A Malware Evasion Technique for Auditing Android Anti-Malware Solutions

Samrah Mirza[§], Haider Abbas[*], Waleed Bin Shahid[*], Narmeen Shafqat[*], Mariagrazia Fugini[‡], Zafar Iqbal[*] and Zia Muhammad[*]
[§*]Department of Information Security, National University of Sciences and Technology, Islamabad, Pakistan
[‡]Professor of Computer Engineering at Politecnico di Milano, Italy
Email: *{haider, waleed.shahid, narmeen_shafqat, zafar.ncsael, zia.ncsael}@mcs.edu.pk
Email: §samrahsamrah.bete50@students.mcs.edu.pk
Email: ‡mariagrazia.fugini@polimi.it

***Abstract*—In the past few years, Android security is enhanced and state-of-the-art anti-malware tools have been introduced to counter Android malware. These tools use both static and dynamic analysis techniques to detect malicious applications. Despite these, the attack surface against Android phones has risen exponentially and malware detection tools are failed to counter sophisticated threats. Therefore, it is a need to audit and evaluate Anti Malware Solutions (AMTs). In our research, we have analyzed various Android malware evasion techniques, along with their pros and cons. Moreover, we conducted a detailed comparison of existing anti-malware tools and measured their efficacy against the discussed evasion techniques. Finally, a more sophisticated anti-malware evasion technique is proposed that uses exhaustive obfuscation and remote code execution to audit static and dynamic detection capabilities of AMTs. The proposed technique is practically validated and results prove that it evades all known anti-malware solutions. This technique can be utilized by anti-malware solution providers for making their products more resilient and powerful.**

***Index Terms*—Antivirus Evasion, Android Security, Malware Analysis, Code Obfuscation, Anti-malware Tools (AMTs)**

I. Introduction

Android Operating System (OS) is among one of the most widely used platforms, deployed on over 2 billion smartphones. This extensive usage of Android OS and its open-source nature [1] have made it a lucrative target for hackers and cyber offenders to spread malicious applications and compromise the confidentiality, integrity, and availability of victims' data. According to the Check Point researchers, mobile malware have become doubled as compared to their count in 2018 [2]. An average of 23,795 malicious mobile applications is estimated to be blocked on mobile devices each day. Although, to enhance security and user experience, annual Android updates are released. However, these updates are not directed to all Android devices but, these are specific to mobile device vendor, phone model, and users' geographic location. Therefore, only 20 percent of Android devices run updated versions. This alarming situation makes it easier for attackers to compromise outdated Android devices.

Malware authors deploy several sophisticated evasion techniques like packing, obfuscation, steganography, and code reuse to create malware variants that can evade antivirus and other security solutions [3]. This kind of devious malware tends to stay hidden while successfully carrying out its desired illicit actions [4]. Therefore, there is a dire need to analyze various antivirus evasion and bypassing techniques in order to critically evaluate and improve the detection efficacy of the current state-of-the-art Anti Malware Tools (AMTs). The existing malware repositories such as Genome [5] and Drebin [6] are outdated, anti-malware tools trained on these data-sets are unable to counter contemporary malware. The proposed research fills this gap by reviewing existing malware evasion techniques with a prime focus on the Android OS and uses this study to propose a holistic harmonized framework for auditing anti-malware tools.

The paper is organized in the following way. Section. II gives a brief overview of Android OS's security. Section. III gives an overview and comparison of various malware evasion techniques. Section. IV provides a discussion on why anti-malware tools fail to detect sophisticated malware. Section. V covers performance analysis, testing and validation of proposed methodology. Section. VI presents an evasion technique for Android malware. Section VII validates the proposed methodology through a practical application. Lastly, Section. VIII concludes the paper.

II. Background

This section explains the Android security environment, vulnerabilities, and tools for the detection of contemporary malware.

