A Critical Analysis on Android Vulnerabilities, Malware, Anti-malware and Anti-malware Bypassing

Muath Alrammal[1*], Munir Naveed[1], Suzan Sallam[2], Georgios Tsaramirsis[1]

[1] Abu Dhabi Women's College, Higher Colleges of Technology, UAE
[2] General Education Department, Khawarizmi International College, UAE
malrammal@hct.ac.ae, mnaveed@hct.ac.ae, suzan.sallam@khawarizmi.com, gtsaramirsis@hct.ac.ae

Abstract

Android has become the dominant operating system for portable devices, making it a valuable asset that needs protection. Though Android is very popular; it has several vulnerabilities which attackers use for malicious intents. In this paper, we present a comprehensive study on the threats in Android OS that various malware developers exploit and the different malware functionality based on Android's threats. Furthermore, we analyze and evaluate the anti-malware approaches implemented to face the malware functionalities. Finally, we analyze and categorize malware developers' most common anti-analysis techniques to evade anti-malware approaches. It comes to our attention that many papers covered each topic separately; however, we could not find one comprehensive study that covers Android with such details that it could be used as a research handbook on Android malware. This is the main novelty and contribution of this work.

Keywords: Malware detection, Dynamic analysis, Static analysis, Hybrid analysis, Reinforcement learning

1 Introduction

As the world evolves and technology around us changes its face every day, the number of smartphone users and other smart devices for business or personal purposes is dramatically increasing. Android is the most commonly used OS in that category, with 87% Android users [1]. Many smart device manufacturers support Android. Android's first release was in September 2008; it is a Linux-based open-source platform that supports over 100 languages. The android applications are widely available on various App stores such as Google Play, Amazon, Aptoide, Galaxy, and others, with millions of Apps available to choose from that leads to people favoring Android over other OS [2]. However, with the increasing numbers of Apps and users, Android Apps APK has become the biggest target for attacks; on average, more than 600,000 malware applications per month are distributed, as AVTest security report [3] stated in their latest security report. The AVTest report narrates is based on analysis of huge number of attacks and signifies the statistically common attack behaviors e.g. in 2019 report, the most damaging but frequent attacks were exploiting the hardware architecture base vulnerabilities to read the content of memory which may contain confidential information e.g. password. The report also summarizes the attack targets year-wise e.g. in 2019, Windows devices were attacked most of the times. According to this AVTest reports, the android OS has been facing tremendous increase in number of attacks per year. Those attacks vary in their specialization level and target, which creates a pressing need for in-depth analysis of their techniques to develop effective detection and classification tools [4]. Therefore, we are targeting Android malware; we are covering different types of malware, anti-malware techniques.

Malware is a code that a cybercriminal has developed intending to gain unauthorized access to a device, data, or network. Its effect may vary from mild to severe depending on the permission it gains. Additionally, we have discovered in our research that due to the speed of App development, some Apps do not go through enough security checks before being published, leading to unintentional security breaches to users' devices. Moreover, malware developers are increasingly more innovative when hiding their malicious code behind complex GUI widgets (are miniature application views that can be embedded in other applications (such as the home screen) and receive periodic updates) to hinder malware analysis tools.

Furthermore, due to the comparatively limited capabilities of smart devices in processing, storage, and battery life, the traditional anti-malware "PC anti-malware" techniques are not suitable as it requires a lot of processing and storage capabilities [5]. These techniques have been adopted for the mobile devices with scarce resources. The traditional techniques are typically constructed using standard signature based approach. However, these techniques pave way for the exploration of more modern and behavior based scalable solutions to constructor state-of-the-art antimalware.

In order to detect and deter new malware definitions, the traditional anti-malware, which are largely based on signature-based Anti Virus (AV) are not sufficient as the malware keep changing signature pattern of their attack. Such attack require use of more sophisticated approaches to detect them and contain them. Signature-based AV uses a kind of fix set malware characteristics to identify and categorize malware; however, malware can easily surmount it using obfuscation or encryption, as explained later in sections 3.4 and 3.5. Consequently, other anti-malware techniques are employed, such as Static and Dynamic analysis.

*Corresponding Author: Muath Alrammal; E-mail: malrammal@hct.ac.ae
DOI: 10.53106/160792642022122307019

Static analysis is performed using a reverse-engineer to analyze the code without executing it; by utilizing the App manifest, static analysis can collect constructive information about the App behavior, permissions, activities, etc. However, this method can indeed be defrauded using camouflage techniques such as encryption and obfuscation. On the other hand, dynamic analysis performs an exhaustive analysis of the App by running the code in a controlled save environment "emulator" and monitor and analyze its functionality at run time. This technique gives wider latitude of the malware behavior and intention; then again, in 2013, Obad malware was the first malware to defeat dynamic analysis by detecting emulators' use and not executing their malicious act on it. Accordingly, various malware evades the use of emulators by delaying their malicious act until they inspect the environment and detect whether an emulator has been employed; in that cases, they do not execute their malicious code and qualify as benign App [5-6]. Hence, there is a need to develop a new technique that can understand the previously stated downsides of the static and dynamic analysis and outsmart malware by learning their attribute and detecting them.

It comes to our attention that many papers covered each topic separately; however, we could not find one comprehensive study that covers Android with such details that it could be used as a research handbook on Android malware. This is the main novelty and contribution of this work.

The paper structure is as follows: section 2 illustrates the different Android vulnerabilities. Section 3 explains how malware developers use android vulnerabilities to perform attacks. Section 4 analyzes and evaluates the several anti-malware techniques to confront the attacks of section 3. Section 5 presents some of the most common anti-analysis techniques that malware developers use to evade anti-malware detection. While we conclude and present the future work in section 6.

2. Android Vulnerabilities

Before discussing the vulnerabilities available in Android, we need to establish a basic understanding of the Android Platform structure Figure1 [1, 7-9]:

1. Android application layer - allows developers to utilize the existing functionality of the device.

2. Java API framework - allows the App developer to access a broad collection of APIs that provide the building blocks for the App layer.

