

# Kann Malware in Android-Apps automatisch gefunden werden?

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

**Zusammenfassung**—An dieser Stelle steht eine kurze Zusammenfassung des Inhaltes des Dokuments.

## Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen	1
2.1	Was ist Android? . . . . .	1
2.2	Android Update Problematik . . . . .	2
2.3	Application Sandbox . . . . .	2
2.4	Permission Model . . . . .	2
2.5	Sicherheit des Google Play Store . . . . .	2
2.6	Malware . . . . .	3
2.7	Dynamische Analyse . . . . .	3
2.8	Statische Analyse . . . . .	3
2.8.1	Kontrollflussanalyse . . . . .	4
2.8.2	Datenflussanalyse . . . . .	4
2.9	Gegenüberstellung von statischer und dynamischer Analyse . . . . .	4
3	Erkennungsmechanismen	4
3.1	signaturbasierte Malwareerkennung . . . . .	4
3.2	Virenschutz durch Machine Learning . . . . .	4
3.3	Integritätsprüfung . . . . .	5
3.4	Behavior Blocker . . . . .	5
3.4.1	Taint Analysis . . . . .	5
4	Bouncer	5
5	Fazit	5
	Abkürzungen	5
	Abbildungen	5
	Literatur	5

## 1. Einleitung

Smartphones bieten aufgrund ihrer vielen Schnittstellen 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 [14]. Da Malware-Autoren immer mehr auf Verschleierungsmechanismen

wie Komprimierung, Verschlüsselung oder sich selbst modifizierender Programmcode setzen, wird es für Antiviren-Herstellern zunehmend schwieriger diese als schädliche Software zu erkennen [13].

## 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 eine beschleunigte Innovation im Mobilbereich, um dem Benutzer ein reicheres, kostengünstigeres und besseres Mobilerlebnis zu bieten"[11]. Die Android Architektur besteht aus insgesamt fünf Schichten, dem Linux Kernel, Bibliotheken, Android-Laufzeitumgebung, Application-Frameworks und auf der obersten Schicht die Anwendungen.

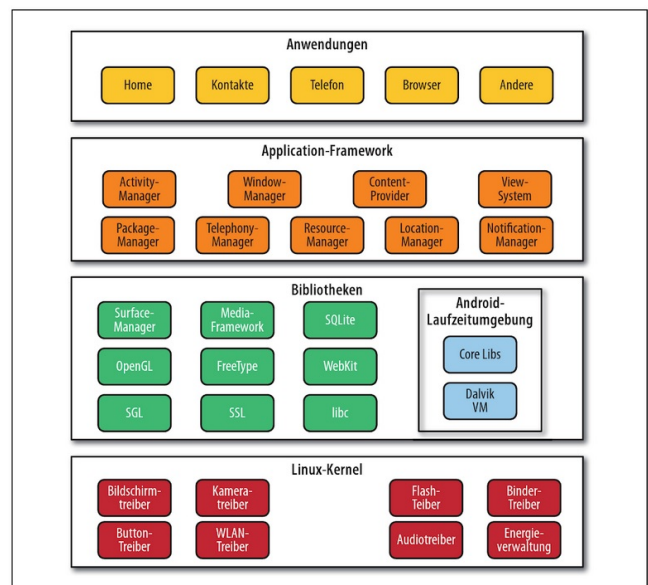


Abbildung 1. Android Architektur [11]

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. Zudem dient der Linux-Kernel 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 [18], greift Android auf viele Schutzmechanismen des Linux Kernels zurück oder benutzt diese als ihre Basis[17]. 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 Betriebssystems. 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 Schicht sind die Anwendungen. Diese sind teilweise schon vorinstalliert oder können heruntergeladen und installiert werden.

## 2.2. 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.

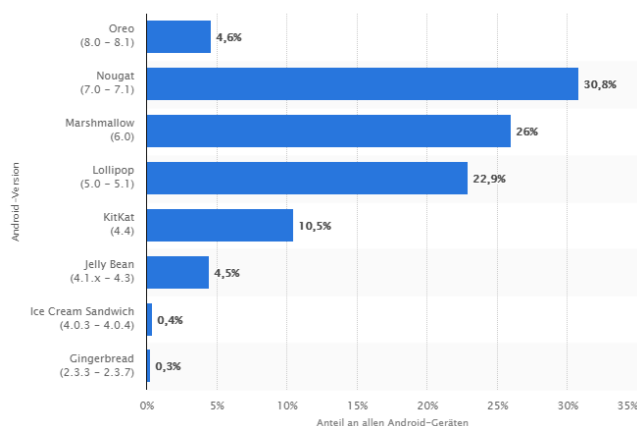


Abbildung 2. Verteilung der Android-Versionen im Jahr 2018 [1]

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. [17].