A. Android Security Environment

In order to get an insight into the working of AMTs, essential components of the Android security model are briefly described in this subsection. The Android security model is based on application sandboxing [7]. Android achieves application sandboxing through Linux User IDs (UIDs) to isolates running applications from other applications. According to the Android's permission model, sandboxed applications communicate with each other, and the system uses intent filters to control the permissions explicitly declared in the AndroidManifest.xml file.

B. Android Security Vulnerabilities

The monthly Android Security Bulletin maintains a database of evolving Android-based vulnerabilities and respective security remediation [7]. The vulnerabilities have been divided into following categories. Table. I lists some of the severe security vulnerabilities recently found in Android's framework.

TABLE I: CVE Android Security Bulletin, The Year 2019.

Month	CVE	Description	Type
Mar 2019	CVE-2019-2004	The presence of uninitialized data in the events of InputTransport.cpp results in information disclosure without additional execution privileges.	ID
Apr 2019	CVE-2019-2026	A possible escape from the Setup Wizard due to missing permission check in Editor.java can cause locale EOP without requiring any additional execution privileges and user interaction.	RCE
Jun 2019	CVE-2019-2090	Due to a missing permission check in PackageManagerService.java, a possible permission bypass leading to a locale escalation of privilege.	EOP
Jul 2019	CVE-2019-2104	A framework vulnerability that can enable a local malicious application to gain access to added permissions by exploiting user interaction requirements.	RCE

- **Information Disclosure (ID):** These vulnerabilities are employed to gain valuable information regarding system or user, thereby causing privacy issues and information leakage.
- **Remote Code Execution (RCE):** It allows an attacker to remotely execute commands or code to targeted devices.
- **Elevation of Privilege (EoP):** Attacker employs to gain access to protected services/ resources by exploiting vulnerabilities in OS or applications.

C. Anti-malware tools for Android

Android anti-malware tools use both static and dynamic techniques to analyze malware. Both of these methods have some pros and cons. For example, in the static analysis, applications are reverse engineered, and resultant source code is analyzed using tools like Apktool [8] and Dex2Jar [9]. Despite being a robust technique, Static Analysis cannot detect an obfuscated or zero-day malware. On the contrary, the dynamic analysis evaluates malicious applications' behavior after their execution and, hence, detects them successfully. Table. II lists various known anti-malware solutions that are used to detect malware in Android devices. The table is designed to distinguish their feature set and protection score given by AV-TEST (an organization that evaluates antivirus and anti-malware solutions).

III. Literature Review

We find a relatively limited amount of associated literature that is specifically relevant to our domain due to our work's novelty. Several researchers have called various aspects of primary malware evasion strategies that are included in this section. Android malware can be categorized based on evasion techniques, core functionality and behaviour such as, trojan, adware, spyware, privacy leaker, root exploit and credential stealer. Table. III gives a quick overview of malware and their evasion techniques along with their pros and cons. Marpaung, Sain, and Lee [10], in their research, have outlined primary evasion techniques such as:

TABLE II: ANDROID SECURITY APPLICATIONS.

Security Application	Features	Protection score (6.0)
Norton Mobile Security 4	Proactive Malware blocker, Malware protection, Anti-theft, Wi-Fi security.	6.0
BullGuard Mobile Security 14.0	Antivirus, Anti-theft, Backup, Security Manager.	4.5
Trend Micro Mobile Security 10.1	Malware Detection, Phishing sites, Privacy Scanner for Facebook.	6.0
Avast Mobile Security 6.11	Antivirus protection, Web shield for malicious URLs, Firewall, application scanner.	5.5
Sophos Mobile Security 8.6	Antivirus, Web filtering, Privacy and Security advice.	6.0
Avira Antivirus Security 5.4	Antivirus, Antispyware, Anti-theft, recovery tools.	6.0
AVG Antivirus Free 6.11	Antivirus, Photo vault, Camera and Trap feature.	5.5
Kaspersky Internet 11.18 Security	Privacy Protection, Encryption, Anti-Spam, Anti-malware, Firewall.	6.0
McAfee 5.0	Backup, and Privacy Data.	6.0

A. Obfuscation

Obfuscation is a process that makes it hard to understand textual and binary data. It deceives simple methods of string-matching used in signature-based detection by concealing the attack payload of malware.

