electronics

MDPI

Article

Benchmarking Android Malware Analysis Tools

Javier Bermejo Higuera, Javier Morales Moreno, Juan Ramón Bermejo Higuera *, Juan Antonio Sicilia Montalvo, Gustavo Javier Barreiro Martillo and Tomas Miguel Sureda Riera

Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Avda. de La Paz 137, 26006 Logroño, La Rioja, Spain; javier.bermejo@unir.net (J.B.H.); moralesjmoreno@gmail.com (J.M.M.); juanantonio.sicilia@unir.net (J.A.S.M.); gustabakn18@hotmail.com (G.J.B.M.); tomasmiguel.sureda@unir.net (T.M.S.R.)

* Correspondence: juanramon.bermejo@unir.net

Abstract: Today, malware is arguably one of the biggest challenges organisations face from a cybersecurity standpoint, regardless of the types of devices used in the organisation. One of the most malware-attacked mobile operating systems today is Android. In response to this threat, this paper presents research on the functionalities and performance of different malicious Android application package analysis tools, including one that uses machine learning techniques. In addition, it investigates how these tools streamline the detection, classification, and analysis of malicious Android Application Packages (APKs) for Android operating system devices. As a result of the research included in this article, it can be highlighted that the AndroPytool, a tool that uses machine learning (ML) techniques, obtained the best results with an accuracy of 0.986, so it can be affirmed that the tools that use artificial intelligence techniques used in this study are more efficient in terms of detection capacity. On the other hand, of the online tools analysed, Virustotal and Pithus obtained the best results. Based on the above, new approaches can be suggested in the specification, design, and development of new tools that help to analyse, from a cybersecurity point of view, the code of applications developed for this environment.

Keywords: malware analysis; sandbox; Android malware; IoT

Citation: Bermejo Higuera, J.; Morales Moreno, J.; Bermejo Higuera, J.R.; Sicilia Montalvo, J.A.; Barreiro Martillo, G.J.; Sureda Riera, T.M. Benchmarking Android Malware Analysis Tools. *Electronics* **2024**, *13*, 2103. https://doi.org/10.3390/electronics13112103

Academic Editors: Juan-Carlos Cano and Aryya Gangopadhyay

Received: 20 April 2024
Revised: 19 May 2024
Accepted: 27 May 2024
Published: 28 May 2024

1. Introduction

In recent years, the amount of malware on smartphones running Android operating systems has increased rapidly, mainly due to the complexity of the development and maintaining modern operating systems that manage these devices. Today, this type of threat has become one of the biggest security problems facing any organisation. Because of the current advancements in programming, the creative ways that developers hide malicious code (malware jumbling), and the added factor of hyper-availability and hyperconnectivity in today's world, malware investigation, analysis, identification, and classification are becoming a real and increasingly difficult problem to deal with. Android is an open-source operating system with more than 1 billion users, covering many devices like smartphones, tablets, Internet of Things (IoT) devices, gadgets, and so on.

Cybercriminals are well aware of the weaknesses of a large percentage of ordinary users, who are unaware of the importance of the data that are exposed every day and every minute on the network, waiting to be "stolen" for fraudulent use.

The amount of sensitive data currently processed and stored on these devices is increasing the number of attacks [1], which is a problem of concern to society. It is a priority for organisations to use tools to analyse, detect, and classify malware on devices using the Android operating system.

The malicious payload available at these malware executables can be defined as "any code added, changed or removed from a software system to intentionally cause harm

or subvert the intended function of the system", the definition used by McGraw and Morrissette in [2].

In the last decade, many methods based on machine learning and data mining were applied to detect intrusions, malware, and their classification, where many clustering and classification techniques involved cataloguing malware into known families or identifying new families of malicious code.

This problem, commonly addressed by manual procedures, has taken on additional dimensions involving the use of new tools capable of automating this process with large numbers of suspicious Android Application Packages (APKs). Among these, ML techniques address a hopeful arrangement.

The use of ML techniques for the specific task of malware analysis is largely due to the idea that artificial intelligence (AI) can automatically learn from the study of data, identify patterns, and make decisions with little human interference, and thus automate the building of analytical models. In other words, this technique allows data to be taken and broken down and then converted into predictions.

