The Rise of Obfuscated Android Malware and Impacts on Detection Methods

Wael F. Elsersy[1], and Ali Feizollah[1], Nor Badrul Anuar[1]

[1] Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia

Corresponding Author:
Nor Badrul Anuar[1]
 Jln Profesor Diraja Ungku Aziz, Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur , 50603, Country
Email address: badrul@um.edu.my
Wael F. Elsersy[1]
 Jln Profesor Diraja Ungku Aziz, Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur , 50603, Country
Email address: wfarouk@siswa.um.edu.my

18 **Abstract**
19 The various application markets are facing an exponential growth of Android malware. Every
20 day, thousands of new Android malware applications emerge. At the same time, Android
21 malware hackers adopt reverse engineering and repackage benign applications with their
22 malicious code. Therefore, Android applications developers tend to use state-of-the-art
23 obfuscation techniques to mitigate the risk of application plagiarism. The malware authors adopt
24 the obfuscation and transformation techniques to defeat the anti-malware detections, which this
25 paper refers to as evasions. Malware authors use obfuscation techniques to generate new
26 malware variants from the same malicious code. The concern of encountering difficulties in
27 malware reverse engineering motivates researchers to secure the source code of benign Android
28 applications using evasion techniques. This study reviews the state-of-the-art evasion tools and
29 techniques. The study criticizes the existing research gap of detection in the latest Android
30 malware detection frameworks and challenges the classification performance against various
31 evasion techniques. The study concludes the research gaps in evaluating the current Android
32 malware detection framework robustness against state-of-the-art evasion techniques. The study
33 concludes the recent Android malware detection-related issues and lessons learned which require
34 researchers' attention in the future.

36 **Introduction**
37 Since, the advent of Android systems, smartphone devices are seen everywhere with a market
38 share of 87% (Chau & Reith 2019). Hence Android devices have become the most popular
39 devices for hackers and malware authors to target. With many open-source libraries in Android,
40 Android application development tools enable young developers to develop Android malware
41 applications. Hence, the number of Android malware increases exponentially. Therefore, in the
42 Google Android market, Android applications exponentially grow from 2.8 million as of
43 September 2018 (Statista 2016; Statista 2021) to almost double to reach 3.4 million Apps as of
44 the first quarter of 2021 (Statista 2021). Nevertheless, Android malware authors attract end-users
45 using cracked games, free applications, and video downloader applications. They mainly aim to
46 spy on private data (e.g., contact lists, photos, videos, documents, and account details) or control
47 devices by remote servers as botnets (Karim et al. 2015). Android applications use Java as a
48 developing language because Java provides a very flexible code, dynamic code loading (Liang &
49 Bracha 1998), and many other features to make Android application development more
50 accessible and efficient. Likewise, Java uses obfuscation tools (Aonzo et al. 2020; GuardSquare
51 2014) to protect commercial software companies from software plagiarism issues; professional
52 developers protect their source codes from being stolen using advanced evasion techniques
53 (Aonzo et al. 2020) as protection mechanisms. However, malware authors use the above-
54 mentioned advanced Java features and evasion tools to reproduce more sophisticated Android
55 malware, evading professional anti-malware (Preda & Maggi 2016). Google introduced Google
56 Bouncer (Rahman et al. 2016); however, Android malware successfully defeats Google Bouncer
57 using different evasion techniques (Maiorca et al. 2015). Furthermore, Google Play Protect (Xu

58 et al. 2016) service is the default device protection tool available on Google Android from
59 Version 6.0 onwards; however, the previous versions are deprecated.
60 The rationale behind this study is the ability of evasion techniques to hinder the analysis process
61 and thus the detection of Android malware. In 2021, PetaDoid (Karbab & Debbabi 2021)
62 proposed Android malware detection using Deep learning techniques. PetaDroid builds static
63 analysis Android malware detection framework using a 10 million Android Apps dataset.
64 PetaDriod addressed obfuscations in his study and concluded in his experimental results that his
65 trained machine learning model that reaches 99.2% using static analysis would not detect
66 complex obfuscated malware applications. The complex obfuscation techniques defeat Android
67 malware detection PetaDroid model, which results into false detection. Though PetaDroid
68 focused on trivial and some non-trivial obfuscation techniques. PetaDroid admitted that further
69 deep analysis is required to address the sophisticated obfuscation techniques. The study focused
70 on several evasion techniques, such as package transformation, string encryption, bytecode
71 encryption, code obfuscation, injecting new codes via dynamic code loading, junk/dead code
72 injection, emulation detection running sandboxing, and user interaction emulation detection.
73 Android malware modifies the package, developer signature, and other information using the
74 repacking evasion technique.
75 Moreover, the availability of various evasion techniques to the malware attackers increases the
76 fear of developing very advanced obfuscation techniques, as such newly developed malware
77 applications adopt advanced obfuscation techniques. It creates a challenge between preventing
78 source code piracy and malicious attacks (Gurulian et al. 2016; Zhang et al. 2014) and struggling
79 to decompile the malware application packages for further analysis (Gonzalez et al. 2015).
80 Android malware detection frameworks (Arp et al. 2015; Elish et al. 2015; Poeplau et al. 2014)
81 (Chen et al. 2015) suffer from False Negative (FN) detection, which means the Android malware
82 detection frameworks fail to detect some malware applications. The main reason behind FN is
83 the malware evasion techniques that malware applications adopt to hinder detection. For
84 instance, Drebin (Arp et al. 2015) achieved 94% detection accuracy because it fails to detect
85 malware with dynamic code loading transformation, one of the advanced evasion techniques.
86 Likewise, Elish (Elish et al. 2015) used trigger-based dependence for privileged API calls, but it
87 is unable to detect malware families with code obfuscation and reflection transformation.
88 Poeplau (Poeplau et al. 2014) used the call graph methodology to detect malicious code loading,
89 and the native code dynamically loads the code.
90 Similarly, Chen (Chen et al. 2015) identifies a repackaged application in 10 seconds using code
91 graph similarity but is incapable of tracking junk code insertion transformation. (You & Yim
92 2010) reviewed the obfuscation technique, metamorphic and polymorphic malware types. They
93 discussed the metamorphic and polymorphic evasion techniques; however, they neglected
94 transformation and anti-emulation evasions. Furthermore, they merely reviewed evasion methods
95 and failed to evaluate current evasion detection systems to evaluate whether they can detect
96 evasive malware. Sharma and Sahay (Sharma & Sahay 2014) reviewed polymorphic and
97 metamorphic malware and discussed their characteristics. They failed to mention evasion

98   detection methods and evaluate the currently proposed methods. Sufatrio et al. (Tan et al. 2015)
99   surveyed Android malware detection methods and briefly assessed a handful of related works in
100 terms of evasion detection.
101
102 This study is intended for Android malware detection research highlighting the research gaps in
103 malware detection caused by different evasion techniques. This study highlights the obfuscation
104 and transformation techniques that need more attention from the research authors in future. It
105 also provides guidelines and lesson learned to face this challenge. Due to the above facts, the
106 authors take the challenge to introduce the following foremost contributions.
107   -   We present evasion taxonomy, particularly in the Android platform. Our goal is to
108   systematize the Android malware evasion techniques using a taxonomy methodology,
109   which clearly shows various evasion techniques and how they affect malware analysis
110   and detection accuracy.
111   -   We scrutinise Android malware detection academic and commercial frameworks while a
112   large portion of the past work concentrated on commercial Anti-malware solutions. This
113   study examines different evasion techniques that hinder detecting malicious parts of
114   applications and affect detection accuracy by reviewing state-of-the-art Android malware
115   studies and issues limiting the detection of evasion techniques. It is worth noting that this
116   work differs from related works that examine detection methods, as we go through
117   evasion techniques that let malware eludes detection methods. Given the vast number in
118   this study field, our investigation focuses on studies written between 2011 and early 2021
119   and innovative contributions that appeared in high-ranked journals or conferences such as
120   IEEE, ACM, and Springer, hence the identified related papers are 511research papers.
121   -   We highlight the existing problems and gaps in Android malware evasion detection by
122   examining the previous frameworks and identifying the Android malware evasion
123   detection research gap.
124   -   We introduce a decent number of recommendations and lessons learned to consider in
125   future work around research. We also aim to highlight the contribution of each study,
126   challenges, countermeasures, and open issues for future research.
127
128 Table 1 presents the differences between this study and the recent evasions detection reviews.
129 Vikas (Sihag et al. 2021a) evaluated the hardening code obfuscation tools against the reverse
130 engineering process; however, it focused on development advantage more than malware
131 detection perspectives. FeCO (Jusoh et al. 2021) focused on Android application static analysis
132 and Android malware detection using machine learning and deep learning methods. It
133 highlighted the type of code obfuscations techniques and previous research obfuscation
134 solutions. AAMO (Preda & Maggi 2016) and Droidchameleon (Rastogi et al. 2013) study the
135 effectiveness of evading commercial anti-malware applications by using their evaluation tools;
136 Droidchameleon (Rastogi et al. 2013) examines trivial transformation, which easily evades the
137 detection of Android malware using the most popular anti-malware commercial packages.

138   However, Droidchameleon (Rastogi et al. 2013) misses studying the effect of the evasion
139   techniques on current detection accuracy. Likewise, Rastogi (Rastogi et al. 2014) continued his
140   study of Droidchameleon (Rastogi et al. 2013) and added more composite transformation attacks
141   that consist of more than evasion attacks and investigated evasion chains' capability for hindering
142   malware detection. Hoffmann (Hoffmann et al. 2016) develops a tool to thwart malware
143   detection and evaluates the accuracy of a few typical static and dynamic malware analysis
144   frameworks. Hoffmann (Hoffmann et al. 2016) concludes that code obfuscation evasion evades
145   Android malware detection frameworks. Nevertheless, Hoffmann excludes some evasion
146   techniques from the evaluation of malware detection frameworks.
147   The rest of the paper is organized as follows: the survey methodology and background section
148   provide essential background information for this study; we explore the Android operating
149   environment and its weaknesses. Evasion techniques section presents the evasion techniques
150   taxonomy with regards to different categories of evasions. Android evasion detection
151   frameworks section investigates the current state-of-the-art evasion detection frameworks and
152   evasion test benches tools. We discuss the lessons learned and future directions in discussion and
153   lessons learned sections. Finally the last section represents the conclusion of this study.
154
155   Table 1
156   Comparison of the recent reviews
157

Survey methodology

159   **Methodology**. The methodology of retrieving Android malware obfuscation detection related
160   articles is presented in this section. This study adopted Web-of-Science search engine to carry
161   over the literature review using search terms with inclusions and exclusion criteria. The review
162   process consists of four phases; first phase is identification, second phase is screening, third
163   phase is eligibility, and fourth phase is analysis phase.
164

165   **Identification**. The adopted Web-of-Science search engine covers hundred years of citation data
166   containing many journals related to computer security, software development, and network
167   security. Clarivate Analytics established this citation database with ranking citations measure
168   (citation per paper). Since this study focused on Android malware obfuscation, we had selected
169   'Android malware, 'malware obfuscation', and 'malware evasion' as our search terms. The
170   search results in 511 research from journals and conferences' proceeding database. The search
171   results mainly records are from IEEE, journals and conferences distributions as per Table 2.
172   The list of collected articles represent the Android malware obfuscation and detection
173   frameworks. It included the three types of the malware analysis techniques static, dynamic and
174   hybrid techniques in the last decade from 2011 to early 2021. Hence, we collected Android
175   malware frameworks for the last decade and innovative contributions that appeared in high-
176   ranked journals or conferences such as IEEE, ACM, and Springer.
177   Table 2

178   The List of Identified Journals and Conferences

179

180   **Screening**. Since, this paper explored the last ten years' research to evaluate the Android

181   detection frameworks against evasion techniques, we focused on experimental malware detection

182   articles using static, dynamic and hybrid analysis techniques, excluding the unrelated articles.

183   We excluded articles that are not Android specific malware detection such as IOS and Windows

184   based operating system. In addition, we excluded all other languages and include only English

185   language research to avoid translation overhead in future.

186

187   **Eligibility**.

188   As shown in Figure 1, the review process presented four phases flow diagram, the identification

189   collect the articles from web of science (WOS) database using above mentioned search terms,

190   next, screening identified the criteria of article inclusion and exclusion. After removing the

191   duplicates and excluded the non-related articles, we categorize Android malware detection by the

192   analysis methodology static, dynamic, and hybrid features. This paper decides to put metadata

193   analysis out of this research scope. The screening phase resulted into 342 article from 511

194   collected in identification phase. However, we have examined 74 static analysis based

195   frameworks. The number of dynamic based analysis frameworks are 35, the number of hybrid

196   analysis frameworks is 26. Hence the total number of examined papers are 135 research paper

197   that this study selected from top rank journals and conferences.

198

199   **Data Analysis**. We scrutinise Android malware detection academic and commercial frameworks

200   while a large portion of the past work concentrated on commercial Anti-malware solutions. This

201   study examines different evasion techniques that hinder detecting malicious parts of applications

202   and affect detection accuracy by reviewing state-of-the-art Android malware studies and issues

203   limiting the detection of evasion techniques. It is worth noting that this work differs from related

204   works that examine detection methods, as we go through evasion techniques that let malware

205   evades detection methods.

206

207   Figure 1. The Review Process Flow Diagram

208

209   **Android Applications and Weaknesses.** In the section, we discussed the Android application

210   architecture. Subsequently, we investigate the Android operating system (OS) weaknesses. This

211   background highlights the seriousness of some drawbacks to rationalize the necessity of

212   establishing this review and explain the essential terms to support the readers of this study.

213

214   **Android Application.** Android application, Android app, or APK refers to the Android

215   application from now on and throughout this paper. APK is a compressed file; an unzipping

216   program extracts its files and folders. This segment explains the APK components and their

217   contents, as some terms are essential in this study. APK developers use development tools that

218 occasionally require simple programming experience from young developers. The Android app
219 runs on Dalvik or ART equivalent to Java Virtual Machine (JVM) in a desktop environment. The
220 APK structure consists of many files and directories; the main file is Classes.dex Java bytecode;
221 it includes the classes and is packed together in a single .dex file. The AndroidManifest.xml file
222 contains deployment specifications and the required permissions from Android OS. Resources
223 .arsc is compiled resources, and Res folder is un-compiled resources.
224 The Android system must install the APK file so that the end-user can utilize the application's
225 functionalities. The Android system only accepts APK with a valid developer certificate, called
226 developer identifier. Individual developers keep their certificate keys; there is no Central
227 Authority (CA) server to maintain developers' keys, and thus no chain of trust between app
228 stores and developers.
229 End-users need to run the installed applications, while other apps run as a service in the OS
230 background. Therefore, the Android application's main components are as follows:
231   a) Activities: The user interface that end-users interact with and that communicates with
232   other activities using intents.
233   b) Services: Android application component runs as a background process and bonds or
234   un-bonds with other Android system components.
235   c) Broadcast and Receivers Intents: send messages that all other applications or
236   individual applications receive.
237   d) Content Providers: It is the intermediate unit to share data between applications.
238
239 **Android Weaknesses.** With some insight into the Android applications' development design, we
240 list the Android system's weaknesses and definitions for the readers of this study. The following
241 is a list of Android flaws and characteristics that malware authors and attackers abuse.
242   a) Open Source:
243   The advantage of Android source code's openness helps developers' communities enhance
244   the OS and add more features. Therefore, the Android community improves Android OS
245   daily. But, this contradicts with the security concerns when malware writers take this
246   advantage. It makes their job more straightforward than in closed source firmware, which
247   commonly triggers new vulnerabilities and malware attacks (Xu et al. 2016).
248   b) End-users Security Awareness:
249   End-users understanding malware's seriousness plays a vital role in early prevention and
250   detection when using feedback and reviews. However, the end-users feedback system is
251   insecure and easily polluted by fake comments (Rashidi et al. 2015). End-users click on
252   malicious URL links in emails, web browsers, pop-ups, or Android application dialogues
253   that download and install malicious applications. The end-users grant permissions to the
254   apps without studying the apps' actual requirements; they believe and follow fake
255   advertisements of permissions greedy apps.
256   c) Third-party Apps Market:

257   Android lets end-users download applications from third-party markets and install such
258   application offline by enabling installations from unknown sources in the phone settings
259   menu. Several untrusted or well-verified application stores offer Android the third-party
260   application, such as Amazon, GoApk, Slide ME, and other apps markets. In addition, there
261   are four Chinese App markets Anzhi, Mumayi, Baidu, and eoe app third party markets,
262   since Google Play restricted access to the Android Play Store for the Chinese population
263   (Fsecure 2013). End-users download mobile applications from any website to their
264   mobiles devices, personal computers, or laptops via tools such as the ADB tool in Android
265   SDK, which increases the probability of installing malicious apps (Sufatrio et al. 2015b).
266   d)   The Coarse Granularity of Android Permissions:
267   The Android system controls the users' application access using coarse granulated
268   permissions, i.e., one permission that provides access to entire Internet protocols and all
269   sites. There is no competent permission administration or sufficient permission
270   documentation, leading to excess permissions (Fang et al. 2014).
271   e)   Developers' Signatures:
272   Android application developers have to sign their apps with their developer key before
273   uploading the developed application to the market. There is no external party to
274   authenticate developers' signatures and thus no confidentiality or integrity (Holla & Katti
275   2012). Hence, malware developers clone benign applications and sign the APK with their
276   developer key after injecting malicious codes (Zhang et al. 2014). Later, malware
277   developers upload malicious APK to third-party application markets or share the infected
278   applications directly with their victims.
279   f)   Application Version Update:
280   Android applications usually enhance their functionalities in the form of version updates.
281   The security frameworks analyze the application during installation, and the update
282   process downloads new services/features without security precautions or checks (Luyi et
283   al. 2014).
284   g)   USB Debugging:
285   USB debugging is a valuable feature for Android Application development; it helps
286   developers be more productive and efficiently troubleshoot applications. It allows direct
287   installation of an application to the Android device using Android SDK tools such as the
288   ADB tool. In addition Expo framework (Zhang et al. 2016) has the possibility of live
289   reloading and dynamic code loading online. On the other hand, malware writers utilize
290   live loading features to gain remote access to install malicious applications using static
291   and dynamic methods. The static method injects JAR (Java) or *.SO (JNI) files to the
292   application before running, while the dynamic method call external files during runtime
293   (Zhang et al. 2016).
294   h)   Dynamic Code Loading (DCL):
295   DCL is an Android OS feature that enables benign Android applications to call another
296   APK or malicious code to compile and execute it in real-time. However, malware

297   developers use this feature to load their malicious codes dynamically after the detection

298   framework ranked the malicious app as benign.

299   i)   Inter-application Communication (intent):

300   Android OS uses the inter-application intent system to deliver a message from and to

301   applications. Malware developers sniff, modify, or gain knowledge, compromising data

302   integrity and privacy (Chin et al. 2011). The intent provides flexibility in Android

303   application development, but it is an entry point for security threats (Feizollah et al. 2017;

304   Salva & Zafimiharisoa 2015).

305

306

307   **Evasion Techniques**

308   This section represents our taxonomy of the currently used evasion techniques and research

309   studies on detecting obfuscated malware. Our taxonomy focuses on classifying the related

310   studies with the same objectives and goals to harvest a comprehensive collection of material and

311   comparative conclusions. When scrutinizing many existing studies, we find it more appropriate

312   to study the evasion detection capabilities of each studied framework after introducing the

313   evasion techniques that hinder malware analysis and detection. This section presents the

314   taxonomy of detection techniques for the ground truth relation between the detection

315   methodology and the evasion ability. Android applications have powerful tools and techniques to

316   secure and protect their applications from being reverse-engineered. Conversely, malware

317   authors are using obfuscation tools and techniques to evade detection. Therefore, evasions, or in

318   other terms, transformation techniques, are techniques that try to defeat Android malware

319   detection and rank the malware applications as benign.

320   As displayed in Figure 2, we categorize evasion techniques into two main types. The first

321   category is polymorphism; it transforms the malicious malware code without changing the

322   original code of the mobile application. The second category is metamorphism, which mutates

323   the application code, but maintains the same behaviour. Malware authors employ obfuscation

324   tools, such as Obfuscapk (Aonzo et al. 2020), ProGuard (Lafortune 2002), DashO (Wang et al.

325   2016), KlassMaster (Kuhnel et al. 2015), and JavaGuard (Sihag et al. 2021a) to encrypt their

326   code and decrypt during runtime; they modify the code itself to evade the heuristic detection and

327   signature analysis of the malware detection techniques.

328   **Polymorphism.** Polymorphic malware is the malware category that keeps changing its

329   characteristics to generate different malware variants evading malware detectors. Polymorphic

330   malware encrypts part of the code embedding malicious code. The polymorphic malwares

331   encrypt itself with variable encryption keys but maintaining the malicious code body unaltered.

332   Polymorphic malware is an advanced version of oligomorphic malware. The oligomorphic

333   malware encrypts the malicious code to defeat source code static analysis based malware

334   detection. Usually, the malware decrypts the malware using the same techniques. However, the

335   oligomorphic malware decrypts the encrypted malicious code using different deyrptor to make

336   decryptor analysis more difficult. The static analysis analyze the decryptor to find the encryption

337 key that enable the detection of the malware.Hence, the static analysis approach is not effective
338 with oligomorphic malware. Polymorphic malware continuously change the decryptor technique
339 to make it more diffcult to the source code static analysis approach. These symptoms are
340 indications of the presence of malicious code in an application. In this section, we discuss the
341 polymorphism evasions subcategories, which are package transformation and encryption.
342 **Package Transformation.** In this section, we study types of package transformation, which are
343 Repacking (RPK), Package Renaming (PKR), and Identifier Renaming (IDR).
344   a) *Repacking (RPK):* It is the process of unpacking the APK file and repacking the original
345   application files but signing the APK file with a developer security key (Rastogi et al. 2013).
346   This way, the code remains unchanged and signed the application with a different key. To
347   repackage Android application, attackers unzips the APK file into DEX file, hence, attackers
348   adopts reverse engineering tools to extract Java or smali code from the DEX file. Using
349   classes, string, and methods rearrangement in DEX file, attacker modifies the architecture of
350   the DEX arrangement resulting into defeating signature based Android malware detection.
351   Canfora (Canfora et al. 2015b) considers a simple repacking evasion technique. It hinders
352   malware detection using all of the commercial anti-malware that uses signature-based
353   detection techniques. Thus, with every iteration, the malware's signature is changed, after
354   which the malware can evade detection. For instance, one AnserverBot malware sample
355   repackaged and disguised as a paid application is available on the official Android Market.
356
357   b) *Package Renaming (PKR):* Every Android application has a unique package name. For
358   instance, com.android.chrome is the package name of Google Chrome. PKR uses multilevel
359   techniques to obfuscate the application classes except for the main Class, for instance,
360   "FlattenPackageHireachey" or "RepackageClass" options(Lafortune 2002). As shown in
361   Algorithm 1, PKR changes all classes' names except the "MyMain" class.
362
363 Figure 2: Evasion Technique Taxonomy
364
365 This algorithm is applied relatedly to form the multilevel PKR obfuscation. The GinMaster
366 family contains a malicious service that can root devices to escalate privileges, steal confidential
367 information. Later, it receives instructions from a remote server to download and install
368 applications without user interaction. The malware can successfully avoid detection by mobile
369 anti-virus software by using polymorphic techniques to hide malicious code, obfuscating class
370 names for each infected object, and randomizing package names and self-signed certificates for
371 applications. Therefore, PKR evades the malware detection technique and causes false negatives,
372 proven by (Faruki et al. 2015c) by applying PKR to malware applications and scanned using
373 Virustotal platform. It shows that the repackage malware detection accuracy dropped to half in
374 all malware categories.
375   d) *Identifier Renaming (IDR):* Identifier is another APK parameter representing the
376   application developer's signature. Classes, methods, and fields consider bytecode identifiers,

377   as a signature is generated based on. Malware authors change developer identifiers using

378   many obfuscation tools such as ProGuard (Lafortune 2002) and DexGuard (GuardSquare

379   2014) to appear as a variant application from the previously detected malicious application,

380   leading to a different signature and evading detection methods. Real-world malware families

381   that rename identifiers are as follows: DroidDream, Geinimi, Fakeplayer, Bgserv,

382   BaseBridge, and Plankton.

383

384   **Encryption Transformation.** Some Android malware families encrypt data values inside the

385   code, compiled code or payload, and decrypt the payload whenever desirable. This paper refers

386   to *Data Encryption as* DEN, Bytecode Encryption as BEN, and Payload Encryption as PEN.

387   This paper examines the following types of evasions:

388   a) *Data Encryption (DEN):* This evasion technique tends to encrypt specific data vital for

389   the malicious action and decrypt the encrypted data later, which modifies the malware

390   application characteristics to evade the detection techniques (Kuhnel et al. 2015). The

391   data refers to strings or network addresses embedded in the code. By encrypting such

392   components, the malware can avoid detection methods (Shrestha et al. 2015), in which

393   the authors extracted strings from APK files and analyzed the decrypted strings to detect

394   malware. Real-world malware families that encrypt payload are as follows: DroidDream,

395   Geinimi, Bgserv, BaseBridge, and Plankton.

396   b) *Bytecode Encryption (BEN):* using ProGuard (Lafortune 2002) or DashO (Maiorca et al.

397   2015) obfuscation tools, the BEN evasion hinders reverse engineering by encrypting

398   original code and makes it almost impossible to read. It divides the code into two parts,

399   the encrypted and non-encrypted parts. The non-encrypted code part includes the

400   decryption code for the encrypted part (Faruki et al. 2014; Rastogi et al. 2014) during

401   run-time. Therefore, dynamic analysis is required to detect this decryption process.

402   However, some static analysis-based detection frameworks propose BEN evasion

403   detection, such as DroidAPIminer (Aafer et al. 2013) and Wang (Wang & Wu 2015) that

404   successfully detect BEN evasion but fail in DEN or PEN evasions detection.

405   c) *Payload Encryption (PEN)*: Malware authors use payload encryption as in DroidDream

406   (Foremost 2012) malware to carry malicious payloads inside applications and install

407   malicious applications at runtime once the system is compromised. The code is encrypted

408   and decrypted during run time, which calls a decrypting function (Cho et al. 2018) and

409   runs it in real-time.

410

411   Algorithm 1

412   Package Renaming (PKR)

413

414   **Metamorphism.** Metamorphic malware is more complex than polymorphic malware that shows

415   a better ability to evade detection frameworks. Malware authors adopt metamorphic malware so

416   to make metamorphic malware detection harder than leveraging polymorphic malware. The

417 metamorphic malware writes new malicious code that varies in each iteration using the same
418 encryption and decryption key. For example, Opcode ngrams (Canfora et al. 2015a) adopts the
419 ngrams feature extraction algorithm to extract the suspected string with n count in the Opcode. It
420 assumes that the Malware writers rarely develop metamorphic Android malware variants. Based
421 on that assumption, it ignored the evaluation of the ngrams' detection framework against
422 metamorphic evasions (Canfora et al. 2015a). Metamorphic malware rewrites itself in every
423 iteration to evade detection methods.
424 **Code Obfuscation.** Code obfuscation is an evasion technique initially used to protect
425 applications from piracy and illegal use by many obfuscation techniques. Conversely, malware
426 authors use code obfuscation techniques to evade malware detections. In this study, we highlight
427 three types of code obfuscation the *Code Reordering (CRE), Call Indirection (CIN), and Dead*
428 *Code Insertion (DCI).*
429   a) *Code Reordering (CRE):* This transformation changes the order of the code by inserting
430   the standard "goto" command to maintain the proper program instruction order.
431   b) *Call Indirections (CIN): CIN* is an object-oriented feature used dynamically to reference
432   specific values inside the code; CIN creates code transformation evasion, obfuscating the
433   call graph detection techniques (Castellanos et al. 2016; Gascon et al. 2013). Malware
434   families such as DroidDream, Geinimi, and FakePlayer incorporate call indirection to
435   evade static analysis based Android malware detection.
436   c) Dead Code Insertion (DCI): Malware inserts junk code into the sequence of the
437   application to ruin its semantics. This type of transformation makes the malware more
438   difficult to analyze (Kwon et al. 2014). AnDarwin (Crussell et al. 2015) experimented
439   with detecting Android malware based on code similarity. Their used method is unable to
440   detect dead code insertion transformation (Crussell et al. 2015). The code similarity
441   approach uses a distance-vector technique, representing the distances between the
442   original code or the DCI transformation representing a distance vector. The far the
443   distance vector, the more complex the detection of such obfuscation.
444
445 **Advanced Code Transformation.** This section explains the advanced code transformation
446 techniques that are more sophisticated in hindering the malware detection frameworks. We
447 include advanced evasion techniques, such as *Native Exploits (NEX), Function Inlining and*
448 *outlining (FIO), Reflection API (REF), Dynamic Code Loading/Modification (DCL/DCM),* and
449 *Anti-debugging (ADE)*.
450   a) *Native Exploits (NEX):* Android applications call native libraries to run system-related
451   functions. The malware uses a native code exploit to escalate the root privilege while
452   running (Xu et al. 2016). Unfortunately, many exploits' source code is available for
453   download. Official Android suppliers are working on a solution using regular system
454   updates and fixes. Additionally, DroidDream malware (Wu et al. 2015) packs native code
455   exploits with application payload, bypassing Android security monitoring and logging
456   systems.

b) *Function Inlining and Outlining (FIO):* Inlining and outlining are compiler optimization techniques options. Inlining replaces the function call with the entire function body, and the outlining function divides the function into smaller functions. This type of transformation obfuscates the call graph detection technique by redirecting function calls and creating a maze of calls (Gascon et al. 2013).

c) *Reflection API (REF):* Reflection API is a technique to initiate a private method or get a list of parameters from another function or class, whether this class is private or public. Android developers legitimately use it to provide genericity, maintain backward compatibility, and reinforce application security. However, malware authors take advantage of this feature and use it to bypass detection methods. Reflection evasion facilitates the possibility to call private functions from any technique outside the main class. Recently few studies highlighted the reflection effect on code analysis and considered reflection during the analysis process (Kuhnel et al. 2015; Li et al. 2016).

d) *Dynamic code loading/Modification (DCL/DCM):* Since Java has the capability of loading code at runtime using class loader methods, Android malware application dynamically download malicious code using the dynamic code loading (DCL). The DCL and DCM techniques provide advanced evasion capability to malware authors, and improper use can make benign applications vulnerable to inject malicious code. For instance, the Plankton malware family uses dynamic code loading to evade detection methods. As being the first malware with DCL that stealthy extend its capabilities on Android devices. It installs an auto-launching background application or service to the device, collecting device critical information to a server. The server sends the malicious class payload URL link to the background service using an HTTP_POST message containing a Dalvik Bytecode jar malicious payload file. In the following trigger of "init()" event of the main application, the malicious payload is invoked using the "DexClassLoader" class. Due to the unavailability of the dynamically loaded code during Android malware static analysis, the DCL and DCM evasion technique is another transformation technique that is a big challenge for static analysis (Hsieh et al. 2016; Li et al. 2016). Although some researchers (Poeplau et al. 2014; Zhang et al. 2015b; Zhauniarovich et al. 2015) studied how DCL evades malware detection, it is still an open issue that needs more attention. Grab'n run (Falsina et al. 2015) uses code verification techniques to secure dynamic code loading and protect it from misuse by malware authors and attackers.

e) *Anti-debugging (ADE):* The malware developer presumes the limitation of Android that only one debugger can be attached to a process using ptrace functionality (Zhang et al. 2015b). Hence, it prohibits attaching a debugger to the suspected application. If the malware detects the running debugging tool like Java Debug Wiring Protocol (JDWP), it discovers the operating environment running under an Android emulator or physical device. Andro-Dumpsys (Jang et al. 2016) is a hybrid Android malware analysis

496   framework that claimed that it disables the attachment of "ptrace" monitoring app service
497   to monitor the running applications, which lack ADE detection.

498

499   **Anti-emulation Transformation.** The primary objective of anti-emulation evasion is to detect
500   the running environment of the sandbox and benignly masquerade as a clean application instead
501   of launching the malicious code, which we refer to as *Virtual Machine Aware (VMA)*. Another
502   side of anti-emulation evasion is detecting automatic user interaction emulation, which refers to
503   as *Programmed Interaction Detection* like the monkeyrunner tool used in many frameworks, for
504   instance, the Droidbox (Desnos & Lantz 2014) sandbox tool in the Mobile-Sandbox
505   (Spreitzenbarth et al. 2015).

506   a) *Virtual Machine Aware (VMA):* The dynamic analysis requires either an Android virtual
507   machine emulator or a physical device to install the suspected application. Scientists
508   studied the possibility of detecting the running environment fingerprints to differentiate
509   between an emulator and a physical device (Jing et al. 2014; Maier et al. 2014; Maier et
510   al. 2015; Vidas & Christin 2014). Android.obad (Faruki et al. 2015b; Singh et al. 2015) is
511   an emulator-aware malware, which complicates the analysis process. The malware looks
512   for the "Android.os.build.MODEL" value throughout the code and exits if it matches the
513   emulator's model. The malware only runs in an emulator after patching WMA checks.

514   b) *Programmed Interaction Detection (PID):* Android malware is an event-driven
515   application that needs a particular series of user interactions to launch malicious actions.
516   Therefore, dynamic analysis requires a running environment user/gesture interaction.
517   Malware writer refers to PID obfuscation as code coverage. Some researchers have tried
518   to address code coverage; however, it remains a challenge to detect it.