B. Code Reuse

This exploits legitimate system requests by executing arbitrary code on a compromised machine. An attacker directs control flow through existing code with a malicious result thus avoids the need for explicit attack code injection on the stack.

C. Steganography

It refers to hiding the data in another medium like an image, without incurring noticeable changes.

D. Packing

A DEX (Dalvik Executable) file is an executable file saved in a format that contains compiled code written for Android. The packing method encrypts malicious DEX files using an Executable and Linkable Format (ELF) binary that only gets decrypted in the memory at runtime and executed using DexClassLoader.

E. Cryptography

It makes the code unreadable by applying encryption algorithms such as polymorphic XOR. This section reviews these techniques with respect to their (i) pros and cons, (ii) evasion

TABLE III: SUMMARY OF EVASION TECHNIQUES ON ANDROID.

Malware Type	Evasion Technique(s)	Pros	Cons
Privacy leaker	Obfuscation (both control and data based).	Maximizes no. of ways to attack, minimizes detection.	Only bypass dynamic analysis based AMTs.
Dynamically assembled and loaded malware	Mystique-S: a service-oriented tool.	Mystique-S developed malware that are undetectable in case of offline detection.	Dynamic analysis tools can detect dynamically loaded malicious code.
Malicious Android application (APKs)	Genetic Programming (GP).	Most successful AMTs can be evaded via GP's attack patterns.	Application limited to few malware, ignores dynamically loaded code.
Root exploit, information exfiltration, SMS Trojan, dynamic code loading	Repacking, renaming identifier, package name, disassembling reassembling, call indirections, data encoding/ reordering, junk code insertion, byte code encryption, and composite transformations.	This is efficient, and it is capable to vade almost all anti-malware tools that are available in market.	Only thwart static analysis, and not dynamic analysis, Ignores code-level transformations.
Credentials stealer, adware, spyware	Repackaging, and obfuscation.	Can evade anti-malware tools with little effort.	Less comprehensive transformations, and lacks composite obfuscation.
Genome Malware dataset (AAMO)	Obfuscation (Android specific, simple/ advanced control-flow, resource renaming/ encryption).	Uses sophisticated/ automated obfuscation techniques to evade top AMTs (Avast, Norton), is open source.	Only evades scan-time static analysis.
Spyware, ransomware, banking Trojan	Code reordering based obfuscation techniques: Method overloading, and opaque predicate.	Decreased detection rate by 50%, employs updated malware samples that retains its malicious operation.	Evades signature- based detection only and uses code reordering obfuscation technique only.
Data extortion, root exploits, bot activity, and SMS Trojan	RealDroid: Static, dynamic and hypervisor level heuristics disguise.	AMTs failed to infer malicious behavior of new malware. Also, no tool detected VM evasion.	Analysis services lacking support for native execution couldn't be evaded.
Genome malware dataset	Fast Fourier Transform, signal steganography-based evasion.	Exploits malware scan and engine update's null-protection window.	Lack of new malware dataset used for evasion.
Trojan: Android/ OpFake	Monitoring of interaction patterns, Scan-code, Device ID and Name, Motion Events to differentiate human user and tool.	Provides malware protection even when analyzed on a bare-metal platform. Detect AMTs on the target device by integrated with Android malware - evades dynamic runtime analysis too.	The limited scope doesn't incorporate static analysis evasion.
OpFake alike Malicious application from Andro MalShare	Obfuscation (using ProGuard), Cryptography (XOR), Steganography and their blend along with dynamic code loading and reflection.	Easy and effective evasion approach. It also provides metrics for evaluating AMTs against new Android malware. It evades both static and dynamic analysis.	The approach couldn't be evaluated as experiments carried out using a single malicious APK only.
Malicious Android Applications	AngeCryption: encrypt APK to valid PNG and embed into a benign-looking wrapping APK.	Embeds undetectable, valid, and runnable bytecode in a benign-looking APK which evades static analysis.	Works only on Android 4.4.2.
Malicious Android Applications	AVPass: automatically bypasses AMTs using both obfuscation and inferring detection rules for AMTs.	Bypasses AMTs and gives a good insight into the detection rules of AMTs using inferring and imitation mode.	Bypasses only static analysis, and certain features such as inferring AV features don't work.

tools, and (iii) detection mechanisms details are provided in the ensuing paragraphs [11].

