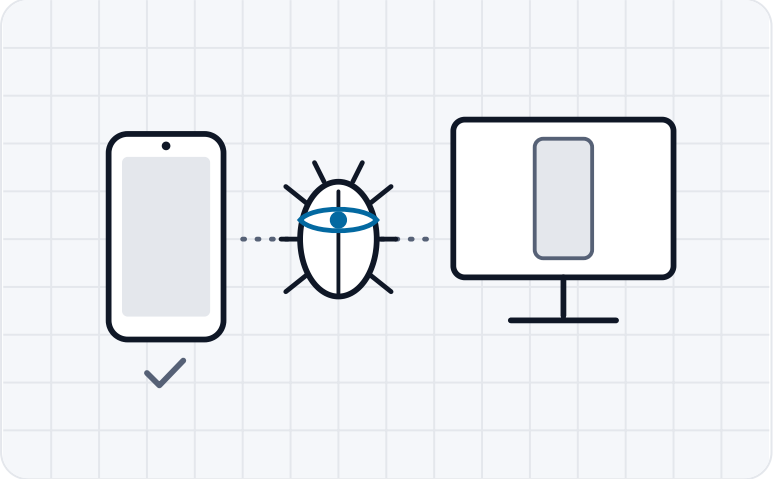
A-SIT Technology
E-Government Innovationszentrum

23. Oktober 2023 Analyse · Mobile Sicherheit · 14 Min. Lesezeit

Erkennung von Emulatoren in der Malware-Analyse unter Android

Gerald Palfinger



Autorinnen und Autoren	Gerald Palfinger
Stand	2023-10-06
Quelle	https://technology2.a-sit.at/veroeffentlichungen/erkennung-von-emulatoren-in-der-malware-analyse-unter-android/

Kernaussagen

- Emulator und Pixel 4 (beide Android 13) unterscheiden sich in 178 Methoden, 123 Feldern und 49 Content Providern der Programmierschnittstelle.
- Die Unterschiede betreffen Software, Hardware, vorkonfigurierte und nutzerbeeinflusste Einstellungen sowie andere Artefakte.
- Der Emulator meldet immer T-Mobile USA als Netzwerkbetreiber; die Google Mobile Services fehlen trotz Google-Play-Konfiguration.
- Der Emulator unterstützt keine Attestation, da ihm kryptografische Hardware fehlt.
- Schädliche Apps können diese Unterschiede ausnutzen, um Analyseumgebungen zu erkennen und ihr Verhalten dort zu verbergen.

Um zu verhindern, dass sich Schadprogramme verbreiten können, werden Applikationen vor der Veröffentlichung in App Stores untersucht. Dabei spielt die dynamische Analyse, bei der die Applikation ausgeführt wird, eine wichtige Rolle. Aus praktischen Gründen werden hierbei oft Android-Emulatoren anstatt echter Geräte eingesetzt. Kleine Unterschiede zwischen Emulatoren und echten Geräten ermöglichen es schädlichen Programmen jedoch potenziell, die Analyse zu erkennen und so ihr schädliches Verhalten während dieser zu verstecken. In diesem Bericht wird darauf eingegangen, welche Unterschiede in der Literatur gefunden wurden, um Emulatoren von echten Geräten zu unterscheiden. Weiters wird in einem Experiment untersucht, welche Unterschiede mittels der Programmierschnittstelle zwischen echten Geräten und dem Emulator gefunden werden können.