519   We scrutinize the top Android malware detection frameworks against the two main evasion
520   categories based on the introduced definitions of Android malware evasion techniques. The first
521   category is polymorphism, which consists of package transformation and encryption
522   transformation. Package transformation includes *Repacking (RPK), Package Renaming (PKR),*
523   *and Identifier Renaming (IDR)*. Encryption transformation includes *Data Encryption (DEN),*
524   *Bytecode Encryption (BEN), and Payload Encryption (PEN)*. The metamorphism subcategories
525   are obfuscation transformation, advanced code transformation, and anti-emulations
526   transformation. The code obfuscation subcategory includes *Code Reordering (CRE), Call*
527   *Indirection (CIN), and Dead Code Insertion (DCI)*. Advanced code transformation includes
528   *Native Exploits (NEX), Function Inlining and outlining (FIO), Reflection API (REF), Dynamic*
529   *Code Loading/Modification (DCL/DCM), and Anti-debugging (ADE)* evasion techniques. Last
530   but not least, anti-emulation transformation includes *Virtual Machine Aware (VMA) and*
531   *Programmed Interaction Detection (PID)*.

532   **Android Evasion Detection Frameworks**
533   Many researchers (Apvrille & Apvrille 2015; Bagheri et al. 2015; Battista et al. 2016; Chenxiong
534   et al. 2015; Elish et al. 2015; Fratantonio et al. 2016; Gonzalez et al. 2014; Gurulian et al. 2016;
535   Kuhnel et al. 2015; Lei et al. 2015; Li et al. 2016; Martín et al. 2016; Preda & Maggi 2016;

536   Sheen et al. 2015; Shen et al. 2015; Sun et al. 2015; Wang et al. 2016; Wu et al. 2016; Zhang et
537   al. 2016) examine their frameworks against different evasion techniques, and they take
538   countermeasures to overcome evasion techniques, which prevent the anti-malware framework
539   from detecting malicious applications. These evasions are the leading cause of false negatives, as
540   they allow many malware applications to penetrate freely into Android smart devices. This
541   section investigates the latest frameworks with different approaches, finding a robust solution to
542   detect evasion techniques. We are aiming to discover the gap in this area of research. We also
543   review the different evasion test benches and tools that researchers and commercial enterprises
544   use to secure their codes. We review the latest detection frameworks and their resilience against
545   five different evasion categories and 16 different subcategories distributed into 56% static
546   analysis, 28% dynamic, and 16% hybrid frameworks.
547   **Android Malware Detection Techniques.** There are three leading techniques for Android
548   malware detection Figure 3 presents the three main categories of Android malware detection
549   techniques, the *first* category is logic-based techniques (Lee et al. 2014; Zhang et al. 2015a),
550   based on hard-coded safe lists and predefined alarms stored in text files or a small database like
551   Amamra (Amamra et al. 2015). The *second* category is signature based malware detection
552   techniques (Niazi et al. 2015; Tchakounté et al. 2021), it based the malware detection on
553   comparing the suspicious application with malware application signature. The *third* category of
554   Android malware detection uses machine learning (ML) classification algorithms to classify the
555   application as benign or malware (Afonso et al. 2015; Alzaylaee et al. 2016; Amamra et al. 2015;
556   Baskaran & Ralescu 2016; Canfora et al. 2016; Canfora et al. 2015c; Castellanos et al. 2016;
557   Faruki et al. 2015a; Feizollah et al. 2015; Fratantonio et al. 2016; Kurniawan et al. 2015; Lei et
558   al. 2015; Lindorfer et al. 2015; Lopez & Cadavid 2016; Meng et al. 2016; Nissim et al. 2016;
559   Spreitzenbarth et al. 2015; Spreitzer et al. 2016; Wang & Wu 2015; Wu et al. 2016; Xu et al.
560   2016; Yerima et al. 2014; Yuan et al. 2016; Zhang et al. 2016). The ML-based techniques extract
561   the Android devices feature that represent the Android application characteristics such as the
562   application's permission, code hierarchy from reverse engineering process, or monitoring
563   application behaviour in runtime. The collected feature is a result of static, dynamic, or hybrid
564   analysis of anlysing Android applications. The collected features are used to build machine
565   learning classification model that decides whether the application is malware or benign.
566
567   Android malware detection methodologies are classified from a different point of view, as
568   depicted in Figure 4, defining the Android malware detection taxonomy as post-installation and
569   pre-installation methods.
570   **Post-installation Detection.** This section explains the Android vulnerability check and monitors
571   the system logs after installing the application. Therefore, post-installation analysis reports the
572   security issues and malicious activity to the end-users.
573   a) Vulnerability Check: The vulnerability check method scans all existing Android Apps
574   and Android system versions against common security threats. APSET (Salva &
575   Zafimiharisoa 2015) collects the vulnerability pattern using the Android application's test

576   case execution framework, which supports receiving exceptions. However, using more
577   vulnerability patterns or generating more test cases per pattern improves the APSET
578   malware detection performance.
579   b) Monitoring Logs: Android systems use process monitoring tools and network monitoring
580   tools. Mobile- Sandbox (Spreitzenbarth et al. 2015) uses the process ltrace monitoring
581   tool and PCAP network monitoring tool to capture the required data for analyzing the
582   Android applications.
583
584   Figure 3: The Main Categories of Android Malware Detection Techniques
585
586   **Pre-installation Detection.** Android malware detection frameworks perform static, dynamic, or
587   hybrid analyses to analyze features for malware detection techniques, which classify the apps as
588   benign or malware. Hence, we identify the following application analysis methodologies.
589   **Static Analysis.** It is a technique to reverse engineer the APK statically without installing it; the
590   analysis requires reading configuration settings, decompiling executable bytecode, and extracts
591   the source code for further analysis.
592   a) *Signature-based:* This paper classifies the signature-based method under static analysis
593   detection because the signature-based detection approach builds its frameworks with static
594   Android application characteristics. As such, DroidAnalytics (Zheng et al. 2013b) uses a
595   signature-based manner in which it dynamically collects and creates a signature for each
596   malware and stores malware signature into a central database. This model has limitations
597   where each of the new malware family variants needs a different signature. LimonDroid
598   (Tchakounté et al. 2021) proposed a signature-based database of Android malware signature
599   based on fuzzy hashing technique. It builds a signature database for literature purposes rather
600   than a malware detection framework.
601   b) *Permission-based*: APK Auditor (Talha et al. 2015) is a static model that leverages
602   *permission-based* detection castoff decompressing the APK package; it extracts the malicious
603   symptoms using permission and signature matching analysis. Likewise, Triggerscope
604   (Fratantonio et al. 2016) uses permissions characteristics as an input to classify the
605   application using different machine learning algorithms (Abdulla & Altaher 2015; Alazab et
606   al. 2020; Arora et al. 2019; Dharmalingam & Palanisamy 2021; Fang et al. 2014; Glodek &
607   Harang 2013; Li et al. 2018b; Niazi et al. 2015; Şahin et al. 2021; Shalaginov & Franke 2014;
608   Talha et al. 2015; Tiwari & Shukla 2018).
609   c) *Source Code based Analysis*: Drebin (Arp et al. 2015) extracts features from the
610   application's *Androidmanifest file* and *source code*; it scrutinizes the code by listing the
611   *native calls*, *API calls,* and *URL addresses*. It uses machine learning classification to
612   discriminate between malware and benign apps. Likewise, DroidMat (Wu et al. 2012) uses
613   the configuration file to get the required permission by the APK and counts the method that
614   has API calls from the decompiled source code; it uses 1,500 benign APK applications and
615   238 malware, evaluates the accuracy of the framework, and achieves 97.87% accuracy.

616   However, Lei (Lei et al. 2015) proposed a probabilistic discriminative model based on
617   decompiled source code with permissions. It classified Apps as benign and malware using
618   machine learning classification techniques. Hanna (Hanna et al. 2013) tried to find the code
619   similarity among Android applications to detect similar code patterns with the same
620   vulnerabilities and the repackaged or cloned applications in Android markets.
621 **Dynamic Analysis.** Dynamic analysis is the process of running the suspect app in an isolated
622 Android environment. It starts by receiving the Android application APK files, either using an
623 online scanning portal VirusTotal (Google) or a scanning agent on an Android
624 smartphone/device. Next is opening a suitable Android operating environment in a physical
625 device or emulator, which we hereafter refer to as a sandbox. The sandbox isolates the
626 application to protect the analysis device from possible malicious attacks. Later, the dynamic
627 analysis starts system logging and network monitoring tools and captures the default system logs.
628
629 Figure 4**:** Taxonomy of Android Malware Detection Methodologies
630
631 Once the sandbox and the logging or monitoring tools are ready, the APK installation follows,
632 and once the installation is successful, the logging system captures all system logs. Dynamic
633 analysis requires the application to start and run all codes and capture all changes to the Android
634 system environment. The sandbox captures the system logs before installing the application and
635 compares the system logs after installing and running the suspect Android application. The
636 sandbox uses a monkeyrunner tool to randomly emulate user gestures and cover all the possible
637 alleged code in an Android application. Dynamic analysis sandboxing techniques install and run
638 Android applications in a virtual environment, emulator, or physical device and monitor the
639 application's behaviour. It considers network traffic, opened ports, and system calls. One of the
640 main issues during the monitoring process is the user interaction simulation tool, which simulates
641 the user interaction gestures that must cover all possible interactions. The following are types of
642 sandboxing: *Sandbox Emulator*: Most researchers (Afonso et al. 2015; Desnos & Lantz 2014;
643 Faruki et al. 2015a; Spreitzenbarth et al. 2015) use Android emulators like Droidbox (Desnos &
644 Lantz 2014), TantDroid (Chao et al. 2020), and CuckooDroid (Technologies 2015), which run an
645 Android image as a virtual machine. Later, the framework destroys the used OS image and
646 prepares a factory reset Android OS for the following analysis process. *Physical sandbox device*:
647 The dynamic analysis algorithm resets the physical device to factory settings to make sure the
648 analysis captures only the suspected application's behaviour. It overcomes the limitations of
649 using emulators and uses physical devices to analyze suspicious applications (Shrestha et al.
650 2015) dynamically.
651 Android malware dynamic analysis faces some challenges; some malware families evade the
652 dynamic malware analysis environment by halting the malicious download until the dynamic
653 analysis finishes the monitoring period. The sandbox environment suffers from the
654 computational time required to load the Android operating system, create log files, install APK,
655 capture system logs and network traffic, and copy the log files to form understandable

656   characteristics. User gestures emulation using Android tools, such as monkeyrunner, is less
657   precise and partially covers the code of an application. Phone calls, SMS, GPS, and NFC
658   hardware emulation is another challenge in Android malware dynamic analysis, as they are not
659   as realistic as a physical device. The dynamic analysis kills the emulator after the dynamic
660   analysis time. Therefore, the dynamic analysis launches a new emulator instance needs for every
661   App analysis. These challenges prevent the dynamic analysis from performing effective malware
662   detection. Some studies have considered dynamic analysis to overcome the limitations of static
663   analysis (Afonso et al. 2015; Amos et al. 2013; Desnos & Lantz 2014; Enck et al. 2014a; Enck et
664   al. 2014b; Lindorfer et al. 2015; Spreitzenbarth et al. 2015; Wang & Shieh 2015; Zhao et al.
665   2014).
666   **Hybrid Analysis.** The hybrid-based detection frameworks, like Mobile-Sandbox (Spreitzenbarth
667   et al. 2015), Droiddetector (Yuan et al. 2016), and Andro-Dumpsys (Jang et al. 2016), combine
668   the dynamic analysis and static analysis techniques to reconcile the limitations of the static
669   analysis. The hybrid analysis extracts static features using reverse engineering techniques(Lim et
670   al. 2016). Static features are Apps permissions, code analysis, intent, network address, string, and
671   hardware features. Likewise, it extracts the dynamic analysis of the application by capturing the
672   network traffic, system calls, user interaction, and system components using sandbox
673   methodologies. Later, it combines a group of static and dynamic features, driving the machine
674   learning algorithms to classify the application to benign or malware.
675   **Android Malware Dataset.** Most Android malware detection frameworks adopt machine
676   learning algorithms to build a detection model; hence researchers crawl apps from the official
677   apps market store Google Play to build its dataset (Arp et al. 2015; Parkour 2013; Yajin &
678   Xuxian 2012). It also crawls sample applications from third-party application stores, such as
679   Soc.io Mall, Samsung Galaxy Apps, SlideME, AppsLib, GetJar, Mobango, Opera Mobile Store,
680   Amazon Appstore, and 1Mobile markets. To label the crawled applications as benign or
681   malware, researchers employ online security scanning tools as listed in Table 3. For instance,
682   Virustotal and AndroTotal, and the online service are used to scan the crawled Apps and cluster
683   the found malware apps into malware families. Researchers label all crawled apps using
684   VirusTotal to build Android malware detection datasets. Many of the dataset are published for
685   future academic research such as Drebin (Arp et al. 2015), Genome (Yajin & Xuxian 2012),
686   Kharon (Kiss et al. 2016), AMD (Li et al. 2017), AAGM (Lashkari et al. 2017), PRAGuard
687   (Maiorca et al. 2015), AndroZoo (Allix et al. 2016) datasets.
688   **Machine Learning in Android Malware Detection.** Based on collected characteristics or so-
689   called features (Feizollah et al. 2015), different machine learning classification techniques
690   classify APK as benign or malware. However, deep insight into machine learning techniques is
691   outside the scope of this study. Android malware detection classifies Android Apps into two
692   classes benign and malware. However, some papers detect Android Ransomware (Andronio et
693   al. 2015; Maiorca et al. 2017) considering three classes benign, malware, and ransomware.
694   Hence, we briefly explain the evaluation measures of ML classification. Machine learning
695   comprises three main categories, namely supervised, unsupervised, and reinforcement learning.

696   *a)* Supervised Model:
697   Supervised machine learning bases its model on a labelled dataset. The framework splits the
698   dataset into two subsets; first subset is for training and creating the classification model, and
699   the second subset is for testing and validating the trained classification model. Most
700   researchers split the data into 70% training and 30% testing subsets, but some split the data
701   into 50% for training and 50% for testing (Adebayo & AbdulAziz 2014).
702   *b)* Unsupervised Model:
703   In the unsupervised model, Apps are unlabeled. The unsupervised model recognizes the class
704   of the applications without knowing which App is malware or benign. Researchers use
705   unsupervised models to learn the covert pattern of the unlabeled data (Akpojaro et al. 2014;
706   Kohout & Pevny 2015; Tang et al. 2014).
707   *c)* Reinforcement Learning:
708   The machine exposes itself to an environment where it trains itself continually using trial and
709   error. This machine learns from experience and tries to capture the best possible knowledge
710   to make accurate business decisions. An example of reinforcement learning is the Markov
711   Decision Process (Kaelbling et al. 1996).
712   To understand the supervised model classification performance, ML introduces the confusion
713   matrix to calculate the performance measures as per Table 4. Let D be the total number of test
714   apps, which we use to examine the supervised ML model performance that classifies apps as
715   benign or malware, let M be the number of malware samples, and B the number of benign
716   samples.
717   True Positive (TP) represents the number of malware correctly classified.
718   False Positive (FP) accounts for the number of benign apps classified erroneously as malware.
719   True Negative (TN) represents the number of correctly classified benign apps.
720   False Negative (FN) accounts for the number of malware Apps classified erroneously as benign.
721
722   The ML performance measures represent the accuracy of the Android malware detection
723   classification frameworks. Table 5 explains the ML performance measure formulas and their
724   direct mathematical relation to the confusion matrix.
725   The Receiver Operating Characteristic (ROC) curve plots the TPR against FPR where TRP is the
726   y-axis and FPR is the x-axis. Every point in the ROC curve represents one confusion; it is all
727   based on TP and FP values. Area Under the Curve (AUC) is the area under the ROC curve
728   representing the aggregation of the ML trained model (Afifi et al. 2016; Baskaran & Ralescu
729   2016; Feizollah et al. 2015).
730
731   Table 3
732   Online Malware Scanning Frameworks
733
734   Table 4
735   Confusion Matrix

736
737 **Evasion Test benches Tools.** Researchers or commercial companies have developed the evasion
738 test benches to study the robustness of the currently available anti-malware applications or
739 protect their software packages from piracy issues. The first Test benches trials were ADAM
740 (Zheng et al. 2013a) and Droidchameleon Rastogi (Rastogi et al. 2013), which conclude that
741 there is a detection performance degradation when applying trivial obfuscation techniques.
742 However, researchers developed evasions tools to evaluate commercial anti-malware
743 performance, such as PANDORA (Protsenko & Muller 2013), Mystique (Meng et al. 2016),
744 AAMO (Preda & Maggi 2016), ProGuard (Lafortune 2002), and others as listed in Table 6.
745 Evasion tools were initially aiming to protect commercial software companies' applications from
746 piracy, such as DexGuard (GuardSquare 2014), which is an extension of ProGuard (Lafortune
747 2002), and Klassmaster (Klassmaster 2013). Recently, a pretty good number of researchers
748 develop frameworks targeting obfuscation and malware variant resiliency. PetaDroid (Karbab &
749 Debbabi 2021) introduces the severe first obfuscation dataset, which is a good initial. However,
750 it proves that the accuracy degrades with time and needs malware variant and obfuscation
751 adaptations. Dynamic analysis frameworks (Chen et al. 2018; Cho et al. 2018; De Lorenzo et al.
752 2020; Feng et al. 2018; Sihag et al. 2021b; Xue et al. 2017) declare the ability to detect all types
753 of obfuscated malware; however, most of it misses the evaluation report of each obfuscation
754 technique using obfuscated malware datasets. Researchers who evaluated their framework
755 against particular evasions are identified by mentioning the detected evasion, which represents
756 that the respective study either evaluated or presumed its ability to detect the evasion technique,
757 while "Failed to detected or ignored" means the respective study is defeated the corresponding
758 evasion technique. The "stared" cell indicates the framework that ignores the evaluation
759 experiments on evasion techniques or assumptions to that effect, or the study misses evaluating
760 its framework performance against this evasion technique.
761
762 Table 5
763 ML Classification Performance Measures
764

Evaluation of Evasion Detection Frameworks

766 We have explored the last ten years' research to evaluate the Android detection frameworks
767 against evasion techniques discussed in evasion techniques section. We studied Android malware
768 detection frameworks for the last decade from 2011 to early 2021, as listed in Table 7. We
769 categorize malware detection framework by the analysis methodology static, dynamic, and
770 hybrid features. This paper decides to put metadata analysis out of this research scope. We have
771 examined 74 static analysis based frameworks. The number of dynamic based analysis
772 frameworks are 35. The number of hybrid analysis frameworks is 26. Hence the total number of
773 examined papers are 135 research paper that this study selected from top rank journals and
774 conferences.
775

776 Table 6
777 Android Malware Evasion Test Benches
778
779 **Polymorphism Evasion Detection.** We examine the three main static, dynamic, and hybrid
780 frameworks versus polymorphism evasions. Table 8 represents static, dynamic, and hybrid
781 analysis based detection; we scrutinize each framework against polymorphism transformation
782 techniques in the two categories package transformation and encryption transformation. Each
783 framework uses various samples of Android malware and benign applications' datasets in the
784 evaluation process; each dataset contains a certain number of malware and benign applications.
785 For instance, APK Auditor (Talha et al. 2015) tested its framework against 6,909 malware and
786 1,853 benign applications; a total of 8,762 apps that APK Auditor crawled from Google play
787 store and other datasets such as Genome Project and Contagio. APK Auditor achieved 88%
788 malware detection accuracy. As it is signature-based, most of the evasion techniques prevent the
789 APK Auditor detection framework from detecting malware applications.
790   a) Package Transformation:
791 - *RPK - Repacking Evasion Detection*:
792 Detecting repacking evasion is possible using static analysis and detection techniques;
793 Dempster–Shafe (Du et al. 2015) investigate repacking characteristics using a control flow
794 graph and claimed better resistance to code obfuscation techniques. Likewise, Droidgraph
795 (Kwon et al. 2014) used the hierarchical class levels to determine the repackaged malicious
796 code to the original payload; it also considered the API calls, junk code, and code
797 obfuscation. It reduced the code comparison time compared to the polynomial time-
798 consuming native call graphs algorithm. Though, reflection successfully evades the detection
799 framework that uses the control flow graph. Other static detection approaches such as
800 MysteryChecker (Jeong et al. 2014), AnDarwin (Crussell et al. 2015), AndroSimilar (Faruki
801 et al. 2015d), ngrams (Canfora et al. 2015a), DroidEagle (Sun et al. 2015), DroidKin
802 (Gonzalez et al. 2014), DroidOlytics (Faruki et al. 2013), Gurulian (Gurulian et al. 2016),
803 Shen (Shen et al. 2015), and AAMO (Preda & Maggi 2016) have indicated their ability to
804 detect RPK evasions. While studying dynamic analysis papers, we notice that most dynamic
805 studies provide less attention to this evasion type. Similarly, Soh (Soh et al. 2015) and Wu
806 2015 (Wu et al. 2015) stressed that RPK evasion detection could detect RPK evasion, as
807 illustrated in Table 8. The study spotted 20 papers that scrutinized the RPK evasion using
808 static analysis, and only 2 papers scrutinized RPK using dynamic analysis.
809
810   - PKR - Package Renaming Detection:
811 Static analysis frameworks such as DroidoLytics (Faruki et al. 2013) and Droidkin (Gonzalez
812 et al. 2014) examine their capability in detecting PKR evasion techniques. However, many
813 other papers insufficiently evaluate its framework against PKR, such as APK Auditor (Talha
814 et al. 2015), DroidGraph (Kwon et al. 2014), Andro-tracer (Kang et al. 2015), Vulhunter
815 (Chenxiong et al. 2015), and COVERT (Bagheri et al. 2015), as presented in Table 8.

816   Dynamic and Hybrid analysis frameworks studies incompetently examine its robustness
817   against PKR, except one research, Shen (Shen et al. 2015) highlighted the issue of PKR and
818   its capability of detecting it as per Table 8. The study spotted 9 papers that scrutinized the
819   PKR evasion using static analysis, and only one papers scrutinized PKR using dynamic
820   analysis.
821
822   Table 7
823   List of examined Android malware detection frameworks
824
825   -   IDR Identifier Renaming Evasion Detection:
826   DroidOlytics (Faruki et al. 2013), AndroSimilar (Faruki et al. 2015d), Droidkin (Gonzalez et
827   al. 2014), Kuhnel (Kuhnel et al. 2015), Triggerscope (Fratantonio et al. 2016), AAMO(Preda
828   & Maggi 2016), and Battista (Battista et al. 2016) claim they can detect IDR evasion by
829   using their static Android malware detection frameworks as presented in Table 8.
830   Nevertheless, many other researchers inadequately evaluate its robustness against IDR
831   evasion. Table 8 demonstrates the issue of assuring the Android malware detection
832   frameworks' robustness against IDR evasion and scrutinizes the researchers' framework
833   against IDR evasion techniques.
834   In summary, most Android malware detection frameworks based on static analysis can detect
835   package transformation techniques (RPK, PKR, and IDR). However, most detection
836   frameworks based on dynamic and hybrid analysis inadequately evaluate or report their
837   resilience against IDR evasion techniques. The study spotted 20 papers that scrutinized the
838   RPK evasion using static analysis, and only 10 papers scrutinized IDR. The study spotted 9
839   papers that scrutinized the IDR evasion using static analysis, and only one paper scrutinized
840   IDR using dynamic analysis.
841
842   b)   Encryption Transformation Evasion Detection:
843   Static analysis detects encryption evasion techniques; many studies, such as DexHunter
844   (Zhang et al. 2015b), DroidKin (Gonzalez et al. 2014), Sherlockdroid (Apvrille & Apvrille
845   2015), Kuhnel (Kuhnel et al. 2015), and AAMO(Preda & Maggi 2016), have proved that
846   they detect the three encryption evasions (DEN, BEN, and PEN). Static based detection
847   studies, such as AndroSimilar (Faruki et al. 2015d), MysteryChecker (Jeong et al. 2014),
848   DroidKin (Gonzalez et al. 2014), SherlockDroid (Apvrille & Apvrille 2015), Kuhnel (Kuhnel
849   et al. 2015), Shen (Shen et al. 2015), a nd AAMO (Preda & Maggi 2016), are able to detect
850   DEN evasions. Likewise, Soh (Soh et al. 2015) and Q-floid (Castellanos et al. 2016) claimed
851   robustness against BEN evasion. The dynamic analysis based detection DwroidDump (Kim
852   et al. 2015) used code extraction executable code from the memory of Dalvik Virtual
853   Machine (DVM) instead of using a decompilation tool, which is subject to obstruction by the
854   three encryption evasions techniques as shown in Table 8. Nevertheless, the RiskRanker
855   (Grace et al. 2012) hybrid based detection framework successfully detected DEN, BEN, and

856 PEN. Hybrid detection frameworks such as RiskRanker (Grace et al. 2012), AMDetector
857 (Zhao et al. 2014), MARVIN (Lindorfer et al. 2015), and Mobile-Sandbox (Spreitzenbarth et
858 al. 2015) evaluated their frameworks against DEN evasion; they claim the ability to detect
859 BEN evasion techniques. Two dynamic detections papers evaluate their frameworks against
860 RPK evasion techniques: Soh (Soh et al. 2015) and Wu 2015 (Wu et al. 2015). Likewise,
861 DwroidDump (Kim et al. 2015) examines its framework against encryption evasion
862 techniques. Kumawate et al. also developed a system to detect cryptographic vulnerabilities
863 in Android applications and to detect malware (Kumawat et al. 2017). This study spotted 7
864 papers that scrutinized the DEN evasion using static analysis, only one paper scrutinized
865 DEN using dynamic analysis, and 2 papers scrutinized DEN using hybrid analysis. However,
866 this study spotted 6 papers that scrutinized the BEN evasion using static analysis, only one
867 paper scrutinized BEN using dynamic analysis, and 2 papers scrutinized BEN using hybrid
868 analysis. In addition, this study spotted 5 papers that scrutinized the PEN evasion using static
869 analysis, only one paper scrutinized PEN using dynamic analysis, and 2 papers scrutinized
870 PEN using hybrid analysis.
871
872
873 **Metamorphism Evasion Detection.** Table 8 represents static, dynamic, and hybrid-based
874 Android malware detection frameworks and their robustness against metamorphism evasion
875 detection techniques.
876 a) Code Obfuscation Detection:
877 Code obfuscation consists of CRE, CIN, and DCI; we explain each evasion detection framework
878 in the following list:
879 - CRE - Code Reordering Evasion Detection:
880 ResDroid (Shao et al. 2014), AnDarwin (Crussell et al. 2015), and Seqmalspec (Sufatrio et
881 al. 2015a) proposed static analysis based detection and managed to detect CRE evasion.
882 Likewise, Q-floid (Castellanos et al. 2016) detected CRE using the dynamic sandboxing
883 methodology. Mobile-Sandbox (Spreitzenbarth et al. 2015) hybrid based detection
884 frameworks detect CRE evasions. Nonetheless, CRE evades ngrams (Canfora et al. 2015a)
885 and Elish (Elish et al. 2015) static detection frameworks, which results in many false
886 negatives (FN), as shown in Table 9. This study spotted 17 papers that scrutinized the CRE
887 evasion using static analysis, only 2 papers scrutinized CRE using dynamic analysis, and 4
888 papers scrutinized CRE using hybrid analysis.
889 - CIN - Call Indirections Evasion Detection:
890 As shown in Table 9, the CIN evasion technique successfully evades the call graph based
891 Android malware detection frameworks (Chenxiong et al. 2015; Poeplau et al. 2014; Wu et
892 al. 2016). Despite the fact that many static frameworks easily detect CIN evasion (Faruki et
893 al. 2015d; Faruki et al. 2013; Gurulian et al. 2016; Kwon et al. 2014; Martín et al. 2016;
894 Narayanan et al. 2014; Wei et al. 2015; Zhang et al. 2015b), CIN still defeats other
895 frameworks such as APK Auditor (Talha et al. 2015), Andro-Tracer (Kang et al. 2015),

896 ngrams (Canfora et al. 2015a), Elsih (Elish et al. 2015) and Wu (Wu et al. 2016). Few
897 dynamic analysis based detection frameworks (Castellanos et al. 2016; Soh et al. 2015) and
898 hybrid detection frameworks such as (Grace et al. 2012; Lindorfer et al. 2015; Zhao et al.
899 2014) can detect- Call Indirections Evasion CIN. Choliy et al. developed a system called
900 ACTS (App topologiCal signature through graphleT Sampling) in which they detected
901 obfuscated function calls in malware samples (Choliy et al. 2017). This study spotted 15
902 papers that scrutinized the CIN evasion using static analysis, only 2 papers scrutinized CIN
903 using dynamic analysis, and 3 papers scrutinized CIN using hybrid analysis.
904
905 - *DCI - Dead Code Insertion Evasion Detection:*
906 AnDarwin (Crussell et al. 2015) conducted dead code insertion detection experiments based
907 on code similarity. AnDarwin reported that it is less robust to dead code insertion
908 transformation (Crussell et al. 2015) that adopts code's similarity approach with semantic
909 analysis, as shown in
910 Table 9. The similarity approach examines the distance vector values using semantic
911 analysis. The distance vector increases with the code alteration between the original and after
912 dead code insertion obfuscation. This study spotted 14 papers that scrutinized the DCI
913 evasion using static analysis, and 4 papers scrutinized DCI using hybrid analysis.
914
915 In general, the dynamic analysis framework Q-floid (Castellanos et al. 2016) introduces the
916 Qualitative Data Flow Graph (QDFG) to analyze the dynamic behaviour of a suspicious app. It
917 states that it detects code obfuscation, basing this assumption on PC-based malware detection
918 using Q-floid (Castellanos et al. 2016). It detects code obfuscation transformation using the
919 QDFG (Banescu et al. 2015; Wüchner et al. 2015). However, it claims that Q-floid (Castellanos
920 et al. 2016) inadequately detects Android malware when restricting using monitoring services.
921 MysteryChecker (Jeong et al. 2014) proposes a novel software-based attestation approach to
922 detect the repackaged malware with code obfuscation and a randomly selected encryption chain.
923 Likewise, Gurulian (Gurulian et al. 2016) introduces a DCI evasion resilient framework by
924 maintaining the attack vector; similarly, DroidOLytics (Faruki et al. 2013) uses statistical
925 similarity to detect application repackaging and code obfuscation. It builds a signature repository
926 that changes its length dynamically for code cloning detection. AndroSimilar (Faruki et al.
927 2015d) uses signature-based detection and attains 76% accuracy, but its detection rate of
928 repacking and code obfuscation transformation evasions is relatively low. Until today, AndrODet
929 (Mirzaei et al. 2019) adopts static analysis to detect Android malware applications with CRE,
930 CIN, and DCI evasions; however, the average achieved performance for detection CRE, CIN,
931 and DCI evasions is 63%.
932
933 Table 8
934 Polymorphism evaluation of frameworks
935

936 b) Advanced Code Transformation Detection:

937 It consists of NEX, FIO, REF, DCL, and ADE evasions explained in this section.

938   -   *NEX Evasion Detection*:

939   DroidAPIMiner (Aafer et al. 2013) uses static analysis to detect NEX evasion and, as listed

940   in Table 9, claims success; likewise, the dynamic analysis DroidBarrier (Almohri et al. 2014)

941   and hybrid analysis MARVIN (Lindorfer et al. 2015) claim the same. In contrast, many static

942   frameworks such as AdDetect (Narayanan et al. 2014), APK Auditor (Talha et al. 2015),

943   Andro-Tracer (Kang et al. 2015), and ngrams (Canfora et al. 2015a) stated their limitations in

944   countermeasures of NEX evasion as shown in Table 9. This study spotted one paper that

945   scrutinized the CIN evasion using static analysis, one paper scrutinized CIN using dynamic

946   analysis, and one paper scrutinized CIN using hybrid analysis.

947

948   -   *FIO Evasion Detection:*

949   AAMO (Preda & Maggi 2016) evaluates anti-virus packages versus function inlining and

950   outlining FIO evasion, as shown in Table 9. However, dynamic analysis and hybrid analyses

951   inadequately consider the evaluation of their framework against FIO evasion. This study

952   spotted one paper that scrutinized the FIO evasion using static analysis, and two papers

953   scrutinized FIO using dynamic analysis.

954

955   -   *REF Evasion Detection:*

956   As shown in Table 9, many static analysis frameworks examine the robustness of their

957   detection frameworks against REF evasion, such as DroidAPIMiner (Aafer et al. 2013),

958   DexHunter (Zhang et al. 2015b), SherLockDroid (Apvrille & Apvrille 2015), Kuhnel

959   (Kuhnel et al. 2015), DroidRA (Li et al. 2016), and AAMO. Likewise, Maier (Maier et al.

960   2015), which uses Dynamic analysis, RiskRanker (Grace et al. 2012), and StaDyna

961   (Zhauniarovich et al. 2015), which use hybrid analysis, study REF evasion detection using

962   dynamic and hybrid analysis based detection techniques. This study spotted 6 papers that

963   scrutinized the REF evasion using static analysis, only 2 papers scrutinized REF using

964   dynamic analysis, and 2 papers scrutinized REF using hybrid analysis.

965

966   -   *DCL Evasion Detection:*

967   Some Android malware detection frameworks propose and evaluate their methods to detect

968   DCL evasion, for instance, DroidAPIMiner (Aafer et al. 2013), Poeplau (Poeplau et al.

969   2014), Dexhunter, Maier (Maier et al. 2015), RiskRanker (Grace et al. 2012), and StaDyna

970   (Zhauniarovich et al. 2015). However, AndroSimilar (Faruki et al. 2015d) insufficiently

971   evaluates its mechanism against dynamic code loading, reflection, and other transformation

972   techniques, as shown in Table 9. This study spotted 4 papers that scrutinized the DCL

973   evasion using static analysis, only 2 papers scrutinized DCL using dynamic analysis, and 2

974   papers scrutinized DCL using hybrid analysis.