Mystique [12] is a malware generation framework that used gene crossover and mutation techniques to generate evasive malware. Mystique-S, a variant of Mystique, is focused on malware specific to financial charge, phishing and extortion cases [13]. It gathers client data, delivers the malware at run time, and can evaluate real devices rather than virtual emulators. Moreover, Using genetic operators on existing malware, Sen, Aydogan, and Aysan [14] developed an effective attack with evasion capability that challenges the effectiveness of most successful security solutions.

Rastogi, Chen, and Jiang [15] developed DroidChameleon [16] that applies various transformation techniques on malware samples and audits ten popular mobile AMTs being vulnerable to these transformations. However, such evasion is not very effective owing to the signature-based detection paradigm.

Zheng, Lee, and Lui [17] developed ADAM that employs obfuscation and repackaging techniques like repacking, assembling/disassembling, string encoding, code reordering, junk code insertion, and renaming identifiers, but ignores sophisticated ones such as payload and native code encryption, array data encoding, reflection and bytecode encryption.

Preda and Maggi [18] proposed an Automatic Android Malware Obfuscator (AAMO) to obfuscate exhaustive datasets of Android malware using existing and new obfuscation techniques. It employs 1,260 malware applications from the Genome repository. Subsequently, Badhani, and Muttoo [19] developed eight different evasion techniques to hide malware inside an image of a wrapper Android application using obfuscation, concatenation, steganography, cryptography and their combinations.

Chua and Balachandran [20] presented a detailed framework having various obfuscation techniques like switch function, method overloading, try-catch function, and opaque predicate. The latest malware use these techniques to bypass the detection of AMTs as listed on VirusTotal [21].

RealDroid [22] highlighted a broad range of techniques to evade dynamic analysis in virtualized environments. A set of repackaged malware with developed heuristics incorporation almost evaded all malware analysis services deceiving numerous analysis tools. A comprehensive analysis of the top 30 AVD (Android Virus Detectors) is presented in [23].

A mechanism to evade automated runtime analysis is pro-

posed by Diao, Liu., Li, and Zhang [24]. The proposed mechanism gives an insight into the efficacy of the current dynamic analysis platforms, and could be used in integration with Android malware to monitor the system events before the execution of actual malware.

Using Angecryption [25], it is possible to embed imperceptible, valid, and executable bytecode in a benign-looking application and static analysis can be bypassed easily. Another tool AVPass [26], is developed to bypass Android malware detection systems and it offers several obfuscation techniques.

All of these works are comprehensive efforts but have limited scope. For example, Mystique and Mystique-S provided reasonable evasion in offline detection, addressed privacy leakage, and dynamically assembled and loaded malware. However, Mystique only e Dynamic Analysis Tool (DAT) and is less effective. Mystique-S, too failed to evade when dynamically loaded malware was subjected to DATs. GP based evasion tool claimed to evade most successful AMTs. However, it lacks dynamically loaded code features.

Among the evasion approaches discussed, DroidChameleon, ADAM, AAMO, the system proposed by Badhani, and Muttoo are used to test the efficacy of the current AMTs being used for the detection of Android malware. Trivial obfuscation techniques developed by DroidChameleon successfully thwart static analysis but fail when DATs are employed for detection. ADAM and RealDroid proposed both evasion and detection frameworks. RealDroid fails to detect VM evasion. ADAM provides reasonable evasion by repackaging and obfuscation but lacks composite obfuscation techniques. AAMO provides exhaustive obfuscation techniques and is flexible in terms of its application but fails to evade DATs.