1. Einleitung

Durch die Vielzahl an gespeicherten Informationen, bereitgestellten Funktionen und sicherheitsrelevanten Anwendungen wie die Verwendung als zweitem Faktor sind Smartphones ein attraktives Ziel für Schadsoftware. Anders als Desktopanwendungen, welche oft manuell über den Browser aus dem Internet heruntergeladen werden, werden Smartphoneanwendungen in der Regel aus einer beziehungsweise wenigen zentralen Applikationsquellen bezogen. Dadurch kommt den Betreibern eine wichtige Rolle als „Gatekeeper“ zwischen den (potenziell schädlich agierenden) Entwicklern und den Endanwendern zu. Durch die Vielzahl an eingereichten Applikationen ist es jedoch schwierig, jede Applikation einzeln manuell zu untersuchen. Ebenso ist ein schädliches Verhalten oft nicht sofort ersichtlich und die Applikation liegt den Betreibern der Stores nicht im Quellcode vor. Deswegen werden Applikationen vor der Veröffentlichung durch automatisierte statische und dynamische Analyseanwendungen untersucht. Bei der dynamischen Untersuchung wird oft auf Emulatoren zurückgegriffen, da diese in der Handhabung gewisse Vorteile haben. So kann ein Emulator nach der Analyse schnell wieder auf den Ausgangszustand zurückgesetzt werden. Ebenso haben Emulatoren, welche auf normaler Serverhardware ausgeführt werden, eine höhere Skalierbarkeit als spezialisierte Gerätefarmen. Während Android-Emulatoren versuchen, Geräte zu emulieren, gelingt ihnen dies jedoch nicht immer bis ins Detail. So können bestimmte statische Informationen, wie Modellbezeichnungen oder Netzwerkdetails auf einen Emulator hinweisen. Darüber hinaus kann die Emulation bestimmter Hardware fehlen oder sich anders als ein echtes Gerät verhalten. Ebenso kann das Vorhandensein von bestimmten Artfakten oder das Fehlen von Benutzungsmerkmalen wie eine geringe Anzahl an Dateien, Applikationen oder Kontakten auf einen Emulator schließen lassen. In diesem Bericht wird gezeigt, welche Möglichkeiten in der Literatur gefunden wurden, um einen Emulator von einem echten Gerät zu unterscheiden. Weiters wird in einem Experiment untersucht, welche Unterschiede mittels der Programmierschnittstelle zwischen Android-Emulator und echtem Gerät gefunden werden können.

2. Verwandte Arbeiten

In [1] wurde versucht, Malwareanalyseumgebungen anhand verschiedener Eigenschaften zu erkennen. Solche Eigenschaften könnten von Malware genutzt werden, um diese Umgebungen zu erkennen. Die Forscher teilten die Erkennungsheuristiken in drei Kategorien ein. Dazu zählen statische Eigenschaften, Messwerte der Gerätesensoren und intrinsische Eigenschaften der verwendeten Emulationsumgebung. Als statische Erkennungsmerkmale wurden Informationen über das verbundene Netzwerk, Build-Informationen und Geräte-IDs erkannt. Viele der verwendeten Emulatoren konnten zwar Gerätesensoren wie den

Beschleunigungssensor emulieren, die emulierten Werte wiesen aber ein eindeutiges Muster auf, welches sich über einen kurzen Zeitraum mit nur geringer Abweichung wiederholte. Schlussendlich weisen auch Unterschiede in der Prozessorarchitektur und Optimierungen bei der Emulation dieser auf die Verwendung eines Emulators hin. So wird beispielsweise der virtuelle Befehlszähler nicht nach jeder Ausführung eines Befehls erhöht, wodurch die Performance des Emulators verbessert werden kann. Ebenso unterscheidet sich das Verhalten der Prozessoren bei der Ausführung von Code, der sich selbst verändert. Dies liegt an Unterschieden im Aufbau von Instruktions- und Datencaches in den Prozessorarchitekturen. Diese beiden Unterschiede ermöglichen es, zu erkennen, ob Code auf einem echten Gerät oder im Emulator ausgeführt wird. In der Analyse wurden verschiedene Malwareanalyseumgebungen auf ihre Erkennbarkeit anhand der aufgestellten Heuristiken überprüft. Nur eine der untersuchten Umgebungen konnte nicht anhand der festgelegten statischen Erkennungsmerkmalen erkannt werden. Die Messwerte der Gerätesensoren wurden bei vielen Umgebungen emuliert. Die Erkennung der intrinsischen Eigenschaften der verwendeten Emulationsumgebung benötigt nativen Code. Dieser wird nicht von jeder Analyseumgebung unterstützt. Bei allen untersuchten Umgebungen, die nativen Code unterstützen, erlauben diese aber eine eindeutige Identifizierung der Emulation.