975

976   *ADE Evasion Detection:* Only the static analysis DexHunter (Zhang et al. 2015b) considered
977   the ADE evasion technique in evaluating the framework. On the contrary, the dynamic
978   analysis Q-floid (Castellanos et al. 2016) reported ineffective ADE evasion detection, as
979   shown in Table 9. This study spotted one paper that scrutinized the ADE evasion using static
980   analysis.
981   -   Anti-emulation Detection
982   Anti-emulation evasions consist of VMA and PID evasion techniques; the following is the
983   insight of detection framework analysis:
984   -   VMA Evasion Detection:
985   As a countermeasure for the VMA evasion technique, researchers (David & Netanyahu
986   2015; Mutti et al. 2015) equip an emulator sandbox with physical devices to dynamically run
987   the application analyzes. Dietzel (Dietzel 2014), Gajrani (Gajrani et al. 2015), and Hu (Hu &
988   Xiao 2014) propose a fake response agent, which feeds the in the dynamic analysis based
989   testing and a masquerade emulator as a physical device. In late 2015 and the beginning of
990   2016, several studies analyze the nature of anti-emulation malware with false values about
991   the environment request. This study spotted 6 papers that scrutinized the WMA using
992   dynamic analysis, and 3 papers scrutinized WMA using hybrid analysis.
993
994
995   Singh (Singh et al. 2015) enhances the dynamic malware detection robustness, using anti-
996   emulator and user interaction detection. Petas (Petsas et al. 2014) proposes countermeasures for
997   different evasion detections, such as anti-emulation using realistic sensor simulation and IMEI
998   modification. However, it inadequately evaluates this countermeasure. Dynalog (Alzaylaee et al.
999   2016) proposes a performance-enhanced Android malware dynamic analysis that uses the
1000   emulation tool, subject to emulation detection evasions. Likewise, Dynalog (Alzaylaee et al.
1001   2016) highlights the issue of dynamic analysis evasion without proposing a solution. To
1002   overcome VMA evasion, Vidas (Vidas et al. 2014) proposes system logs and network traffic
1003   classification features using a physical device A5 instead of emulator evasion techniques. Some
1004   studies only hoist the red flag to indicate that neither enough malware samples nor test benches
1005   exist for examining anti-emulation evasion (works such as Chaugule (Chaugule et al. 2011) and
1006   Tao (Tao et al. 2012)). Nevertheless, Mair (Maier et al. 2015) studied VWA evasion and
1007   proposed a solution based on comparing the behaviour of the APK when installing on a physical
1008   device and emulator, as shown in Table 9.
1009
1010   -   *PID Evasion Detection:*
1011   Programmed Interaction Detection is fortunate to evade automated dynamic analysis using
1012   the inherent difference between key runner and human interaction patterns (Diao et al. 2016).
1013   Instead of relying on identifying old virtualization or emulation techniques, Dai 2016 (Diao
1014   et al. 2016) focuses on detecting the automated gesture, which simulates user input, to

1015 conclude whether the application is under analysis or working under normal conditions, as
1016 shown in
1017 Table 9. As this anti-emulation evasion targeted sandboxing, which takes place during the
1018 dynamic analysis based detection, most of the efforts to countermeasure this type of evasion
1019 have used dynamic or hybrid analysis detection frameworks. This study spotted 4 papers that
1020 scrutinized the PID using dynamic analysis, and one paper scrutinized PID using hybrid
1021 analysis.

1022

1023 Table 9
1024 Metamorphism evaluation of frameworks

1025

Discussion

1027 In this section, this paper synthesizes the last decade's Android malware detection framework
1028 using three methodologies. First is identifying the evasions techniques requiring more attention
1029 from the research community. The second represents the potential evasion resilient detection
1030 techniques by reporting each framework's number of considered evasion techniques. The third
1031 summarizes the three types of Android application analysis with the number of frameworks that
1032 evaluated evasions techniques by bubble plot chart. Finally, we provide a to-do list and learned
1033 lessons from all the examined frameworks.
1034 The static analysis radar graph shown in Figure 5 signifies the evasion detection capabilities of
1035 static based detection. It serves to understand the evaluation of the static analysis based detection
1036 frameworks.
1037 Figure 5 presents the static analysis based Android malware detection frameworks using the
1038 radar graph approach. The radar graph represents the number of frameworks in circular layers,
1039 starting with the outside circle, which means zero frameworks. The second circular layer
1040 represents five frameworks. The inner-circle layer represents the largest number of frameworks
1041 that examined evasion techniques. Each evasion technique is labelled point such as PID, WMA,
1042 ADE, DCL, etc. Besides each point number representing the number of Android malware
1043 detection frameworks that evaluated its proposed model against this evasion technique or point
1044 in the radar graph. For example, 15 malware detection frameworks consider the RPR evasion
1045 technique; thus, the RPK label points to 15, as displayed in
1046 Figure 5. The evasion techniques that avoid Android malware detection using VMA and PID
1047 have zero values besides their points, as shown in
1048 Figure 5.
1049 We selected the Radar graph to demonstrate that static detection studies could detect package
1050 transformation evasions and basic code obfuscation; however, advanced transformation
1051 techniques and anti-emulation were neither studied nor evaluated. Concerning DCL, Pektas
1052 (Pektas & Acarman 2014), in 2014, detected anti-emulation evasion by using a dynamic
1053 analyzing tool developed just to deal with the DCL evasion malware samples, which achieved
1054 92% accuracy. Many researchers avoid using dynamic-based detection techniques because they

1055 are time-consuming and risk installing malware into their testing devices. In Mobile-Sandbox
1056 (Spreitzenbarth et al. 2015), the dynamic analysis required an average of eighteen minutes to
1057 accomplish the dynamic analysis tasks. This time depends on the size of the APK file and the
1058 dynamic analysis server hardware specifications.
1059 Figure 5: Evasion Techniques Radar for Static Frameworks
1060 Figure 6: Static Analysis Based Frameworks and Considered Evasion
1061 Until today, many static analysis researchers depends on permissions (Arora et al. 2019;
1062 Dharmalingam & Palanisamy 2021; Li et al. 2018a; Li et al. 2018b; Şahin et al. 2021); however,
1063 many are relying on API calls (Alazab et al. 2020; Jung et al. 2018; Maiorca et al. 2017; Mirzaei
1064 et al. 2019; Pektaş & Acarman 2020; Tiwari & Shukla 2018; Zhang et al. 2020; Zhang et al.
1065 2016; Zou et al. 2021) and deep code analysis and other types of features as discussed earlier in
1066 Android evasion detection frameworks section. Many of examined researches ignored the
1067 evasion techniques evaluation. Other frameworks assumed the impossibility of the evasion
1068 detection using static analysis and advise the research community to use dynamic analysis to
1069 detect it. Android Malware detection frameworks assumed their capability of detecting
1070 obfuscation techniques without evaluating their framework against obfuscated malware datasets.
1071 This paper examined 74 static frameworks, but only 35 research papers consider or evaluate their
1072 framework using at least one evasion technique, as shown in Figure 6. The dynamic analysis
1073 evasion radar graph demonstrates the capabilities of dynamic analysis based.
1074 Researchers assume that dynamic analysis covers all the simple obfuscations and transformation
1075 techniques. Hence many of the dynamic analysis frameworks (Abuthawabeh & Mahmoud 2019;
1076 Chen et al. 2018; de la Puerta et al. 2019; De Lorenzo et al. 2020; Feng et al. 2020; Feng et al.
1077 2018; Pang et al. 2017; Sihag et al. 2021b; Wang et al. 2019) ignored the metamorphic evasion
1078 techniques. The overall performance accuracy of the most current malware detection frameworks
1079 is measured against randomly selected malware samples representing certain malware families.
1080 If the randomly chosen malware families overlook evasion techniques, the selected malware
1081 insufficiently reflect the actual robustness of the proposed detection framework against evasion
1082 techniques; this was the main reason behind excluding the accuracy in evaluation tables. This
1083 paper examined 35 Android malware detection using different dynamic techniques. However,
1084 only 14 of 35 dynamic analysis based detection framworks have tried to include obfuscation into
1085 their evaluation processes, as shown in
1086 Figure 7. Figure 8 shows the number of considered evasion techniques in each research is
1087 between 1 and 5 evasions. In its evaluation, Soh (Soh et al. 2015) considered three types of
1088 repackaging evasion, indirectly considered code reordering, and called indirection evasion. It
1089 defines many limitations to its approach and planned to consider the hybrid analysis in its future
1090 plan.
1091 However, a few researchers evaluate their frameworks against specific evasion techniques, as
1092 reflected in the radar graph of the hybrid malware detection frameworks, as shown in Figure 9
1093 and Figure 10. For instance, four frameworks claimed that their method detected the CRE and
1094 DCI evasions (Grace et al. 2012; Lindorfer et al. 2015; Spreitzenbarth et al. 2015; Zhao et al.

1095   2014), and three frameworks claimed the detection of CIN (Grace et al. 2012; Spreitzenbarth et

1096   al. 2015; Zhao et al. 2014) and WMA (Grace et al. 2012; Petsas et al. 2014; Yuan et al. 2016).

1097   The hybrid based detection requires enormous effort to collect both static and dynamic

1098   characteristics and logs. RiskRanker (Grace et al. 2012) started highlighting the evasion

1099   problems and their impacts on detection accuracy. However, Petsas (Petsas et al. 2014) in 2014

1100   and Tap-Wave-Rub (Shrestha et al. 2015) battled anti-emulation evasions and used the device

1101   hardware (proximity sensor) to differentiate between maliciously driven actions and end-user

1102   physical interactions

1103

1104   Figure 7: Dynamic Analysis and Evasion Radar Graph

1105

1106   Figure 8: Dynamic Analysis Based Frameworks and Considered Evasion

1107

1108   Most of the recent dynamic analysis researches (Feng et al. 2020; Mahindru & Sangal 2021;

1109   Sihag et al. 2021b) confirmed the ability to detect obfuscated Android malware. Unfortunately,

1110   none of dynamic analysis based detection has evaluated their framework using specific evasion

1111   techniques; most of dynamic analysis studies just randomly select from the publicly available

1112   Android malware datasets. For example, Droidetec (Ma et al. 2020) proposed a dynamic analysis

1113   based framework by analyzing the process behavior in an ordered manner. Still, the evaluation

1114   process was generic and included few malware families that exclude obfuscated malware.

1115   The Hybrid analysis techniques are suggested by many researchers and have been set in their

1116   future plan to overcome the resiliency issue of complex obfuscation techniques. However, it is a

1117   shocking fact that the examined 26 Android malware detection frameworks using hybrid

1118   analysis, that only nine frameworks just consider few evasion techniques such as RiskRanker

1119   (Grace et al. 2012) that has initiated the issue in 2012, Mobile-Sandbox (Hoffmann et al. 2016),

1120   Marvin (Lindorfer et al. 2015). Recently some hybrid analysis based detection Puerta(de la

1121   Puerta et al. 2019), Surendrean (Surendran et al. 2020), Lu(Lu et al. 2020), Dhalaria (Dhalaria &

1122   Gandotra 2021), Zhu (Zhu et al. 2021), Nawaz (Nawaz 2021), Liu (Liu et al. 2021), PNSDroid

1123   (Kandukuru & Sharma 2018), Bacci (Bacci et al. 2018), DAMBA (Zhang et al. 2020) has

1124   highlighted the complex evasions detection resiliency issue in their research literature; however,

1125   the proposed malware detection methods and experiments of excluded the obfuscated malware

1126   from their evaluation sheets.

1127

1128   Figure 9: Hybrid Analysis and Evasion Radar

1129

1130   Figure 10: Hybrid Analysis Based Frameworks and Considered Evasion

1131 The systematic evasion detection map is illustrated in Figure 11; the horizontal axis represents
1132 each type of evasion in this study. The bubble size represents the accumulative number of
1133 detection techniques developed by the research community to fight each evasion technique. It is
1134 divided into three main categories in the vertical axis: static, dynamic, and hybrid detection
1135 techniques. For instance, the circle with the number "17" represents static Android malware
1136 detection frameworks, which consider CRE evasions on the framework evaluation process. As
1137 per the systematic map, the NEX, FIO, and ADE need more attention from the research
1138 community. Likewise, the overall dynamic analysis studies that considered evasion evaluation is
1139 shallow.
1140 Researchers have concentrated on Android malware static analysis in the last few years, which
1141 requires less time and effort than dynamic analysis. They tried to overcome the static analysis
1142 weaknesses against evasion attacks, which is why many researchers evaluated their frameworks
1143 to check the anti-obfuscation capabilities, as presented in Figure 11. Dynamic analysis
1144 researchers concentrated on avoiding virtualization detection and random interaction, which is
1145 the main reason for False Negative malware detection. Figure 11 shows the number of existing
1146 Android malware detection frameworks in each circle, which consider each evasion technique in
1147 the framework evaluations. It shows the necessity of more insights regarding evaluation against
1148 all types of evasions, as currently, available standard malware datasets cover some evasion
1149 techniques that are randomly selected during evaluation. In summary, all the above
1150 investigations demonstrate the absolute need for standard evasion benchmarking tools to
1151 evaluate the newly developed frameworks against all evasion techniques.
1152

Lessons Learned and Future Directions

1154 Android malware development is always one step ahead of malware detection techniques, which
1155 means malware detection still requires many efforts to catch up with malware development. To
1156 achieve this objective, we share several insights drawn from our analysis.
1157   *a)* Obfuscation dataset
1158 One of the most important is to keep on updating and standardizing obfuscated malware datasets.
1159 We recommend standardizing this dataset by the research community trusted institutions and
1160 being available upon validated requests for research purposes. Despite some available obfuscated
1161 datasets such as PRAGuard (Karbab & Debbabi 2021) sharing ten thousands obfuscated malware
1162 by obfuscating MalGnome and the Contagio MiniDump dataset, however the PRAGuard stopped
1163 sharing the dataset starting from April 2021.
1164   *b)* Obfuscation detection framework performance
1165 The performance of the Android malware framework degraded over time since new malware
1166 variants, and obfuscations techniques were generated PetaDroid (Karbab & Debbabi 2021).
1167 Hence, we recommend researchers extend their research to keep an eye on their framework
1168 performance over time.
1169   *c)* Metamorphism evasion:

1170 Static detection is unable to detect most of the metamorphism evasion techniques because of the
1171 dynamic characteristics of metamorphism. However, there is still a lack of dynamic and hybrid
1172 frameworks to detect metamorphism evasions. It is therefore beneficial to focus more on
1173 developing dynamic and hybrid methods.
1174   *d)* Standard Evasion Benchmarking:
1175 We suggest building a comprehensive and collaborative benchmarking framework for Android
1176 malware detection evasion techniques that aims to improve the quality of research and add to the
1177 body of knowledge in Android malware detection studies. The benchmark consists of a list of
1178 evasion techniques based on the detection methods that have been evaluated. As a result,
1179 detection methods are tested against a standardized list of malware evasion techniques to
1180 determine whether they are capable of detecting malware evasions.
1181
1182 Figure 11: Systematic Map of Accumulative Number of Detection Frameworks versus Evasions
1183 Techniques
1184
1185   *e)* Android Exploits:
1186 As mentioned earlier, Android is based on Linux OS; it has inherited Linux exploits. Recently,
1187 malware authors developed and published the Android exploit code Dirty-Cow CVE-2016-5195
1188 (2016). The Dirty Cow exploit has been existing in Linux since 2007; it affects all Android
1189 versions. Existing fixes for Linux exploits are inefficient; Android fixes are still expected from
1190 vendors like Google or Samsung. Researchers must study such exploits and recommend proper
1191 ways to fix newly discovered exploits. Additionally, researchers need to examine the Android
1192 operating system and identify potential exploits and offer solutions before attackers abuse such
1193 exploits.
1194   *f)* Code Integrity Verification:
1195 Verification means that the application integrity is authenticated against repackaging by
1196 guaranteed third-party authentication authorities. Vidas (Vidas & Christin 2013) proposed a
1197 simple mechanism that alleviates the specific problem of verifying the authenticity of an App to
1198 protect the user from repackaged apps that contain malicious code. Their approach is based on
1199 creating a simple public-key infrastructure backed by the domain name system. This area of
1200 research needs more attention compared to others. App integrity significantly increases the effort
1201 required for a successful attack. Under this new model, the attacker must either obtain the
1202 original publisher's secret signing key, control the publisher's web server, or commit a man-in-
1203 the-middle (MitM) attack on the publisher's DNS records and web server. The attacker must
1204 now conduct two successful attacks in all cases, and the secondary attack requires more effort
1205 than application repackaging. It is worth noting that code verification, and not code analysis, is
1206 recommended, as it is necessary to consider the complexity of the available applications. Code
1207 verification does not require much effort, as it involves checking the code's integrity by using the
1208 public-key infrastructure.
1209   *g)* Process Authentication:

1210   Some researchers leverage the process of model authentication to eliminate the need for an
1211   external Certification Authority (CA) that protects the system from many exploits (Almohri et al.
1212   2014). However, they are still unable to detect the payload that is downloaded to install
1213   malicious applications. For example, DroidBarrier is designed to prevent such installations by
1214   detecting their unauthenticated processes, thereby foiling this form of attack. However,
1215   DroidBarrier (Almohri et al. 2014) cannot guarantee the isolation of hijacked processes
1216   described under attacks. Therefore, it is generally advisable to monitor processes running on the
1217   device. If an unauthenticated process is launched, the process must be isolated to hinder
1218   damaging the device and analyze and detect the malicious application. This way, if a malicious
1219   application bypasses the detection barrier and downloads a malicious payload, it is caught when
1220   running an unauthenticated process to execute that payload.
1221   *h)*   Triggering Malicious Code Assurance:
1222   The process of ensuring the malicious code runs during the dynamic analysis sandboxing.
1223   TriggerScope (Fratantonio et al. 2016) statically tries to detect suspicious triggering; however, its
1224   limitation as static analysis makes it easy to be evaded by code obfuscation. Likewise, Groddroid
1225   (Abraham et al. 2015) developed a framework to launch the branches of each function to make
1226   sure that the malicious code starts. However, it fails to follow the components of background
1227   services, which misses the main activity. Groddroid is still an open issue among researchers and
1228   is known as code coverage. It is essential to address this issue by covering possible branches in
1229   the source code of the applications.
1230
1231

Conclusions

1232

1233   Global evasion techniques make Android malware more advanced and sophisticated, which was
1234   our motivation for this study. We aim to highlight the most critical weaknesses of Android
1235   malware detection frameworks, mainly when malware uses different evasions techniques.
1236   Therefore, this study scrutinizes top Android malware detection frameworks against eighteen
1237   evaluation test benches to evaluate the effectiveness of the evasions detection techniques in
1238   Android malware detection frameworks. Therefore, the study introduces a new evasion
1239   taxonomy that categorizes the evasion techniques into two main groups, polymorphism and
1240   metamorphism, where each group has branches; the polymorphism group includes package
1241   transformation, and the encryption metamorphism group contains code obfuscation, advanced
1242   transformation, and anti-emulation branches. The study also pointed out the lack of research in
1243   evaluating the malware detection against different evasion techniques; hence we scrutinized the
1244   frameworks based on every evasion technique and categorized the evaluations based on the
1245   malware detection methods. Our analysis results conclude a lack of research evaluating the
1246   current Android malware detection framework robustness against state-of-the-art evasion
1247   techniques. We also concluded that static analysis based detection is easily evaded with simple
1248   obfuscation.

1249 On the contrary, dynamic and hybrid analyses address advanced code transformation techniques
1250 and other advanced evasions. However, preliminary studies have evaluated their frameworks
1251 against evasion techniques. The missing framework evaluations are due to the lack of standard
1252 benchmark evasion datasets with updated standard malware datasets and the lack of
1253 comprehensive test benches tools to assess the efficiency of the existing and future frameworks.
1254 This study advises the research community to exert more effort into detecting anti-emulation
1255 evasion as indicated in the map of evasions and detection techniques. We also plan to create a
1256 standard evaluation framework to include all types of evasion techniques and consider the new
1257 generation of malware that combines multiple evasion techniques.
1258

1259 ## References

1260
1261
1262

1263 Aafer Y, Du WL, and Yin H. 2013. DroidAPIMiner: Mining API-Level Features for Robust
1264   Malware Detection in Android. *Security and Privacy in Communication Networks,*
1265   *Securecomm 2013* 127:86-103.
1266 Abdulla S, and Altaher A. 2015. Intelligent approach for android malware detection. *KSII*
1267   *Transactions on Internet and Information Systems* 9:2964-2983.
1268   10.3837/tiis.2015.08.012
1269 Abraham A, Andriatsimandefitra R, Brunelat A, Lalande JF, and Viet Triem Tong V. 2015.
1270   GroddDroid: A gorilla for triggering malicious behaviors. 10th International Conference
1271   on Malicious and Unwanted Software, MALWARE 2015: Institute of Electrical and
1272   Electronics Engineers Inc. p 119-127.
1273 Abuthawabeh MKA, and Mahmoud KW. 2019. Android malware detection and categorization
1274   based on conversation-level network traffic features. 2019 International Arab Conference
1275   on Information Technology (ACIT): IEEE. p 42-47.
1276 Adebayo OS, and AbdulAziz N. 2014. Android malware classification using static code analysis
1277   and Apriori algorithm improved with particle swarm optimization. Information and
1278   Communication Technologies (WICT), 2014 Fourth World Congress on. p 123-128.
1279 Afifi F, Anuar NB, Shamshirband S, and Choo K-KR. 2016. DyHAP: Dynamic Hybrid ANFIS-
1280   PSO Approach for Predicting Mobile Malware. *PLOS ONE* 11:e0162627.
1281   10.1371/journal.pone.0162627
1282 Afonso VM, de Amorim MF, Grégio ARA, Junquera GB, and de Geus PL. 2015. Identifying
1283   Android malware using dynamically obtained features. *Journal of Computer Virology and*
1284   *Hacking Techniques* 11:9-17. 10.1007/s11416-014-0226-7
1285 Akpojaro J, Aigbe P, and Onwudebelu U. 2014. Unsupervised Machine Learning Techniques for
1286   Detecting Malware Applications in Wireless Devices. *Transactions on Machine Learning*
1287   *and Artificial Intelligence* 2:20-29.
1288 Alazab M, Alazab M, Shalaginov A, Mesleh A, and Awajan A. 2020. Intelligent mobile malware
1289   detection using permission requests and api calls. *Future Generation Computer Systems*
1290   107:509-521.
1291 Allix K, Bissyandé TF, Klein J, and Le Traon Y. 2016. Androzoo: Collecting millions of android
1292   apps for the research community. 2016 IEEE/ACM 13th Working Conference on Mining
1293   Software Repositories (MSR): IEEE. p 468-471.

1294 Almohri HMJ, Yao D, and Kaiyfura D. 2014. DroidBarrier: Know what is executing on your
1295   Android. 4th ACM Conference on Data and Application Security and Privacy, CODASPY
1296   2014. San Antonio, TX: Association for Computing Machinery. p 257-264.
1297 Alzaylaee MK, Yerima SY, and Sezer S. 2016. DynaLog: An automated dynamic analysis
1298   framework for characterizing android applications. 2016 International Conference on
1299   Cyber Security and Protection of Digital Services, Cyber Security 2016: Institute of
1300   Electrical and Electronics Engineers Inc.
1301 Amamra A, Robert JM, and Talhi C. 2015. Enhancing malware detection for Android systems
1302   using a system call filtering and abstraction process. *Security and Communication*
1303   *Networks* 8:1179-1192. 10.1002/sec.1073
1304 Amos B, Turner H, and White J. 2013. Applying machine learning classifiers to dynamic Android
1305   malware detection at scale. Wireless Communications and Mobile Computing
1306   Conference (IWCMC), 2013 9th International. p 1666-1671.
1307 Andronio N, Zanero S, and Maggi F. 2015. Heldroid: Dissecting and detecting mobile
1308   ransomware. International workshop on recent advances in intrusion detection: Springer.
1309   p 382-404.
1310 Aonzo S, Georgiu GC, Verderame L, and Merlo A. 2020. Obfuscapk: An open-source black-box
1311   obfuscation tool for Android apps. *SoftwareX* 11:100403.
1312   https://doi.org/10.1016/j.softx.2020.100403
1313 Apvrille A, and Apvrille L. 2015. SherlockDroid: a research assistant to spot unknown malware
1314   in Android marketplaces. *Journal of Computer Virology and Hacking Techniques* 11:235-
1315   245. 10.1007/s11416-015-0245-z
1316 Arora A, Peddoju SK, and Conti M. 2019. Permpair: Android malware detection using
1317   permission pairs. *IEEE Transactions on Information Forensics and Security* 15:1968-
1318   1982.
1319 Arp D, Spreitzenbarth M, Hubner M, Gascon H, and Rieck K. 2015. Drebin: Efficient and
1320   explainable detection of android malware in your pocket. Proc of 17th Network and
1321   Distributed System Security Symposium, NDSS.
1322 Bacci A, Bartoli A, Martinelli F, Medvet E, Mercaldo F, and Visaggio CA. 2018. Impact of Code
1323   Obfuscation on Android Malware Detection based on Static and Dynamic Analysis.
1324   ICISSP. p 379-385.
1325 Bagheri H, Sadeghi A, Garcia J, and Malek s. 2015. COVERT: Compositional Analysis of
1326   Android Inter-App Permission Leakage. *Software Engineering, IEEE Transactions on*
1327   PP:1-1. 10.1109/TSE.2015.2419611
1328 Banescu S, Wuchner T, Salem A, Guggenmos M, and Pretschner A. 2015. A framework for
1329   empirical evaluation of malware detection resilience against behavior obfuscation. 2015
1330   10th International Conference on Malicious and Unwanted Software (MALWARE): IEEE.
1331   p 40-47.
1332 Baskaran B, and Ralescu A. 2016. A study of Android malware detection techniques and
1333   machine learning. In: Glass M, Shen J, and Phung PH, editors. 27th Modern Artificial
1334   Intelligence and Cognitive Science Conference, MAICS 2016: CEUR-WS. p 15-23.
1335 Battista P, Mercaldo F, Nardone V, Santone A, and Visaggio CA. 2016. Identification of android
1336   malware families with model checking. In: Furnell S, Camp O, and Mori P, editors. 2nd
1337   International Conference on Information Systems Security and Privacy, ICISSP 2016:
1338   SciTePress. p 542-547.
1339 Canfora G, De Lorenzo A, Medvet E, Mercaldo F, and Visaggio CA. 2015a. Effectiveness of
1340   opcode ngrams for detection of multi family android malware. Availability, Reliability and
1341   Security (ARES), 2015 10th International Conference on: IEEE. p 333-340.
1342 Canfora G, Di Sorbo A, Mercaldo F, and Visaggio CA. 2015b. Obfuscation techniques against
1343   signature-based detection: A case study. 1st Mobile Systems Technologies Workshop,
1344   MST 2015: Institute of Electrical and Electronics Engineers Inc. p 21-26.

1345 Canfora G, Medvet E, Mercaldo F, and Visaggio CA. 2016. Acquiring and analyzing app metrics
1346   for effective mobile malware detection. 2016 2nd ACM International Workshop on
1347   Security and Privacy Analytics, IWSPA 2016: Association for Computing Machinery, Inc.
1348   p 50-57.
1349 Canfora G, Mercaldo F, Moriano G, and Visaggio CA. 2015c. Composition-malware: Building
1350   android malware at run time. 10th International Conference on Availability, Reliability
1351   and Security, ARES 2015: Institute of Electrical and Electronics Engineers Inc. p 318-
1352   326.
1353 Castellanos JH, Wüchner T, Ochoa M, and Rueda S. 2016. Q-floid: Android malware detection
1354   with quantitative data flow graphs. In: Roychoudhury A, and Mathur A, editors. 1st
1355   Singapore-Cybersecurity R and D Conference, SG-CRC 2016: IOS Press. p 13-25.
1356 Chao W, Qun L, XiaoHu W, TianYu R, JiaHan D, GuangXin G, and EnJie S. 2020. An android
1357   application vulnerability mining method based on static and dynamic analysis. 2020
1358   IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC):
1359   IEEE. p 599-603.
1360 Chau M, and Reith R. 2019. Smartphone market share. *IDC Corporate USA* 444.
1361 Chaugule A, Xu Z, and Zhu S. 2011. A Specification Based Intrusion Detection Framework for
1362   Mobile Phones. In: Lopez J, and Tsudik G, editors. Applied Cryptography and Network
1363   Security: Springer Berlin Heidelberg. p 19-37.
1364 Chen K, Wang P, Lee Y, Wang X, Zhang N, Huang H, Zou W, and Liu P. 2015. Finding
1365   unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale.
1366   24th USENIX Security Symposium (USENIX Security 15). p 659-674.
1367 Chen Z, Yan Q, Han H, Wang S, Peng L, Wang L, and Yang B. 2018. Machine learning based
1368   mobile malware detection using highly imbalanced network traffic. *Information Sciences*
1369   433:346-364.
1370 Chenxiong Q, Xiapu L, Yu L, and Guofei G. 2015. VulHunter: Toward Discovering
1371   Vulnerabilities in Android Applications. *Micro, IEEE* 35:44-53. 10.1109/MM.2015.25
1372 Chin E, Felt AP, Greenwood K, and Wagner D. 2011. Analyzing inter-application communication
1373   in Android. Proceedings of the 9th international conference on Mobile systems,
1374   applications, and services: ACM. p 239-252.
1375 Cho H, Yi JH, and Ahn G. 2018. DexMonitor: Dynamically Analyzing and Monitoring Obfuscated
1376   Android Applications. *IEEE Access* 6:71229-71240. 10.1109/ACCESS.2018.2881699
1377 Choliy A, Li F, and Gao T. 2017. Obfuscating function call topography to test structural malware
1378   detection against evasion attacks. Computing, Networking and Communications (ICNC),
1379   2017 International Conference on: IEEE. p 808-813.
1380 Crussell J, Gibler C, and Chen H. 2015. AnDarwin: Scalable Detection of Android Application
1381   Clones Based on Semantics. *IEEE Transactions on Mobile Computing* 14:2007-2019.
1382   10.1109/TMC.2014.2381212
1383 David OE, and Netanyahu NS. 2015. DeepSign: Deep learning for automatic malware signature
1384   generation and classification. Neural Networks (IJCNN), 2015 International Joint
1385   Conference on. p 1-8.
1386 de la Puerta JG, Pastor-López I, Sanz B, and Bringas PG. 2019. Network traffic analysis for
1387   android malware detection. International Conference on Hybrid Artificial Intelligence
1388   Systems: Springer. p 468-479.
1389 De Lorenzo A, Martinelli F, Medvet E, Mercaldo F, and Santone A. 2020. Visualizing the
1390   outcome of dynamic analysis of Android malware with VizMal. *Journal of Information*
1391   *Security and Applications* 50:102423.
1392 Desnos A, and Lantz P. 2014. Droidbox: An android application sandbox for dynamic analysis
1393   (2011).

1394   Dhalaria M, and Gandotra E. 2021. A Hybrid Approach for Android Malware Detection and
1395   Family Classification. *International Journal of Interactive Multimedia & Artificial*
1396   *Intelligence* 6.
1397   Dharmalingam VP, and Palanisamy V. 2021. A novel permission ranking system for android
1398   malware detection—the permission grader. *Journal of Ambient Intelligence and*
1399   *Humanized Computing* 12:5071-5081.
1400   Diao W, Liu X, Li Z, and Zhang K. 2016. Evading android runtime analysis through detecting
1401   programmed interactions. 9th ACM Conference on Security and Privacy in Wireless and
1402   Mobile Networks, WiSec 2016: Association for Computing Machinery, Inc. p 159-164.
1403   Dietzel C. 2014. Porting and Improving an Android Sandbox for Automated Assessment of
1404   Malware. Hochschule Darmstadt.
1405   Du Y, Wang X, and Wang J. 2015. A static Android malicious code detection method based on
1406   multi-source fusion. *Security and Communication Networks* 8:3238-3246.
1407   10.1002/sec.1248
1408   Elish KO, Shu X, Yao D, Ryder BG, and Jiang X. 2015. Profiling user-trigger dependence for
1409   Android malware detection. *Computers and Security* 49:255-273.
1410   10.1016/j.cose.2014.11.001
1411   Enck W, Gilbert P, Chun BG, Cox LP, Jung J, McDaniel P, and Sheth AN. 2014a. TaintDroid:
1412   An Information Flow Tracking System for Real-Time Privacy Monitoring on
1413   Smartphones. *Communications of the Acm* 57:99-106. 10.1145/2494522
1414   Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, Mcdaniel P, and Sheth AN.
1415   2014b. TaintDroid: An Information-Flow Tracking System for Realtime Privacy
1416   Monitoring on Smartphones. *Acm Transactions on Computer Systems* 32. Artn 5
1417   10.1145/2619091
1418   Falsina L, Fratantonio Y, Zanero S, Kruegel C, Vigna G, and Maggi F. 2015. Grab'n Run:
1419   Secure and Practical Dynamic Code Loading for Android Applications. Proceedings of
1420   the 31st Annual Computer Security Applications Conference: ACM. p 201-210.
1421   Fang Z, Han W, and Li Y. 2014. Permission based Android security: Issues and
1422   countermeasures. *Computers & Security* 43:205-218. 10.1016/j.cose.2014.02.007
1423   Faruki P, Bhandari S, Laxmi V, Gaur M, and Conti M. 2015a. Droidanalyst: Synergic app
1424   framework for static and dynamic app analysis. Studies in Computational Intelligence:
1425   Springer Verlag. p 519-552.
1426   Faruki P, Bharmal A, Laxmi V, Ganmoor V, Gaur MS, Conti M, and Rajarajan M. 2015b.
1427   Android security: A survey of issues, malware penetration, and defenses. *IEEE*
1428   *Communications Surveys and Tutorials* 17:998-1022. 10.1109/COMST.2014.2386139
1429   Faruki P, Bharmal A, Laxmi V, Gaur M, Conti M, and Rajarajan M. 2014. Evaluation of Android
1430   Anti-malware Techniques against Dalvik Bytecode Obfuscation. Trust, Security and
1431   Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International
1432   Conference on: IEEE. p 414-421.
1433   Faruki P, Bharmal A, Laxmi V, Gaur MS, Conti M, and Rajarajan M. 2015c. Evaluation of
1434   android anti-malware techniques against dalvik bytecode obfuscation. 13th IEEE
1435   International Conference on Trust, Security and Privacy in Computing and
1436   Communications, TrustCom 2014: Institute of Electrical and Electronics Engineers Inc. p
1437   414-421.
1438   Faruki P, Laxmi V, Bharmal A, Gaur MS, and Ganmoor V. 2015d. AndroSimilar: Robust
1439   signature for detecting variants of Android malware. *Journal of Information Security and*
1440   *Applications* 22:66-80. 10.1016/j.jisa.2014.10.011
1441   Faruki P, Laxmi V, Ganmoor V, Gaur MS, and Bharmal A. 2013. DroidOLytics: Robust feature
1442   signature for repackaged android apps on official and third party android markets. 2nd
1443   International Conference on Advanced Computing, Networking and Security, ADCONS
1444   2013. Surathkal, Karnataka: IEEE Computer Society. p 247-252.