Although, aforementioned techniques are great motivation in research, however, none of these studies audit image-based malware. In contrast, our research work Audits 50+ malware tools that are available on Virustotal. Furthermore, it uses application obfuscation, dynamic code loading and stenography, it successfully bypasses static and dynamic analysis based malware detection techniques.

IV. Proposed Malware Evasion Technique

To audit the detection efficacy of known anti-malware tools against simple yet sophisticated evasion techniques, a simple, resilient, and lightweight methodology has been proposed in this section. Our methodology is based on application obfuscation, dynamic code loading, and malware propagation using images. The proposed framework consists of a series of necessary steps illustrated in Fig. 1.

1) Firstly, the malicious application is obfuscated using the obfuscation module. Any obfuscators can be used such as ProGuard [27], AAMO [18], and AVPass [26]. Here we have used AVPass - an open-source obfuscator. This step is important and can't be skipped because it introduced randomization and play a vital role to avoid signature-base detection.

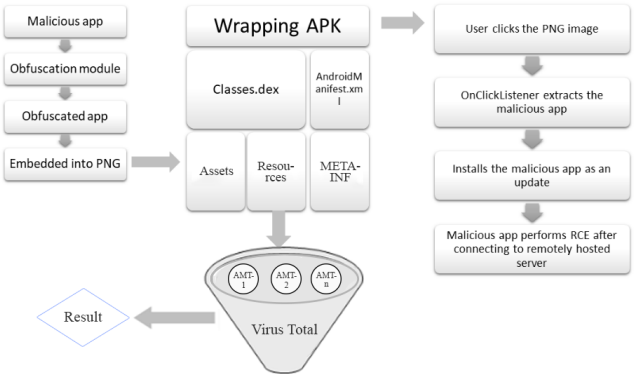


Fig. 1: Framework for creating malware and auditing AMTs.

2) In the second step, the obtained obfuscated application is encrypted using AES-128 or AES-256 such that this encryption comprises of the malicious application followed by the inverse AES of CRC32 checksum of the target PNG image, header chunk (IHDR), Data chunk (IDAT), and end of data chunk (IEND). This is followed by some dummy bytes to make the file multiple of 16 bytes as AES operates on 16 bytes block size.
3) In the third step, the malware obtained from step 2 is crafted in an image with extension type PNG (Portable Network Graphics) that can be done either by concatenation or stenography.
4) The fourth step involves placing the PNG image into a wrapping application's assets folder. Wrapping application can be any benign application. Afterwards, We created an event **onClickListener** that triggers the process to decrypt the malicious application that was placed in the assets folder. The process converts PNG into the APK file and dynamically loads it to the phone storage and then install it as an application update on run-time.
5) The final step is to upload the wrapping application to the repository of known anti-malware solutions. Also, we upload the application to VirusTotal at each step and gradually check the detection rates. Moreover, installation of the application on the target device is done at each stage to validate the application's malicious intent and make sure the application never crashes. Similarly, we can evade dynamic analysis using Remote Code Execution (RCE) script residing on a remote website instead of the target device. The malicious application would be installed at the run time as an update to the wrapping application. This would resist malicious code detection on runtime hence giving complete evasion from static and dynamic malware detection techniques. Later, most importantly, all known licensed and open source AMTs will be audited against the proposed methodology to ascertain their weaknesses. Furthermore, we have performed a case study on an existing malware to validate the proposed technique.

V. Performance Analysis, Testing and Validation

In this section, we analyzed the performance of our methodology by applying it to real malware and tested it on the emulator. The methodology is not malware specific and it can be implemented on any malware designed for the Android platform. To validate the working of the proposed system, a real malware application namely Dendroid [28] is selected. Dendroid was designed to evade Android devices and it is a sophisticated malware designed to spy and remotely acquire backups. The key features of the Dendroid are as follow:

- Records messages and calls.
- Obtains the accounts that are stored in the device.
- Downloads media and images from the target device.
- Takes pictures, record audio and video of the user.