Mit Morpheus [2] können automatisiert Heuristiken gefunden werden, die es erlauben, emulierte Umgebungen zu erkennen. Dazu wird das Framework sowohl auf Emulatoren als auch auf echten Geräten ausgeführt. In einem ersten Schritt werden mit Hilfe eines erstellten Programms Daten von echten Geräten und Emulatoren gesammelt. Dabei wurden die Daten aus drei verschiedenen Quellen gesammelt: Informationen über den Aufbau und den Inhalt des Dateisystems, Informationen aus der Programmierschnittstelle und Informationen aus den Systemeigenschaften. Diese wurden dann zwischen den Geräten und den Emulatoren verglichen. Dabei wurden über 10.000 unterschiedliche Informationsquellen gefunden, die potenziell als Heuristik verwendet werden können. Ein Großteil der gefundenen Quellen stammt aus dem Dateisystem, mit über 9.800 potenziell verwendbaren Heuristiken. Weiters wurden jeweils etwas weniger als 400 potenzielle Heuristiken in der Programmierschnittstelle und den Systemeigenschaften entdeckt. Die gefundenen Heuristiken wurden evaluiert, indem sie zur Unterscheidung von Malwareanalysesystemen und einem Set an echten Geräten verwendet wurden. Dabei wurden aus jeder der Kategorien nur die 10 wichtigsten Heuristiken ausgewählt. Dies deshalb, da auf vielen der nur online verfügbaren Emulatoren die Ausführungszeit stark begrenzt war und dadurch nicht alle Heuristiken abgerufen werden konnten. Bei der Analyse wurde festgestellt, dass die Heuristiken aus dem Dateisystem am besten abschnitten und eine Genauigkeit von 97% erreichten, gefolgt von den Systemeigenschaften mit 90%. Die Informationen aus der Programmierschnittstelle schnitten hierbei jedoch um einiges schlechter ab, mit einer Genauigkeit von nur 63%.

In [3] wird die Effizienz der Malwareanalyse auf Emulatoren und auf echten Geräten verglichen. Dadurch kann erkannt werden, ob sich Applikationen auf echten Geräten anders verhalten als im Emulator. Ebenso konnten mehr Applikationen erfolgreich untersucht werden, da echte Geräte mehr Funktionen haben als der Emulator. Zum Vergleich wurden verschiedene Machine-Learning-Algorithmen verwendet. In der Analyse wurde festgestellt, dass beinahe 99% der 1222 untersuchten Malwareproben auf echten Geräten ausgeführt werden konnte, während nur etwa 77% auf dem Emulator ausgeführt werden konnte. Weiters wurde festgestellt, dass manche Methoden der Programmierschnittstelle nur beziehungsweise von mehr Applikationen auf echten Geräten ausgeführt werden. Dies führte dann auch dazu, dass auf den echten Geräten mehr Malware korrekt erkannt werden konnte.

[4] zeigt Erkennungsmethoden von Emulatoren anhand von Diskrepanzen im Verhalten auf Befehlsebene zwischen softwarebasierten Emulatoren und realen ARM-CPUs. Dazu wurden in großem Umfang Spuren der Anwendungsausführung gesammelt, insbesondere die Änderungen an Registern und dem Speicher. Dazu wurde ein akkurates Software-Modell einer ARM-CPU erstellt. In der Arbeit wurden verschiedene Test-Applikationen sowohl in diesem Modell als auch im Emulator ausgeführt. Das erstellte Framework identifiziert automatisch Anweisungen, die ein abweichendes Verhalten zwischen emulierten CPUs und dem erstellten Modell verursachen. Dabei konnten die erkannten Unterschiede in drei Hauptklassen eingeteilt werden. Manche der gefundenen Ursachen konnte ohne Leistungseinbußen durch Änderungen am Emulator behoben werden.