1445   Feizollah A, Anuar NB, Salleh R, Suarez-Tangil G, and Furnell S. 2017. AndroDialysis: Analysis
1446   of Android Intent Effectiveness in Malware Detection. *Computers & Security* 65:121-134.
1447   Feizollah A, Anuar NB, Salleh R, and Wahab AWA. 2015. A review on feature selection in
1448   mobile malware detection. *Digital Investigation* 13:22-37. 10.1016/j.diin.2015.02.001
1449   Feng J, Shen L, Chen Z, Wang Y, and Li H. 2020. A two-layer deep learning method for android
1450   malware detection using network traffic. *IEEE Access* 8:125786-125796.
1451   Feng P, Ma J, Sun C, Xu X, and Ma Y. 2018. A novel dynamic Android malware detection
1452   system with ensemble learning. *IEEE Access* 6:30996-31011.
1453   Foremost J. 2012. DroidDream mobile malware. *Available at*
1454   *https://www.virusbulletin.com/virusbulletin/2012/03/droiddream-mobile-malware*
1455   (accessed 14-8-2018 2018).
1456   Fratantonio Y, Bianchi A, Robertson W, Kirda E, Kruegel C, and Vigna G. 2016. TriggerScope:
1457   Towards Detecting Logic Bombs in Android Applications. 2016 IEEE Symposium on
1458   Security and Privacy, SP 2016: Institute of Electrical and Electronics Engineers Inc. p
1459   377-396.
1460   Fsecure. 2013. Mobile Threat Report Q1 2013. F-Secure: , (Online; Last Accessed Oct 15)]
1461   Gajrani J, Sarswat J, Tripathi M, Laxmi V, Gaur M, and Conti M. 2015. A robust dynamic
1462   analysis system preventing SandBox detection by Android malware. Proceedings of the
1463   8th International Conference on Security of Information and Networks: ACM. p 290-295.
1464   Gascon H, Yamaguchi F, Arp D, and Rieck K. 2013. Structural detection of Android malware
1465   using embedded call graphs. 2013 6th Annual ACM Workshop on Artificial Intelligence
1466   and Security, AISec 2013, Co-located with the 20th ACM Conference on Computer and
1467   Communications Security, CCS 2013. Berlin. p 45-54.
1468   Glodek W, and Harang R. 2013. Rapid permissions-based detection and analysis of mobile
1469   malware using random decision forests. 2013 IEEE Military Communications
1470   Conference, MILCOM 2013. San Diego, CA. p 980-985.
1471   Gonzalez H, Kadir AA, Stakhanova N, Alzahrani N, and Ghorbani AA. 2015. Exploring reverse
1472   engineering symptoms in android apps. 8th European Workshop on System Security,
1473   EuroSec 2015: Association for Computing Machinery, Inc.
1474   Gonzalez H, Stakhanova N, and Ghorbani AA. 2014. Droidkin: Lightweight detection of android
1475   apps similarity. International Conference on Security and Privacy in Communication
1476   Systems: Springer. p 436-453.
1477   Google. VirusTotal https://virustotal.com/ (accessed 1/6/2016.
1478   Grace M, Zhou Y, Zhang Q, Zou S, and Jiang X. 2012. Riskranker: scalable and accurate zero-
1479   day android malware detection. Proceedings of the 10th international conference on
1480   Mobile systems, applications, and services: ACM. p 281-294.
1481   GuardSquare. 2014. DexGuard.
1482   Gurulian I, Markantonakis K, Cavalaro L, and Mayes K. 2016. You can't touch this: Consumer-
1483   centric android application repackaging detection. *Future Generation Computer Systems*
1484   65:1-9. 10.1016/j.future.2016.05.021
1485   Hanna S, Huang L, Wu E, Li S, Chen C, and Song D. 2013. Juxtapp: A scalable system for
1486   detecting code reuse among android applications. *Detection of Intrusions and Malware,*
1487   *and Vulnerability Assessment*: Springer, 62-81.
1488   Hoffmann J, Rytilahti T, Maiorca D, Winandy M, Giacinto G, and Holz T. 2016. Evaluating
1489   analysis tools for android apps: Status Quo and robustness against obfuscation. 6th
1490   ACM Conference on Data and Application Security and Privacy, CODASPY 2016:
1491   Association for Computing Machinery, Inc. p 139-141.
1492   Holla S, and Katti MM. 2012. Android based mobile application development and its security.
1493   *International Journal of Computer Trends and Technology* 3:486-490.
1494   Hsieh WC, Wu CC, and Kao YW. 2016. A study of android malware detection technology
1495   evolution. 49th Annual IEEE International Carnahan Conference on Security

1496   Technology, ICCST 2015: Institute of Electrical and Electronics Engineers Inc. p 135-
1497   140.
1498 Hu W, and Xiao Z. 2014. Guess where i am-android: detection and prevention of emulator
1499   evading on android. XFocus Information Security Conference (XCon): HitCon.
1500 Jang JW, Kang H, Woo J, Mohaisen A, and Kim HK. 2016. Andro-Dumpsys: Anti-malware
1501   system based on the similarity of malware creator and malware centric information.
1502   *Computers & Security* 58:125-138. 10.1016/j.cose.2015.12.005
1503 Jeong J, Seo D, Lee C, Kwon J, Lee H, and Milburn J. 2014. MysteryChecker: Unpredictable
1504   attestation to detect repackaged malicious applications in Android. 9th IEEE
1505   International Conference on Malicious and Unwanted Software, MALCON 2014: Institute
1506   of Electrical and Electronics Engineers Inc. p 50-57.
1507 Jing Y, Zhao Z, Ahn G-J, and Hu H. 2014. Morpheus: automatically generating heuristics to
1508   detect Android emulators. Proceedings of the 30th Annual Computer Security
1509   Applications Conference: ACM. p 216-225.
1510 Jung J, Kim H, Shin D, Lee M, Lee H, Cho S-j, and Suh K. 2018. Android malware detection
1511   based on useful API calls and machine learning. 2018 IEEE First International
1512   Conference on Artificial Intelligence and Knowledge Engineering (AIKE): IEEE. p 175-
1513   178.
1514 Jusoh R, Firdaus A, Anwar S, Osman MZ, Darmawan MF, and Ab Razak MF. 2021. Malware
1515   detection using static analysis in Android: a review of FeCO (features, classification, and
1516   obfuscation). *PeerJ Computer Science*.
1517 Kaelbling LP, Littman ML, and Moore AW. 1996. Reinforcement learning: A survey. *Journal of
1518   artificial intelligence research* 4:237-285.
1519 Kandukuru S, and Sharma R. 2018. PNSDroid: A Hybrid Approach for Detection of Android
1520   Malware. *Recent Findings in Intelligent Computing Techniques*: Springer, 361-367.
1521 Kang H, Jang JW, Mohaisen A, and Kim HK. 2015. Detecting and classifying android malware
1522   using static analysis along with creator information. *International Journal of Distributed
1523   Sensor Networks* 2015. 10.1155/2015/479174
1524 Karbab EB, and Debbabi M. 2021. Resilient and Adaptive Framework for Large Scale Android
1525   Malware Fingerprinting using Deep Learning and NLP Techniques. *arXiv preprint
1526   arXiv:210513491*.
1527 Karim A, Shah SAA, Bin Salleh R, Arif M, Noor RM, and Shamshirband S. 2015. Mobile Botnet
1528   Attacks - an Emerging Threat: Classification, Review and Open Issues. *Ksii
1529   Transactions on Internet and Information Systems* 9:1471-1492.
1530   10.3837/tiis.2015.04.012
1531 Kim D, Kwak J, and Ryou J. 2015. DWroidDump: Executable Code Extraction from Android
1532   Applications for Malware Analysis. *International Journal of Distributed Sensor Networks*
1533   2015. 10.1155/2015/379682
1534 Kiss N, Lalande J-F, Leslous M, and Tong VVT. 2016. Kharon dataset: Android malware under
1535   a microscope. The {LASER} Workshop: Learning from Authoritative Security Experiment
1536   Results ({LASER} 2016). p 1-12.
1537 Klassmaster. 2013. Zelix Klassmaster [online] Available: http://www.zelix.com/klassmaster/.
1538 Kohout J, and Pevny T. 2015. Unsupervised detection of malware in persistent web traffic.
1539   Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International
1540   Conference on. p 1757-1761.
1541 Kuhnel M, Smieschek M, and Meyer U. 2015. Fast identification of obfuscation and mobile
1542   advertising in mobile malware. 14th IEEE International Conference on Trust, Security
1543   and Privacy in Computing and Communications, TrustCom 2015: Institute of Electrical
1544   and Electronics Engineers Inc. p 214-221.

1545 Kumawat A, Sharma AK, and Kumawat S. 2017. Identification of Cryptographic Vulnerability
1546   and Malware Detection in Android. *International Journal of Information Security &*
1547   *Privacy* 11:15.
1548 Kurniawan H, Rosmansyah Y, and Dabarsyah B. 2015. Android anomaly detection system
1549   using machine learning classification. 5th International Conference on Electrical
1550   Engineering and Informatics, ICEEI 2015: Institute of Electrical and Electronics
1551   Engineers Inc. p 288-293.
1552 Kwon J, Jeong J, Lee J, and Lee H. 2014. DroidGraph: Discovering Android malware by
1553   analyzing semantic behavior. 2014 IEEE Conference on Communications and Network
1554   Security, CNS 2014: Institute of Electrical and Electronics Engineers Inc. p 498-499.
1555 Lafortune E. 2002. ProGuard.
1556 Lashkari AH, Kadir AFA, Gonzalez H, Mbah KF, and Ghorbani AA. 2017. Towards a network-
1557   based framework for android malware detection and characterization. 2017 15th Annual
1558   Conference on Privacy, Security and Trust (PST): IEEE. p 233-23309.
1559 Lee C, Kim J, Cho S-j, Choi J, and Park Y. 2014. Unified security enhancement framework for
1560   the Android operating system. *Journal of Supercomputing* 67:738-756. 10.1007/s11227-
1561   013-0991-y
1562 Lei C, Gates CS, Luo S, and Ninghui L. 2015. A Probabilistic Discriminative Model for Android
1563   Malware Detection with Decompiled Source Code. *Dependable and Secure Computing,*
1564   *IEEE Transactions on* 12:400-412. 10.1109/TDSC.2014.2355839
1565 Li J, Sun L, Yan Q, Li Z, Srisa-An W, and Ye H. 2018a. Significant permission identification for
1566   machine-learning-based android malware detection. *IEEE Transactions on Industrial*
1567   *Informatics* 14:3216-3225.
1568 Li J, Sun L, Yan Q, Li Z, Srisa-an W, and Ye H. 2018b. Significant Permission Identification for
1569   Machine Learning Based Android Malware Detection. *IEEE Transactions on Industrial*
1570   *Informatics*.
1571 Li L, Bissyandé TF, Octeau D, and Klein J. 2016. DroidRA: Taming reflection to support whole-
1572   program analysis of android apps. In: Roychoudhury A, and Zeller A, editors. 25th
1573   International Symposium on Software Testing and Analysis, ISSTA 2016: Association for
1574   Computing Machinery, Inc. p 318-329.
1575 Li Y, Jang J, Hu X, and Ou X. 2017. Android malware clustering through malicious payload
1576   mining. International Symposium on Research in Attacks, Intrusions, and Defenses:
1577   Springer. p 192-214.
1578 Liang S, and Bracha G. 1998. Dynamic class loading in the Java virtual machine. *Acm sigplan*
1579   *notices* 33:36-44.
1580 Lim K, Jeong Y, Cho S-j, Park M, and Han S. 2016. An Android Application Protection Scheme
1581   against Dynamic Reverse Engineering Attacks. *Journal of Wireless Mobile Networks,*
1582   *Ubiquitous Computing, and Dependable Applications*.
1583 Lindorfer M, Neugschwandtner M, and Platzer C. 2015. MARVIN: Efficient and Comprehensive
1584   Mobile App Classification through Static and Dynamic Analysis. In: Huang G, Yang J,
1585   Ahamed SI, Hsiung PA, Chang CK, Chu W, and Crnkovic I, editors. 39th IEEE Annual
1586   Computer Software and Applications Conference, COMPSAC 2015: IEEE Computer
1587   Society. p 422-433.
1588 Liu Z, Wang R, Japkowicz N, Tang D, Zhang W, and Zhao J. 2021. Research on unsupervised
1589   feature learning for Android malware detection based on Restricted Boltzmann
1590   Machines. *Future Generation Computer Systems* 120:91-108.
1591 Lopez CCU, and Cadavid AN. 2016. Machine learning classifiers for android malware analysis.
1592   In: Garcia L, editor. 2016 IEEE Colombian Conference on Communications and
1593   Computing, COLCOM 2016: Institute of Electrical and Electronics Engineers Inc.
1594 Lu T, Du Y, Ouyang L, Chen Q, and Wang X. 2020. Android malware detection based on a
1595   hybrid deep learning model. *Security and Communication Networks* 2020.

1596   Luyi X, Xiaorui P, Rui W, Kan Y, and XiaoFeng W. 2014. Upgrading Your Android, Elevating My
1597   Malware: Privilege Escalation through Mobile OS Updating. Security and Privacy (SP),
1598   2014 IEEE Symposium on. p 393-408.
1599   Ma Z, Ge H, Wang Z, Liu Y, and Liu X. 2020. Droidetec: Android Malware Detection and
1600   Malicious Code Localization through Deep Learning. *arXiv preprint arXiv:200203594*.
1601   Mahindru A, and Sangal A. 2021. MLDroid—Framework for Android malware detection using
1602   machine learning techniques. *Neural Computing and Applications* 33:5183-5240.
1603   Maier D, Muller T, and Protsenko M. 2014. Divide-and-conquer: Why android malware cannot
1604   be stopped. 9th International Conference on Availability, Reliability and Security, ARES
1605   2014: Institute of Electrical and Electronics Engineers Inc. p 30-39.
1606   Maier D, Protsenko M, and Müller T. 2015. A game of Droid and Mouse: The threat of split-
1607   personality malware on Android. *Computers and Security* 54:2-15.
1608   10.1016/j.cose.2015.05.001
1609   Maiorca D, Ariu D, Corona I, Aresu M, and Giacinto G. 2015. Stealth attacks: An extended
1610   insight into the obfuscation effects on Android malware. *Computers and Security* 51:16-
1611   31. 10.1016/j.cose.2015.02.007
1612   Maiorca D, Mercaldo F, Giacinto G, Visaggio CA, and Martinelli F. 2017. R-PackDroid: API
1613   package-based characterization and detection of mobile ransomware. Proceedings of
1614   the symposium on applied computing: ACM. p 1718-1723.
1615   Martín A, Menéndez HD, and Camacho D. 2016. MOCDroid: multi-objective evolutionary
1616   classifier for Android malware detection. *Soft Computing*:1-11. 10.1007/s00500-016-
1617   2283-y
1618   Meng G, Xue Y, Mahinthan C, Narayanan A, Liu Y, Zhang J, and Chen T. 2016. Mystique:
1619   Evolving android malware for auditing anti-malware tools. 11th ACM Asia Conference on
1620   Computer and Communications Security, ASIA CCS 2016: Association for Computing
1621   Machinery, Inc. p 365-376.
1622   Mirzaei O, de Fuentes JM, Tapiador J, and Gonzalez-Manzano L. 2019. AndrODet: An adaptive
1623   Android obfuscation detector. *Future Generation Computer Systems* 90:240-261.
1624   https://doi.org/10.1016/j.future.2018.07.066
1625   Mutti S, Fratantonio Y, Bianchi A, Invernizzi L, Corbetta J, Kirat D, Kruegel C, and Vigna G.
1626   2015. Baredroid: Large-Scale analysis of android apps on real devices. 31st Annual
1627   Computer Security Applications Conference, ACSAC 2015: Association for Computing
1628   Machinery. p 71-80.
1629   Narayanan A, Chen L, and Chan CK. 2014. AdDetect: Automated detection of Android ad
1630   libraries using semantic analysis. 9th IEEE International Conference on Intelligent
1631   Sensors, Sensor Networks and Information Processing, IEEE ISSNIP 2014. Singapore:
1632   IEEE Computer Society.
1633   Nawaz A. 2021. Feature Engineering based on Hybrid Features for Malware Detection over
1634   Android Framework. *Turkish Journal of Computer and Mathematics Education
1635   (TURCOMAT)* 12:2856-2864.
1636   Niazi RH, Shamsi JA, Waseem T, and Khan MM. 2015. Signature-based detection of privilege-
1637   escalation attacks on Android. Conference on Information Assurance and Cyber
1638   Security, CIACS 2015: Institute of Electrical and Electronics Engineers Inc. p 44-49.
1639   Nissim N, Moskovitch R, BarAd O, Rokach L, and Elovici Y. 2016. ALDROID: efficient update of
1640   Android anti-virus software using designated active learning methods. *Knowledge and
1641   Information Systems*:1-39. 10.1007/s10115-016-0918-z
1642   Oester P. 2016. Dirty Cow (CVE-2016-5195)
1643   Pang Y, Chen Z, Li X, Wang S, Zhao C, Wang L, Ji K, and Li Z. 2017. Finding Android malware
1644   trace from highly imbalanced network traffic. 2017 IEEE International Conference on
1645   Computational Science and Engineering (CSE) and IEEE International Conference on
1646   Embedded and Ubiquitous Computing (EUC): IEEE. p 588-595.

1647   Parkour M. 2013. Contagio Mobile. Mobile malware mini dump
1648   http://contagiodump.blogspot.my/2011/03/take-sample-leave-sample-mobile-
1649   malware.html.
1650   Pektas A, and Acarman T. 2014. A dynamic malware analyzer against virtual machine aware
1651   malicious software. *Security and Communication Networks* 7:2245-2257.
1652   10.1002/sec.931
1653   Pektaş A, and Acarman T. 2020. Deep learning for effective Android malware detection using
1654   API call graph embeddings. *Soft Computing* 24:1027-1043.
1655   Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, and Ioannidis S. 2014. Rage
1656   against the virtual machine: Hindering dynamic analysis of android malware. 7th
1657   European Workshop on System Security, EuroSec 2014. Amsterdam: Association for
1658   Computing Machinery.
1659   Poeplau S, Fratantonio Y, Bianchi A, Kruegel C, and Vigna G. 2014. Execute this! analyzing
1660   unsafe and malicious dynamic code loading in android applications. Proceedings of the
1661   20th Annual Network & Distributed System Security Symposium (NDSS).
1662   Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
1663   systematization of knowledge and unified methodology. *Journal of Computer Virology*
1664   *and Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2
1665   Protsenko M, and Muller T. 2013. PANDORA applies non-deterministic obfuscation randomly to
1666   Android. 2013 8th International Conference on Malicious and Unwanted Software: "The
1667   Americas", MALWARE 2013. Fajardo, PR: IEEE Computer Society. p 59-67.
1668   Rahman M, Rahman M, Carbunar B, and Chau DH. 2016. Fairplay: Fraud and malware
1669   detection in google play. Proceedings of the 2016 SIAM International Conference on
1670   Data Mining: SIAM. p 99-107.
1671   Rashidi B, Fung C, and Tam V. 2015. Dude, ask the experts!: Android resource access
1672   permission recommendation with RecDroid. Integrated Network Management (IM), 2015
1673   IFIP/IEEE International Symposium on. p 296-304.
1674   Rastogi V, Chen Y, and Jiang X. 2013. DroidChameleon: Evaluating Android anti-malware
1675   against transformation attacks. 8th ACM SIGSAC Symposium on Information, Computer
1676   and Communications Security, ASIA CCS 2013. Hangzhou. p 329-334.
1677   Rastogi V, Chen Y, and Jiang X. 2014. Catch me if you can: Evaluating android anti-malware
1678   against transformation attacks. *IEEE Transactions on Information Forensics and*
1679   *Security* 9:99-108. 10.1109/TIFS.2013.2290431
1680   Şahin DÖ, Kural OE, Akleylek S, and Kılıç E. 2021. A novel permission-based Android malware
1681   detection system using feature selection based on linear regression. *Neural Computing*
1682   *and Applications*:1-16.
1683   Salva S, and Zafimiharisoa SR. 2015. APSET, an Android aPplication SEcurity Testing tool for
1684   detecting intent-based vulnerabilities. *International Journal on Software Tools for*
1685   *Technology Transfer* 17:201-221. 10.1007/s10009-014-0303-8
1686   Shalaginov A, and Franke K. 2014. Automatic rule-mining for malware detection employing
1687   Neuro-Fuzzy Approach. Norsk informasjonssikkerhetskonferanse (NISK).
1688   Shao Y, Luo X, Qian C, Zhu P, and Zhang L. 2014. Towards a scalable resource-driven
1689   approach for detecting repackaged android applications. 30th Annual Computer Security
1690   Applications Conference, ACSAC 2014. December ed: Association for Computing
1691   Machinery. p 56-65.
1692   Sharma A, and Sahay SK. 2014. Evolution and detection of polymorphic and metamorphic
1693   malwares: A survey. *arXiv preprint arXiv:14067061*.
1694   Sheen S, Anitha R, and Natarajan V. 2015. Android based malware detection using a
1695   multifeature collaborative decision fusion approach. *Neurocomputing* 151:905-912.
1696   10.1016/j.neucom.2014.10.004

1697 Shen T, Zhongyang Y, Xin Z, Mao B, and Huang H. 2015. Detect android malware variants
1698   using component based topology graph. 13th IEEE International Conference on Trust,
1699   Security and Privacy in Computing and Communications, TrustCom 2014: Institute of
1700   Electrical and Electronics Engineers Inc. p 406-413.
1701 Shrestha B, Ma D, Zhu Y, Li H, and Saxena N. 2015. Tap-Wave-Rub: Lightweight Human
1702   Interaction Approach to Curb Emerging Smartphone Malware. *Information Forensics and*
1703   *Security, IEEE Transactions on* PP:1-1. 10.1109/TIFS.2015.2436364
1704 Sihag V, Vardhan M, and Singh P. 2021a. A survey of android application and malware
1705   hardening. *Computer Science Review* 39:100365.
1706 Sihag V, Vardhan M, Singh P, Choudhary G, and Son S. 2021b. De-LADY: Deep learning
1707   based Android malware detection using Dynamic features. *Journal of Internet Services*
1708   *and Information Security (JISIS)* 11:34-45.
1709 Singh S, Mishra B, and Singh S. 2015. Detecting intelligent malware on dynamic Android
1710   analysis environments. 10th International Conference for Internet Technology and
1711   Secured Transactions, ICITST 2015: Institute of Electrical and Electronics Engineers Inc.
1712   p 414-419.
1713 Soh C, Tan HBK, Arnatovich YL, and Wang L. 2015. Detecting Clones in Android Applications
1714   through Analyzing User Interfaces. 23rd IEEE International Conference on Program
1715   Comprehension, ICPC 2015: IEEE Computer Society. p 163-173.
1716 Spreitzenbarth M, Schreck T, Echtler F, Arp D, and Hoffmann J. 2015. Mobile-Sandbox:
1717   combining static and dynamic analysis with machine-learning techniques. *International*
1718   *Journal of Information Security* 14:141-153. 10.1007/s10207-014-0250-0
1719 Spreitzer R, Griesmayr S, Korak T, and Mangard S. 2016. Exploiting data-usage statistics for
1720   website fingerprinting attacks on android. 9th ACM Conference on Security and Privacy
1721   in Wireless and Mobile Networks, WiSec 2016: Association for Computing Machinery,
1722   Inc. p 49-60.
1723 Statista. 2016. Number of available applications in the Google Play Store from December 2009
1724   to Sep 2016. https://wwwstatistacom/statistics/266210/number-of-available-applications-
1725   in-the-google-play-store/.
1726 Statista. 2021. Number of apps available in leading app stores as of first quarter of 2021.
1727   *Available at https://www.statista.com/statistics/276623/number-of-apps-available-in-*
1728   *leading-app-stores/* (accessed 2-July-2021 2021).
1729 Sufatrio, Chua TW, Tan DJJ, and Thing VLL. 2015a. Accurate specification for robust detection
1730   of malicious behavior in mobile environments. In: Weippl E, Pernul G, and Ryan PYA,
1731   editors. 20th European Symposium on Research in Computer Security, ESORICS 2015:
1732   Springer Verlag. p 355-375.
1733 Sufatrio, Tan DJJ, Chua T-W, and Thing VLL. 2015b. Securing Android: A Survey, Taxonomy,
1734   and Challenges. *ACM Comput Surv* 47:1-45. 10.1145/2733306
1735 Sun M, Li M, and Lui JCS. 2015. DroidEagle: Seamless detection of visually similar android
1736   apps. 8th ACM Conference on Security and Privacy in Wireless and Mobile Networks,
1737   WiSec 2015: Association for Computing Machinery, Inc.
1738 Surendran R, Thomas T, and Emmanuel S. 2020. A TAN based hybrid model for android
1739   malware detection. *Journal of Information Security and Applications* 54:102483.
1740 Talha KA, Alper DI, and Aydin C. 2015. APK Auditor: Permission-based Android malware
1741   detection system. *Digital Investigation* 13:1-14. 10.1016/j.diin.2015.01.001
1742 Tan DJ, Chua T-W, and Thing VL. 2015. Securing android: a survey, taxonomy, and challenges.
1743   *ACM Computing Surveys (CSUR)* 47:58.
1744 Tang A, Sethumadhavan S, and Stolfo SJ. 2014. Unsupervised anomaly-based malware
1745   detection using hardware features. *Research in Attacks, Intrusions and Defenses*:
1746   Springer, 109-129.

1747 Tao F, Ziyi L, Kyeong-An K, Weidong S, Carbunar B, Yifei J, and Nguyen N. 2012. Continuous
1748   mobile authentication using touchscreen gestures. Homeland Security (HST), 2012 IEEE
1749   Conference on Technologies for. p 451-456.
1750 Tchakounté F, Ngassi RCN, Kamla VC, and Udagepola KP. 2021. LimonDroid: A system
1751   coupling three signature-based schemes for profiling Android malware. *Iran Journal of*
1752   *Computer Science* 4:95-114.
1753 Technologies CPS. 2015. CuckooDroid (accessed 18/12/2016 2016).
1754 Tiwari SR, and Shukla RU. 2018. An android malware detection technique based on optimized
1755   permissions and API. 2018 International Conference on Inventive Research in
1756   Computing Applications (ICIRCA): IEEE. p 258-263.
1757 Vidas T, and Christin N. 2013. Sweetening android lemon markets: measuring and combating
1758   malware in application marketplaces. Proceedings of the third ACM conference on Data
1759   and application security and privacy: ACM. p 197-208.
1760 Vidas T, and Christin N. 2014. Evading android runtime analysis via sandbox detection.
1761   Proceedings of the 9th ACM symposium on Information, computer and communications
1762   security: ACM. p 447-458.
1763 Vidas T, Tan J, Nahata J, Tan CL, Christin N, and Tague P. 2014. A5: Automated Analysis of
1764   Adversarial Android Applications. Proceedings of the 4th ACM Workshop on Security
1765   and Privacy in Smartphones & Mobile Devices. Scottsdale, Arizona, USA: ACM. p
1766   39-50.
1767 Wang C, and Shieh SW. 2015. DROIT: Dynamic alternation of dual-level tainting for malware
1768   analysis. *Journal of Information Science and Engineering* 31:111-129.
1769 Wang S, Chen Z, Yan Q, Yang B, Peng L, and Jia Z. 2019. A mobile malware detection method
1770   using behavior features in network traffic. *Journal of Network and Computer Applications*
1771   133:15-25.
1772 Wang Z, Li C, Guan Y, and Xue Y. 2016. Anti-obfuscation method for detecting similarity of
1773   Android application. *Huazhong Keji Daxue Xuebao (Ziran Kexue Ban)/Journal of*
1774   *Huazhong University of Science and Technology (Natural Science Edition)* 44:60-64 and
1775   76. 10.13245/j.hust.160312
1776 Wang Z, and Wu F. 2015. Android malware analytic method based on improved multi-level
1777   signature matching. 5th International Conference on Information Science and
1778   Technology, ICIST 2015: Institute of Electrical and Electronics Engineers Inc. p 93-98.
1779 Wei TE, Tyan HR, Jeng AB, Lee HM, Liao HYM, and Wang JC. 2015. DroidExec: Root exploit
1780   malware recognition against wide variability via folding redundant function-relation
1781   graph. 17th IEEE International Conference on Advanced Communications Technology,
1782   ICACT 2015: Institute of Electrical and Electronics Engineers Inc. p 161-169.
1783 Wu D-J, Mao C-H, Wei T-E, Lee H-M, and Wu K-P. 2012. Droidmat: Android malware detection
1784   through manifest and api calls tracing. Information Security (Asia JCIS), 2012 Seventh
1785   Asia Joint Conference on: IEEE. p 62-69.
1786 Wu S, Wang P, Li X, and Zhang Y. 2016. Effective Detection of Android Malware Based on the
1787   Usage of Data Flow APIs and Machine Learning. *Information and Software Technology*.
1788   10.1016/j.infsof.2016.03.004
1789 Wu X, Zhang D, Su X, and Li W. 2015. Detect repackaged Android application based on HTTP
1790   traffic similarity. *Security and Communication Networks* 8:2257-2266. 10.1002/sec.1170
1791 Wüchner T, Ochoa M, and Pretschner A. 2015. Robust and effective malware detection through
1792   quantitative data flow graph metrics. International Conference on Detection of Intrusions
1793   and Malware, and Vulnerability Assessment: Springer. p 98-118.
1794 Xu M, Song C, Ji Y, Shih MW, Lu K, Zheng C, Duan R, Jang Y, Lee B, Qian C, Lee S, and Kim
1795   T. 2016. Toward engineering a secure android ecosystem: A survey of existing
1796   techniques. *ACM Computing Surveys* 49. 10.1145/2963145

1797 Xue Y, Meng G, Liu Y, Tan TH, Chen H, Sun J, and Zhang J. 2017. Auditing anti-malware tools
1798   by evolving android malware and dynamic loading technique. *IEEE Transactions on
1799   Information Forensics and Security* 12:1529-1544.
1800 Yajin Z, and Xuxian J. 2012. Dissecting Android Malware: Characterization and Evolution.
1801   Security and Privacy (SP), 2012 IEEE Symposium on. p 95-109.
1802 Yerima SY, Sezer S, and Muttik I. 2014. Android Malware Detection Using Parallel Machine
1803   Learning Classifiers. Next Generation Mobile Apps, Services and Technologies
1804   (NGMAST), 2014 Eighth International Conference on. p 37-42.
1805 You I, and Yim K. 2010. Malware obfuscation techniques: A brief survey. Broadband, Wireless
1806   Computing, Communication and Applications (BWCCA), 2010 International Conference
1807   on: IEEE. p 297-300.
1808 Yuan Z, Lu Y, and Xue Y. 2016. Droiddetector: android malware characterization and detection
1809   using deep learning. *Tsinghua Science and Technology* 21:114-123.
1810   10.1109/TST.2016.7399288
1811 Zhang F, Huang H, Zhu S, Wu D, and Liu P. 2014. ViewDroid: Towards obfuscation-resilient
1812   mobile application repackaging detection. 7th ACM Conference on Security and Privacy
1813   in Wireless and Mobile Networks, WiSec 2014. Oxford: Association for Computing
1814   Machinery. p 25-36.
1815 Zhang H, She D, and Qian Z. 2015a. Android root and its providers: A double-edged sword.
1816   22nd ACM SIGSAC Conference on Computer and Communications Security, CCS
1817   2015: Association for Computing Machinery. p 1093-1104.
1818 Zhang W, Wang H, He H, and Liu P. 2020. DAMBA: Detecting Android Malware by ORGB
1819   Analysis. *IEEE Transactions on Reliability* 69:55-69.
1820 Zhang X, Breitinger F, and Baggili I. 2016. Rapid Android Parser for Investigating DEX files
1821   (RAPID). *Digital Investigation* 17:28-39. 10.1016/j.diin.2016.03.002
1822 Zhang Y, Luo X, and Yin H. 2015b. Dexhunter: toward extracting hidden code from packed
1823   android applications. *Computer Security--ESORICS 2015*: Springer, 293-311.
1824 Zhao S, Li X, Xu G, Zhang L, and Feng Z. 2014. Attack tree based android malware detection
1825   with hybrid analysis. 13th IEEE International Conference on Trust, Security and Privacy
1826   in Computing and Communications, TrustCom 2014: Institute of Electrical and
1827   Electronics Engineers Inc. p 380-387.
1828 Zhauniarovich Y, Ahmad M, Gadyatskaya O, Crispo B, and Massacci F. 2015. StaDynA:
1829   Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android
1830   Applications. Proceedings of the 5th ACM Conference on Data and Application Security
1831   and Privacy. San Antonio, Texas, USA: ACM. p 37-48.
1832 Zheng M, Lee PP, and Lui JC. 2013a. ADAM: an automatic and extensible platform to stress
1833   test android anti-virus systems. *Detection of Intrusions and Malware, and Vulnerability
1834   Assessment*: Springer, 82-101.
1835 Zheng M, Sun M, and Lui JCS. 2013b. Droid analytics: A signature based analytic system to
1836   collect, extract, analyze and associate android malware. 12th IEEE International
1837   Conference on Trust, Security and Privacy in Computing and Communications,
1838   TrustCom 2013. Melbourne, VIC. p 163-171.
1839 Zhu H, Wang L, Zhong S, Li Y, and Sheng VS. 2021. A Hybrid Deep Network Framework for
1840   Android Malware Detection. *IEEE Transactions on Knowledge and Data Engineering*.
1841 Zou D, Wu Y, Yang S, Chauhan A, Yang W, Zhong J, Dou S, and Jin H. 2021. IntDroid: Android
1842   Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software
1843   Engineering and Methodology (TOSEM)* 30:1-32.
1844