3. Native Libraries – core android components are built from native code C and C++.

4. Android Runtime

a. Older Android versions used - Dalvik Virtual Machine - the previous two layers are Java programed, and they get executed inside Dalvik VM, which is responsible for interpreting the Dalvik Executable "DEX" into byte code format. That allows App components to communicate and share resources such as user interface and stored data.

b. Android 5 and above use - Runtime ART – multiple virtual machines with a low memory device usage.

5. Hardware Abstraction layer – allows API framework access to particular device hardware such as camera and Bluetooth.

6. Linux Kernel is built on the open-source kernel of Linux provide Android OS with the core OS infrastructure; some of the userspace services and libraries communicate with the kernel layer.

Before analyzing the different Android malware techniques, we need to discuss the inherently available threats in Android OS that various malware developers exploit.

It is obvious from the Figure 1 that the android runtime environment is a layered system which executes the app methods by using several callbacks. This architecture also pose challenges for the security solution to predict the app behavior, construct the context and data flows etc. For example, a method "*sendMessage()*" could be an execution pattern based on call to the code from Application layer to Android framework layer and eventually to linux kernel layer. Each layer generates its down data and maintains it in its local space. To generalize this behavior, the security solution must maintain the information flow about the callbacks, contexts of the object, the use of data generated by method etc. To maintain such rich data, the security solutions can face challenge of consuming many resources, which can yield high overheads. The rest of paper will be this framework (given in Figure 1) to narrate the significance and weakness of the current security solutions of the android apps.

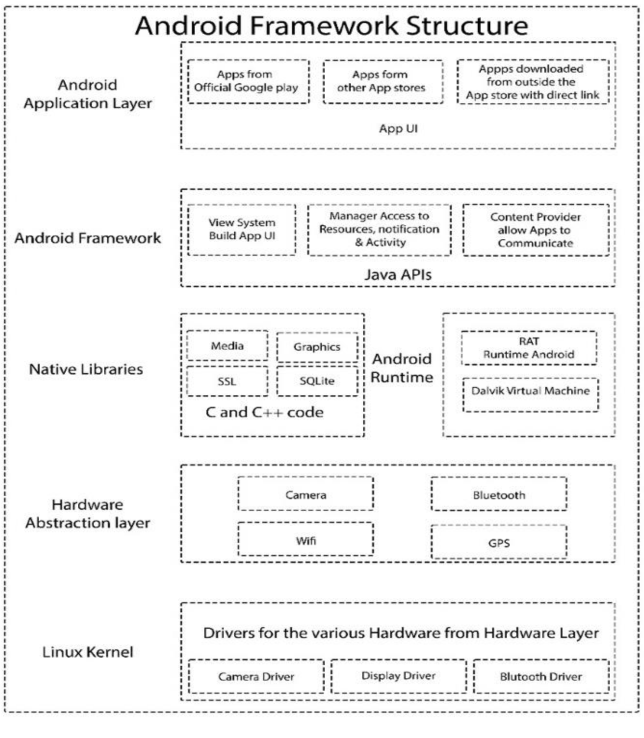
Android Framework Structure
Android Application Layer
Apps from Official Google play
Apps form other App stores
Appps downloaded from outside the App store with direct link
App UI
Android Framework
View System Build App UI
Manager Access to Resources, notification & Activity
Content Provider allow Apps to Communicate
Java APIs
Native Libraries
Media
Graphics
SSL
SQLite
C and C++ code
Android Runtime
RAT Runtime Android
Dalvik Virtual Machine
Hardware Abstraction layer
Camera
Bluetooth
Wifi
GPS
Linux Kernel
Drivers for the various Hardware from Hardware Layer
Camera Driver
Display Driver
Blutooth Driver

Figure 1. Android platform structure

2.1 Application Permissions

Android restrict application access to different resources using permission-based security level. Permissions are categorized into four levels normal, dangerous, signature and Signature OrSystem permission. Apps are required to declare their permission requests in their manifest [10]. However, when users download and install an App, they get prompted to allow all or deny all permissions. If the user reject the permission granting step, the App will not be installed; therefore, most users obliviously accept all permissions without comprehending their implications or risks. Additionally, Apps can request extra dangerous permissions at runtime from users [11].

2.2 Fragmentation Problem

On average, Google Android publish an update every month; however, for the update to reach every user from every manufacture worldwide, it takes months, which leads to different versions of Android remain available around the world, those with older versions remain vulnerable to security cracks that have been resolve in the newer updates [12]. For instance, some permissions declared as dangerous in the newer update are still normal on the un-updated device, leading to user data exploitation [13-14].

2.3 Colluding Attack

When users unknowingly install different Apps that share the same signed certificate, created by the same developer, they also share the permissions and access to the resources. In contrast, each App asks for single permission does not seem suspicious, but their permissions combined allow them to perform malicious activities. Additionally, each App separately gains access to a different resource. All Apps with the same certificate share these resources, leading to one App having access to voluminous resources without arousing any suspicion [10, 15].

2.4 System Server

Researchers in [16] has discussed the system server as a single point of failure in Android OS. System server is a multi-thread process that is responsible for most of the system process and App functionality. However, this process is vulnerable to exploitation by App developers. If an App developer writes a loop to invoke API in an App, it can loop endlessly and keep freezing and rebooting the system, which can later be used by malware to conduct a DoS attack. Moreover, [17] has shown that callback on SS can cause a system freeze and repeated system reboot. Various malware developers can exploit all the previously named vulnerabilities in Android to perform attacks on users, as explained in the following section.

3. Malware Functionalities

Malware varies in its purpose and attacks; some cause disturbance of the user device functionality by performing massive Ads attacks. Others steal users' contacts and use them to spread and target others' attacks. While others are more harmful, causing financial charges on the user or even stealing their bank account details to perform transactions on the behaves. Below we present the malware functionalities.

3.1 Aggressive Advertisement

One of the most widespread malware known to users is those annoying Apps that keep popup and disturb their device usage. Some malware takes control of the user device and starts bombarding them with advertisements, change their default search engine, and more such as Plankton [15].