In [5] wurde untersucht, wie sich die bei der dynamischen Analyse gesammelten Daten zwischen echten Geräten und Emulatoren unterscheiden. Dazu wurden jeweils 110 harmlose und 110 bösartige Applikationen untersucht. Diese wurden sowohl auf einem echten Gerät als auch auf einem emulierten Gerät untersucht. Bei der Untersuchung wurden alle Systemaufrufe aufgezeichnet, welche von den untersuchten Applikationen ausgelöst wurden. Bei der Auswertung der Daten wurde festgestellt, dass es hierbei Unterschiede zwischen den echten Geräten und Emulatoren gibt. So wurden am Emulator mehr Systemaufrufe ausgeführt, diese korrelierten aber stärker als auf echten Geräten.

In [6] wurden aktuelle Malwareanalyseumgebungen erneut auf ihre Erkennbarkeit gegenüber echten Geräten untersucht. Dabei wurde eine Applikation erstellt, welche sowohl statische Informationen wie Gerätedetails, als auch sogenannte Verwendungsartefakte der untersuchten Android-Umgebung abruft. Darunter fallen in der Studie Informationen wie die Anzahl der Kontakte und Kalendereereignisse, Anzahl der gespeicherten Mediendateien und Erinnerungen sowie gewisse Geräteeinstellungen wie die Anzahl der gespeicherten Drahtlosnetzwerke. Um diese Informationen abzurufen, wird jedoch oft eine Berechtigung durch den Nutzer beziehungsweise die Nutzerin benötigt. Die untersuchten Informationsquellen wurden durch die Autoren manuell aus der Dokumentation der Programmierschnittstelle von Android ausgewählt. In den durchgeführten Experimenten wurden 12 verschiedene Malwareuntersuchungsumgebungen sowie die Aufnahmeprozesse von mehreren Applikationsmärkten untersucht. Mit Hilfe von maschinellem Lernen wurde anhand der gesammelten Informationen versucht, die untersuchten Analyseumgebungen von echten Geräten zu unterscheiden. Dazu wurden vorab Informationen von mehr als 1.200 echten und aktiv verwendeten Android-Geräten gesammelt. In der Evaluierung wurde festgestellt, dass das erstellte Modell eine Genauigkeit von über 98% bei der Unterscheidung zwischen echtem Gerät und Malwareanalyseumgebung hat. Ebenso wurde gezeigt, dass viele Analyseumgebungen Gerätemodelle unzureichend emulieren, indem sich zum Beispiel Herstelleranpassungen wie vorinstallierte Applikationen oder die Hardware von einem echten Gerät unterscheidet.

CamoDroid [7] ist eine Malwareanalyseumgebung, die mit dem Ziel erstellt wurde, echte Geräte möglichst realitätsgetreu nachzubilden, um so die Erkennung zu verhindern. Dazu werden Sensoren, statische Informationen, Netzwerkdetails und Nutzereingaben emuliert. Ebenso werden Zugriffe auf potenziell gefährliche Methoden der Programmierschnittstelle überwacht und realistische Werte zurückgegeben. In den Experimenten wurde CamoDroid sowohl mit Morpheus [2] als auch gegen echte Malware getestet. Dabei konnten 219 der insgesamt 227 getesteten Malwareproben erfolgreich ausgeführt werden, ohne dass diese ihr bösartiges Verhalten durch die Erkennung der Malwareanalyseumgebung verschleierten.