Figure 1

The Review Process Flow Diagram

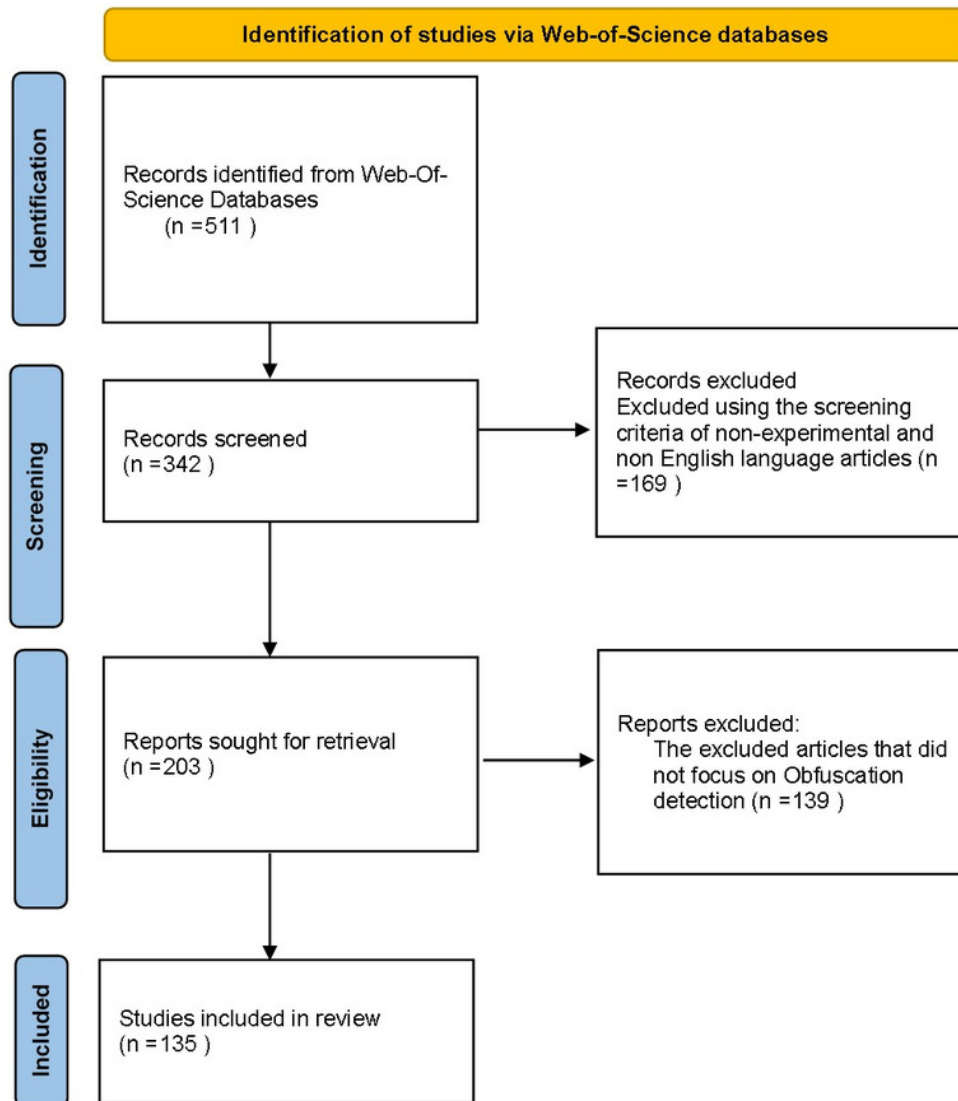



Figure 2

Evasion Technique Taxonomy

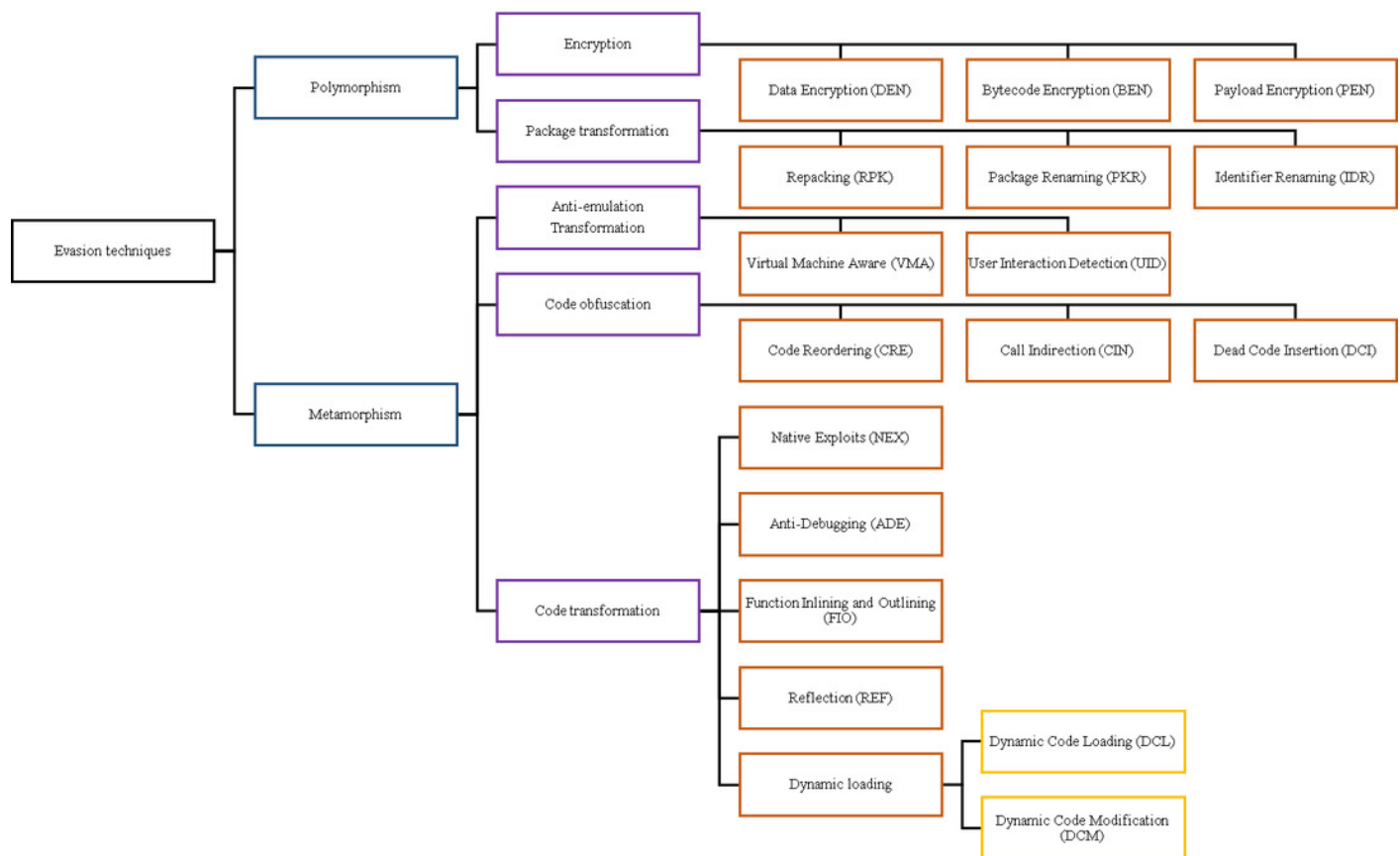



Figure 3

The Main Categories of Android Malware Detection Techniques

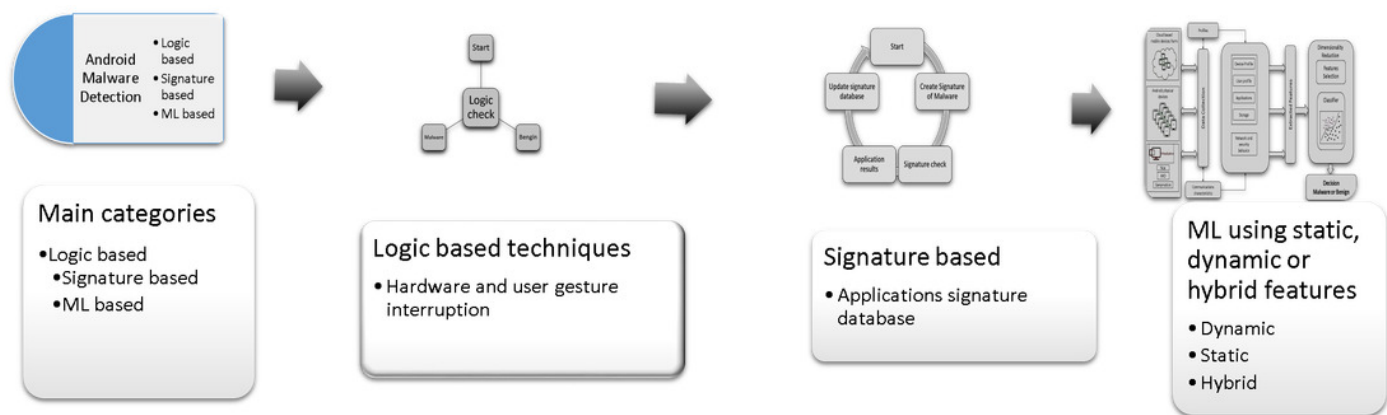
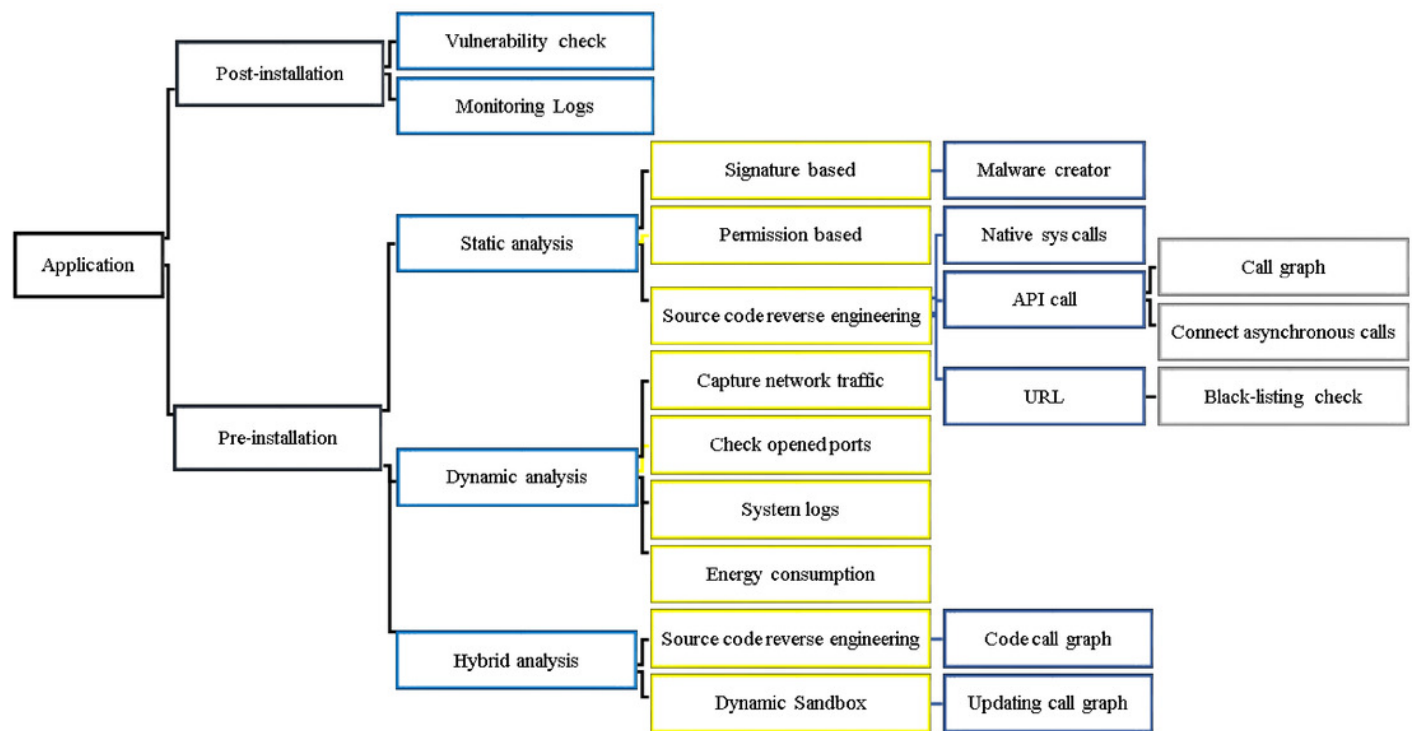



Figure 4

Taxonomy of Android Malware Detection Methodologies

Figure 5

Evasion Techniques Radar for Static Frameworks

—•—Number of static Frameworks vs Evasions Techniques

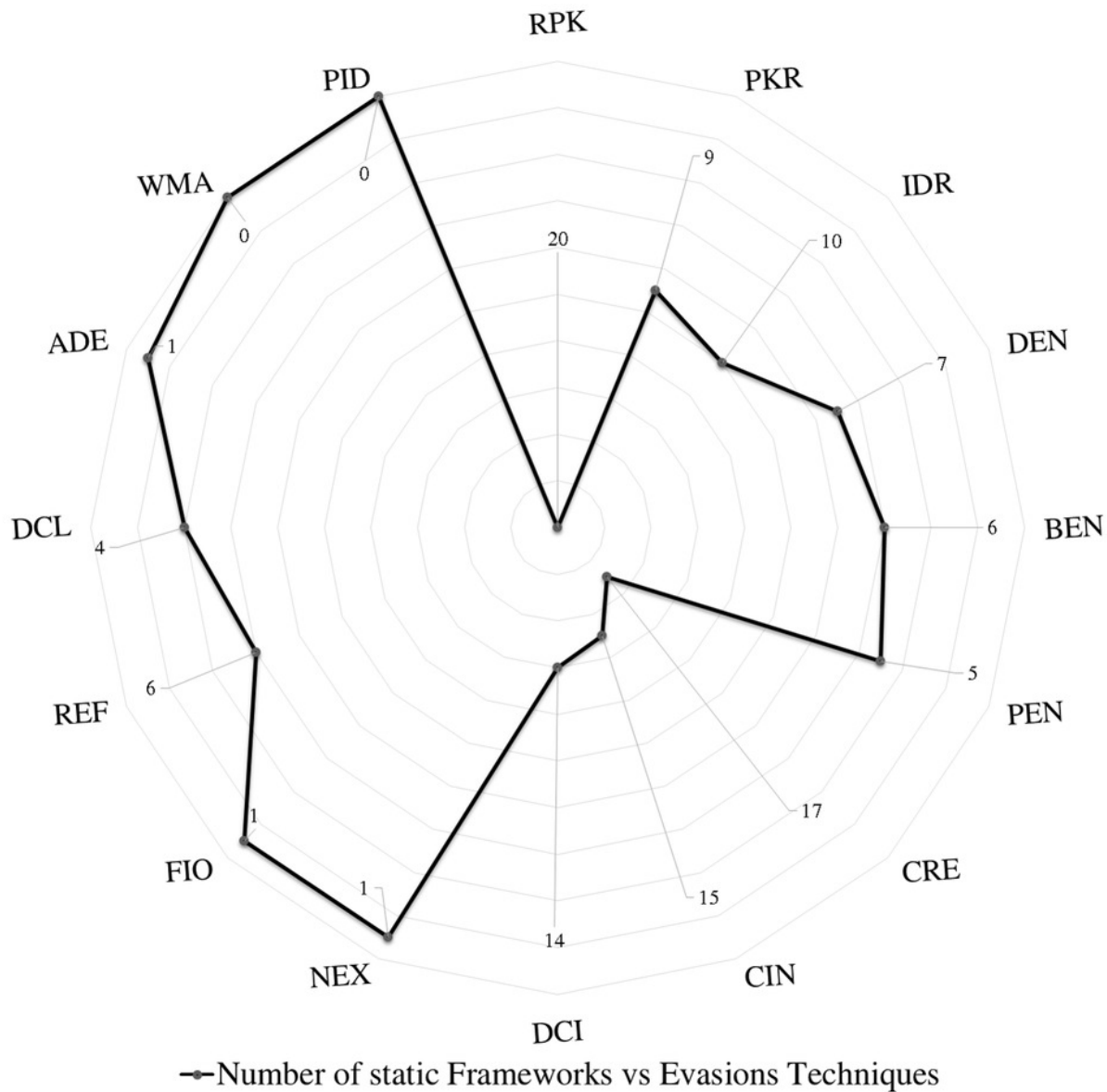



Figure 6

Static Analysis Based Frameworks and Considered Evasion

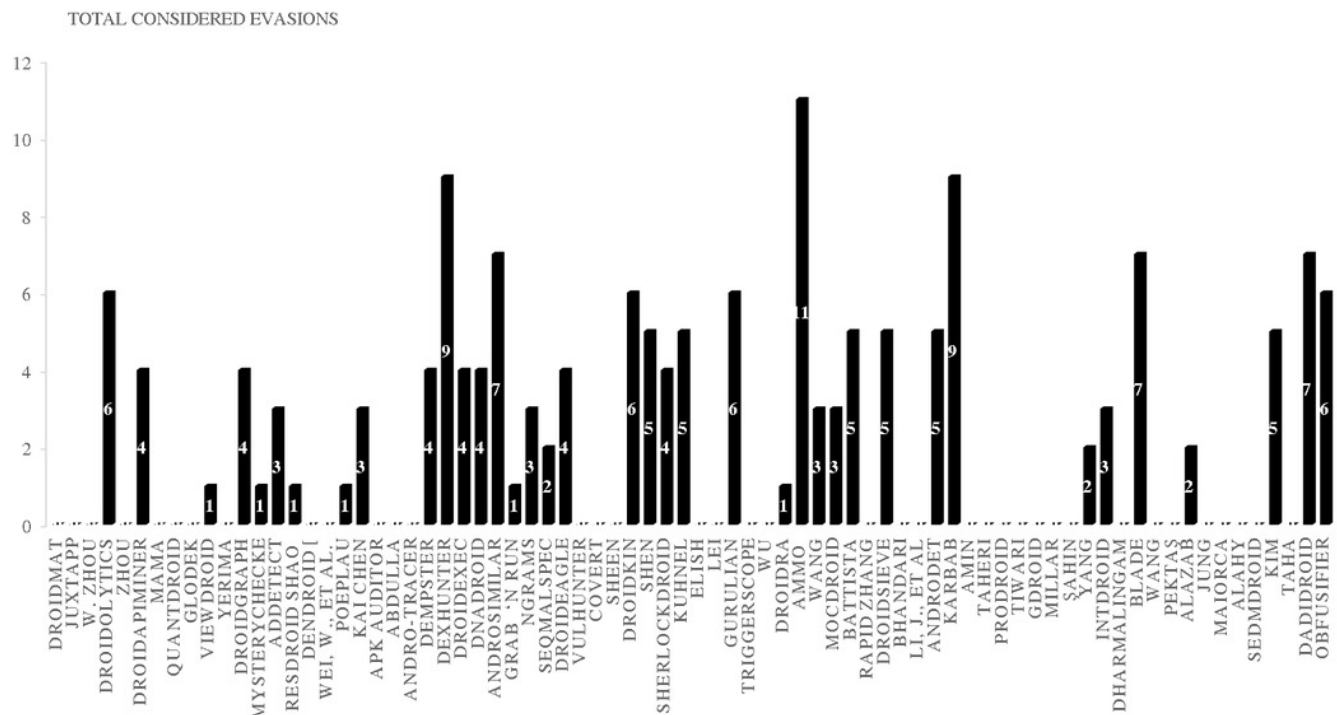



Figure 7

Dynamic Analysis and Evasion Radar Graph

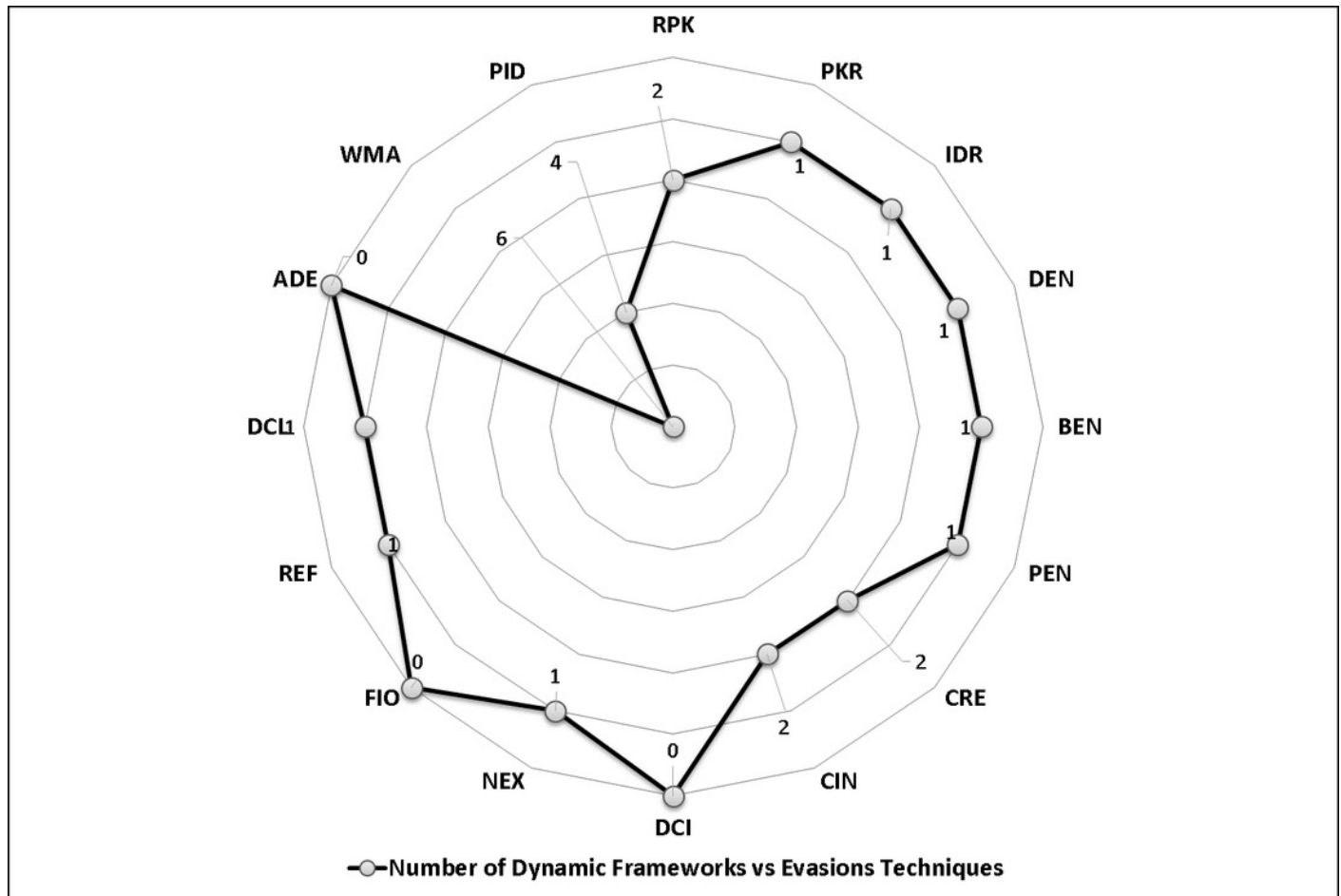



Figure 8

Dynamic Analysis Based Frameworks and Considered Evasion

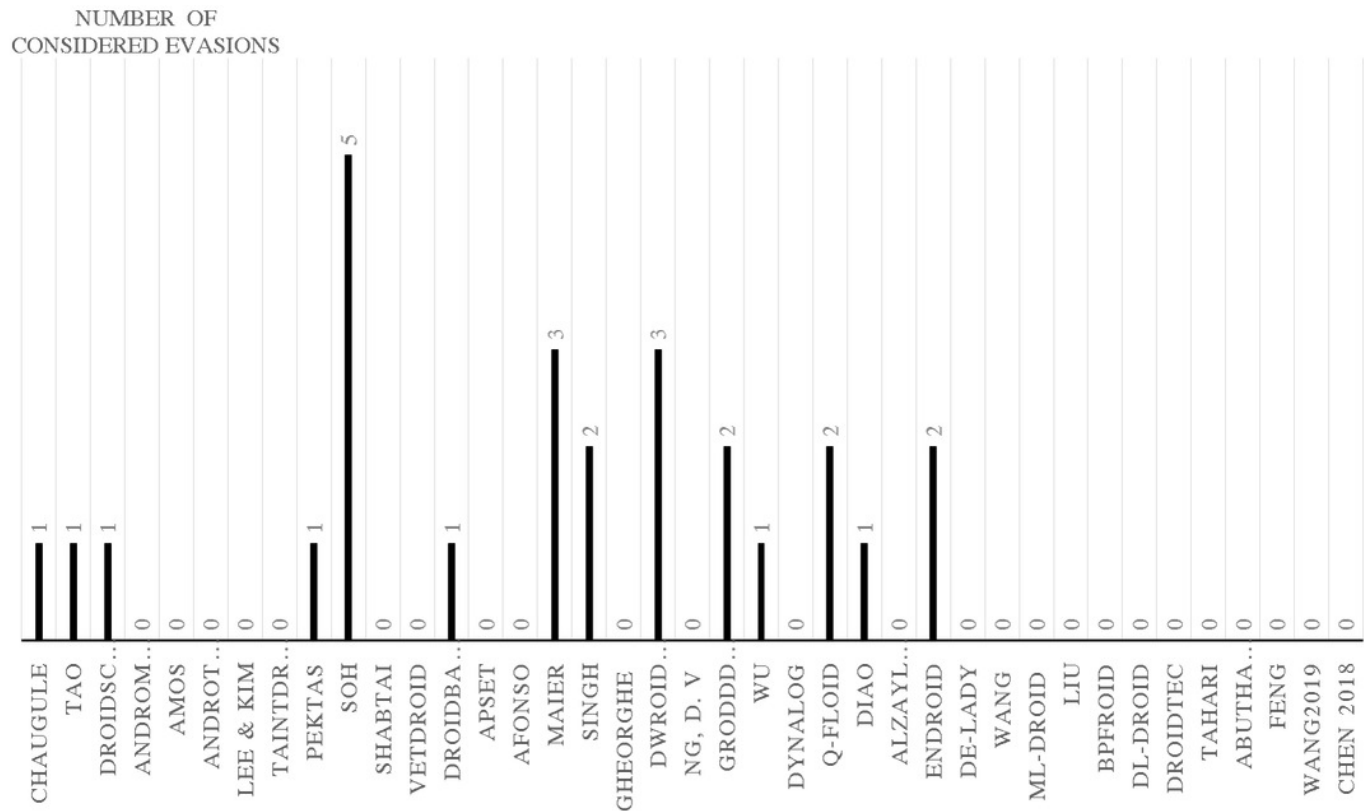



Figure 9

Hybrid Analysis and Evasion Radar

─●─Number of Hybrid Frameworks vs Evasions Techniques

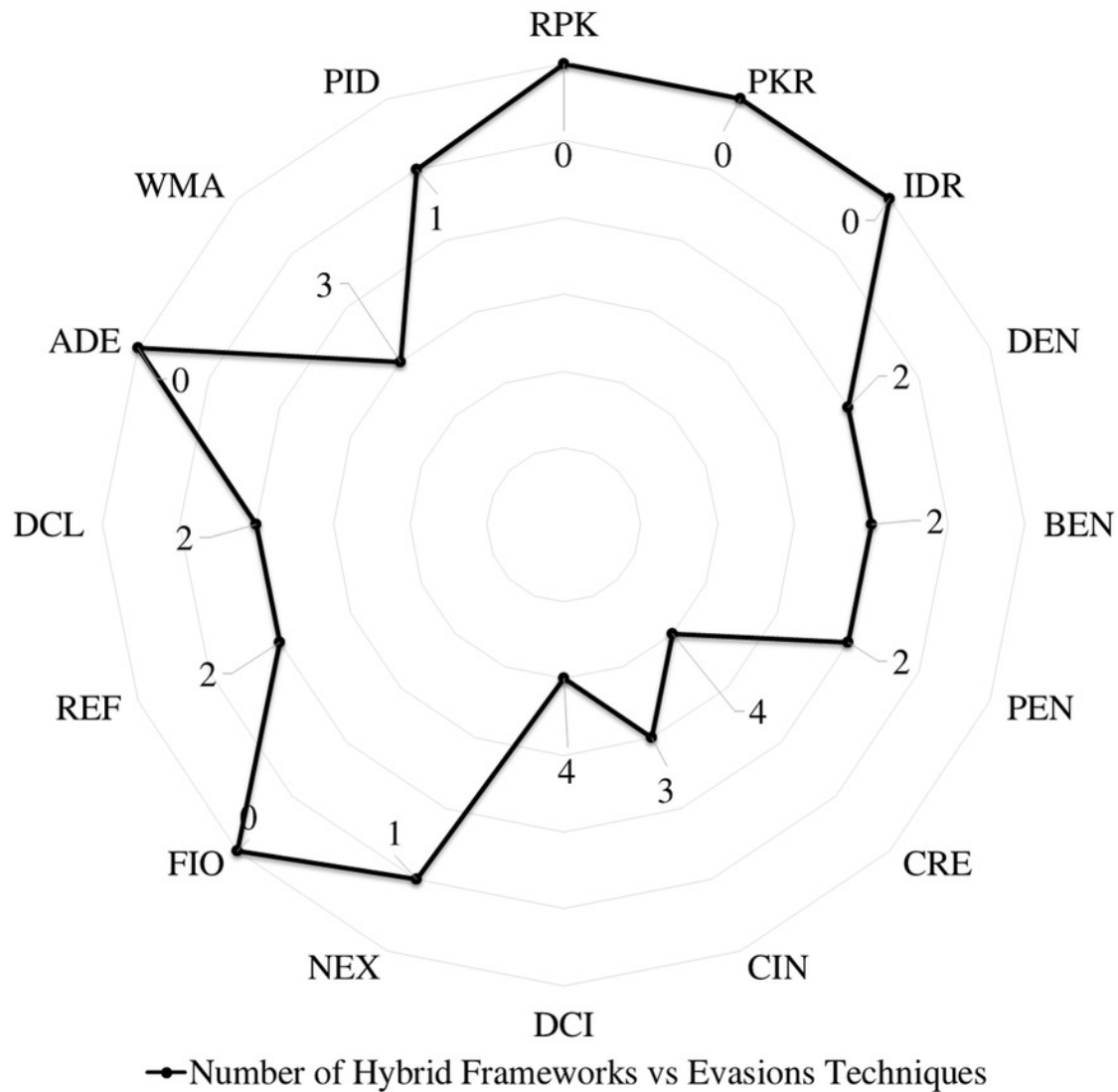



Figure 10

Hybrid Analysis Based Frameworks and Considered Evasion

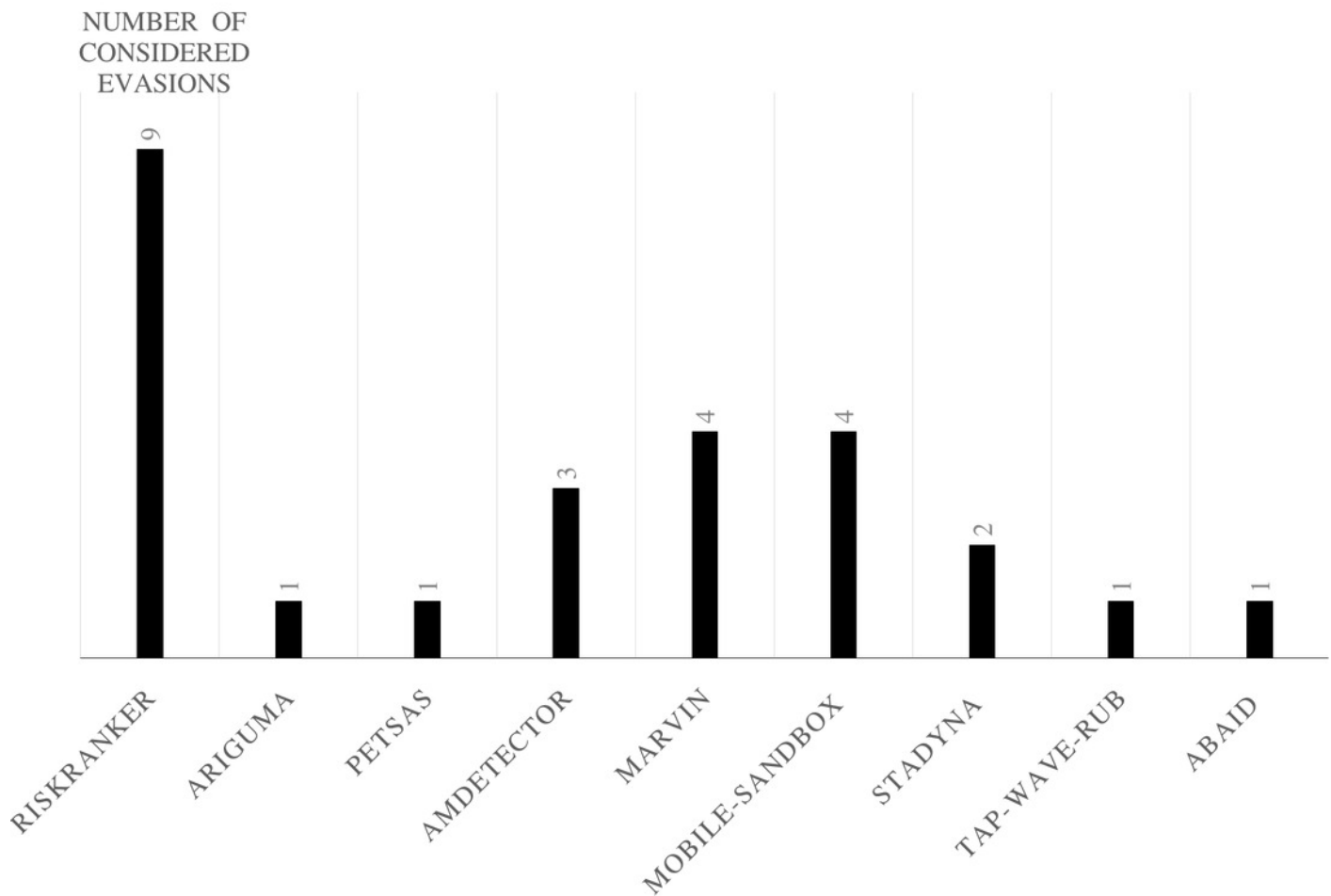
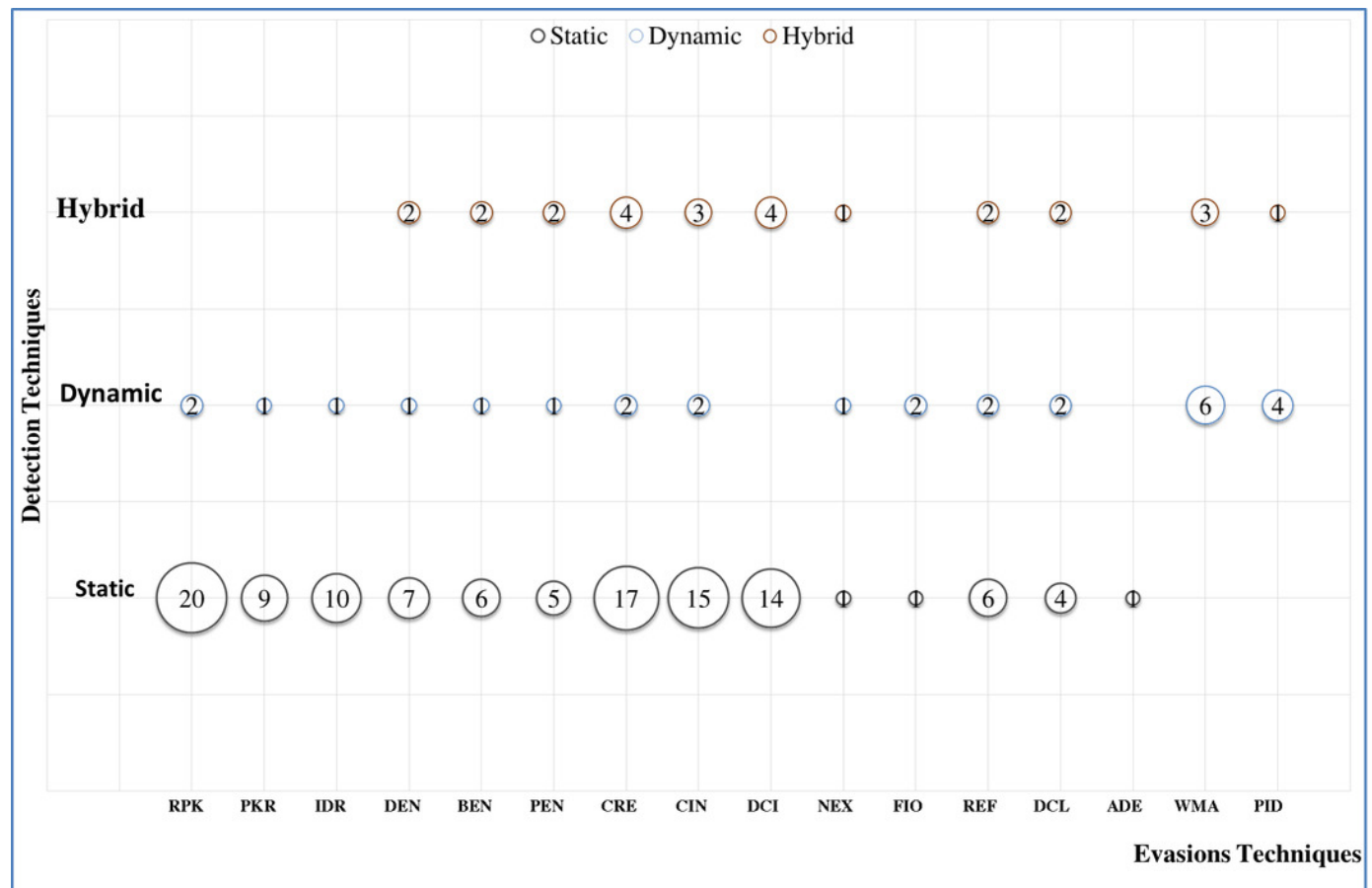