3.2 Remote Control

93% of malware use the infected device for bots [18]. By gaining access to the device, these types of malware take control of the device, and by using a remote server, they can utilize the device as a bot which is a part of a botnet "group of devices controlled by a remote server" either to steal their data or to perform attacks such as denial of service attacks. Beanbot, Anserverbot are famous malware that create botnet [15]. Beanbot attack the devices with stealing the information e.g. IMEI number and phone number etc and send it to a remote server. It can also send high priced SMS from the device to consume the phone credits. Anserverbot also install the code on victim device to give remote control to the hacker via this code. The malware hidden in app, prompts users to install an update and in this disguise, the remote control program is download and install on the victim machines.

3.3 Privilege and Permission Escalation

Around 36% of malware use at least one root exploit, but it is ubiquitous to use more than one. By exploiting the android vulnerability colluding attack explained in section 2.3, various malware can collude to share permissions and gain higher privileges [19]. SMS-related permission is the most used exploit among malware; around 45% of malware requests to gain various SMS-related access, such as reading, writing, receiving, and sending messages [18].

Malware uses these escalations for various purposes, such as Obad to hide, Fobus to prevent users from disabling their privileges. In addition, the malware sways the user to give privileges for various reasons such as security-related services "Updtkiller," enhance the device productivity "Fobus," or forcefully ask for permissions till the user gives in "SmsZombie" [20].

Additional observation, malware Apps usually ask for more permissions than benign Apps. On average, malicious Apps ask for 11 permissions. In contrast, benign Apps, on average, asks for four permissions.

3.4 Financial Charges

Some malware uses remote control and privilege escalation to make a financial gain of the compromised device. This financial gain can send messages or subscribing to a premium-rate number, sending messages to the contact list, or even making phone calls in the background without the users' knowledge [18]. Such as DroidSMS subscribing for premium numbers, Zitmo steals users' login details to perform financial transactions from users' bank accounts [15]. Zitmo is designed to steal mobile transaction authorization numbers (mTAN). It is a Trojan that aims to forward incoming text messages with mTAN codes to malicious users or servers. Based on that, it can execute financial transactions using hacked bank accounts. Another prevalent malware that causes finical charges in users is ransomware, where the malware locks the device until payment is made to unlock the device data again, such as FakeDefender [15].

3.5 Leaking Information

Apps, in general, need access to users' information to function and communicate; however, transferring this information outside the user's device without their knowledge or consent is considered information leaking [19]. More than 80% of malware collects some user's and device information and sends it to their remote servers. Device information such as IMEI , IMSI , Kernel version, phone manufacturer, network operator [20]. SMS messages, phone numbers, user accounts, email addresses, usernames, and passwords. Using this information, malware attackers can perform a fraud on the user without their knowledge [18]—FakeNetflix masquerade as the famous App Netflix to steal the users' login details and leak them.

4. Anti-Malware Techniques

In order to monitor, detect and mitigate the malware functionality stated in section 4, intensive research has been conducted over the years. As a result, anti-malware techniques vary in their methodology of detection. Following, we discuss static and dynamic anti-malware techniques with state- of-the- art tools used in each.

4.1 Static Analysis

Static analysis has been widely employed in detecting malware. The static analysis relies on reverse engineering by analyzing the Dalvik bytecode without executing the code. This technique advances over dynamic analysis as the malware will not hide or delay their malicious act while being analyzed in a safe environment. Dalvik Virtual Machines (DVM) is not more used as runtime environment for the newer version of Android, however, the dex format is still in use. DVM is different than typical java virtual machine at its execution architecture is based registered-based bycote. Empirically, DVM is slower than JVM [63], however, performance of apps running on DVM is not deteriorated and significantly affected due to running in registered-based architecture. The dalvik format is commonly used in the app security models which can scan the dalvik byte code.

Serval previous work has been done in that area. CHEX [1] is one of the early attempts to employ static analysis on Android apps. CHEX is a component hijacking examiner that checks apps for multiple entry points, resulting in sensitive data leaking or permission escalation. Chex has reported a low false rate in identifying all entry points of an app, and then by introducing the app splitting concept, they divide the app code into multiple segments according to their used entry point to the code. Then they were able to track all potential data flow for each app entry point and identify the potential hijacking vulnerability exploited by the app, whether intentional or accidental. However, CHEX cannot detect those hijacking attacks that do not use explicit data flow, resulting in false-negative results.

Flowdroid [2] is a static taint analysis tool specifically designed for the Android platform that examines the potential App byte code and configuration files for sensitivity leaks, and the first taint analysis tool detects sensitive leaks from context, flow, field, and object flow by modeling the altogether App life cycle and UI widgets. FlowDroid combines forward taint analysis and on-demand backward alias analysis to determine if the data would get tainted at the received method "sink." However, Flowdroid is unable to resolve reflective calls if their arguments are not a string constant.

Motivated by Flowdroid, Amandroid [22] creates IDGF "inter-component data-flow graph," which covers all the Apps access points to sensitive information. That graph identifies any sensitive information request and flows between the same App component or between different Apps. Then it builds DDG "data dependence graph" to track the data flow through the App, which then can be generalized and used to analyze different Apps. Thus, Amandroid can be utilized to detect data leaks, data injection, and misuse of API. However, Amandroid has limited competency in handling exceptions, reflections, concurrency, and implicit flows.

Another tool that targets sensitive data leaks is Apposcopy [23] which uses signature base and taint analysis to identify an App as malware. Apposcopy firstly creates a high-level language datalog to identify shared behavior characteristics of each malware family, then using deep taint analysis to match an App data-flow and control-flow with the pre-defined datalog. This tool advantage does not depend on the existence of particular instruction or byte code to identify malware, making it resilient to code obfuscation. However, Apposcopy cannot detect all variants of the same malware family when they vary in characteristics if it is not listed in their datalog; additionally, it can be defeated by using dynamic payload explained in section 5.2.

Flowdroid creates structure (i.e. graph) to generalize the behavior signature of the malware while Apposcopy creates sematic-based representation for defining the malware patterns. The Flowdroid structure captures information about flow of calls and constructs context while Apposcopy is semantically capture the pattern structure using inter component call graph.