3. Methodik

Um die Unterschiede zwischen echten Geräten und Emulatoren zu erkennen, werden die Daten aus der Programmierschnittstelle systematisch gesammelt. Dazu werden die Methoden der Programmierschnittstelle aufgerufen und deren Rückgabewerte gesammelt. Ebenso werden die Felder von Klassen abgerufen. Schlussendlich werden auch die Android-spezifischen Content Provider ausgelesen. Die gesammelten Daten werden in textueller Repräsentation gespeichert. Um temporäre Informationen zu filtern, wird die Sammlung der Informationen auf jedem Gerät beziehungsweise Emulator zweimal ausgeführt. Dazwischen wird das Gerät beziehungsweise der Emulator neu gestartet. Nach der Sammlung der Informationen werden diese zwischen den untersuchten Geräten und Emulatoren verglichen. Informationen aus derselben Quelle, die sich zwischen Geräten und Emulatoren unterscheiden, werden als potenzielles Unterscheidungsmerkmal markiert. Diese wurden dann manuell inspiziert und zur Übersicht in Kategorien eingeteilt. Die Ergebnisse sind im folgenden Kapitel dargestellt. Während bei den verwandten Arbeiten oft Informationsquellen gewählt wurden, welche eine Berechtigung benötigen, konzentriert sich diese Studie auf Quellen, welche keine gesonderte Berechtigung durch den Nutzer beziehungsweise die Nutzerin benötigen. Dadurch kann die Erkennung eines Emulators möglichst unauffällig erfolgen.

4. Ergebnisse

In den Experimenten wurde der durch Google bereitgestellte Android Emulator mit einem physischen Pixel 4-Gerät verglichen. Auf beiden wurde Android 13 installiert. Bei der Konfiguration des Emulators wurde als Emulationsziel ein Pixel 4 mit Google Play ausgewählt. Die gefundenen Informationsquellen sind in Tabelle 1 ersichtlich. Darunter sind sowohl Quellen, welche bereits in der Literatur gefunden wurde, als auch neue Informationsquellen. Zu den bereits bekannten Quellen zählen beispielsweise die Build-Informationen, CPU-Architektur oder Details über das Netzwerk. Prinzipiell lassen sich die gefundenen Informationsquellen in fehlende beziehungsweise anders konfigurierte Softwarekomponenten, Hardwarefunktionen sowie vorkonfigurierte Einstellungen als auch durch den Benutzer oder die Benutzerin beeinflusste Einstellungen sowie anderer Artefakte aufteilen. Im Folgenden wird auf die gefundenen Unterschiede genauer eingegangen.

Über verschiedene Methoden und Content Provider konnte festgestellt werden, dass gewisse Softwarekomponenten von Google fehlen, obwohl der Emulator mit Google Play konfiguriert wurde. Dazu zählen unter anderem die Abwesenheit der Google Mobile Services (com.google.android.gms), die Feedback-Komponente als auch verschiedenen Benachrichtigungskomponenten von Google. Ebenso ist die Anzahl der installierten Applikationen als auch die Anzahl der vorinstallierten Codecs auf dem Emulator geringer. Auch vorinstallierte Module, Content Provider und Applikationen sowie deren Konten unterscheiden sich zwischen Emulator und Gerät. Zwischen Emulator und echtem Gerät unterscheiden sich ebenso die verfügbaren Berechtigungsgruppen. Darüber hinaus unterscheiden sich gewisse Konstanten sowie bestimmte Betriebssystemeigenheiten wie Parallelitätseinstellungen beim Multithreading oder die Größe des Zwischenspeichers von Sockets. Ebenso fehlen am Emulator einige geteilte Bibliotheken, besonders jene, welche vom SoC-Hersteller Qualcomm kommen und nur am echten Gerät zu finden sind. Das echte Gerät unterstützt zwei zusätzliche Bedienungshilfen im Vergleich zum Emulator. Gewisse Funktionen wie SIP oder VoIP sind ebenso nur am echten Gerät verfügbar. Auf der Softwareseite unterscheiden sich darüber hinaus gewisse Labels, Systemeigenschaften, unterstützte Sprachen sowie die Zeitzonenversion.