Presently, all the known anti-malware solutions, particularly Avast, AVG, Kaspersky, Symantec, and McAfee are able to detect it a malicious file. According to the VirusTotal, it has a detection rate of 32/56, as illustrated in Fig. 2.

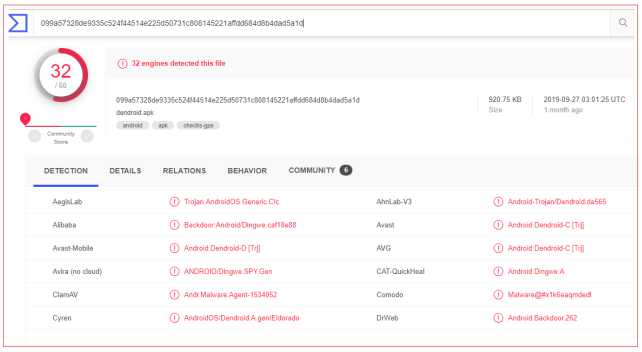
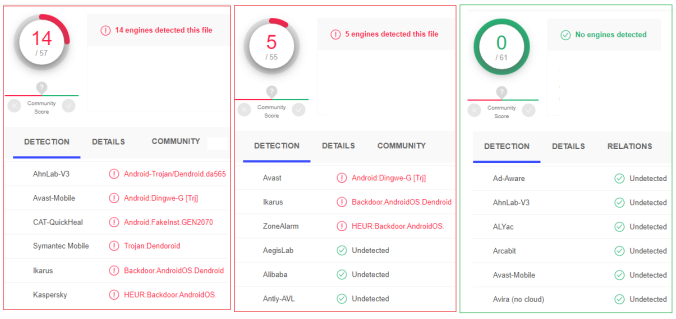


Fig. 2: VirusTotal Results for Orignal Dendroid malware.

To validate our proposed methodology, AVPass (An open source application obfuscator) has been used as an obfuscation module. AVPass uses obfuscation techniques such as API reflection, string, and Variable techniques. A new APK file is generated by applying AVPass. Then the APK was uploaded to VirusTotal to check the results. The detection rate dropped to almost 25%, as 14/56 antivirus solutions were able to detect it as malicious. Results are illustrated in Fig. 3 (a).



(a) VirusTotal Results after the Application of Obfuscation Module.

(b) VirusTotal Results for Encrypted Application.

(c) VirusTotal Result for the Application Containing Maliciously Crafted PNG Embedded into the Assets

Fig. 3: Step-by-Step Auditing of Anti Malware Solutions.

Next, we encrypted this obfuscated APK file in such a way that the new file is a combination of payload apk, inverse AES of the (CRC32+IHDR+IDAT+IEND) where IHDR is the header chunk of the target PNG image, IDAT is the data chunk where the actual image data resides and IEND being the end chunk which is an end-of-file marker for the PNG. This yielded quite effective results decreasing the detection rate further, as only 5/55 anti-malware solutions were able to detect it. Thus, we successfully achieved a visual reduction that can be seen in Fig. 3 (b). The application was installed on an Android emulator after each step to validate its malicious intent and to ensure its proper working.

For better results, we further embedded the application, attained as a result of the previous step, into a target PNG file. At this step, we simply use the Steganography technique to disguise a malware in an image. Afterwards, this maliciously crafted PNG image was embedded into the assets folder of a benign wrapping application. Subsequently, the wrapping application had an event onClickListener that fetched the PNG image from the assets folder, decrypted it into the malicious app, dynamically loaded it into the SD card and installed it at the runtime as an update to the wrapping application.

In the final step, the resultant application is uploaded again at the VirusTotal. It is important to mention that no anti-malware solution was able to detect the malicious application, as indicated in Fig. 3 (c). Moreover, the application was also successfully installed on the victim's device. Upon installation, the application executed its behavior as intended.