On the other hand, another attempt to track evidence of malicious behavior and sensitive data-flow has been published under EviHunter [24]. In their work, their analysis goes through two phases online and offline. In the offline phase, they build an App Evidence Database AED to store all files that include evidentiary data, the data type under investigation, and the exact file path on Android. The online phase compares the Apps with the database that has been build in phase 1. They have improved the existing static analysis tools "Horndroid [25]" to include the sensitive data and its type and include its path in the device, which allows this tool to be typically utilized for forensics investigation slightly more than malware detection and prevention. The limitation of EviHunter, like its preliminary attempts, cannot detect dynamic payload that can only be detected at runtime.

Other attempts such as CHEX [26], SafeDroid [27], AnaDroid [28], ScanDal [29], DroidEnsemble [30], DroidSieve [31], COVA [32] AndroDialysis [33], DroidNative [34], Vulvet [35] and many others; were all limited by the inclination of static analysis and the possibility of malware hiding its malicious act until runtime or using dynamic load, or any other anti-analysis techniques explained in section 5. Researchers in [36] detected the presence of malware using power consumption data. While their approach was mainly focus in detecting in desktop computers, conceptually it can be applied to mobile apps.

SafeDroid [27] is a client-server based antivirus for the android apps which scans the apps and then assigns a label to them based on its signature matching technique which is run by a remote module of the SafeDroid. The client module installed on a device reads the required information from the dex file and sends data to remote service which classify if the app is malicious or not.

Anadroid [28] combines taint analysis with pushdown control flow analysis (CFA) for the java code (but could be used for any high level language). It also uses the approximation based on asynchronous entry points in an app to predict a malicious agenda in an android code.

4.2 Dynamic Analysis

Another technique that has been employed in detecting and mitigating malware is Dynamic analysis. In this technique, analysts execute and monitor App functionality and behavior in a safe environment, such as emulators, simulators, sandboxing, and others. In this technique, the suspected App run and user interaction are imitated to observe the App's actions, behavior, and control-flow to identify malicious actions and categorize the App as malware or benign.

ServiceMonitor [37] is a host-based lightweight detection tool that monitors the application interaction with system services and constructs a statistical Markov chain model as a feature vector to classify the application. Then using the previously constructed model, the Forest algorithm identifies the App either as benign or malicious. It extracts a set of behaviors (methods) by monitoring the app interaction with other services like camera, messaging (i.e. sendSMS). This extraction can build a true behavior model for a malicious app, which hides malicious behaviours into code. To map the extract methods from interaction to a class label (malicious or benign) using a variation of decision tree called Random Forest. Their experiment published results stated that ServiceMonitor could detect malware trying to escalate their privilege, acquire unnecessary permission, collect user information, or gain financial benefits. However, some malware could detect the safe environment and did not trigger their malicious act and therefore been classified as a benign App which caused several false negatives. Additionally, they reported a lesser percentage of false-negative, which has been reported due to some benign Apps acquiring permission that has been classified as dangerous.

TaintDroid [38] is another attempt to identify malware Apps by tagging data, then track the tainted data dissemination through the system. If the data leaves the system through the network or any other port, TiantDroid logs the data, and the Application used it, and the transferred data destination. However, as reported in their paper, TaintDroid can detect only explicit data-flow and be deceived using implicit data-flow.

Motivated by TaintDroid, TaintART [39] comes to be the first information-flow tracking on ART "Android Run time" system. The tool is based on a multi-level data-flow tracking system to minimize the storage overhaul. Firstly, taint logic is performed by tagging and tracking tainted data; when data is released, the tag gets cleared. Then, using tags, the data is tracked throughout the system and record when it leaves the system either via network or any other way. However, similar to its preceding TaintDroid, TaintARt cannot detect explicit data-flow.

On the other hand, Droid-AntiRM [40] aims to support dynamic analysis techniques by taming the anti-analysis techniques such as logic bomb section 5.6. Droid-AntiRMtr identifies those anti-analysis techniques and rewrites the condition statements in the App code to force the malicious behavior to be executed at analysis time, improving the performance of other dynamic analysis tools. Although their work mainly focused on SMS-related services such as sending, blocking, deleting SMS, or leak sensitive data through SMS, they also focus on privilege escalation and gaining root privilege. The limitation of this tool is its inability to detect dynamic code loading or other obfuscation techniques.

DroidScope [41] is a droid-based emulator that performs taint analysis at the machine code level, it is capable of detecting data leakage and root exploit. They additionally developed four system at the API level, taint tracker to trace how the App acquire and leak the tainted data through Java objects.

Similar to EviHunter, [42] proposes a tool that dynamically analyzes App and identifies sensitive data leaks such as GPS location, device ID, browsing history, or any other data that might the type of data they collect and how they use it. However, similar to DroidAntiRm, they cannot detect implicit data-flow. In addition to facing problems with event sequence, if the tool failed to identify and follow the correct sequence of events in the App, the App will not be analyzed accurately.

Other attempts have been published for dynamic analysis such as DL-Droid [43], CopperDroid [44], MAdFraud [45], VetDroid [46], PREC [47], DTAInjection [48] however, they all faced common limitations, that is malware can detect dynamic analysis tools, in that case, they either do not lunch their malicious activities or crashes itself.

4.3 Hybrid Analysis

Other tools use a hybrid technique that combines static and dynamic analysis, such as WifiLeaks [49]. It detects permission requests and data collected using these permissions. They use static analysis to categorize the permissions attained by the App then use dynamic analysis to identify the uses of those permissions for data collection and leaks after that. Their primary focus is WiFi access permission and the Apps that use that permission to leak personal identification information. EspyDroid [50] targets reflection analysis and obfuscation malware by using static analysis to rewrite conditional statements using bytecode instrumentation, optimizing the number of paths to be traveled at the dynamic analysis stage.

Table 1. summarizes anti-malware tools and their techniques; additionally, it covers the dataset they have used for testing and malware functionality they are targeting.