## 2.3. 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 2.2 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 [11].

## 2.4. 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. [15]. 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.

## 2.5. 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 [12] und jeden Monat kommen ca. 30.000 neue Applikationen hinzu [4]. 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 [16]. Da nicht alle Anwendungen von Mitarbeitern geprüft werden können, hat Google das Programm Bouncer ins Leben gerufen

(siehe Kapitel 4). 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.

## 2.6. Malware

Bei dem Begriff Malware handelt es sich um ein Kunstwort, welches sich aus malicious und software zusammensetzt und bezeichnet Programme, welche unerwünschte oder auch schädliche Funktionen ausführen. Malware kann man prinzipiell in drei unterschiedliche Hauptkategorien unterteilen, Viren, Würmer und Trojaner. Ein Virus ist eine Software, welche sich selbst vervielfältigen 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 Viren, der Hauptunterschied liegt darin, dass sie sich über das Intra- oder Internet vermehren können und oft keine Interaktion mit den betroffenen Benutzern benötigen. Die dritte und gefährlichste Kategorie 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 Malware-Authoren durch Verschleierrungsmechanismen wie Komprimierung, Verschlüsselung oder sich selbst modifizierender Programmcode verschleiert werden, was das Auffinden durch Virenprogramme um einiges erschwert. [13] 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 [7]. 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 2.8.1 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. [5] 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.

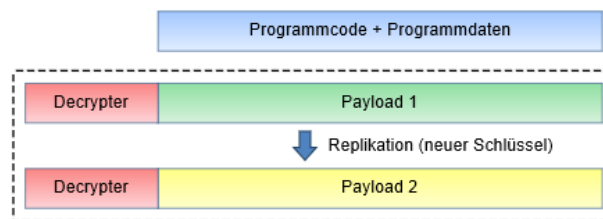


Abbildung 3. Polymorphie des Programmcodes [9]

**Metamorphie** hat die Bedeutung Verwandlung. Sollte nun nicht nur, wie in Abbildung 3 gezeigt ist, die Payload jedes mal geändert werden, sondern auch der Decrypter, spricht man von einem Metamorphosen Programm. Dies hat zur Folge, dass der gesamte Quellcode nach jeder Replikation verändert ist. [9]

Malware stellt wegen all dieser Verschleierrungsmechanismen 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 [19]. Bei der Analyse kann grundsätzlich in zwei Arten unterschieden werden: Statische und Dynamische Analyse (siehe Kapitel 2.8 und 2.7).

## 2.7. 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 [8]. 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 2.8 vorgestellt wird.

## 2.8. Statische Analyse

Bei der statischen Analyse wird versucht durch reverse engineering Code aus der Malware zu extrahieren um somit auf das Verhalten des Programmes schließen zu können. Dieser extrahierte Code kann anschließend eingehend untersucht werden. [19]. 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 Kontrollflussanalyse 2.8.1 oder Datenflussanalyse 2.8.2 durchgeführt werden kann.

**2.8.1. Kontrollflussanalyse.** Bei der Kontrollflussanalyse wird untersucht, welcher Block im zu untersuchenden 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. [21] Die Abbildung 4 zeigt ein Beispiel eines solchen Kontrollflussgraphen.

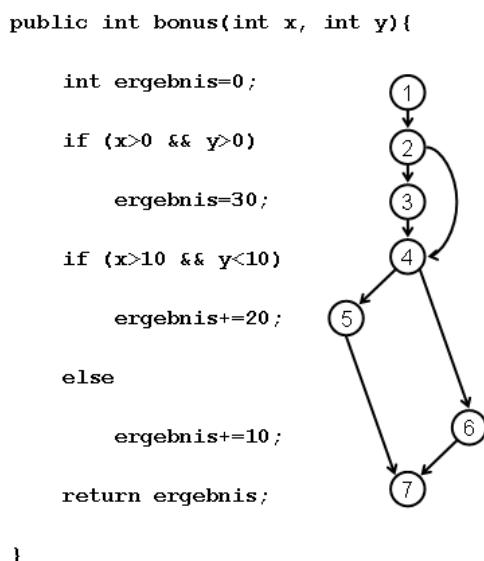


Abbildung 4. Kontrollflussgraph [6]

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.