Ähnlich wie bei den Softwarekomponenten gibt es auch auf Hardwareseite eine Vielzahl an Unterschieden zwischen dem Emulator und dem echten Gerät. Dazu zählt die Anzahl an Kameras, welche am echten Gerät höher ist als am Emulator. Ebenso unterstützt der Emulator nur ein Modem, während sich das echte Gerät mit zwei Mobilfunkzugängen gleichzeitig nutzen lässt. Das echte Gerät bietet mehr Audiofunktionen, wie beispielsweise Geräuschunterdrückung. Darüber hinaus unterscheidet sich die angegebene Audiolatenz, welche am Emulator geringer ist. Im Vergleich zum Emulator hat das Gerät mehr Streams in der Audiodatenbank. Weiters unterscheiden sich bestimmte Konstanten bei der Unterstützung von Bluetooth-Geräten. Ebenfalls unterscheiden sich die unterstützten Funktionen der GPS-Hardware als auch das Herstellungsjahr der Hardware. Auch die Größe des Hauptspeichers ist bei der Verwendung des Pixel 4-Profils des Emulators kleiner als am echten Gerät. Im Bezug auf den Bildschirm konnte eine geringere Pixeldichte am Emulator festgestellt werden, welcher mit dem verwendeten Computermonitor übereinstimmt. Aufgrund fehlender kryptographischer Hardware bietet der Emulator keine Attestation an. Auch die verfügbaren kryptographischen Algorithmen unterscheiden sich zwischen den beiden. Ebenso unterscheiden sich MMS-User Agent als auch MMS-Profil. Dieses nimmt am Emulator noch Bezug auf Nexus-Geräte. Bei Nutzung der Standardeinstellungen des Emulators unterscheiden sich auch die Eigenschaften des persistenten Speichers, wie die Größe des internen Speichers oder das Vorhandensein einer Speicherkarte auf dem Emulator. Im Bezug auf die Hardware unterscheidet sich auch die Anzahl der Eingabegeräte, das SoC-Modell sowie Funktionen der WiFi-Hardware.

Im Vergleich zum Gerät sind am Emulator andere Benachrichtigungstöne vorkonfiguriert. Ebenso unterscheidet sich der voreingestellte Gerätename. Zwischen dem echten Gerät und dem Emulator unterscheiden sich auch Einstellungen zur Gerätesicherheit, im Besonderen die Verwendung eines PINs. Dieser wird auf dem echten Gerät in der Regel bereits bei der Einrichtung eingestellt, während dies beim Emulator nicht vorgeschlagen wird. Ebenso konnten über einen Content Provider zahlreiche andere benutzeranpassbare Einstellungen gefunden werden. Da diese in der Regel am Emulator nicht geändert werden, kann durch eine Änderung einer Vielzahl dieser Einstellungen auf ein echtes Gerät geschlossen werden. Zwischen Emulator und Gerät unterscheidet sich in der Regel auch der Netzwerkbetreiber. So wird am Emulator immer T-Mobile USA gemeldet.

Tabelle 1 Gefundene Unterschiede zwischen Emulator und echtem Gerät

Art	# Methoden	# Felder	# Content Provider
(Fehlende) Google-Komponenten	2		1
Andere Konstanten	7	2	3
android.os.Build Informationen		18	
Anzahl installierte Apps	8		
Anzahl installierte Codecs	1		
Anzahl Kameras	2		
Anzahl Modems	2		
Audio-Funktionen	9		
Audio-Latenz	1		
Audio-Streams	1		
Benachrichtigungstöne	19		6
Berechtigungsgruppen	1		
Betriebssystemeinstellungen	2		
Bluetooth-Funktionen	5		
CPU-Architektur		5	
Eingabegeräte	3		
Feedback-Komponente	4		
Funktionen zur Bedienungshilfe	2		
Gerätename			2
Gerätesicherheit (PIN eingestellt)	3		
Geteilte Bibliotheken	2		
GPS-Hardware	4		
Größe des Hauptspeichers	2		
Insets Typ	2	1	
Java System Eigenschaften	1		
MMS-Profil & Useragent	2		
Mobilfunkanbieter	8		