AndroShield [57] also employs hybrid analysis; it utilizes static analysis to perform reverse engineering on the apk file to get code and manifest files. It then utilizes dynamic analysis to monitor the App runtime behavior to detect Data Leak, Intent Crashes, and Insecure Network requests. Other hybrid analysis research is published such as SamaDroid [52], AspectDroid [55], mad4a [56], AndroPyTool [53-54], TAN [58]. However, these tools are confronted with some inherent limitations of static and dynamic analysis (they do not handle all the malware functionality stated in section 3), such as obfuscation or change of package signature. Furthermore, these techniques still faced a problem with zero-day attacks, "new malware striking for the first time." Therefore, comes the need for a new technique that can learn and develop itself, independent of malware signature or library.

Table 1. Anti-malware comparison - tools, techniques, dataset, and malware functionality
(1 Aggressive ads, 2 Remote control, 3 Privilege and permission escalation, 4 Financial charges, 5 Leaking information)

Anti-malware tools	Analysis technique	First released	Dataset	Malware functionality
CHEX [26]	Static	2012	General Apps from Google Play	3 and 5
DroidScope [41]	Dynmaic	2012	Droid-KungFu, DroidDream	3 and 5
ScanDal [29]	Static	2012	General Apps from Google Play	5
AnaDroid [28]	Static	2013	Contagio	3 and 5
VetDroid [46]	Dynamic	2013	Genome, BaseBridge, Zitmo	3
Amandroid [22, 51]	Static	2014	General Apps from Google Play	5
Apposcopy [23]	Static	2014	Droid-KungFu, Geinimi, and GoldDream	5
Flowdroid [21]	Static	2014	DroidBench	5
MAdFraud [45]	Dynamic	2014	Airpush	1 and 2
PREC [47]	Dynamic	2014	Genome, DroidDream, DroidKunfgFu1, DroidKungFu2, Ginger Master, BaseBridge, DroidKungFuSapp	3
TaintDroid [38]	Dynamic	2014	General Apps from Google Play	2 and 5
WifiLeaks [49]	Static and Dynamic	2014	General Apps from Google Play	3 and 5
CopperDroid [44]	Dynamic	2015	Genome, Contagio, McAfee	3 and 5
Horndroid [25]	Static	2016	DroidBench	5
TaintART [39]	Dynamic	2016	General Apps from Google Play	2 and 5
AndroDialysis [33]	Static	2017	Drebin	3
Droid-AntiRM [40]	Dynamic	2017	Drebin, Contagio and Genome	2, 3, 4 and 5
DroidSieve [31]	Static	2017	Drebin, MalGenome	3
DTAInjection [48]	Dynamic	2017	DroidBench	5
SamaDroid [52]	Static & Dynamic	2017	Drebin, Genome	3 and 5
AndroPyTool [53-54]	Static & Dynamic	2018	OmniDroid [54]	3 and 5
AspectDroid [55]	Static & Dynamic	2018	DroidBench	3, 4, and 5
DroidEnsemble [30]	Static	2018	FakeInst, Opfake, FakeInstaller, DroidKungFu, GinMaster, Plankton	3
EviHunter [24]	Static	2018	DroidBench	5
mad4a [56]	Static & Dynamic	2018	ASHISHB, Genome, Drebin, Contagio	3
AndroShield [57]	Static & Dynamic	2019	DroidBench	3 and 5
COVA [32]	Static	2019	AndroZoo	5
EspyDroid [50]	Static & Dynamic	2019	F-Droid, Genome	5
ServiceMonitor [37]	Dynamic	2019	AndroZoo, Drebin, Genome	3 and 5
TAN [58]	Static & Dynamic	2020	Drebin, AMD, AndroZoo	3
Vulvet [35]	Static	2020	Ghera, Mobomarket, Androidpur	3 and 5

5. Anti- Analysis Techniques

As malware spread increases, there are several anti-malware techniques to detect them; however, malware developers are also evolving and updating their ways of bypassing anti-malware techniques. Following are some of the most common anti-analysis techniques that malware developers use to evade anti-malware detection.

5.1 Repackaging

Malware developers use reverse engineering to infect legitimate android Apps. Firstly, they download a legitimate popular App, then reassemble it again after adding a payload of malicious code. Then the new infected App is published back either on the official App store or other stores. When the user unknowingly installs the App, they are vulnerable to those malware attacks to steal their information or make purchases on the App. It is a prevalent malware technique as more than 85% of malware are employing repackaging techniques such as DroidDream, and DroidKungFu 1, 2, 3, and 4. [15, 18, 59].

5.2 Update Payload

Another technique to bypass anti-malware tools that malware developers use is an update attack or dynamic payload. Instead of piggybacking the entire malicious code into the altered App, malware developers embed the malicious payload as a source in the form of an apk/jar file then asks the user to install some critical App updates, which will get the user malicious payload from a remote server. In that way, it overcomes signature-based and static scanning tools in the user device. This technique is adopted by malware families such as BaseBridge, and Plankton [15, 18]. Other malware use polymorphism to change its code every time it gets updated without changing its functionality, such as Opfake [10]. The main advantages polymorphism provides to the malicious code is the exploitation of same methods but doing different behaviors via code overriding using inheritance. The interfacing is exploited to override the actual behavior for the changing the code to include the malicious behavior.

5.3 Dynamic Execution

Malware register and listen to system-wide events such as Boot, Call, and SMS, then when the event happens it triggers its payload of malicious code [18]. Slembunk uses this technique to monitor user activities, and once the user opens their banking application, Slembunk overlays a phishing screen on top of the legitimate App that looks exactly alike and collects all users' banking information from the compromised device [20].

5.4 Code Obfuscation

App developer usually uses code obfuscation techniques to protect their intellectual property from being misused or theft by complicating their code to prevent reverse engineering; additionally, obfuscation leads to a compact App that is faster to run on users' device. For instance, Progurad is a prevalent tool used by App developers to optimize the App code by removing unused classes and methods and replacing lengthy class names with shorter ones [15]. However, malware developers are using this technique to evade manual analysis. Different obfuscation techniques can be used, such as junk code insertion, package renaming, altering control-flow [60-61]. For instance, Obad malware renames all classes and methods to an unreadable form [20]. It runs in the background of your phone. It is well hidden and can't be detected without root privileges. Actually, once Device Administrator privileges have been granted, the malware does not appear in the device administrator list.