Hence, using the above mentioned simple and easy-to-implement framework, we were able to achieve the desired result. Fig. 4 shows stagewise Antivirus detection. This can be seen that, initially malware was detected by 32 AMTs, but this detection decreased gradually after applying proposed methodology and no AMT was able to detect our final malware. Moreover, Table. IV shows that our proposed solution has evaded well known AMTs and gives a comparison against existing evasion techniques.

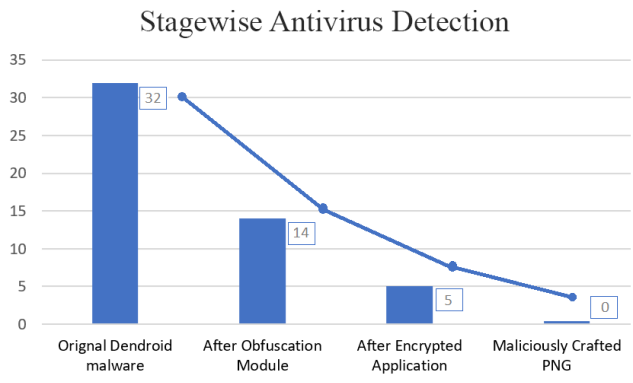


Fig. 4: Validation of Technique Across Malware Detection Platform.

TABLE IV: COMPARISON OF DIFFERENT EVASION TECHNIQUES.

Security Application	**Raw**	**Angecr-ytpion**	**AVPass**	**AAMO**	**Our Solu-tion**
BitDefender	×	×	×	✓	✓
Fortinet	×	×	×	×	✓
Trend Micro	×	×	×	×	✓
Avast	×	×	×	×	✓
Sophos	×	×	×	×	✓
Avira	×	×	×	×	✓
AVG	×	×	×	×	✓
Kaspersky	×	×	×	×	✓
McAfee	×	×	×	×	✓

VI. Conclusion And Future Work

The proposed framework focus on malware creation and several malware evasion techniques. There are many social engineering and malware propagation techniques to trick Android users into installing malware but they are not in our scope. In our research, we proved that every new evasion technique aims to render AMTs useless and motivates security researchers to enhance and revamp their malware detection suites. After conducting a critical analysis of existing evasion techniques, the research proposes a mechanism to audit advanced Android AMTs and sets a benchmark for the progressive and sophisticated class of evasive malware against which anti-malware tools can be tested. This research invites Android researchers, developers, and Anti-malware companies to investigate, audit, and enhance their malware solutions against the latest evasion techniques. Furthermore, this work can be implemented to test other malware variants and platforms like iOS, Windows, and Linux.

VII. Acknowledgement

This research is supported by the Higher Education Commission (HEC), Pakistan through its initiative of National Center for Cyber Security for the affiliated lab "National Cyber Security Auditing and Evaluation Lab" (NCSAEL), Grant No: 2(1078)/HEC/M&E/2018/707.

References

[1] N. Elenkov, Android security internals. San Francisco, CA: No Starch Press, 2015.

[2] Symantec.com, 2019. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/eports/istr-23-2018-en.pdf.

[3] Alharbi, Faris Auid, Abdurhman Mansour Alghamdi, and Ahmed S. Alghamdi. "A Systematic Review of Android Malware Detection Techniques." International Journal of Computer Science and Security (IJCSS) 15.1 (2021): 1.

[4] Sinha, Anukriti, et al. "Emulation Versus Instrumentation for Android Malware Detection." Digital Forensic Investigation of Internet of Things (IoT) Devices. Springer, Cham, 2021. 1-20.

[5] "Android Malware Genome Project", Malgenomeproject.org, 2019. [Online]. Available: http://www.malgenomeproject.org/.

[6] D. Arp, "The Drebin Dataset", Sec.cs.tu-bs.de, 2019. [Online]. Available: https://www.sec.cs.tu-bs.de/ danarp/drebin/.