Art	# Methoden	# Felder	# Content Provider
Netzwerkdetails	7		
Persistenter Speicher (Einstellungen, Volumen, Größe)	27		
Pixeldichte	1		
Ressourcen-Labels	13	96	
SIP-Unterstützung	1		
SoC-Modell		1	
Tethering	1		
Unterstützte kryptographische Algorithmen	3		
Unterstützte Sprachen	1		
Unterstützung von Attestation	3		
Verfügbare Kontentypen	1		
Version des Modems	2		
VoIP-Support	1		
Vorinstallierte Apps	6		
Vorinstallierte Content Provider	1		
Vorinstallierte Module	1		
Weitere benutzeranpassbare Einstellungen			36
Weitere Funktionen	5		
WiFi Funktionen	10		1
Zeitzonenversion	1		
Gesamt	178	123	49

5. Fazit

In diesem Bericht wurde gezeigt, welche Arten von Unterschieden zwischen echten Geräten und Android-Emulatoren bestehen, die von schädlichen Anwendungen ausgenutzt werden können, um ihr schädliches Verhalten während Untersuchungen zu unterbinden und so unentdeckt zu bleiben. Darunter fallen statische Informationen wie Modellnummer, Geräteinformationen oder Netzwerkdetails. Ebenso kann bestimmte Hardware, welche auf Smartphones Standard ist, fehlen oder sich anders verhalten. Vor allem Sensoren liefern oft statische oder unglaubhafte Werte. Darüber hinaus kann es Artefakte geben, welche nur während der Emulation vorhanden sind, wie bestimmte Treiber oder Applikationen. Durch die Emulation der

Prozessorarchitektur kann es wahrnehmbare Unterschiede bei der Ausführung gewisser Befehle geben. Darüber hinaus fehlen oft Nutzungsartefakte. Dies macht sich bemerkbar durch eine geringe Anzahl an gespeicherten Dateien, Kontakten, Kalendereinträgen oder eine geringe Anzahl an installierten Applikationen.

Referenzen

[1] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis und S. Ioannidis, „Rage against the virtual machine: hindering dynamic analysis of Android malware," EuroSec '14: Proceedings of the Seventh European Workshop on System Security, p. 1–6, 2014.

[2] Y. Jing, Z. Zhao, G.-J. Ahn und H. Hu, „Morpheus: automatically generating heuristics to detect Android emulators," Proceedings of the 30th Annual Computer Security Applications Conference, p. 216–225, 2014.

[3] M. K. Alzaylaee, Y. Y. Suleiman und S. Sezer, „EMULATOR vs REAL PHONE: Android Malware Detection Using Machine Learning," IWSPA '17: Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics, p. 65–72, 2017.

[4] O. Sahin, A. K. Coskun und M. Egele, „Proteus: Detecting Android Emulators from Instruction-Level Profiles," International Symposium on Research in Attacks, Intrusions, and Defenses, p. 3–24, 2018.

[5] A. Guerra-Manzanares, H. Bahsi und S. Nõmm, „Differences in Android Behavior Between Real Device and Emulator: A Malware Detection Perspective," Sixth International Conference on Internet of Things: Systems, Management and Security, 2019.

[6] B. Kondracki, B. A. Azad, N. Miramirkhani und N. Nikiforakis, „The Droid is in the Details: Environment-aware Evasion of Android Sandboxes," Annual Network and Distributed System Security Symposium, 2022.

[7] F. Faghihi, M. Zulkernine und S. Ding, „CamoDroid: An Android application analysis environment resilient against sandbox evasion," Journal of Systems Architecture, Bd. 125, 2022.

[8] J. Lin, C. Liu und B. Fang, „Out-of-Domain Characteristic Based Hierarchical Emulator Detection for Mobile," 2nd International Conference on Information Technologies and Electrical Engineering, p. 1–5, 2019.

[9] J.-d. Lim, I.-k. Kim, N. Kim, B. Kang und S.-j. Cho, „A Study on Android Emulator Detection Using Build Properties," The 8th International Conference on Next Generation Computing, pp. 321-324, 2022.