The junk code insertion creates additional instructions, which might not execute ever but makes it challenge for the malware analysis to detect a malware after reverse engineering the app due to presence of junk code. The obad takes benefit of renaming the source code files or classes such that they seems genuine to achieve a legitimate goal, however, the code has a high degree obfuscation that it goes undetected by Android OS security. It is a highly sophisticated Trojan for android platform.

The common solution to avoid the attack of such malwares is to use android operating system features such as keeping off the option of auto discover and to use sophisticated anti-virus to defend the device against taks.

5.5 Encryption

Another way of defeating analysis techniques is by encrypting the code that gets decrypted only at runtime. Several encryption techniques can be applied for App hardenings, such as string encryption or class encryption. For analysts to manually examine malware, they need to decrypt it first and then map the ciphered text to plain text to understand their behavior; however, it remains a challenge to the analysts to attain the encryption key. For instance, Fobus uses the fourth entry's class and method name to the JVM stack as a key, while Obad uses a particular Facebook page to generate its key [20, 59].

5.6 Logic Bomb

Some malware to defy dynamic and static analysis techniques do not launch their malicious code once executed. Instead, they wait for a specific event before triggering their malicious action, such as RCSAndroid. Furthermore, some wait for a certain amount of time before triggering their malicious code, referred to as a time bomb, such as HolyColbert. Others start with a login screen that requires the user's credentials to start; without valid credentials, the analysis tool will not be able to proceed on investigating the App functionality and behavior such as Zitmo [9, 62].

6. Conclusion

This paper highlighted the main vulnerabilities of Android OS and how malware developers exploit these vulnerabilities to create several types of attacks such as aggressive ads, remote control, financial charges, privilege and permission escalation, and information leak. The paper introduced several anti-malware techniques to detect and mitigates the malware/attacks. In addition, it classified these techniques into static, dynamic, and hybrid. It also analyzes and evaluates the anti-malware techniques based on the type of attack and the dataset used. Finally, it introduced the most common practices used by malware developers to bypass the anti-malware techniques such as repackaging, update payload, dynamic execution, encryption, and logic bomb. As future work, we will focus on the impact of reinforcement learning and how it may solve the problem of sustainability for the anti-malware approaches.

References

[1] M. Marko, CRAZY Android vs iOS Market Share Discoveries in 2021, leftronic, 2021. https://leftronic.com/blog/android-vs-ios-market-share/. (accessed May 24, 2021).

[2] Diffen.com, Android vs iOS, Diffen LLC, n.d., 2021. https://www.diffen.com/difference/Android_vs_iOS (accessed May 24, 2021).

[3] Av-Test, Security Report 2018/19, July, 2019. [Online]. Available: www.av-test.org.

[4] W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang, Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers, *Future Generation Computer System*, Vol. 78, pp. 987-994, January, 2018.

[5] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, L. Cavallaro, The evolution of android malware and android analysis techniques, *ACM Computing Surveys*, Vol. 49, No. 4, pp. 1-41, December, 2017.

[6] M. K. Alzaylaee, S. Y. Yerima, S. Sezer, Emulator vs real phone: Android malware detection using machine learning, *Proceedings of the 3rd ACM International Workshop on Security and Privacy Analytics*, Scottsdale, Arizona, 2017, pp. 65-72.

[7] DewangNautiyal, Android System Architecture, GeeksforGeeks, 2018. https://www.geeksforgeeks.org/android-system-architecture/ (accessed June 05, 2021).

[8] Platform Architecture, Android Developers. https://developer.android.com/guide/platform (accessed June 20, 2021).

[9] M. Xu, C. Song, Y. Ji, M. Shih, K. Lu, C. Zheng, R. Duan, Y. Jang, B. Lee, C. Qian, S. Lee, T. Kim, Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques, *ACM Computing Survey*, Vol. 49, No. 2, pp. 1-47, June, 2016.

[10] V. Sihag, M. Vardhan, P. Singh, A survey of android application and malware hardening, *Computer Science Review*, Vol. 39, Article No. 100365, February, 2021.

[11] H. Bagheri, E. Kang, S. Malek, D. Jackson, A formal approach for detection of security flaws in the android permission system, *Formal Aspects of Computing*, Vol. 30, No. 5, pp. 525-544, September, 2018.

[12] D. Bohn, Google can't fix the Android update problem, The Verge, 2019, https://www.theverge.com/2019/9/4/20847758/google-android-update-problem-pie-q-treble-mainline (accessed June 09, 2021).

[13] P. Faruki, H. Fereidooni, V. Laxmi, M. Conti, M. Gaur, *Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions*, pp. 1-37, November, 2016, https://arxiv.org/abs/1611.10231.

[14] D. R. Thomas, A. R. Beresford, A. Rice, Security metrics for the android ecosystem, *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, Denver, Colorado, USA, 2015, pp. 87-98.

[15] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, M. Rajarajan, Android security: A survey of issues, malware penetration, and defenses, *IEEE Communcations Surveys and Tutorials*, Vol. 17, No. 2, pp. 998-1022, Second Quarter, 2015.

[16] H. Huang, S. Zhu, K. Chen, P. Liu, From System Services Freezing to System Server Shutdown in Android: All You Need Is a Loop in an App, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, Denver, Colorado, USA, 2015, pp. 1236-1247.

[17] K. Wang, Y. Zhang, P. Liu, Call me back! Attacks on system server and system apps in Android through synchronous callback, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016, pp. 92-103.

[18] Y. Zhou, X. Jiang, Dissecting Android malware: Characterization and evolution, *Proceedings of IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2012, pp. 95-109.

[19] S. Bhandari, W. B. Jaballah, V. Jain, V. Laxmi, A. Zemmari, M. S. Gaur, M. Mosbah, M. Conti, Android inter-app communication threats and detection techniques, *Computers and Security*, Vol. 70, pp. 392-421, September, 2017.