[7] "Android Open Source Project", Android Open Source Project, 2019. [Online]. Available: https://source.android.com/security. [Accessed: 10- July- 2019]

[8] "Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.", Ibotpeaches.github.io, 2019. [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Accessed: 10- Mar- 2019]

[9] Tools.kali.org. [online] Available at: https://tools.kali.org/reverse-engineering/dex2jar [Accessed 6 Mar. 2020].

[10] J. Marpaung, M. Sain and H. Lee, "Survey on malware evasion techniques: State of the art and challenges", in 2012 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang, South Korea, 2012.

[11] DexClassLoader — Android Developers. [online] Available at: https://developer.android.com/reference/dalvik/system/DexClassLoader [Accessed 6 Mar. 2020].

[12] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang and T. Chen, "Mystique: Evolving Android Malware for Auditing Anti-Malware Tools", 2019.

[13] Y. Xue, G. Meng, Y. Liu, T. Tan, H. Chen, J. Sun and J. Zhang, "Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique", IEEE Transactions on Information Forensics and Security, vol. 12, no. 7, pp. 1529-1544, 2017.

[14] S. Sen, E. Aydogan and A. Aysan, "Coevolution of Mobile Malware and Anti-Malware", IEEE Transactions on Information Forensics and Security, vol. 13, no. 10, pp. 2563-2574, 2018.

[15] V. Rastogi, Y. Chen and X. Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks", IEEE Transactions on Info Forensics and Security, vol. 9, no. 1, pp. 99-108, 2014.

[16] V. Rastogi, Y. Chen and X. Jiang, "DroidChameleon", Proceedings of the 8th ACM SIGSAC symposium on information, computer and communications security - ASIA CCS '13, 2013.

[17] M. Zheng, P. Lee and J. Lui, "ADAM: Automatic and Extensible Platform to Stress Test Android Antivirus Systems", Detection of Intrusions, Malware and Vulnerability Assessment, pp. 82-101, 2013.

[18] M. Preda and F. Maggi, "Testing android malware detectors against code obfuscation: a systematization of knowledge and unified methodology", Journal of Computer Virology and Hacking Techniques, vol. 13, no. 3, pp. 209-232, 2016.

[19] S. Badhani and S. Muttoo, "Evading android anti-malware by hiding malicious application inside images", International Journal of Sys Assurance Engineering and Management, vol. 9, no. 2, pp. 482-493, 2017.

[20] M. Chua and V. Balachandran, "Effectiveness of Android Obfuscation on Evading Anti-malware", Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy - CODASPY '18, 2018.

[21] Virustotal.com. (2019). VirusTotal. [online] Available at: https://www.virustotal.com/ [Accessed 11 Aug. 2019].

[22] L. Liu, Y. Gu, Q. Li and P. Su, "RealDroid: Large-Scale Evasive Malware Detection on "Real Devices"", 2017 26th International Conference on Computer Communication and Networks (ICCCN).

[23] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu and D. Wu, "Towards Discovering and Understanding Unexpected Hazards in Tailoring Antivirus Software for Android", Proceedings of the 10th ACM Symposium on Information, Computer and Communications Sec - ASIA CCS '15, 2015.

[24] W. Diao, X. Liu., Z. Li, K. Zhang, "Evading Android Runtime Analysis Through Detecting Programmed Interactions". 10.1145/2939918.2939926. In: WiSec '16: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks. NY, USA: Association for Computing Machinery, pp.159-164.

[25] A. Apvrille. A. Albertini, "Hide Android Applications in Images," 2014 in paper presented at BlackHat Europe, Amsterdam, NH.

[26] J. Jung, C. Jeon, M. Wolotsky, I. Yun and T. Kim, "AVPASS: Automatically Bypassing Android Malware Detection Sys," Las Vegas, 2017.

[27] Guardsquare. (2020). ProGuard. [online] Available at: https://www.guardsquare.com/en/ [Accessed Mar 2020].

[28] F-Secure," [Online]. Available: https://www.f-secure.com/v-descs/backdoor_android_dendroid_a.shtml. [Accessed Sep 2019].