Figure 11

Systematic Map of Accumulative Number of Detection Frameworks versus Evasions
Techniques

Comparison of the recent reviews

Related Studies	Evasion Techniques Discussion	Evasion detection Tools Evaluation
This study	Encryption, package and code transformation, code obfuscation, anti-emulation	Commercial + Academic
Droidchameleon (Rastogi et al. 2013)	Transformation	Commercial
Vikas(Sihag et al. 2021)	Code Obfuscation, repackaging	Academic
FeCO (Jusoh et al. 2021)	Code Obfuscation, Encryption	Academic
Rastogi (Rastogi et al. 2014)	Encryption + Transformation	Commercial
AAMO (Preda & Maggi 2016)	None	Commercial
Hoffmann (Hoffmann et al. 2016)	Obfuscation	Commercial
Tam et al. (Tam et al. 2017)	Transformation + Obfuscation	None
Nguyen-Vu et al. (Nguyen-Vu et al. 2017)	Transformation	None
Kim et al. (Kim et al. 2016)	Anti-emulation	None
Xue et al. (Xue et al. 2017)	Encryption	Commercial
Bulazel (Bulazel & Yener 2017)	Virtualization and performance Case studies	Academic

3   References

7   Bulazel A, and Yener B. 2017. A survey on automated dynamic malware analysis evasion and
8   counter-evasion: Pc, mobile, and web. Proceedings of the 1st Reversing and Offensive-
9   oriented Trends Symposium: ACM. p 2.

10   Hoffmann J, Rytilahti T, Maiorca D, Winandy M, Giacinto G, and Holz T. 2016. Evaluating analysis
11   tools for android apps: Status Quo and robustness against obfuscation. 6th ACM
12   Conference on Data and Application Security and Privacy, CODASPY 2016: Association
13   for Computing Machinery, Inc. p 139-141.

14   Jusoh R, Firdaus A, Anwar S, Osman MZ, Darmawan MF, and Ab Razak MF. 2021. Malware
15   detection using static analysis in Android: a review of FeCO (features, classification, and
16   obfuscation). *PeerJ Computer Science*.

17   Kim M, Lee TJ, Shin Y, and Youm HY. 2016. A study on behavior-based mobile malware analysis
18   system against evasion techniques. Information Networking (ICOIN), 2016 International
19   Conference on: IEEE. p 455-457.

20   Nguyen-Vu L, Chau N-T, Kang S, and Jung S. 2017. Android Rooting: An Arms Race between
21   Evasion and Detection. *Security and Communication Networks* 2017:1-13.

22   Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
23   systematization of knowledge and unified methodology. *Journal of Computer Virology and
24   Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2

25   Rastogi V, Chen Y, and Jiang X. 2013. DroidChameleon: Evaluating Android anti-malware against
26   transformation attacks. 8th ACM SIGSAC Symposium on Information, Computer and
27   Communications Security, ASIA CCS 2013. Hangzhou. p 329-334.

28   Rastogi V, Chen Y, and Jiang X. 2014. Catch me if you can: Evaluating android anti-malware
29   against transformation attacks. *IEEE Transactions on Information Forensics and Security*
30   9:99-108. 10.1109/TIFS.2013.2290431

31   Sihag V, Vardhan M, and Singh P. 2021. A survey of android application and malware hardening.
32   *Computer Science Review* 39:100365.

33   Tam K, Feizollah A, Anuar NB, Salleh R, and Cavallaro L. 2017. The Evolution of Android Malware
34   and Android Analysis Techniques. *ACM Computing Surveys* 49:1-41. 10.1145/3017427

35   Xue Y, Meng G, Liu Y, Tan TH, Chen H, Sun J, and Zhang J. 2017. Auditing anti-malware tools
36   by evolving android malware and dynamic loading technique. *IEEE Transactions on
37   Information Forensics and Security* 12:1529-1544.

Comparison of the recent reviews

Article Type	Full Name	Publisher
Journals	ACM Computing Surveys	ACM
	ACM Transaction on Computer system	ACM
	Computers & Security	
	Digital Investigation	
	Future Generation Computer Systems	
	IEEE Transactions on Dependable and Secure Computing	
	IEEE Access	IEEE
	IEEE Transactions on Industrial Informatics	IEEE
	IEEE Transactions on Information Forensics and Security	IEEE
	IEEE Transactions on Knowledge and Data Engineering	
	IEEE Transactions on Mobile Computing	
	IEEE Transactions on Network Science and Engineering	
	IEEE Transactions on Reliability	
	Information and Software Technology	
	Information Sciences	
	International Journal of Distributed Sensor Networks	
	International Journal of Information Security	
	International Journal of Interactive Multimedia & Artificial Intelligence	Springer
	Journal of Ambient Intelligence and Humanized Computing	
	Journal of artificial intelligence research	
	Journal of Computer Virology and Hacking Techniques	Springer
	Journal of Information Science and Engineering	
	Journal of Information Security and Applications	
	Journal of Supercomputing	
	PLOS ONE	
	Soft Computing	
	Security and Communication Networks	
Conferences	Advanced Computing, Networking and Security	IEEE
	Artificial Intelligence and Knowledge Engineering (AIKE)	IEEE
	Inventive Research in Computing Applications (ICIRCA)	IEEE
	International Arab Conference on Information Technology (ACIT)	IEEE
	Information Security	IEEE
	Network Computing and Applications (NCA)	IEEE
	Computer Software and Applications Conference	IEEE
	International Conference on Security and Privacy in Communication Systems	Springer
	International Conference on Security and Privacy in Communication Systems	Springer
	Seventh ACM on Conference on Data and Application Security and Privacy	ACM
	The symposium on applied computing	ACM
	Data and application security and privacy	ACM

Online Malware Scanning Frameworks

Online security scanning	Description	Started	Scanning Rate (app/ day)	Services	License
VirusTotal (Google)	https://www.virustotal.com	2011	Ignored	Web /API	Free
AndroTotal (Maggi et al. 2013) Droydseuss (Coletta 2016)	https://andrototal.org/ http://droydseuss.com	2013	Ignored	Web	Free
ANDRUBIS (Lindorfer et al. 2014)	https://anubis.iseclab.org. commercialized to https://www.lastline.com/	2012	3,500	API	Free/ discontinued – Paid only
APK Auditor (Talha et al. 2015)	http://app.ibu.edu.tr:8080/apkin spectoradmin	2015	Ignored	Web	Discontinued
NVISO(Hoffmann et al. 2016)	https://apkscan.nviso.be/	-	2400	Web /API	Free/Pro
Copperdroid	http://copperdroid.isg.rhul.ac.uk/copperdroid/	2015	NA	Web	NA
Totalhash	https://totalhash.cymru.com		10	Web/API	Commercial

3   References

7   Coletta AaVDV, Victor and Maggi, Federico. 2016. DroydSeuss: A Mobile Banking Trojan Tracker
8   -Short Paper. Financial Cryptography and Data Security. Springer Berlin Heidelberg.

9   Google. VirusTotal https://virustotal.com/ (accessed 1/6/2016.

10   Hoffmann J, Rytilahti T, Maiorca D, Winandy M, Giacinto G, and Holz T. 2016. Evaluating analysis
11   tools for android apps: Status Quo and robustness against obfuscation. 6th ACM
12   Conference on Data and Application Security and Privacy, CODASPY 2016: Association
13   for Computing Machinery, Inc. p 139-141.

14   Lindorfer M, Neugschwandtner M, Weichselbaum L, Fratantonio Y, van der Veen V, and Platzer
15   C. 2014. ANDRUBIS-1,000,000 Apps Later: A View on Current Android Malware
16   Behaviors. Proceedings of the the 3rd International Workshop on Building Analysis
17   Datasets and Gathering Experience Returns for Security (BADGERS).

18   Maggi F, Valdi A, and Zanero S. 2013. AndroTotal: a flexible, scalable toolbox and service for
19   testing mobile malware detectors. Proceedings of the Third ACM workshop on Security
20   and privacy in smartphones & mobile devices: ACM. p 49-54.

21   Talha KA, Alper DI, and Aydin C. 2015. APK Auditor: Permission-based Android malware
22   detection system. *Digital Investigation* 13:1-14. 10.1016/j.diin.2015.01.001

Confusion Matrix

		Classified apps	
	Total Samples	Malware	Benign
True apps	Malware - M	TP[1]	FN[2]
	Benign - B	FP[3]	TN[4]

2
3
4

---

[1] TP True Positive

[2] FN False Negative

[3] FP False Positive

[4] TN True Negative

ML Classification Performance Measures

Performance measure	Short-form	Formulas	Description
Recall or Sensitivity	TPR	$=\dfrac{TP}{M}=\dfrac{TP}{TP+FP}$	True Positive Rate
Miss rate	FNR	$=\dfrac{FN}{M}=\dfrac{FN}{TP+FP}$	False Negative Rate
Fall-out	FPR	$=\dfrac{FP}{B}=\dfrac{FP}{TP+FN}$	False Positive Rate
Specificity	TNR	$=\dfrac{TN}{B}=\dfrac{TN}{TP+FN}$	True Negative Rate
Precision	PPV	$=\dfrac{TP}{TP+FP}$	Positive Predictive Value
False Discovery Rate	FDR	$=\dfrac{FP}{TP+FP}$	False Discovery Rate
False Omission Rate	FOR	$==\dfrac{FN}{TN+FN}$	False Omission Rate
Negative Predictive Value	NPV	$=\dfrac{TN}{TN+FN}$	Negative Predictive Value
Accuracy	ACC	$=\dfrac{TP+TN}{D}=\dfrac{TP+TN}{TP+TN+FP+FN}$	Total truly detected apps over total examined apps
F-measure	F1	$=\dfrac{2\times TP}{2\times TP+FN+FP}$	The harmonic mean of precision and sensitivity

2
3
4
5

Android Malware Evasion Test benches

1 (RPK)[1], (PKR[2]), (IDR[3]), (DEN[4]), (BEN[5]), (PEN[6]), (CRE[7]), (CIN[8]), (DCI[9]), (NEX[10]), (FIO[11]),
2 API(REF[12]), (DCL/DCM[13]), (ADE[14]), (VMA[15]), (PID[16])

| Framework | Polymorphism | | | | | | Metamorphism | | | | | | | | | |
| | Package transformation | | | Encryption | | | Code obfuscation | | | Advanced Code transformation | | | | | Anti-emulator | |
	(RPK)	(PKR)	(IDR)	(DEN)	(BEN)	(PEN)	(CRE)	(CIN)	(DCI)	(NEX)	(FIO):	(REF)	DCL/DCM	(ADE)	(VMA)	(PID)
ADAM(Zheng et al. 2013)	✓	*	*	✓	*	*	✓	*	✓	*	*	*	*	*	*	*
DroidChameleon (Rastogi et al. 2013)	✓	*	*	*	*	*	✓	*	*	*	*	✓	*	*	*	*

---

[1] REPACKING

[2] PACKAGE RENAMING

[3] IDENTIFIER RENAMING

[4] DATA ENCRYPTION

[5] BYTECODE ENCRYPTION

[6] PAYLOAD ENCRYPTION

[7] CODE REORDERING

[8] CALL INDIRECTIONS

[9] DEAD CODE INSERTION

[10] NATIVE EXPLOITS

[11] FUNCTION INLINING AND OUTLINING

[12] REFLECTION

[13] DYNAMIC CODE LOADING/MODIFICATION

[14] ANTI-DEBUGGING

[15] VIRTUAL MACHINE AWARE

[16] PROGRAMMED INTERACTION DETECTION

	Polymorphism						Metamorphism									
Framework	Package transformation			Encryption			Code obfuscation			Advanced Code transformation					Anti-emulator	
	(RPK)	(PKR)	(IDR)	(DEN)	(BEN)	(PEN)	(CRE)	(CIN)	(DCI)	(NEX)	(FIO):	(REF)	(DCL/DCM)	(ADE)	(VMA)	(PID)
ProGuard(Lafortune 2002)	*	*	*	✓	✓	✓	*	*	*	*	*	*	*	*	*	*
DexGuard(Guard Square 2014)	*	*	*	✓	*	*	✓	✓	*	*	*	*	*	*	*	*
Klassmaster (Klassmaster 2013)	*	*	*	✓	✓	*	✓	✓	*	*	*	*	*	*	*	*
Maiorca (Maiorca et al. 2015)	✓	*	*	✓	✓	✓	*	*	*	*	*	✓	*	*	*	*
Vidas (Vidas & Christin 2014)	*	*	*	*	*	*	*	*	*	*	*	*	*	*	✓	*
Petsas (Petsas et al. 2014)	*	*	*	*	*	*	*	*	*	*	*	*	*	*	✓	*
Morpheus (Jing et al. 2014)	*	*	*	*	*	*	*	*	*	*	*	*	*	*	✓	*
Garcia(Garcia et al. 2015)	*	✓	*	✓	✓	*	*	✓	*	*	*	*	*	*	*	*
DroidSieve (Suarez-Tangil et al. 2017)	*	*	*	✓	✓	✓	*	*	*	*	*	✓	✓	*	*	*
MysteryChecker (Jeong et al. 2014)	✓	*	*	*	✓	✓	✓	✓	*	*	*	*	*	*	*	*
PANDORA(Protsenko & Muller 2013)	*	*	*	✓	*	*	*	*	*	*	✓	✓		*	*	*
Mystique) Meng et al. 2016(	*	*	✓	✓	*	*	*	*	*	*	✓	*	*	*	*	*
Canfora (Canfora et al. 2015)	✓	✓	✓	✓	*	*	✓	*	✓	*	*	*	*	*	*	*
Hatwar (Hatwar & Shelke 2014)	*	*	*	*	*	*	*	*	*	*	*	*	✓	*	*	*
AAMO (Preda &	✓	✓	*	*	✓	*	✓	✓	✓	*	✓	✓	*	✓	*	*

| Framework | Polymorphism | | | | | | Metamorphism | | | | | | | | | |
| | Package transformation | | | Encryption | | | Code obfuscation | | | Advanced Code transformation | | | | | Anti-emulator | |
	(RPK)	(PKR)	(IDR)	(DEN)	(BEN)	(PEN)	(CRE)	(CIN)	(DCI)	(NEX)	(FIO):	(REF)	DCL/DC...	(ADE)	(VMA)	(PID)
Maggi 2016)																
Abid (Abaid et al. 2017)	*	*	*	*	*	*	*	*	*	*	*	*	✓	*	*	*
EnDroid (Feng et al. 2018)	*	*	*	*	*	*	*	*	*	*	*	✓	✓	*	*	*
Bacci (Bacci et al. 2018)	✓	✓	✓	✓	*	*	✓	✓	✓	*	*	*	*	*	*	*
DexMoinitor(Cho et al. 2018)	*	*	*	✓	✓	✓	*	*	*	*	*	*	*	*	*	*
Kim(Kim et al. 2019)	*	✓	✓	✓	*	*	*	✓	✓	*	*	*	*	*	*	*
DAMBA (Zhang et al. 2020)	*	*	*	✓	✓	✓	*	✓	*	*	*	*	✓	*	*	*
IMCFN (Vasan et al. 2020)	✓	✓	✓	✓	*	*	✓	*	✓	*	*	*	*	*	*	*
PetaDroid(Karbab & Debbabi 2021)	✓	✓	✓	✓	*	✓	✓	✓	✓	*	*	✓	*	*	*	*
BLADE (Sihag et al. 2021)	✓	✓	✓	✓	✓	✓	✓	*	*	*	*	*	*	*	*	*
DANDroid(Millar et al. 2020)	*	*	*	✓	✓	✓	*	*	*	*	*	*	*	*	*	*
AndrODet (Mirzaei et al. 2019)	✓	✓	✓	✓	*	*	*	✓	*	*	*	*	*	*	*	*
Dadidroid(Ikram et al. 2019)	✓	✓	✓	✓	✓	✓	*	✓	*	*	*	*	*	*	*	*
Obfusifier(Li et al. 2019)	✓	✓	✓	*	*	*	✓	✓	✓	*	*	*	*	*	*	*

4   References

9   Abaid Z, Kaafar MA, and Jha S. 2017. Quantifying the impact of adversarial evasion attacks on
10   machine learning based android malware classifiers. Network Computing and
11   Applications (NCA), 2017 IEEE 16th International Symposium on: IEEE. p 1-10.

12   Bacci A, Bartoli A, Martinelli F, Medvet E, Mercaldo F, and Visaggio CA. 2018. Impact of Code
13   Obfuscation on Android Malware Detection based on Static and Dynamic Analysis.
14   ICISSP. p 379-385.

15   Canfora G, Di Sorbo A, Mercaldo F, and Visaggio CA. 2015. Obfuscation techniques against
16   signature-based detection: A case study. 1st Mobile Systems Technologies Workshop,
17   MST 2015: Institute of Electrical and Electronics Engineers Inc. p 21-26.

18   Cho H, Yi JH, and Ahn G. 2018. DexMonitor: Dynamically Analyzing and Monitoring Obfuscated
19   Android Applications. *IEEE Access* 6:71229-71240. 10.1109/ACCESS.2018.2881699

20   Feng P, Ma J, Sun C, Xu X, and Ma Y. 2018. A novel dynamic Android malware detection system
21   with ensemble learning. *IEEE Access* 6:30996-31011.

22   Garcia J, Hammad M, Pedrood B, Bagheri-Khaligh A, and Malek S. 2015. Obfuscation-resilient,
23   efficient, and accurate detection and family identification of android malware. *Department*
24   *of Computer Science, George Mason University, Tech Rep* 202.

25   GuardSquare. 2014. DexGuard.

26   Hatwar MS, and Shelke C. 2014. An Assess Android Antimalware that Detects Malicious Dynamic
27   Code in Apps. *International Journal of Computer Science and Mobile Computing* 3:263.

28   Ikram M, Beaume P, and Kâafar MA. 2019. Dadidroid: An obfuscation resilient tool for detecting
29   android malware via weighted directed call graph modelling. *arXiv preprint*
30   *arXiv:190509136*.

31   Jeong J, Seo D, Lee C, Kwon J, Lee H, and Milburn J. 2014. MysteryChecker: Unpredictable
32   attestation to detect repackaged malicious applications in Android. 9th IEEE International
33   Conference on Malicious and Unwanted Software, MALCON 2014: Institute of Electrical
34   and Electronics Engineers Inc. p 50-57.

35   Jing Y, Zhao Z, Ahn G-J, and Hu H. 2014. Morpheus: automatically generating heuristics to detect
36   Android emulators. Proceedings of the 30th Annual Computer Security Applications
37   Conference: ACM. p 216-225.

38   Karbab EB, and Debbabi M. 2021. Resilient and Adaptive Framework for Large Scale Android
39   Malware Fingerprinting using Deep Learning and NLP Techniques. *arXiv preprint*
40   *arXiv:210513491*.

41   Kim T, Kang B, Rho M, Sezer S, and Im EG. 2019. A Multimodal Deep Learning Method for
42   Android Malware Detection Using Various Features. *IEEE Transactions on Information*
43   *Forensics and Security* 14:773-788.

44   Klassmaster. 2013. Zelix Klassmaster [online] Available: http://www.zelix.com/klassmaster/.

45   Lafortune E. 2002. ProGuard.

46 Li Z, Sun J, Yan Q, Srisa-an W, and Tsutano Y. 2019. Obfusifier: Obfuscation-resistant Android
47   malware detection system. International Conference on Security and Privacy in
48   Communication Systems: Springer. p 214-234.

49 Maiorca D, Ariu D, Corona I, Aresu M, and Giacinto G. 2015. Stealth attacks: An extended insight
50   into the obfuscation effects on Android malware. *Computers and Security* 51:16-31.
51   10.1016/j.cose.2015.02.007

52 Meng G, Xue Y, Mahinthan C, Narayanan A, Liu Y, Zhang J, and Chen T. 2016. Mystique:
53   Evolving android malware for auditing anti-malware tools. 11th ACM Asia Conference on
54   Computer and Communications Security, ASIA CCS 2016: Association for Computing
55   Machinery, Inc. p 365-376.

56 Millar S, McLaughlin N, Martinez del Rincon J, Miller P, and Zhao Z. 2020. DANdroid: A Multi-
57   View Discriminative Adversarial Network for Obfuscated Android Malware Detection.
58   Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy.
59   p 353-364.

60 Mirzaei O, de Fuentes JM, Tapiador J, and Gonzalez-Manzano L. 2019. AndrODet: An adaptive
61   Android obfuscation detector. *Future Generation Computer Systems* 90:240-261.
62   https://doi.org/10.1016/j.future.2018.07.066

63 Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, and Ioannidis S. 2014. Rage against
64   the virtual machine: Hindering dynamic analysis of android malware. 7th European
65   Workshop on System Security, EuroSec 2014. Amsterdam: Association for Computing
66   Machinery.

67 Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
68   systematization of knowledge and unified methodology. *Journal of Computer Virology and
69   Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2

70 Protsenko M, and Muller T. 2013. PANDORA applies non-deterministic obfuscation randomly to
71   Android. 2013 8th International Conference on Malicious and Unwanted Software: "The
72   Americas", MALWARE 2013. Fajardo, PR: IEEE Computer Society. p 59-67.

73 Rastogi V, Chen Y, and Jiang X. 2013. DroidChameleon: Evaluating Android anti-malware against
74   transformation attacks. 8th ACM SIGSAC Symposium on Information, Computer and
75   Communications Security, ASIA CCS 2013. Hangzhou. p 329-334.

76 Sihag V, Vardhan M, and Singh P. 2021. BLADE: Robust malware detection against obfuscation
77   in android. *Forensic Science International: Digital Investigation* 38:301176.

78 Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, and Cavallaro L. 2017. DroidSieve:
79   Fast and Accurate Classification of Obfuscated Android Malware. Seventh ACM on
80   Conference on Data and Application Security and Privacy. Scottsdale, Arizona, USA:
81   ACM. p 309-320.

82 Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, and Zheng Q. 2020. IMCFN: Image-based
83   malware classification using fine-tuned convolutional neural network architecture.
84   *Computer Networks* 171:107138. https://doi.org/10.1016/j.comnet.2020.107138

85 Vidas T, and Christin N. 2014. Evading android runtime analysis via sandbox detection.
86   Proceedings of the 9th ACM symposium on Information, computer and communications
87   security: ACM. p 447-458.

88 Zhang W, Wang H, He H, and Liu P. 2020. DAMBA: Detecting Android Malware by ORGB
89   Analysis. *IEEE Transactions on Reliability* 69:55-69.

90 Zheng M, Lee PP, and Lui JC. 2013. ADAM: an automatic and extensible platform to stress test

91   android anti-virus systems.   *Detection of Intrusions and Malware, and Vulnerability*
92   *Assessment*: Springer, 82-101.

List of examined Android malware detection frameworks

Detection techniques	The examined Android malware detection frameworks	Number of frameworks
Static	DroidMat (Wu et al. 2012), Juxtapp (Hanna et al. 2013), DroidOLytics (Faruki et al. 2013), Zhou (Zhou et al. 2013), DroidAPIMiner (Aafer et al. 2013), MAMA (Sanz et al. 2013), QuantDroid (Markmann et al. 2013), Glodek (Glodek & Harang 2013), ViewDroid (Zhang et al. 2014), Yerima (Yerima et al. 2014), DroidGraph (Kwon et al. 2014), MysteryChecker(Jeong et al. 2014), AdDetect (Narayanan et al. 2014), ResDroid (Shao et al. 2014), Dendroid (Suarez-Tangil et al. 2014), Wei, W., et al. (Wei et al. 2015), Poeplau (Poeplau et al. 2014), Chen (Chen et al. 2015), Apk Auditor (Talha et al. 2015), Abdulla (Abdulla & Altaher 2015), Andro-Tracer (Kang et al. 2015), Dempster–Shafe (Du et al. 2015) , Dexhunter(Zhang et al. 2015), DroidExec(Wei et al. 2015), AnDarwin and DNADroid(Crussell et al. 2015), AndroSimilar (Faruki et al. 2015), Grab 'n Run Falsina (Falsina et al. 2015), Ngrams (Canfora et al. 2015), SeqMalSpec -Sufatrio (Sufatrio et al. 2015), DroidEagle(Sun et al. 2015), VulHunter (Chenxiong et al. 2015), COVERT (Bagheri et al. 2015), Sheen (Sheen et al. 2015), Droidkin (Gonzalez et al. 2014), Shen (Shen et al. 2015), SherlockDroid (Apvrille & Apvrille 2015), Kuhnel (Kuhnel et al. 2015), Elish (Elish et al. 2015), Lei (Lei et al. 2015) , Gurulian (Gurulian et al. 2016), TriggerScope(Fratantonio et al. 2016), Wu 2016 (Wu et al. 2016), DroidRA (Li et al. 2016), AAMO (Preda & Maggi 2016), Wang (Wang et al. 2016), MocDroid (Martín et al. 2016), Battista (Battista et al. 2016), RAPID Zhang (Zhang et al. 2016), DroidSieve (Suarez-Tangil et al. 2017), Bhandari et al. (Bhandari et al. 2017), Jin Li (Li et al. 2018), AndRODet (Mirzaei et al. 2019), PetaDroid (Karbab & Debbabi 2021), Amin (Amin et al. 2020) , Taheri (Taheri et al. 2020), ProDroid(Sasidharan & Thomas 2021), Tiwari (Tiwari & Shukla 2018), GDroid(Gao et al. 2021), Millar (Millar et al. 2021), Şahin (Şahin et al. 2021), DGCNDroid (Yang et al. 2021), IntDroid (Zou et al. 2021), Dharmalingam (Dharmalingam & Palanisamy 2021), BLADE (Sihag et al. 2021a), Wang (Wang et al. 2020), Pektas (Pektaş & Acarman 2020), Alazab (Alazab et al. 2020), Jung (Jung et al. 2018), Tiwari (Tiwari & Shukla 2018), Maiorca(Maiorca et al. 2017), Alahy (Alahy et al. 2020), Hamming (Taheri et al. 2020), SEDMDroid(Zhu et al. 2020), Kim Multimodal (Kim et al. 2019), Taha (Taha & Malebary 2021), Dadidroid (Ikram et al.	74

	2019), Obfusifier (Li et al. 2019)	
Dynamic	Amos (Amos et al. 2013), AndroTotal (Maggi et al. 2013), Lee & Kim (Lee et al. 2014), TaintDroid (Enck et al. 2014), Pektas (Pektas & Acarman 2014), Soh (Soh et al. 2015), Shabtai (Shabtai et al. 2014),VetDroid (Yuan et al. 2014b), DroidBarrier (Almohri et al. 2014), APSET(Salva & Zafimiharisoa 2015), Afonso (Afonso et al. 2015), Maier (Maier et al. 2015), Singh (Singh et al. 2015), Gheorghe (Gheorghe et al. 2015), DwroidDump(Kim et al. 2015), Ng, D. V (Ng & Hwang 2015), GroddDroid (Abraham et al. 2015), Wu 2015 (Wu et al. 2015), DynaLog (Alzaylaee et al. 2016), Q-floid (Castellanos et al. 2016), Diao (Diao et al. 2016), Alzaylaee (Alzaylaee et al. 2017), (Feng et al. 2018), DE-LADY (Sihag et al. 2021b), Wang (Wang & Li 2021),MLDroid (Mahindru & Sangal 2021), Liu (Liu et al. 2021), BPFroid (Agman & Hendler 2021), DL-Droid (Alzaylaee et al. 2020), Droidetec (Ma et al. 2020), Taheri (Taheri et al. 2020), Abuthawabeh (Abuthawabeh & Mahmoud 2019), Feng (Feng et al. 2020), Wang (Wang et al. 2019), Chen (Chen et al. 2018)	35
Hybrid	RiskRanker(Grace et al. 2012), MobSafe (Xu et al. 2013), Shalaginov (Shalaginov & Franke 2014), ARIGUMA (Zhong et al. 2013), Petsas (Petsas et al. 2014), Droid-Sec (Yuan et al. 2014a), AMDetector(Zhao et al. 2014), MARVIN (Lindorfer et al. 2015), Mobile-Sandbox (Spreitzenbarth et al. 2015), StaDyna (Zhauniarovich et al. 2015), Tap-Wave-Rub (Shrestha et al. 2015), Droiddetector (Yuan et al. 2016), Andro-Dumpsys(Jang et al. 2016), Abaid (Abaid et al. 2017), Manto (Mantoo & Khurana 2020), Chao(Chao et al. 2020), Loreenzo(De Lorenzo et al. 2020), Puerta(de la Puerta et al. 2019), Surendrean (Surendran et al. 2020), Lu(Lu et al. 2020), Dhalaria (Dhalaria & Gandotra 2021), Zhu (Zhu et al. 2021), Nawaz (Nawaz 2021), Liu (Liu et al. 2021), PNSDroid (Kandukuru & Sharma 2018), Bacci (Bacci et al. 2018), DAMBA (Zhang et al. 2020)	26

2
3

5   References

9   Aafer Y, Du WL, and Yin H. 2013. DroidAPIMiner: Mining API-Level Features for Robust Malware
10   Detection in Android. *Security and Privacy in Communication Networks, Securecomm*
11   *2013* 127:86-103.