[20] F. Wei, Y. Li, S. Roy, X. Ou, W. Zhou, Deep ground truth analysis of current android malware, in: M. Polychronakis, M. Meier (Eds), *Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2017. Lecture Notes in Computer Science, vol. 10327*, Springer, Cham, 2017, pp. 252-276.

[21] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, P. McDaniel, FLOWDROID: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps, *ACM SIGPLAN Notices*, Vol. 49, No. 6, pp. 259-269, June, 2014.

[22] F. Wei, S. Roy, X. Ou, Robby, Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps, *ACM Transactions on Privacy and Security*, Vol. 21, No. 3, pp. 1-32, August, 2018.

[23] Y. Feng, S. Anand, I. Dillig, A. Aiken, Apposcopy: Semantics-based detection of android malware through static analysis, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, Hong Kong, China, pp. 576-587.

[24] C. C. C. Cheng, C. Shi, N. Z. Gong, Y. Guan, EviHunter: Identifying digital evidence in the permanent storage of android devices via static analysis, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, Canada, 2018, pp. 1338-1350.

[25] S. Calzavara, I. Grishchenko, M. Maffei, HornDroid: Practical and sound static analysis of android applications by SMT solving, *Proceedings of 2016 IEEE European Symposium on Security and Privacy* (EURO S and P), Saarbruecken, Germany, 2016, pp. 47-

62.

[26] L. Lu, Z. Li, Z. Wu, W. Lee, G. Jiang, CHEX: Statically vetting Android apps for component hijacking vulnerabilities, *Proceedings of the 2012 ACM conference on Computer and communications security*, Raleigh, North Carolina, USA, 2012, pp. 229-240.

[27] M. Argyriou, N. Dragoni, A. Spognardi, Analysis and Evaluation of SafeDroid v2.0, a Framework for Detecting Malicious Android Applications, *Security and Communication Networks*, Vol. 2018, pp. 1-16, September, 2018.

[28] S. Liang, A. W. Keep, M. Might, S. Lyde, T. Gilray, P. Aldous, D. V. Horn, Sound and precise malware analysis for Android via pushdown reachability and entry-point saturation, *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, Berlin, Germany, 2013, pp. 21-32.

[29] J. Kim, Y. Yoon, K. Yi, J. Shin, SCANDAL: Static Analyzer for Detecting Privacy Leaks in Android Applications, *IEEE Symposium on Security & Privacy*, San Francisco, CA, USA, 2012, pp. 1-10.

[30] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, X. Zhang, DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features, *IEEE Access*, Vol. 6, pp. 31798-31807, May, 2018.

[31] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, L. Cavallaro, DroidSieve: Fast and accurate classification of obfuscated android malware, *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Scottsdale, Arizona, USA, 2017, pp. 309-320.

[32] L. Luo, E. Bodden, J. Spath, A qualitative analysis of android taint-analysis results, *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, 2019, pp. 102-114.

[33] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, S. Furnell, AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection, *Computers & Security*, Vol. 65, pp. 121-134, March, 2017.

[34] S. Alam, Z. Qu, R. Riley, Y. Chen, V. Rastogi, DroidNative: Automating and optimizing detection of Android native code malware variants, *Computers and Security*, Vol. 65, pp. 230-246, March, 2017.

[35] J. Gajrani, M. Tripathi, V. Laxmi, G. Somani, A. Zemmari, M. S. Gaur, Vulvet: Vetting of Vulnerabilities in Android Apps to Thwart Exploitation, *Digital Threats: Research and Practice*, Vol. 1, No. 2, pp. 1-25, June, 2020.

[36] M. Almshari, G. Tsaramirsis, A. O. Khadidos, S. M. Buhari, F. Q. Khan, A. O. Khadidos, Detection of potentially compromised computer nodes and clusters connected on a smart grid, using power consumption data, *Sensors*, Vol. 20, No. 18, pp. 1-16, September, 2020.

[37] M. Salehi, M. Amini, B. Crispo, Detecting malicious applications using system services request behavior, *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Houston, Texas, USA, 2019, pp. 200-209.

[38] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones, *ACM Transactions on Computer Systems*, Vol. 32, No. 2, pp. 1-29, June, 2014.

[39] M. Sun, T. Wei, J. C. S. Lui, TaintART: A practical multi-level information-flow tracking system for Android RunTime, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria 2016, pp. 331-342.

[40] X. Wang, S. Zhu, D. Zhou, Y. Yang, Droid-AntiRM: Taming control flow anti-Analysis to support automated dynamic analysis of android malware, *Proceedings of the 33rd Annual Computer Security Applications Conference*, Orlando, FL, USA, 2017, pp. 350-361.

[41] L. K. Yan, H. Yin, DroidScope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis, *Security'12: Proceedings of the 21st USENIX conference on Security symposium*, Bellevue, WA, USA, 2012, pp. 569-584.

[42] Z. Xu, C. Shi, C. C. C. Cheng, N. Z. Gong, Y. Guan, A dynamic taint analysis tool for android app forensics, *Proceedings of 2018 IEEE Symposium on Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, 2018, pp. 160-169.

[43] M. K. Alzaylaee, S. Y. Yerima, S. Sezer, DL-Droid: Deep learning based android malware detection using real devices, *Computers and Security*, Vol. 89, Article No. 101663, February, 2020.

[44] K. Tam, S. J. Khan, A. Fattori, L. Cavallaro, CopperDroid: Automatic Reconstruction of Android Malware Behaviors, *Network and Distributed System Security Symposium*, San Diego, CA, USA, 2015, pp. 1-15.

[45] J. Crussell, R. Stevens, H. Chen, MAdFraud: Investigating ad fraud in Android applications, *MobiSys 2014 - Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, Bretton Woods, New Hampshire, USA, 2014, pp. 123-134.

[46] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, B. Zang,Vetting undesirable behaviors in Android apps with permission use analysis, *CCS '13: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, Berlin, Germany, 2013, pp. 611-622.

