen viele der hier vorgestellten Verfahren auf die zu untersuchende Software angewandt werden. Dies ist durch den rasanten Wachstum des App-Stores leider nicht möglich. Es gibt einige gute Ansätze, aber es wird immer Malware geben, welche aus Grund fehlender Zeit nicht entdeckt wird.

Abbildungsverzeichnis

Literatur

[1] *Anteile der verschiedenen Android-Versionen an allen Geräten mit Android OS weltweit im Zeitraum 10. bis 16. April 2018*. https://de.statista.com/statistik/daten/studie/180113/umfrage/anteil-der-verschiedenen-android-versionen-auf-geraeten-mit-android-os/. Accessed: 2018-04-25.

[2] *Anzahl der Apps, die im Google Play Store heruntergeladen wurden*. https://de.statista.com/statistik/daten/studie/243412/umfrage/anzahl-von-downloads-im-google-play-store/. Accessed: 2018-04-25.

[3] *ART vs. Dalvik*. http://www.areamobile.de/specials/26470-art-vs-dalvik-im-test-die-neue-android-runtime-im-vergleich. Accessed: 2018-05-28.

[4] Steven Arzt u.a. „FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps“. In: *SIGPLAN Not.* 49.6 (Juni 2014), S. 259–269. ISSN: 0362-1340. DOI: 10.1145/2666356.2594299. URL: http://doi.acm.org/10.1145/2666356.2594299.

[5] Steffen Bartsch u.a. „Zertifizierte Datensicherheit für Android-Anwendungen auf Basis statischer Programmanalysen.“ In: *Sicherheit*. 2014, S. 283–291.

[6] Ralf Benzmüller. „Machine Learning und Virenschutz“. In: *Datenschutz und Datensicherheit - DuD* 42.4 (2018), S. 224–230. ISSN: 1862-2607. DOI: 10.1007/s11623-018-0912-6. URL: https://doi.org/10.1007/s11623-018-0912-6.

[7] *Überdeckungsmaße*. http://home.edvsz.fh-osnabrueck.de/skleuker/CSI/Methoden/kombiquUeberdeckung.html. Accessed: 2018-05-29.

[8] Simon Bierbaum. „Malicious Code: Code Obfuscation“. In: (). Accessed: 2018-05-31.

[9] *dynamische Analyse*. https://www.informatik.uni-bremen.de/st/lehre/re10/dynamische_analyse.pdf. Accessed: 2018-05-28.

[10] Header Entpacker-Stub. *Malware-Techniken und Malware-Analyse [MM-108]*. https://www1.informatik.uni-erlangen.de/filepool/projects/openc3s/mm-Malware.pdf. Accessed: 2018-05-31.

[11] Michael D Ernst. „Static and dynamic analysis: Synergy and duality“. In: *WODA 2003: ICSE Workshop on Dynamic Analysis*. 2003, S. 24–27.

[12] *FlowDroid – Taint Analysis*. https://blogs.uni-paderborn.de/sse/tools/flowdroid/. Accessed: 2018-05-31.

[13] Marko Gargenta. *Einführung in die Android-Entwicklung*. O'Reilly Germany, 2011. URL: https://books.google.de/books?hl=de&lr=&id=34S3Jt1ONTkC&oi=fnd&pg=PR5&dq=android+architektur&ots=Hm7wBINdJG&sig=620owrYAI_6HXB7XDZjvfRM59KE#v=onepage&q&f=false.

[14] *Google Play Apps Statistik*. https://de.statista.com/statistik/daten/studie/74368/umfrage/anzahl-der-verfuegbaren-apps-im-google-play-store/. Accessed: 2018-04-22.

[15] Marcel Lehner und Eckehard Hermann. „Auffinden von verschleierter Malware“. In: *Datenschutz und Datensicherheit - DuD* 30.12 (2006), S. 768–772. ISSN: 1862-2607. DOI: 10.1007/s11623-006-0237-8. URL: https://doi.org/10.1007/s11623-006-0237-8.

[16] *Marktanteil von Android*. https://www.netzwelt.de/news/164146-android-vs-ios-plattform-treue-android-91-prozent-deutlich-hoeher.html. Accessed: 2018-04-25.

[17] *Request App Permissions*. https://developer.android.com/training/permissions/requesting.html. Accessed: 2018-04-25.

[18] *RiskIQ report about Malicious Mobile Apps in Google Play*. https://www.riskiq.com/press-release/riskiq-reports-malicious-mobile-apps-google-play-have-spiked-nearly-400/. Accessed: 2018-04-22.

[19] Julian Scheid. „Sicherheit mobiler Geräte-Schutzmaßnahmen, Angriffsarten & Angriffserkennung auf Android“. In: *Ausgewählte Themen der IT-Sicherheit* (2012), S. 7.

[20] *Sicherheitskonzepte von Linux*. https://wiki.ubuntuusers.de/Archiv/Sicherheitskonzepte/. Accessed: 2018-04-25.

[21] Philipp Trinius. *Visualisierung von Malware-Verhalten*. https://www1.cs.fau.de/filepool/publications/trinius_myphd_abstract.pdf. Accessed: 2018-05-29. 2010.

[22] *What is a Behavior Blocker?* https://www.welivesecurity.com/2006/09/11/what-is-a-behavior-blocker/. Accessed: 2018-05-31.

[23] J. Yousefi, Y. Sedaghat und M. Rezaee. „Masking wrong-successor Control Flow Errors employing data redundancy“. In: *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*. 2015, S. 201–205. DOI: 10.1109/ICCKE.2015.7365827.