12   Abaid Z, Kaafar MA, and Jha S. 2017. Quantifying the impact of adversarial evasion attacks on
13   machine learning based android malware classifiers. Network Computing and
14   Applications (NCA), 2017 IEEE 16th International Symposium on: IEEE. p 1-10.

15   Abdulla S, and Altaher A. 2015. Intelligent approach for android malware detection. *KSII*
16   *Transactions on Internet and Information Systems* 9:2964-2983. 10.3837/tiis.2015.08.012

17   Abraham A, Andriatsimandefitra R, Brunelat A, Lalande JF, and Viet Triem Tong V. 2015.
18   GroddDroid: A gorilla for triggering malicious behaviors. 10th International Conference on
19   Malicious and Unwanted Software, MALWARE 2015: Institute of Electrical and Electronics
20   Engineers Inc. p 119-127.

21   Abuthawabeh MKA, and Mahmoud KW. 2019. Android malware detection and categorization
22   based on conversation-level network traffic features. 2019 International Arab Conference
23   on Information Technology (ACIT): IEEE. p 42-47.

24   Afonso VM, de Amorim MF, Grégio ARA, Junquera GB, and de Geus PL. 2015. Identifying
25   Android malware using dynamically obtained features. *Journal of Computer Virology and*
26   *Hacking Techniques* 11:9-17. 10.1007/s11416-014-0226-7

27   Agman Y, and Hendler D. 2021. BPFroid: Robust Real Time Android Malware Detection
28   Framework. *arXiv preprint arXiv:210514344*.

29   Alahy QE, Chowdhury MN-U-R, Soliman H, Chaity MS, and Haque A. 2020. Android Malware
30   Detection in Large Dataset: Smart Approach. Future of Information and Communication
31   Conference: Springer. p 800-814.

32   Alazab M, Alazab M, Shalaginov A, Mesleh A, and Awajan A. 2020. Intelligent mobile malware
33   detection using permission requests and api calls. *Future Generation Computer Systems*
34   107:509-521.

35   Almohri HMJ, Yao D, and Kafura D. 2014. DroidBarrier: Know what is executing on your Android.
36   4th ACM Conference on Data and Application Security and Privacy, CODASPY 2014. San
37   Antonio, TX: Association for Computing Machinery. p 257-264.

38   Alzaylaee MK, Yerima SY, and Sezer S. 2016. DynaLog: An automated dynamic analysis
39   framework for characterizing android applications. 2016 International Conference on
40   Cyber Security and Protection of Digital Services, Cyber Security 2016: Institute of
41   Electrical and Electronics Engineers Inc.

42   Alzaylaee MK, Yerima SY, and Sezer S. 2017. Emulator vs real phone: Android malware
43   detection using machine learning. Proceedings of the 3rd ACM on International Workshop
44   on Security And Privacy Analytics: ACM. p 65-72.

45   Alzaylaee MK, Yerima SY, and Sezer S. 2020. DL-Droid: Deep learning based android malware
46   detection using real devices. *Computers & Security* 89:101663.

47   Amin M, Tanveer TA, Tehseen M, Khan M, Khan FA, and Anwar S. 2020. Static malware

48 detection and attribution in android byte-code through an end-to-end deep system. *Future*
49 *Generation Computer Systems* 102:112-126.

50 Amos B, Turner H, and White J. 2013. Applying machine learning classifiers to dynamic Android
51 malware detection at scale. Wireless Communications and Mobile Computing Conference
52 (IWCMC), 2013 9th International. p 1666-1671.

53 Apvrille A, and Apvrille L. 2015. SherlockDroid: a research assistant to spot unknown malware in
54 Android marketplaces. *Journal of Computer Virology and Hacking Techniques* 11:235-
55 245. 10.1007/s11416-015-0245-z

56 Bacci A, Bartoli A, Martinelli F, Medvet E, Mercaldo F, and Visaggio CA. 2018. Impact of Code
57 Obfuscation on Android Malware Detection based on Static and Dynamic Analysis.
58 ICISSP. p 379-385.

59 Bagheri H, Sadeghi A, Garcia J, and Malek s. 2015. COVERT: Compositional Analysis of Android
60 Inter-App Permission Leakage. *Software Engineering, IEEE Transactions on* PP:1-1.
61 10.1109/TSE.2015.2419611

62 Battista P, Mercaldo F, Nardone V, Santone A, and Visaggio CA. 2016. Identification of android
63 malware families with model checking. In: Furnell S, Camp O, and Mori P, editors. 2nd
64 International Conference on Information Systems Security and Privacy, ICISSP 2016:
65 SciTePress. p 542-547.

66 Bhandari S, Jaballah WB, Jain V, Laxmi V, Zemmari A, Gaur MS, Mosbah M, and Conti M. 2017.
67 Android inter-app communication threats and detection techniques. *Computers & Security*
68 70:392-421. https://doi.org/10.1016/j.cose.2017.07.002

69 Canfora G, De Lorenzo A, Medvet E, Mercaldo F, and Visaggio CA. 2015. Effectiveness of opcode
70 ngrams for detection of multi family android malware. Availability, Reliability and Security
71 (ARES), 2015 10th International Conference on: IEEE. p 333-340.

72 Castellanos JH, Wüchner T, Ochoa M, and Rueda S. 2016. Q-floid: Android malware detection
73 with quantitative data flow graphs. In: Roychoudhury A, and Mathur A, editors. 1st
74 Singapore-Cybersecurity R and D Conference, SG-CRC 2016: IOS Press. p 13-25.

75 Chao W, Qun L, XiaoHu W, TianYu R, JiaHan D, GuangXin G, and EnJie S. 2020. An android
76 application vulnerability mining method based on static and dynamic analysis. 2020 IEEE
77 5th Information Technology and Mechatronics Engineering Conference (ITOEC): IEEE. p
78 599-603.

79 Chen K, Wang P, Lee Y, Wang X, Zhang N, Huang H, Zou W, and Liu P. 2015. Finding unknown
80 malice in 10 seconds: Mass vetting for new threats at the google-play scale. 24th USENIX
81 Security Symposium (USENIX Security 15). p 659-674.

82 Chen Z, Yan Q, Han H, Wang S, Peng L, Wang L, and Yang B. 2018. Machine learning based
83 mobile malware detection using highly imbalanced network traffic. *Information Sciences*
84 433:346-364.

85 Chenxiong Q, Xiapu L, Yu L, and Guofei G. 2015. VulHunter: Toward Discovering Vulnerabilities
86 in Android Applications. *Micro, IEEE* 35:44-53. 10.1109/MM.2015.25

87 Crussell J, Gibler C, and Chen H. 2015. AnDarwin: Scalable Detection of Android Application
88 Clones Based on Semantics. *IEEE Transactions on Mobile Computing* 14:2007-2019.
89 10.1109/TMC.2014.2381212

90 de la Puerta JG, Pastor-López I, Sanz B, and Bringas PG. 2019. Network traffic analysis for
91 android malware detection. International Conference on Hybrid Artificial Intelligence
92 Systems: Springer. p 468-479.

93 De Lorenzo A, Martinelli F, Medvet E, Mercaldo F, and Santone A. 2020. Visualizing the outcome
94   of dynamic analysis of Android malware with VizMal. *Journal of Information Security and*
95   *Applications* 50:102423.

96 Dhalaria M, and Gandotra E. 2021. A Hybrid Approach for Android Malware Detection and Family
97   Classification. *International Journal of Interactive Multimedia & Artificial Intelligence* 6.

98 Dharmalingam VP, and Palanisamy V. 2021. A novel permission ranking system for android
99   malware detection—the permission grader. *Journal of Ambient Intelligence and*
100   *Humanized Computing* 12:5071-5081.

101 Diao W, Liu X, Li Z, and Zhang K. 2016. Evading android runtime analysis through detecting
102   programmed interactions. 9th ACM Conference on Security and Privacy in Wireless and
103   Mobile Networks, WiSec 2016: Association for Computing Machinery, Inc. p 159-164.

104 Du Y, Wang X, and Wang J. 2015. A static Android malicious code detection method based on
105   multi-source fusion. *Security and Communication Networks* 8:3238-3246.
106   10.1002/sec.1248

107 Elish KO, Shu X, Yao D, Ryder BG, and Jiang X. 2015. Profiling user-trigger dependence for
108   Android malware detection. *Computers and Security* 49:255-273.
109   10.1016/j.cose.2014.11.001

110 Enck W, Gilbert P, Chun BG, Cox LP, Jung J, McDaniel P, and Sheth AN. 2014. TaintDroid: An
111   Information Flow Tracking System for Real-Time Privacy Monitoring on Smartphones.
112   *Communications of the Acm* 57:99-106. 10.1145/2494522

113 Falsina L, Fratantonio Y, Zanero S, Kruegel C, Vigna G, and Maggi F. 2015. Grab'n Run: Secure
114   and Practical Dynamic Code Loading for Android Applications. Proceedings of the 31st
115   Annual Computer Security Applications Conference: ACM. p 201-210.

116 Faruki P, Laxmi V, Bharmal A, Gaur MS, and Ganmoor V. 2015. AndroSimilar: Robust signature
117   for detecting variants of Android malware. *Journal of Information Security and Applications*
118   22:66-80. 10.1016/j.jisa.2014.10.011

119 Faruki P, Laxmi V, Ganmoor V, Gaur MS, and Bharmal A. 2013. DroidOLytics: Robust feature
120   signature for repackaged android apps on official and third party android markets. 2nd
121   International Conference on Advanced Computing, Networking and Security, ADCONS
122   2013. Surathkal, Karnataka: IEEE Computer Society. p 247-252.

123 Feng J, Shen L, Chen Z, Wang Y, and Li H. 2020. A two-layer deep learning method for android
124   malware detection using network traffic. *IEEE Access* 8:125786-125796.

125 Feng P, Ma J, Sun C, Xu X, and Ma Y. 2018. A novel dynamic Android malware detection system
126   with ensemble learning. *IEEE Access* 6:30996-31011.

127 Fratantonio Y, Bianchi A, Robertson W, Kirda E, Kruegel C, and Vigna G. 2016. TriggerScope:
128   Towards Detecting Logic Bombs in Android Applications. 2016 IEEE Symposium on
129   Security and Privacy, SP 2016: Institute of Electrical and Electronics Engineers Inc. p 377-
130   396.

131 Gao H, Cheng S, and Zhang W. 2021. GDroid: Android malware detection and classification with
132   graph convolutional network. *Computers & Security* 106:102264.

133 Gheorghe L, Marin B, Gibson G, Mogosanu L, Deaconescu R, Voiculescu VG, and Carabas M.
134   2015. Smart malware detection on Android. *Security and Communication Networks*
135   8:4254-4272. 10.1002/sec.1340

136 Glodek W, and Harang R. 2013. Rapid permissions-based detection and analysis of mobile
137   malware using random decision forests. 2013 IEEE Military Communications Conference,

138   MILCOM 2013. San Diego, CA. p 980-985.

139   Gonzalez H, Stakhanova N, and Ghorbani AA. 2014. Droidkin: Lightweight detection of android
140   apps similarity. International Conference on Security and Privacy in Communication
141   Systems: Springer. p 436-453.

142   Grace M, Zhou Y, Zhang Q, Zou S, and Jiang X. 2012. Riskranker: scalable and accurate zero-
143   day android malware detection. Proceedings of the 10th international conference on
144   Mobile systems, applications, and services: ACM. p 281-294.

145   Gurulian I, Markantonakis K, Cavalaro L, and Mayes K. 2016. You can't touch this: Consumer-
146   centric android application repackaging detection. *Future Generation Computer Systems*
147   65:1-9. 10.1016/j.future.2016.05.021

148   Hanna S, Huang L, Wu E, Li S, Chen C, and Song D. 2013. Juxtapp: A scalable system for
149   detecting code reuse among android applications. *Detection of Intrusions and Malware,*
150   *and Vulnerability Assessment*: Springer, 62-81.

151   Ikram M, Beaume P, and Kâafar MA. 2019. Dadidroid: An obfuscation resilient tool for detecting
152   android malware via weighted directed call graph modelling. *arXiv preprint*
153   *arXiv:190509136*.

154   Jang JW, Kang H, Woo J, Mohaisen A, and Kim HK. 2016. Andro-Dumpsys: Anti-malware system
155   based on the similarity of malware creator and malware centric information. *Computers &*
156   *Security* 58:125-138. 10.1016/j.cose.2015.12.005

157   Jeong J, Seo D, Lee C, Kwon J, Lee H, and Milburn J. 2014. MysteryChecker: Unpredictable
158   attestation to detect repackaged malicious applications in Android. 9th IEEE International
159   Conference on Malicious and Unwanted Software, MALCON 2014: Institute of Electrical
160   and Electronics Engineers Inc. p 50-57.

161   Jung J, Kim H, Shin D, Lee M, Lee H, Cho S-j, and Suh K. 2018. Android malware detection
162   based on useful API calls and machine learning. 2018 IEEE First International Conference
163   on Artificial Intelligence and Knowledge Engineering (AIKE): IEEE. p 175-178.

164   Kandukuru S, and Sharma R. 2018. PNSDroid: A Hybrid Approach for Detection of Android
165   Malware. *Recent Findings in Intelligent Computing Techniques*: Springer, 361-367.

166   Kang H, Jang JW, Mohaisen A, and Kim HK. 2015. Detecting and classifying android malware
167   using static analysis along with creator information. *International Journal of Distributed*
168   *Sensor Networks* 2015. 10.1155/2015/479174

169   Karbab EB, and Debbabi M. 2021. Resilient and Adaptive Framework for Large Scale Android
170   Malware Fingerprinting using Deep Learning and NLP Techniques. *arXiv preprint*
171   *arXiv:210513491*.

172   Kim D, Kwak J, and Ryou J. 2015. DWroidDump: Executable Code Extraction from Android
173   Applications for Malware Analysis. *International Journal of Distributed Sensor Networks*
174   2015. 10.1155/2015/379682

175   Kim T, Kang B, Rho M, Sezer S, and Im EG. 2019. A Multimodal Deep Learning Method for
176   Android Malware Detection Using Various Features. *IEEE Transactions on Information*
177   *Forensics and Security* 14:773-788.

178   Kuhnel M, Smieschek M, and Meyer U. 2015. Fast identification of obfuscation and mobile
179   advertising in mobile malware. 14th IEEE International Conference on Trust, Security and
180   Privacy in Computing and Communications, TrustCom 2015: Institute of Electrical and
181   Electronics Engineers Inc. p 214-221.

182   Kwon J, Jeong J, Lee J, and Lee H. 2014. DroidGraph: Discovering Android malware by analyzing

183   semantic behavior. 2014 IEEE Conference on Communications and Network Security,
184   CNS 2014: Institute of Electrical and Electronics Engineers Inc. p 498-499.

185   Lee C, Kim J, Cho S-j, Choi J, and Park Y. 2014. Unified security enhancement framework for the
186   Android operating system. *Journal of Supercomputing* 67:738-756. 10.1007/s11227-013-
187   0991-y

188   Lei C, Gates CS, Luo S, and Ninghui L. 2015. A Probabilistic Discriminative Model for Android
189   Malware Detection with Decompiled Source Code. *Dependable and Secure Computing,*
190   *IEEE Transactions on* 12:400-412. 10.1109/TDSC.2014.2355839

191   Li J, Sun L, Yan Q, Li Z, Srisa-an W, and Ye H. 2018. Significant Permission Identification for
192   Machine Learning Based Android Malware Detection. *IEEE Transactions on Industrial*
193   *Informatics*.

194   Li L, Bissyandé TF, Octeau D, and Klein J. 2016. DroidRA: Taming reflection to support whole-
195   program analysis of android apps. In: Roychoudhury A, and Zeller A, editors. 25th
196   International Symposium on Software Testing and Analysis, ISSTA 2016: Association for
197   Computing Machinery, Inc. p 318-329.

198   Li Z, Sun J, Yan Q, Srisa-an W, and Tsutano Y. 2019. Obfusifier: Obfuscation-resistant Android
199   malware detection system. International Conference on Security and Privacy in
200   Communication Systems: Springer. p 214-234.

201   Lindorfer M, Neugschwandtner M, and Platzer C. 2015. MARVIN: Efficient and Comprehensive
202   Mobile App Classification through Static and Dynamic Analysis. In: Huang G, Yang J,
203   Ahamed SI, Hsiung PA, Chang CK, Chu W, and Crnkovic I, editors. 39th IEEE Annual
204   Computer Software and Applications Conference, COMPSAC 2015: IEEE Computer
205   Society. p 422-433.

206   Liu Z, Wang R, Japkowicz N, Tang D, Zhang W, and Zhao J. 2021. Research on unsupervised
207   feature learning for Android malware detection based on Restricted Boltzmann Machines.
208   *Future Generation Computer Systems* 120:91-108.

209   Lu T, Du Y, Ouyang L, Chen Q, and Wang X. 2020. Android malware detection based on a hybrid
210   deep learning model. *Security and Communication Networks* 2020.

211   Ma Z, Ge H, Wang Z, Liu Y, and Liu X. 2020. Droidetec: Android Malware Detection and Malicious
212   Code Localization through Deep Learning. *arXiv preprint arXiv:200203594*.

213   Maggi F, Valdi A, and Zanero S. 2013. AndroTotal: A flexible, scalable toolbox and service for
214   testing mobile malware detectors. 3rd ACM Workshop on Security and Privacy in
215   Smartphones and Mobile Devices, SPSM 2013, Held in Association with the 20th ACM
216   Conference on Computer and Communications Security, CCS 2013. Berlin. p 49-54.

217   Mahindru A, and Sangal A. 2021. MLDroid—Framework for Android malware detection using
218   machine learning techniques. *Neural Computing and Applications* 33:5183-5240.

219   Maier D, Protsenko M, and Müller T. 2015. A game of Droid and Mouse: The threat of split-
220   personality malware on Android. *Computers and Security* 54:2-15.
221   10.1016/j.cose.2015.05.001

222   Maiorca D, Mercaldo F, Giacinto G, Visaggio CA, and Martinelli F. 2017. R-PackDroid: API
223   package-based characterization and detection of mobile ransomware. Proceedings of the
224   symposium on applied computing: ACM. p 1718-1723.

225   Mantoo BA, and Khurana SS. 2020. Static, Dynamic and Intrinsic Features Based Android
226   Malware Detection Using Machine Learning. *Proceedings of ICRIC 2019*: Springer, 31-
227   45.

228 Markmann T, Gessner D, and Westhoff D. 2013. QuantDroid: Quantitative approach towards
229   mitigating privilege escalation on Android. Communications (ICC), 2013 IEEE
230   International Conference on. p 2144-2149.

231 Martín A, Menéndez HD, and Camacho D. 2016. MOCDroid: multi-objective evolutionary
232   classifier for Android malware detection. *Soft Computing*:1-11. 10.1007/s00500-016-
233   2283-y

234 Millar S, McLaughlin N, del Rincon JM, and Miller P. 2021. Multi-view deep learning for zero-day
235   Android malware detection. *Journal of Information Security and Applications* 58:102718.

236 Mirzaei O, de Fuentes JM, Tapiador J, and Gonzalez-Manzano L. 2019. AndrODet: An adaptive
237   Android obfuscation detector. *Future Generation Computer Systems* 90:240-261.
238   https://doi.org/10.1016/j.future.2018.07.066

239 Narayanan A, Chen L, and Chan CK. 2014. AdDetect: Automated detection of Android ad libraries
240   using semantic analysis. 9th IEEE International Conference on Intelligent Sensors, Sensor
241   Networks and Information Processing, IEEE ISSNIP 2014. Singapore: IEEE Computer
242   Society.

243 Nawaz A. 2021. Feature Engineering based on Hybrid Features for Malware Detection over
244   Android Framework. *Turkish Journal of Computer and Mathematics Education
245   (TURCOMAT)* 12:2856-2864.

246 Ng DV, and Hwang JIG. 2015. Android malware detection using the dendritic cell algorithm. 13th
247   International Conference on Machine Learning and Cybernetics, ICMLC 2014: IEEE
248   Computer Society. p 257-262.

249 Pektas A, and Acarman T. 2014. A dynamic malware analyzer against virtual machine aware
250   malicious software. *Security and Communication Networks* 7:2245-2257.
251   10.1002/sec.931

252 Pektaş A, and Acarman T. 2020. Deep learning for effective Android malware detection using API
253   call graph embeddings. *Soft Computing* 24:1027-1043.

254 Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, and Ioannidis S. 2014. Rage against
255   the virtual machine: Hindering dynamic analysis of android malware. 7th European
256   Workshop on System Security, EuroSec 2014. Amsterdam: Association for Computing
257   Machinery.

258 Poeplau S, Fratantonio Y, Bianchi A, Kruegel C, and Vigna G. 2014. Execute this! analyzing
259   unsafe and malicious dynamic code loading in android applications. Proceedings of the
260   20th Annual Network & Distributed System Security Symposium (NDSS).

261 Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
262   systematization of knowledge and unified methodology. *Journal of Computer Virology and
263   Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2

264 Şahin DÖ, Kural OE, Akleylek S, and Kılıç E. 2021. A novel permission-based Android malware
265   detection system using feature selection based on linear regression. *Neural Computing
266   and Applications*:1-16.

267 Salva S, and Zafimiharisoa SR. 2015. APSET, an Android aPplication SEcurity Testing tool for
268   detecting intent-based vulnerabilities. *International Journal on Software Tools for
269   Technology Transfer* 17:201-221. 10.1007/s10009-014-0303-8

270 Sanz B, Santos I, Laorden C, Ugarte-Pedrero X, Nieves J, Bringas PG, and Maranon GA. 2013.
271   Mama: Manifest Analysis for Malware Detection in Android. *Cybernetics and Systems*
272   44:469-488. 10.1080/01969722.2013.803889

273 Sasidharan SK, and Thomas C. 2021. ProDroid—An Android malware detection framework
274   based on profile hidden Markov model. *Pervasive and Mobile Computing* 72:101336.

275 Shabtai A, Tenenboim-Chekina L, Mimran D, Rokach L, Shapira B, and Elovici Y. 2014. Mobile
276   malware detection through analysis of deviations in application network behavior.
277   *Computers & Security* 43:1-18.

278 Shalaginov A, and Franke K. 2014. Automatic rule-mining for malware detection employing
279   Neuro-Fuzzy Approach. Norsk informasjonssikkerhetskonferanse (NISK).

280 Shao Y, Luo X, Qian C, Zhu P, and Zhang L. 2014. Towards a scalable resource-driven approach
281   for detecting repackaged android applications. 30th Annual Computer Security
282   Applications Conference, ACSAC 2014. December ed: Association for Computing
283   Machinery. p 56-65.

284 Sheen S, Anitha R, and Natarajan V. 2015. Android based malware detection using a multifeature
285   collaborative decision fusion approach. *Neurocomputing* 151:905-912.
286   10.1016/j.neucom.2014.10.004

287 Shen T, Zhongyang Y, Xin Z, Mao B, and Huang H. 2015. Detect android malware variants using
288   component based topology graph. 13th IEEE International Conference on Trust, Security
289   and Privacy in Computing and Communications, TrustCom 2014: Institute of Electrical and
290   Electronics Engineers Inc. p 406-413.

291 Shrestha B, Ma D, Zhu Y, Li H, and Saxena N. 2015. Tap-Wave-Rub: Lightweight Human
292   Interaction Approach to Curb Emerging Smartphone Malware. *Information Forensics and
293   Security, IEEE Transactions on* PP:1-1. 10.1109/TIFS.2015.2436364

294 Sihag V, Vardhan M, and Singh P. 2021a. BLADE: Robust malware detection against obfuscation
295   in android. *Forensic Science International: Digital Investigation* 38:301176.

296 Sihag V, Vardhan M, Singh P, Choudhary G, and Son S. 2021b. De-LADY: Deep learning based
297   Android malware detection using Dynamic features. *Journal of Internet Services and
298   Information Security (JISIS)* 11:34-45.

299 Singh S, Mishra B, and Singh S. 2015. Detecting intelligent malware on dynamic Android analysis
300   environments. 10th International Conference for Internet Technology and Secured
301   Transactions, ICITST 2015: Institute of Electrical and Electronics Engineers Inc. p 414-
302   419.

303 Soh C, Tan HBK, Arnatovich YL, and Wang L. 2015. Detecting Clones in Android Applications
304   through Analyzing User Interfaces. 23rd IEEE International Conference on Program
305   Comprehension, ICPC 2015: IEEE Computer Society. p 163-173.

306 Spreitzenbarth M, Schreck T, Echtler F, Arp D, and Hoffmann J. 2015. Mobile-Sandbox:
307   combining static and dynamic analysis with machine-learning techniques. *International
308   Journal of Information Security* 14:141-153. 10.1007/s10207-014-0250-0

309 Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, and Cavallaro L. 2017. DroidSieve:
310   Fast and Accurate Classification of Obfuscated Android Malware. Seventh ACM on
311   Conference on Data and Application Security and Privacy. Scottsdale, Arizona, USA:
312   ACM. p 309-320.

313 Suarez-Tangil G, Tapiador JE, Peris-Lopez P, and Blasco J. 2014. Dendroid: A text mining
314   approach to analyzing and classifying code structures in Android malware families. *Expert
315   Systems with Applications* 41:1104-1117.

316 Sufatrio, Chua TW, Tan DJJ, and Thing VLL. 2015. Accurate specification for robust detection of
317   malicious behavior in mobile environments. In: Weippl E, Pernul G, and Ryan PYA,

318   editors. 20th European Symposium on Research in Computer Security, ESORICS 2015:
319   Springer Verlag. p 355-375.

320 Sun M, Li M, and Lui JCS. 2015. DroidEagle: Seamless detection of visually similar android apps.
321   8th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec
322   2015: Association for Computing Machinery, Inc.

323 Surendran R, Thomas T, and Emmanuel S. 2020. A TAN based hybrid model for android malware
324   detection. *Journal of Information Security and Applications* 54:102483.

325 Taha AA, and Malebary SJ. 2021. Hybrid classification of Android malware based on fuzzy
326   clustering and the gradient boosting machine. *Neural Computing and Applications*
327   33:6721-6732.

328 Taheri R, Ghahramani M, Javidan R, Shojafar M, Pooranian Z, and Conti M. 2020. Similarity-
329   based Android malware detection using Hamming distance of static binary features.
330   *Future Generation Computer Systems* 105:230-247.

331 Talha KA, Alper DI, and Aydin C. 2015. APK Auditor: Permission-based Android malware
332   detection system. *Digital Investigation* 13:1-14. 10.1016/j.diin.2015.01.001

333 Tiwari SR, and Shukla RU. 2018. An android malware detection technique based on optimized
334   permissions and API. 2018 International Conference on Inventive Research in Computing
335   Applications (ICIRCA): IEEE. p 258-263.

336 Wang S, Chen Z, Yan Q, Ji K, Peng L, Yang B, and Conti M. 2020. Deep and broad URL feature
337   mining for android malware detection. *Information Sciences* 513:600-613.

338 Wang S, Chen Z, Yan Q, Yang B, Peng L, and Jia Z. 2019. A mobile malware detection method
339   using behavior features in network traffic. *Journal of Network and Computer Applications*
340   133:15-25.

341 Wang X, and Li C. 2021. Android malware detection through machine learning on kernel task
342   structures. *Neurocomputing* 435:126-150.

343 Wang Z, Li C, Guan Y, and Xue Y. 2016. Anti-obfuscation method for detecting similarity of
344   Android application. *Huazhong Keji Daxue Xuebao (Ziran Kexue Ban)/Journal of*
345   *Huazhong University of Science and Technology (Natural Science Edition)* 44:60-64 and
346   76. 10.13245/j.hust.160312

347 Wei TE, Tyan HR, Jeng AB, Lee HM, Liao HYM, and Wang JC. 2015. DroidExec: Root exploit
348   malware recognition against wide variability via folding redundant function-relation graph.
349   17th IEEE International Conference on Advanced Communications Technology, ICACT
350   2015: Institute of Electrical and Electronics Engineers Inc. p 161-169.

351 Wu D-J, Mao C-H, Wei T-E, Lee H-M, and Wu K-P. 2012. Droidmat: Android malware detection
352   through manifest and api calls tracing. Information Security (Asia JCIS), 2012 Seventh
353   Asia Joint Conference on: IEEE. p 62-69.

354 Wu S, Wang P, Li X, and Zhang Y. 2016. Effective Detection of Android Malware Based on the
355   Usage of Data Flow APIs and Machine Learning. *Information and Software Technology*.
356   10.1016/j.infsof.2016.03.004

357 Wu X, Zhang D, Su X, and Li W. 2015. Detect repackaged Android application based on HTTP
358   traffic similarity. *Security and Communication Networks* 8:2257-2266. 10.1002/sec.1170

359 Xu J, Yu Y, Chen Z, Cao B, Dong W, Guo Y, and Cao J. 2013. MobSafe: cloud computing based
360   forensic analysis for massive mobile applications using data mining. *Tsinghua Science*
361   *and Technology* 18. 10.1109/TST.2013.6574680

362 Yang Y, Du X, Yang Z, and Liu X. 2021. Android Malware Detection Based on Structural Features
363   of the Function Call Graph. *Electronics* 10:186.

364 Yerima SY, Sezer S, and Muttik I. 2014. Android Malware Detection Using Parallel Machine
365   Learning Classifiers. Next Generation Mobile Apps, Services and Technologies
366   (NGMAST), 2014 Eighth International Conference on. p 37-42.

367 Yuan Z, Lu Y, Wang Z, and Xue Y. 2014a. Droid-Sec: deep learning in android malware detection.
368   *SIGCOMM Comput Commun Rev* 44:371-372. 10.1145/2740070.2631434

369 Yuan Z, Lu Y, and Xue Y. 2016. Droiddetector: android malware characterization and detection
370   using deep learning. *Tsinghua Science and Technology* 21:114-123.
371   10.1109/TST.2016.7399288

372 Yuan Z, Min Y, Zhemin Y, Guofei G, Peng N, and Binyu Z. 2014b. Permission Use Analysis for
373   Vetting Undesirable Behaviors in Android Apps. *Information Forensics and Security, IEEE*
374   *Transactions on* 9:1828-1842. 10.1109/TIFS.2014.2347206

375 Zhang F, Huang H, Zhu S, Wu D, and Liu P. 2014. ViewDroid: Towards obfuscation-resilient
376   mobile application repackaging detection. 7th ACM Conference on Security and Privacy
377   in Wireless and Mobile Networks, WiSec 2014. Oxford: Association for Computing
378   Machinery. p 25-36.

379 Zhang W, Wang H, He H, and Liu P. 2020. DAMBA: Detecting Android Malware by ORGB
380   Analysis. *IEEE Transactions on Reliability* 69:55-69.

381 Zhang X, Breitinger F, and Baggili I. 2016. Rapid Android Parser for Investigating DEX files
382   (RAPID). *Digital Investigation* 17:28-39. 10.1016/j.diin.2016.03.002

383 Zhang Y, Luo X, and Yin H. 2015. Dexhunter: toward extracting hidden code from packed android
384   applications. *Computer Security--ESORICS 2015*: Springer, 293-311.

385 Zhao S, Li X, Xu G, Zhang L, and Feng Z. 2014. Attack tree based android malware detection
386   with hybrid analysis. 13th IEEE International Conference on Trust, Security and Privacy in
387   Computing and Communications, TrustCom 2014: Institute of Electrical and Electronics
388   Engineers Inc. p 380-387.

389 Zhauniarovich Y, Ahmad M, Gadyatskaya O, Crispo B, and Massacci F. 2015. StaDynA:
390   Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android
391   Applications. Proceedings of the 5th ACM Conference on Data and Application Security
392   and Privacy. San Antonio, Texas, USA: ACM. p 37-48.

393 Zhong Y, Yamaki H, Yamaguchi Y, and Takakura H. 2013. ARIGUMA Code Analyzer: Efficient
394   Variant Detection by Identifying Common Instruction Sequences in Malware Families.
395   2013 Ieee 37th Annual Computer Software and Applications Conference (Compsac). p
396   11-20.

397 Zhou W, Zhou Y, Grace M, Jiang X, and Zou S. 2013. Fast, scalable detection of piggybacked
398   mobile applications. Proceedings of the third ACM conference on Data and application
399   security and privacy: ACM. p 185-196.

400 Zhu H, Li Y, Li R, Li J, You Z-H, and Song H. 2020. Sedmdroid: An enhanced stacking ensemble
401   of deep learning framework for android malware detection. *IEEE Transactions on Network*
402   *Science and Engineering*.

403 Zhu H, Wang L, Zhong S, Li Y, and Sheng VS. 2021. A Hybrid Deep Network Framework for
404   Android Malware Detection. *IEEE Transactions on Knowledge and Data Engineering*.

405 Zou D, Wu Y, Yang S, Chauhan A, Yang W, Zhong J, Dou S, and Jin H. 2021. IntDroid: Android
406   Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software*

407   *Engineering and Methodology (TOSEM)* 30:1-32.