[47] T.-H. Ho, D. Dean, X. Gu, W. Enck, PREC: Practical Root Exploit Containment for Android Devices, *CODASPY '14: Proceedings of the 4th ACM conference on Data and application security and privacy*, San Antonio, Texas, USA, 2014, pp. 187-198.

[48] J. Schuette, A. Kuechler, D. Titze, Practical application-level dynamic taint analysis of android apps, *Proceedings of 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems*, Sydney, NSW, Australia, 2017, pp. 17-24.

[49] J. P. Achara, M. Cunche, V. Roca, A. Francillon, Short Paper : WifiLeaks : Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission, *WiSec '14: Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile*

networks, Oxford, United Kingdom, 2014, pp. 231-236.

[50] J. Gajrani, U. Agarwal, V. Laxmi, B. Bezawada, M. S. Gaur, M. Tripathi, A. Zemmari, EspyDroid+: Precise reflection analysis of android apps, *Computers and Security*, Vol. 90, Article No. 101688, March, 2020.

[51] F. Wei, S. Roy, X. Ou, Robby, Amandroid: A precise and general inter-component data flow analysis framework for security vetting of Android apps, *Proceedings of the ACM Conference on Computer and Communications Security*, Scottsdale, Arizona, USA, 2014, pp. 1329-1341.

[52] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, H. Yu, SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System, *IEEE Access*, Vol. 6, pp. 4321-4339, January, 2018.

[53] A. Martín, R. Lara-Cabrera, D. Camacho, A new tool for static and dynamic Android malware analysis, *Conference on Data Science and Knowledge Engineering for Sensing Decision Support (FLINS 2018)*, Belfast, Northern Ireland, UK, 2018, pp. 509-516.

[54] A. Martín, R. Lara-Cabrera, D. Camacho, Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset, *Information Fusion*, Vol. 52, pp. 128-142, December, 2019.

[55] A. Ali-Gombe, B. Saltaformaggio, J. Ramanujam, D. Xu, G. G. Richard III, Toward a more dependable hybrid analysis of android malware using aspect-oriented programming, *Computers and Security*, Vol. 73, pp. 235-248, March, 2018.

[56] A. T. Kabakus, I. A. Dogru, An in-depth analysis of Android malware using hybrid techniques, *Digital Investigation*, Vol. 24, pp. 25-33, March, 2018.

[57] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, I. Hegazy, AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach, *Information*, Vol. 10, No. 10, Article No. 326, October, 2019.

[58] R. Surendran, T. Thomas, S. Emmanuel, A TAN based hybrid model for android malware detection, *Journal of Information Security and Applications*, Vol. 54, Article No. 102483, October, 2020.

[59] V. Rastogi, Y. Chen, X. Jiang, DroidChameleon: Evaluating Android anti-malware against transformation attacks, *ASIA CCS 2013 - Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, Hangzhou, China, 2013, pp. 329-334.

[60] M. Zheng, P. P. C. Lee, J. C. S. Lui, ADAM: An automatic and extensible platform to stress test android anti-virus systems, in: U. Flegel, E. Markatos, W. Robertson (Eds), *Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2012. Lecture Notes in Computer Science, vol. 7591*, Springer, Berlin, Heidelberg, 2013, pp. 82-101.

[61] S. Schrittwieser, S. Katzenbeisser, P. Kieseberg, M. Huber, M. Leithner, M. Mulazzani, E. Weippl, Covert Computation - Hiding code in code through compile-time obfuscation, *Computers and Security*, Vol. 42, pp. 13-26, May, 2014.

[62] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, G. Vigna, TriggerScope: Towards Detecting Logic Bombs in Android Applications, *Proceedings of 2016 IEEE Symposium on Security and Privacy (SP 2016)*, San Jose, CA, USA, 2016, pp. 377-396.

[63] H.-S. Oh, B.-J. Kim, H.-K. Choi, S.-M. Moon, Evaluation of Android Dalvik virtual machine, *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES'12)*, Copenhagen, Denmark, 2012, pp. 115-124.

Biographies

Muath Alrammal received his MSc in information technology from Telecom SudParis, Evry, France, in 2007. In 2011, he received his Ph.D. degrees in computer sciences from the University of Paris Est, Paris. In 2011, he joined LACL, University of Paris Est, as post-doctoral researcher. In 2012, he joined LIFO, University of Orleans, France, as a post-doctoral researcher. Between 2013-2017 he joined IT Department in KIC, Abu Dhabi, UAE, where he was an assistant professor. Since 2017 he works for HCT, Abu Dhabi, UAE as assistant professor in CIS department. His current research interests include processing big data in streaming, performance models, selectivity estimation techniques, and Machine learning. Dr. Alrammal is a member of LaMHA research group.

Munir Naveed obtained his first degree in software engineering and a PhD in automated planning for real-time-strategy games in 2012 from University of Huddersfield. He has been exploring AI in different domains eg. computer games, big data and network security since 2014. His focus of his research is designing new algorithms to solve problems for real-time applications.

Suzan Sallam received the M. Sc. degree in Information, Network, & Computer Security (Cybersecurity) from the New York Institute of Technology, UAE, in 2018 and the B.Sc. degree in Electrical Engineering, Computer & Automatic Control from Tanta University, Egypt, in 2005. She currently works as a lecturer at Khawarizmi International College. Her current research interest includes IoT security.

Georgios Tsaramirsis is an academic working as a full time faculty at higher colleges of technologies (HCT), Abu Dhabi, UAE. George received his PhD from King's College London, University of London, UK. Before joining HCT, George worked at King Abdulaziz University, Jeddah, Saudi Arabia for 8 years, teaching various undergraduate and postgraduate courses. Prior to joining academia, George worked for a number of companies including Accenture, UK. George also is a co-founder of Infosuccess3D, Athens, Greece an Interactive content development firm that has published a number of computer games, VR and AR applications. George has a significant research background with a plethora of funded projects and scientific publications including papers in top 2%, Q1 journals. George has delivered numerous webinars in well-known international institutions, organized multiple conferences and workshops and edited a number of scientific books. George has received a number of rewards, including best teacher, best applied research project and 3rd position in Arab Mate Rov competition.