**2.8.2. 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 2.8.1 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. [21]

## 2.9. 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. Beide Analysemethoden haben Vor- und Nachteile, ergänzen sich allerdings gegenseitig gut, daher sollten beide Methoden zusammen verwendet werden. [10]

## 3. Erkennungsmechanismen

### 3.1. signaturbasierte Malwareerkennung

Die am meisten Vertretene Methode der Malwareerkennung ist die auf Signatur basierte Erkennung von Malware. Diese Methode basiert auf der statischen Analyse 2.8 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. Der erste große Nachteil dabei ist, dass nur bereits bekannte Malwarearten gefunden werden können, zudem ist das Gerät in der gesamten Zeit in der die Signaturen verglichen werden gegen Angriffe ungeschützt. [13]

### 3.2. Virenschutz durch Machine Learning

Glaubt man den Firmen, welche Machine Learning für ihr Virenerkennung einsetzen, 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 Forschungsgebietes der Informatik. Allerdings ist Intelligenz nur mangelhaft definiert und somit lässt sich dieses Forschungsgebiet 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 Ansätze, die statische 2.8 und die dynamische Analyse 2.7. 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 Netzwerkcommunication 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 2.6 beschrieben wurden. Zudem können Neuronale Netze eine Aussage nur mit einer gewissen Wahrscheinlichkeit ermöglichen, daher kann keine 100 %ige Erkennung garantiert werden. [5]

### 3.3. 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. Wird 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. [13]

### 3.4. 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. Anschließend 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.[13] 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. [20]

#### 3.4.1. Taint Analysis.

## 4. Bouncer

## 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 raffinierte Verschleierungsmethoden 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 Verschleierungsmethoden 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.

## Abkürzungen

## Abbildungsverzeichnis

1	Android Architektur [11] . . . . .	1
2	Verteilung der Android-Versionen im Jahr 2018 [1] . . . . .	2
3	Polymorphie des Programmcodes [9] . . . . .	3
4	Kontrollflussgraph [6] . . . . .	4

## 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] Steffen Bartsch u.a. „Zertifizierte Datensicherheit für Android-Anwendungen auf Basis statischer Programmanalysen.“ In: *Sicherheit*. 2014, S. 283–291.
- [5] Ralf Benzmlüller. „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>.
- [6] *Überdeckungsmaße*. <http://home.edvsz.fh-osnabrueck.de/skleuker/CSI/Methoden/kombiquUeberdeckung.html>. Accessed: 2018-05-29.
- [7] Simon Bierbaum. „Malicious Code: Code Obfuscation“. In: (). Accessed: 2018-05-31.
- [8] *dynamische Analyse*. [https://www.informatik.uni-bremen.de/st/lehre/re10/dynamische\\_analyse.pdf](https://www.informatik.uni-bremen.de/st/lehre/re10/dynamische_analyse.pdf). Accessed: 2018-05-28.
- [9] 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.
- [10] Michael D Ernst. „Static and dynamic analysis: Synergy and duality“. In: *WODA 2003: ICSE Workshop on Dynamic Analysis*. 2003, S. 24–27.
- [11] 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](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).
- [12] *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.
- [13] Marcel Lehner und Eckehard Hermann. „Auffinden von verschleierte 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>.
- [14] *Marktanteil von Android*. <https://www.netzwelt.de/news/164146-android-vs-ios-plattform-treue-android-91-prozent-deutlich-hoer.html>. Accessed: 2018-04-25.
- [15] *Request App Permissions*. <https://developer.android.com/training/permissions/requesting.html>. Accessed: 2018-04-25.
- [16] *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.
- [17] Julian Scheid. „Sicherheit mobiler Geräte-Schutzmaßnahmen, Angriffsarten & Angriffserkennung auf Android“. In: *Ausgewählte Themen der IT-Sicherheit* (2012), S. 7.
- [18] *Sicherheitskonzepte von Linux*. <https://wiki.ubuntuusers.de/Archiv/Sicherheitskonzepte/>. Accessed: 2018-04-25.
- [19] Philipp Trinius. *Visualisierung von Malware-Verhalten*. [https://www1.cs.fau.de/filepool/publications/trinius\\_myphd\\_abstract.pdf](https://www1.cs.fau.de/filepool/publications/trinius_myphd_abstract.pdf). Accessed: 2018-05-29. 2010.
- [20] *What is a Behavior Blocker?* <https://www.welivesecurity.com/2006/09/11/what-is-a-behavior-blocker/>. Accessed: 2018-05-31.
- [21] 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.