408

409

Polymorphism evaluation of frameworks

			Android Malware Detection Frameworks		
			Static	Dynamic	Hybrid
Polymorphism	Package transformation	Repacking (RPK)	DroidMat (Wu et al. 2012), DroidOLytics (Faruki et al. 2013), ViewDroid (Zhang et al. 2014), DroidGraph (Kwon et al. 2014), MysteryChecker(Jeong et al. 2014), Chen (Chen et al. 2015), Dempster–Shafe (Du et al. 2015), DroidExec(Wei et al. 2015), AnDarwin and DNADroid(Crussell et al. 2015), AndroSimilar (Faruki et al. 2015), Ngrams (Canfora et al. 2015), DroidEagle(Sun et al. 2015), Droidkin (Gonzalez et al. 2014), Gurulian (Gurulian et al. 2016, AAMO (Preda & Maggi 2016), AndroDet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021), Amin (Amin et al. 2020), BLADE (Sihag et al. 2021), Dadidroid (Ikram et al. 2019, Obfusifier (Li et al. 2019)	Soh (Soh et al. 2015)	NA
		Package Renaming (PKR)	DroidMat (Wu et al. 2012), DroidOLytics (Faruki et al. 2013), Chen (Chen et al. 2015), AnDarwin and DNADroid(Crussell et al. 2015), AndroSimilar (Faruki et al. 2015), Ngrams (Canfora et al. 2015), Droidkin (Gonzalez et al. 2014), Gurulian (Gurulian et al. 2016), AAMO (Preda & Maggi 2016), Battista (Battista et al. 2016), Obsifier (Li et al. 2019), Kim(Kim et al. 2019), Dadidroid(Ikram et al. 2019), Balde (Sihag et al. 2021), Dharmalingam (Dharmalingam &	Soh (Soh et al. 2015)	Abaid (Abaid et al. 2017)

		Android Malware Detection Frameworks		
		Static	Dynamic	Hybrid
		Palanisamy 2021), Karbab (Karbab & Debbabi 2021), AndrODet (Mirzaei et al. 2019)		
	Identifier Renaming (IDR)	DroidMat (Wu et al. 2012), Chen (Chen et al. 2015), Ngrams (Canfora et al. 2015), SeqMalSpec -Sufatrio (Sufatrio et al. 2015), Droidkin (Gonzalez et al. 2014), Shen (Shen et al. 2015), Kuhnel (Kuhnel et al. 2015), Gurulian (Gurulian et al. 2016), AAMO (Preda & Maggi 2016), Battista (Battista et al. 2016), AndrODet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021), Dharmalingam (Dharmalingam & Palanisamy 2021), Kim (Kim et al. 2019), Dadidroid (Ikram et al. 2019), Obfusifier (Li et al. 2019)	Soh (Soh et al. 2015), Wu 2015 (Wu et al. 2015)	
Encrypti	Data Encryptio n (DEN)	DroidMat (Wu et al. 2012), MysteryChecker(Jeong et al. 2014), Dexhunter(Zhang et al. 2015), AndroSimilar (Faruki et al. 2015), Droidkin (Gonzalez et al. 2014), Shen (Shen et al. 2015), SherlockDroid (Apvrille & Apvrille 2015), Kuhnel (Kuhnel et al. 2015), AAMO (Preda & Maggi 2016), AndrODet (Mirzaei et al. 2019), DroidSieve (Suarez-Tangil et al. 2017), AndrODet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021),Intdroid (Zou et al. 2021), BLADE (Sihag et al. 2021), Alazab (Alazab et al. 2020), Kim (Kim et al. 2019), Dadidroid	DwroidDump(Kim et al. 2015)	RiskRanker(Grace et al. 2012) , Mobile-Sandbox (Spreitzenbarth et al. 2015)

rpppr

4   References

8   Aafer Y, Du WL, and Yin H. 2013. DroidAPIMiner: Mining API-Level Features for Robust Malware
9   Detection in Android. *Security and Privacy in Communication Networks, Securecomm*
10   *2013* 127:86-103.

11   Abaid Z, Kaafar MA, and Jha S. 2017. Quantifying the impact of adversarial evasion attacks on
12   machine learning based android malware classifiers. Network Computing and
13   Applications (NCA), 2017 IEEE 16th International Symposium on: IEEE. p 1-10.

14   Alazab M, Alazab M, Shalaginov A, Mesleh A, and Awajan A. 2020. Intelligent mobile malware
15   detection using permission requests and api calls. *Future Generation Computer Systems*
16   107:509-521.

17   Amin M, Tanveer TA, Tehseen M, Khan M, Khan FA, and Anwar S. 2020. Static malware
18   detection and attribution in android byte-code through an end-to-end deep system. *Future*
19   *Generation Computer Systems* 102:112-126.

20   Apvrille A, and Apvrille L. 2015. SherlockDroid: a research assistant to spot unknown malware in
21   Android marketplaces. *Journal of Computer Virology and Hacking Techniques* 11:235-
22   245. 10.1007/s11416-015-0245-z

23   Battista P, Mercaldo F, Nardone V, Santone A, and Visaggio CA. 2016. Identification of android
24   malware families with model checking. In: Furnell S, Camp O, and Mori P, editors. 2nd
25   International Conference on Information Systems Security and Privacy, ICISSP 2016:
26   SciTePress. p 542-547.

27   Canfora G, De Lorenzo A, Medvet E, Mercaldo F, and Visaggio CA. 2015. Effectiveness of opcode
28   ngrams for detection of multi family android malware. Availability, Reliability and Security
29   (ARES), 2015 10th International Conference on: IEEE. p 333-340.

30   Chen K, Wang P, Lee Y, Wang X, Zhang N, Huang H, Zou W, and Liu P. 2015. Finding unknown
31   malice in 10 seconds: Mass vetting for new threats at the google-play scale. 24th USENIX
32   Security Symposium (USENIX Security 15). p 659-674.

33   Crussell J, Gibler C, and Chen H. 2015. AnDarwin: Scalable Detection of Android Application
34   Clones Based on Semantics. *IEEE Transactions on Mobile Computing* 14:2007-2019.
35   10.1109/TMC.2014.2381212

36   Dharmalingam VP, and Palanisamy V. 2021. A novel permission ranking system for android
37   malware detection—the permission grader. *Journal of Ambient Intelligence and*
38   *Humanized Computing* 12:5071-5081.

39   Du Y, Wang X, and Wang J. 2015. A static Android malicious code detection method based on
40   multi-source fusion. *Security and Communication Networks* 8:3238-3246.
41   10.1002/sec.1248

42   Faruki P, Laxmi V, Bharmal A, Gaur MS, and Ganmoor V. 2015. AndroSimilar: Robust signature
43   for detecting variants of Android malware. *Journal of Information Security and Applications*
44   22:66-80. 10.1016/j.jisa.2014.10.011

45   Faruki P, Laxmi V, Ganmoor V, Gaur MS, and Bharmal A. 2013. DroidOLytics: Robust feature
46   signature for repackaged android apps on official and third party android markets. 2nd
47   International Conference on Advanced Computing, Networking and Security, ADCONS

48   2013. Surathkal, Karnataka: IEEE Computer Society. p 247-252.

49   Gonzalez H, Stakhanova N, and Ghorbani AA. 2014. Droidkin: Lightweight detection of android
50   apps similarity. International Conference on Security and Privacy in Communication
51   Systems: Springer. p 436-453.

52   Grace M, Zhou Y, Zhang Q, Zou S, and Jiang X. 2012. Riskranker: scalable and accurate zero-
53   day android malware detection. Proceedings of the 10th international conference on
54   Mobile systems, applications, and services: ACM. p 281-294.

55   Gurulian I, Markantonakis K, Cavalaro L, and Mayes K. 2016. You can't touch this: Consumer-
56   centric android application repackaging detection. *Future Generation Computer Systems*
57   65:1-9. 10.1016/j.future.2016.05.021

58   Ikram M, Beaume P, and Kâafar MA. 2019. Dadidroid: An obfuscation resilient tool for detecting
59   android malware via weighted directed call graph modelling. *arXiv preprint*
60   *arXiv:190509136*.

61   Jeong J, Seo D, Lee C, Kwon J, Lee H, and Milburn J. 2014. MysteryChecker: Unpredictable
62   attestation to detect repackaged malicious applications in Android. 9th IEEE International
63   Conference on Malicious and Unwanted Software, MALCON 2014: Institute of Electrical
64   and Electronics Engineers Inc. p 50-57.

65   Karbab EB, and Debbabi M. 2021. Resilient and Adaptive Framework for Large Scale Android
66   Malware Fingerprinting using Deep Learning and NLP Techniques. *arXiv preprint*
67   *arXiv:210513491*.

68   Kim D, Kwak J, and Ryou J. 2015. DWroidDump: Executable Code Extraction from Android
69   Applications for Malware Analysis. *International Journal of Distributed Sensor Networks*
70   2015. 10.1155/2015/379682

71   Kim T, Kang B, Rho M, Sezer S, and Im EG. 2019. A Multimodal Deep Learning Method for
72   Android Malware Detection Using Various Features. *IEEE Transactions on Information*
73   *Forensics and Security* 14:773-788.

74   Kuhnel M, Smieschek M, and Meyer U. 2015. Fast identification of obfuscation and mobile
75   advertising in mobile malware. 14th IEEE International Conference on Trust, Security and
76   Privacy in Computing and Communications, TrustCom 2015: Institute of Electrical and
77   Electronics Engineers Inc. p 214-221.

78   Kwon J, Jeong J, Lee J, and Lee H. 2014. DroidGraph: Discovering Android malware by analyzing
79   semantic behavior. 2014 IEEE Conference on Communications and Network Security,
80   CNS 2014: Institute of Electrical and Electronics Engineers Inc. p 498-499.

81   Li Z, Sun J, Yan Q, Srisa-an W, and Tsutano Y. 2019. Obfusifier: Obfuscation-resistant Android
82   malware detection system. International Conference on Security and Privacy in
83   Communication Systems: Springer. p 214-234.

84   Mirzaei O, de Fuentes JM, Tapiador J, and Gonzalez-Manzano L. 2019. AndrODet: An adaptive
85   Android obfuscation detector. *Future Generation Computer Systems* 90:240-261.
86   https://doi.org/10.1016/j.future.2018.07.066

87   Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
88   systematization of knowledge and unified methodology. *Journal of Computer Virology and*
89   *Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2

90   Shen T, Zhongyang Y, Xin Z, Mao B, and Huang H. 2015. Detect android malware variants using
91   component based topology graph. 13th IEEE International Conference on Trust, Security
92   and Privacy in Computing and Communications, TrustCom 2014: Institute of Electrical and

93   Electronics Engineers Inc. p 406-413.

94   Sihag V, Vardhan M, and Singh P. 2021. BLADE: Robust malware detection against obfuscation
95   in android. *Forensic Science International: Digital Investigation* 38:301176.

96   Soh C, Tan HBK, Arnatovich YL, and Wang L. 2015. Detecting Clones in Android Applications
97   through Analyzing User Interfaces. 23rd IEEE International Conference on Program
98   Comprehension, ICPC 2015: IEEE Computer Society. p 163-173.

99   Spreitzenbarth M, Schreck T, Echtler F, Arp D, and Hoffmann J. 2015. Mobile-Sandbox:
100   combining static and dynamic analysis with machine-learning techniques. *International*
101   *Journal of Information Security* 14:141-153. 10.1007/s10207-014-0250-0

102   Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, and Cavallaro L. 2017. DroidSieve:
103   Fast and Accurate Classification of Obfuscated Android Malware. Seventh ACM on
104   Conference on Data and Application Security and Privacy. Scottsdale, Arizona, USA:
105   ACM. p 309-320.

106   Sufatrio, Chua TW, Tan DJJ, and Thing VLL. 2015. Accurate specification for robust detection of
107   malicious behavior in mobile environments. In: Weippl E, Pernul G, and Ryan PYA,
108   editors. 20th European Symposium on Research in Computer Security, ESORICS 2015:
109   Springer Verlag. p 355-375.

110   Sun M, Li M, and Lui JCS. 2015. DroidEagle: Seamless detection of visually similar android apps.
111   8th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec
112   2015: Association for Computing Machinery, Inc.

113   Wang Z, Li C, Guan Y, and Xue Y. 2016. Anti-obfuscation method for detecting similarity of
114   Android application. *Huazhong Keji Daxue Xuebao (Ziran Kexue Ban)/Journal of*
115   *Huazhong University of Science and Technology (Natural Science Edition)* 44:60-64 and
116   76. 10.13245/j.hust.160312

117   Wei TE, Tyan HR, Jeng AB, Lee HM, Liao HYM, and Wang JC. 2015. DroidExec: Root exploit
118   malware recognition against wide variability via folding redundant function-relation graph.
119   17th IEEE International Conference on Advanced Communications Technology, ICACT
120   2015: Institute of Electrical and Electronics Engineers Inc. p 161-169.

121   Wu D-J, Mao C-H, Wei T-E, Lee H-M, and Wu K-P. 2012. Droidmat: Android malware detection
122   through manifest and api calls tracing. Information Security (Asia JCIS), 2012 Seventh
123   Asia Joint Conference on: IEEE. p 62-69.

124   Wu X, Zhang D, Su X, and Li W. 2015. Detect repackaged Android application based on HTTP
125   traffic similarity. *Security and Communication Networks* 8:2257-2266. 10.1002/sec.1170

126   Zhang F, Huang H, Zhu S, Wu D, and Liu P. 2014. ViewDroid: Towards obfuscation-resilient
127   mobile application repackaging detection. 7th ACM Conference on Security and Privacy
128   in Wireless and Mobile Networks, WiSec 2014. Oxford: Association for Computing
129   Machinery. p 25-36.

130   Zhang Y, Luo X, and Yin H. 2015. Dexhunter: toward extracting hidden code from packed android
131   applications. *Computer Security--ESORICS 2015*: Springer, 293-311.

132   Zou D, Wu Y, Yang S, Chauhan A, Yang W, Zhong J, Dou S, and Jin H. 2021. IntDroid: Android
133   Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software*
134   *Engineering and Methodology (TOSEM)* 30:1-32.

135

136

Metamorphism evaluation of frameworks

			Android Malware Detection Frameworks		
			Static	Dynamic	Hybrid
Metamorphism	Code obfuscation	Code Reordering (CRE)	DroidOLytics (Faruki et al. 2013), DroidGraph (Kwon et al. 2014), MysteryChecker(Jeong et al. 2014), AdDetect (Narayanan et al. 2014), ResDroid (Shao et al. 2014), Apk Auditor (Talha et al. 2015), Dempster–Shafe (Du et al. 2015), Dexhunter(Zhang et al. 2015), DroidExec(Wei et al. 2015), AnDarwin and DNADroid(Crussell et al. 2015), AndroSimilar (Faruki et al. 2015), SeqMalSpec -Sufatrio (Sufatrio et al. 2015), DroidEagle(Sun et al. 2015), Shen (Shen et al. 2015), Gurulian (Gurulian et al. 2016), AAMO (Preda & Maggi 2016), Wang (Wang et al. 2016), MocDroid (Martín et al. 2016), Battista (Battista et al. 2016), DroidSieve (Suarez-Tangil et al. 2017), AndrODet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021), Obfusifier (Li et al. 2019), Kim (Kim et al. 2019), Dadidroid (Ikram et al. 2019)	Soh (Soh et al. 2015), Q-floid (Castellanos et al. 2016)	RiskRanker(Grace et al. 2012), MDetector(Zhao et al. 2014), MARVIN (Lindorfer et al. 2015), Mobile-Sandbox (Spreitzenbarth et al. 2015)
		Call Indirections (CIN)	DroidOLytics (Faruki et al. 2013), DroidGraph (Kwon et al. 2014),	Soh (Soh et al. 2015), Q-floid	RiskRanker(Grace et al. 2012), MDetector(Zhao

	Android Malware Detection Frameworks		
	Static	Dynamic	Hybrid
	AdDetect (Narayanan et al. 2014), Apk Auditor (Talha et al. 2015), Dempster–Shafe (Du et al. 2015), Dexhunter(Zhang et al. 2015), DroidExec(Wei et al. 2015), AnDarwin and DNADroid(Crussell et al. 2015), AndroSimilar (Faruki et al. 2015), DroidEagle(Sun et al. 2015), Shen (Shen et al. 2015), Gurulian (Gurulian et al. 2016), AAMO (Preda & Maggi 2016), Wang (Wang et al. 2016), MocDroid (Martín et al. 2016), Battista (Battista et al. 2016), DroidSieve (Suarez-Tangil et al. 2017), AndRODet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021), Obfusifier (Li et al. 2019)	(Castellanos et al. 2016)	et al. 2014), MARVIN (Lindorfer et al. 2015)
Dead Code Insertion (DCI)	DroidOLytics (Faruki et al. 2013), DroidGraph (Kwon et al. 2014), AdDetect (Narayanan et al. 2014), Apk Auditor (Talha et al. 2015), Dempster–Shafe (Du et al. 2015), Dexhunter(Zhang et al. 2015), DroidExec(Wei et al. 2015), AndroSimilar (Faruki et al. 2015), DroidEagle(Sun et al. 2015), Shen (Shen et al. 2015), Gurulian (Gurulian	No dynamic frameworks	RiskRanker(Grace et al. 2012), ARIGUMA (Zhong et al. 2013), AMDetector(Zhao et al. 2014), MARVIN (Lindorfer et al. 2015)

		Android Malware Detection Frameworks		
		Static	Dynamic	Hybrid
Advanced Code transformation		et al. 2016), AAMO (Preda & Maggi 2016), Wang (Wang et al. 2016), MocDroid (Martín et al. 2016), Battista (Battista et al. 2016), DroidSieve (Suarez-Tangil et al. 2017), AndrODet (Mirzaei et al. 2019), Karbab (Karbab & Debbabi 2021), Obfusifier (Li et al. 2019), Alazab (Alazab et al. 2020), Pektas (Pektaş & Acarman 2020)		
	Native Exploits (NEX)	DroidAPIMiner (Aafer et al. 2013), AdDetect (Narayanan et al. 2014)	DroidBarrier (Almohri et al. 2014)	MARVIN (Lindorfer et al. 2015)
	Function Inlining and Outlining (FIO):	AAMO (Preda & Maggi 2016)	No Dynamic frameworks	No hybrid frameworks
	Reflection API(REF)	Juxtapp (Hanna et al. 2013), DroidAPIMiner (Aafer et al. 2013), Dexhunter(Zhang et al. 2015), SherlockDroid (Apvrille & Apvrille 2015), Kuhnel (Kuhnel et al. 2015), AAMO (Preda & Maggi 2016), DroidSieve (Suarez-Tangil et al. 2017), Yang (Yang et al. 2021), BLADE (Sihag et al. 2021), Karbab (Karbab & Debbabi 2021)	Maier (Maier et al. 2015), EnDroid (Feng et al. 2018)	RiskRanker(Grace et al. 2012), StaDyna (Zhauniarovich et al. 2015)
	Dynamic code loading (DCL)	DroidAPIMiner (Aafer et al. 2013), Yerima (Yerima et al. 2014), ResDroid (Shao et al. 2014), Poeplau	Maier (Maier et al. 2015), EnDroid	RiskRanker(Grace et al. 2012), StaDyna (Zhauniarovich et

Android Malware Detection Frameworks			
		Static	Dynamic	Hybrid
		(Poeplau et al. 2014), Dexhunter(Zhang et al. 2015), Grab 'n Run Falsina (Falsina et al. 2015), DroidSieve (Suarez-Tangil et al. 2017), Yang (Yang et al. 2021)	(Feng et al. 2018)	al. 2015), Abaid (Abaid et al. 2017)
	Anti-debugging (ADE)	Dexhunter(Zhang et al. 2015)		MARVIN (Lindorfer et al. 2015)
Anti-emulator	Virtual Machine Aware (VMA)	No static frameworks	Tao (Tao et al. 2012), DroidScope (Yan & Yin 2012), Pektas (Pektas & Acarman 2014), Maier (Maier et al. 2015), Singh (Singh et al. 2015), GroddDroid (Abraham et al. 2015), Alzaylaee (Alzaylaee et al. 2017)	RiskRanker(Grace et al. 2012), Petsas (Petsas et al. 2014), Tap-Wave-Rub (Shrestha et al. 2015)
	Programmed Interaction Detection (PID)	No static frameworks	Chaugule (Chaugule et al. 2011), Singh (Singh et al. 2015), GroddDroid (Abraham et al. 2015), Diao (Diao	Tap-Wave-Rub (Shrestha et al. 2015)

Android Malware Detection	Frameworks	
Static	Dynamic	Hybrid
	et al. 2016)	

References

Aafer Y, Du WL, and Yin H. 2013. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. *Security and Privacy in Communication Networks, Securecomm 2013* 127:86-103.

Abaid Z, Kaafar MA, and Jha S. 2017. Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers. Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on: IEEE. p 1-10.

Abraham A, Andriatsimandefitra R, Brunelat A, Lalande JF, and Viet Triem Tong V. 2015. GroddDroid: A gorilla for triggering malicious behaviors. 10th International Conference on Malicious and Unwanted Software, MALWARE 2015: Institute of Electrical and Electronics Engineers Inc. p 119-127.

Alazab M, Alazab M, Shalaginov A, Mesleh A, and Awajan A. 2020. Intelligent mobile malware detection using permission requests and api calls. *Future Generation Computer Systems* 107:509-521.

Almohri HMJ, Yao D, and Kafura D. 2014. DroidBarrier: Know what is executing on your Android. 4th ACM Conference on Data and Application Security and Privacy, CODASPY 2014. San Antonio, TX: Association for Computing Machinery. p 257-264.

Alzaylaee MK, Yerima SY, and Sezer S. 2017. Emulator vs real phone: Android malware detection using machine learning. Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics: ACM. p 65-72.

Apvrille A, and Apvrille L. 2015. SherlockDroid: a research assistant to spot unknown malware in Android marketplaces. *Journal of Computer Virology and Hacking Techniques* 11:235-245. 10.1007/s11416-015-0245-z

Battista P, Mercaldo F, Nardone V, Santone A, and Visaggio CA. 2016. Identification of android malware families with model checking. In: Furnell S, Camp O, and Mori P, editors. 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016: SciTePress. p 542-547.

Castellanos JH, Wüchner T, Ochoa M, and Rueda S. 2016. Q-floid: Android malware detection with quantitative data flow graphs. In: Roychoudhury A, and Mathur A, editors. 1st Singapore-Cybersecurity R and D Conference, SG-CRC 2016: IOS Press. p 13-25.

Chaugule A, Xu Z, and Zhu S. 2011. A Specification Based Intrusion Detection Framework for Mobile Phones. In: Lopez J, and Tsudik G, editors. Applied Cryptography and Network Security: Springer Berlin Heidelberg. p 19-37.

Crussell J, Gibler C, and Chen H. 2015. AnDarwin: Scalable Detection of Android Application Clones Based on Semantics. *IEEE Transactions on Mobile Computing* 14:2007-2019. 10.1109/TMC.2014.2381212

Diao W, Liu X, Li Z, and Zhang K. 2016. Evading android runtime analysis through detecting programmed interactions. 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2016: Association for Computing Machinery, Inc. p 159-164.

45 Du Y, Wang X, and Wang J. 2015. A static Android malicious code detection method based on
46   multi-source fusion. *Security and Communication Networks* 8:3238-3246.
47   10.1002/sec.1248

48 Falsina L, Fratantonio Y, Zanero S, Kruegel C, Vigna G, and Maggi F. 2015. Grab'n Run: Secure
49   and Practical Dynamic Code Loading for Android Applications. Proceedings of the 31st
50   Annual Computer Security Applications Conference: ACM. p 201-210.

51 Faruki P, Laxmi V, Bharmal A, Gaur MS, and Ganmoor V. 2015. AndroSimilar: Robust signature
52   for detecting variants of Android malware. *Journal of Information Security and Applications*
53   22:66-80. 10.1016/j.jisa.2014.10.011

54 Faruki P, Laxmi V, Ganmoor V, Gaur MS, and Bharmal A. 2013. DroidOLytics: Robust feature
55   signature for repackaged android apps on official and third party android markets. 2nd
56   International Conference on Advanced Computing, Networking and Security, ADCONS
57   2013. Surathkal, Karnataka: IEEE Computer Society. p 247-252.

58 Feng P, Ma J, Sun C, Xu X, and Ma Y. 2018. A novel dynamic Android malware detection system
59   with ensemble learning. *IEEE Access* 6:30996-31011.

60 Grace M, Zhou Y, Zhang Q, Zou S, and Jiang X. 2012. Riskranker: scalable and accurate zero-
61   day android malware detection. Proceedings of the 10th international conference on
62   Mobile systems, applications, and services: ACM. p 281-294.

63 Gurulian I, Markantonakis K, Cavalaro L, and Mayes K. 2016. You can't touch this: Consumer-
64   centric android application repackaging detection. *Future Generation Computer Systems*
65   65:1-9. 10.1016/j.future.2016.05.021

66 Hanna S, Huang L, Wu E, Li S, Chen C, and Song D. 2013. Juxtapp: A scalable system for
67   detecting code reuse among android applications. *Detection of Intrusions and Malware,*
68   *and Vulnerability Assessment*: Springer, 62-81.

69 Ikram M, Beaume P, and Kâafar MA. 2019. Dadidroid: An obfuscation resilient tool for detecting
70   android malware via weighted directed call graph modelling. *arXiv preprint*
71   *arXiv:190509136*.

72 Jeong J, Seo D, Lee C, Kwon J, Lee H, and Milburn J. 2014. MysteryChecker: Unpredictable
73   attestation to detect repackaged malicious applications in Android. 9th IEEE International
74   Conference on Malicious and Unwanted Software, MALCON 2014: Institute of Electrical
75   and Electronics Engineers Inc. p 50-57.

76 Karbab EB, and Debbabi M. 2021. Resilient and Adaptive Framework for Large Scale Android
77   Malware Fingerprinting using Deep Learning and NLP Techniques. *arXiv preprint*
78   *arXiv:210513491*.

79 Kim T, Kang B, Rho M, Sezer S, and Im EG. 2019. A Multimodal Deep Learning Method for
80   Android Malware Detection Using Various Features. *IEEE Transactions on Information*
81   *Forensics and Security* 14:773-788.

82 Kuhnel M, Smieschek M, and Meyer U. 2015. Fast identification of obfuscation and mobile
83   advertising in mobile malware. 14th IEEE International Conference on Trust, Security and
84   Privacy in Computing and Communications, TrustCom 2015: Institute of Electrical and
85   Electronics Engineers Inc. p 214-221.

86 Kwon J, Jeong J, Lee J, and Lee H. 2014. DroidGraph: Discovering Android malware by analyzing
87   semantic behavior. 2014 IEEE Conference on Communications and Network Security,
88   CNS 2014: Institute of Electrical and Electronics Engineers Inc. p 498-499.

89 Li Z, Sun J, Yan Q, Srisa-an W, and Tsutano Y. 2019. Obfusifier: Obfuscation-resistant Android
90   malware detection system. International Conference on Security and Privacy in
91   Communication Systems: Springer. p 214-234.

92 Lindorfer M, Neugschwandtner M, and Platzer C. 2015. MARVIN: Efficient and Comprehensive
93   Mobile App Classification through Static and Dynamic Analysis. In: Huang G, Yang J,
94   Ahamed SI, Hsiung PA, Chang CK, Chu W, and Crnkovic I, editors. 39th IEEE Annual
95   Computer Software and Applications Conference, COMPSAC 2015: IEEE Computer
96   Society. p 422-433.

97 Maier D, Protsenko M, and Müller T. 2015. A game of Droid and Mouse: The threat of split-
98   personality malware on Android. *Computers and Security* 54:2-15.
99   10.1016/j.cose.2015.05.001

100 Martín A, Menéndez HD, and Camacho D. 2016. MOCDroid: multi-objective evolutionary
101   classifier for Android malware detection. *Soft Computing*:1-11. 10.1007/s00500-016-
102   2283-y

103 Mirzaei O, de Fuentes JM, Tapiador J, and Gonzalez-Manzano L. 2019. AndrODet: An adaptive
104   Android obfuscation detector. *Future Generation Computer Systems* 90:240-261.
105   https://doi.org/10.1016/j.future.2018.07.066

106 Narayanan A, Chen L, and Chan CK. 2014. AdDetect: Automated detection of Android ad libraries
107   using semantic analysis. 9th IEEE International Conference on Intelligent Sensors, Sensor
108   Networks and Information Processing, IEEE ISSNIP 2014. Singapore: IEEE Computer
109   Society.

110 Pektas A, and Acarman T. 2014. A dynamic malware analyzer against virtual machine aware
111   malicious software. *Security and Communication Networks* 7:2245-2257.
112   10.1002/sec.931

113 Pektaş A, and Acarman T. 2020. Deep learning for effective Android malware detection using API
114   call graph embeddings. *Soft Computing* 24:1027-1043.

115 Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, and Ioannidis S. 2014. Rage against
116   the virtual machine: Hindering dynamic analysis of android malware. 7th European
117   Workshop on System Security, EuroSec 2014. Amsterdam: Association for Computing
118   Machinery.

119 Poeplau S, Fratantonio Y, Bianchi A, Kruegel C, and Vigna G. 2014. Execute this! analyzing
120   unsafe and malicious dynamic code loading in android applications. Proceedings of the
121   20th Annual Network & Distributed System Security Symposium (NDSS).

122 Preda MD, and Maggi F. 2016. Testing android malware detectors against code obfuscation: a
123   systematization of knowledge and unified methodology. *Journal of Computer Virology and
124   Hacking Techniques*:1-24. 10.1007/s11416-016-0282-2

125 Shao Y, Luo X, Qian C, Zhu P, and Zhang L. 2014. Towards a scalable resource-driven approach
126   for detecting repackaged android applications. 30th Annual Computer Security
127   Applications Conference, ACSAC 2014. December ed: Association for Computing
128   Machinery. p 56-65.

129 Shen T, Zhongyang Y, Xin Z, Mao B, and Huang H. 2015. Detect android malware variants using
130   component based topology graph. 13th IEEE International Conference on Trust, Security
131   and Privacy in Computing and Communications, TrustCom 2014: Institute of Electrical and
132   Electronics Engineers Inc. p 406-413.

133 Shrestha B, Ma D, Zhu Y, Li H, and Saxena N. 2015. Tap-Wave-Rub: Lightweight Human

134   Interaction Approach to Curb Emerging Smartphone Malware. *Information Forensics and*
135   *Security, IEEE Transactions on* PP:1-1. 10.1109/TIFS.2015.2436364

136 Sihag V, Vardhan M, and Singh P. 2021. BLADE: Robust malware detection against obfuscation
137   in android. *Forensic Science International: Digital Investigation* 38:301176.

138 Singh S, Mishra B, and Singh S. 2015. Detecting intelligent malware on dynamic Android analysis
139   environments. 10th International Conference for Internet Technology and Secured
140   Transactions, ICITST 2015: Institute of Electrical and Electronics Engineers Inc. p 414-
141   419.

142 Soh C, Tan HBK, Arnatovich YL, and Wang L. 2015. Detecting Clones in Android Applications
143   through Analyzing User Interfaces. 23rd IEEE International Conference on Program
144   Comprehension, ICPC 2015: IEEE Computer Society. p 163-173.

145 Spreitzenbarth M, Schreck T, Echtler F, Arp D, and Hoffmann J. 2015. Mobile-Sandbox:
146   combining static and dynamic analysis with machine-learning techniques. *International*
147   *Journal of Information Security* 14:141-153. 10.1007/s10207-014-0250-0

148 Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, and Cavallaro L. 2017. DroidSieve:
149   Fast and Accurate Classification of Obfuscated Android Malware. Seventh ACM on
150   Conference on Data and Application Security and Privacy. Scottsdale, Arizona, USA:
151   ACM. p 309-320.

152 Sufatrio, Chua TW, Tan DJJ, and Thing VLL. 2015. Accurate specification for robust detection of
153   malicious behavior in mobile environments. In: Weippl E, Pernul G, and Ryan PYA,
154   editors. 20th European Symposium on Research in Computer Security, ESORICS 2015:
155   Springer Verlag. p 355-375.

156 Sun M, Li M, and Lui JCS. 2015. DroidEagle: Seamless detection of visually similar android apps.
157   8th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec
158   2015: Association for Computing Machinery, Inc.

159 Talha KA, Alper DI, and Aydin C. 2015. APK Auditor: Permission-based Android malware
160   detection system. *Digital Investigation* 13:1-14. 10.1016/j.diin.2015.01.001

161 Tao F, Ziyi L, Kyeong-An K, Weidong S, Carbunar B, Yifei J, and Nguyen N. 2012. Continuous
162   mobile authentication using touchscreen gestures. Homeland Security (HST), 2012 IEEE
163   Conference on Technologies for. p 451-456.

164 Wang Z, Li C, Guan Y, and Xue Y. 2016. Anti-obfuscation method for detecting similarity of
165   Android application. *Huazhong Keji Daxue Xuebao (Ziran Kexue Ban)/Journal of*
166   *Huazhong University of Science and Technology (Natural Science Edition)* 44:60-64 and
167   76. 10.13245/j.hust.160312

168 Wei TE, Tyan HR, Jeng AB, Lee HM, Liao HYM, and Wang JC. 2015. DroidExec: Root exploit
169   malware recognition against wide variability via folding redundant function-relation graph.
170   17th IEEE International Conference on Advanced Communications Technology, ICACT
171   2015: Institute of Electrical and Electronics Engineers Inc. p 161-169.

172 Yan L-K, and Yin H. 2012. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic
173   Views for Dynamic Android Malware Analysis. USENIX security symposium. p 569-584.

174 Yang Y, Du X, Yang Z, and Liu X. 2021. Android Malware Detection Based on Structural Features
175   of the Function Call Graph. *Electronics* 10:186.

176 Yerima SY, Sezer S, and Muttik I. 2014. Android Malware Detection Using Parallel Machine
177   Learning Classifiers. Next Generation Mobile Apps, Services and Technologies

178   (NGMAST), 2014 Eighth International Conference on. p 37-42.

179   Zhang Y, Luo X, and Yin H. 2015. Dexhunter: toward extracting hidden code from packed android
180   applications. *Computer Security--ESORICS 2015*: Springer, 293-311.

181   Zhao S, Li X, Xu G, Zhang L, and Feng Z. 2014. Attack tree based android malware detection
182   with hybrid analysis. 13th IEEE International Conference on Trust, Security and Privacy in
183   Computing and Communications, TrustCom 2014: Institute of Electrical and Electronics
184   Engineers Inc. p 380-387.

185   Zhauniarovich Y, Ahmad M, Gadyatskaya O, Crispo B, and Massacci F. 2015. StaDynA:
186   Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android
187   Applications. Proceedings of the 5th ACM Conference on Data and Application Security
188   and Privacy. San Antonio, Texas, USA: ACM. p 37-48.

189   Zhong Y, Yamaki H, Yamaguchi Y, and Takakura H. 2013. ARIGUMA Code Analyzer: Efficient
190   Variant Detection by Identifying Common Instruction Sequences in Malware Families.
191   2013 Ieee 37th Annual Computer Software and Applications Conference (Compsac). p
192   11-20.

193

194