ML significantly reduces effort, saves time, and is a cost-effective tool that replaces multiple teams working on analysing, processing, and performing regression tests on data. It provides accurate results and helps organisations build statistical models based on real-time data. It has positioned itself as a powerful mechanism for solving diverse, vast, and complex distinct challenges. This concept is classified as a subfield of artificial intelligence (AI), which is a fundamental part of many Data Mining processes, which are concerned with extracting knowledge from enormous volumes of data (datasets). To define the term "machine learning", Kevin P. Murphy's precise definition is used, included in his book "Machine Learning: A Probabilistic Perspective" [3], comprising "a set of methods that can automatically detect patterns in data and then use the uncovered patterns to predict future data or to perform other types of decision making under conditions of unpredictability".

In this article, firstly, an analysis of malware targeting this platform, existing malware analysis techniques, and related work are presented. Next, a specific research method (Section 3.1 Methodological design) is designed and developed to carry out the research that aims to evaluate the effectiveness of tools that use or do not use ML techniques, to address the detection and classification of malware on Android devices, using different adapted benchmarks.

Then, a series of experiments are performed against the two datasets of malware and goodware (benign applications) APKs (a dataset of 7003 APKs) and a dataset of 106 APKs. We use a method based on several selected metrics to obtain different rankings of the tools, according to different criticality objectives according to the desired weight of TPs and FPs. After analysing the tools, practitioners will be able to choose the most appropriate tools to protect Android-based devices and their malware scanning needs.

In short, this article mainly makes the following contributions:

- Design of a method for carrying out the research presented in this paper.
- A functional analysis of the tools is based on the work in which a comparison of various malware analysis tools available on the Internet is performed.
- A comparison based on defined metrics of different tools for detecting malicious code in Android applications.
- Based on the comparison made in the previous point, it is determined whether the tools that use ML in the malicious code detection method present better results, advantages, or disadvantages.

In conclusion, this research attempts to demonstrate the benefits of using machine learning tools as a method for detecting known families of malicious code for Android applications. The need for the use of more complete tools is justified, providing the essential foundations for establishing a systematic process of malware analysis for Android applications.

The article is structured as follows. Section 2 surveys our current knowledge of the types of malware analysis techniques and available tools for Android operating systems. Then, Section 3 includes the methodological process followed in carrying out the research,

the functional analysis of the selected tools, the process of carrying out the experiments, and the detailed results obtained from the application of the aforementioned method. Finally, conclusions and future research guidelines are included in Section 4.

2. Background and Related Work

2.1. Android Malware

Android is one of the most important operating systems for mobile devices today, being used in many devices such as smartphones, IoT, smart TV gadgets, and many others. The first version was released in November 2007 [4], although it was commercialised at the end of 2008. Since then, it has experienced extraordinary growth that has led it to become the most widely used mobile operating system in the world. It is recognised for its open-source code, architecture, and its multiplatform approach as well as its kernel from the Linux operating system.

It stands out from the rest of the competitors with a market share [5] of 85% according to official sources, and will reach 87% market share in 2022. Android is not only found in mobile terminals but also various environments such as critical industrial systems, servers, network nodes, telephony, IoT devices, gadgets, tablets, etc. [6]. Therefore, cybercriminals have focused their efforts on this operating system, making it the most targeted platform by cybercriminals [7], with over 900 million devices, and over 1 million applications, and increasing its growth year after year.

The characteristics of mobile devices stand out for the presence of sensors (GPS, gyroscopes, microphones), open connections (Bluetooth, Wi-Fi), and hosting of third-party applications. But these are not all advantages; all the above-mentioned aspects present security problems. Apart from storing sensitive data, the sensors that incorporate these technologies have been shown to collect information without the user being aware of it. It is for all these reasons that the proportion of malicious software or "malware" has shot up at the same rate as its use.

Detecting malware is a difficult task, due not only to the numbers but also to the variety of families available to attackers. In addition, cybercriminals have a variety of techniques at their disposal to bypass the controls of malicious applications, such as hiding malware in code (obfuscation), targeted permission elevation attacks, or API calls [8].

Today, there are two ways of detecting whether the software is malicious or benign. One is "signature-based" (reactive) analysis, which involves rules in detection systems, and antivirus software, which recognises the characteristic patterns of known threats. The other method available to classify and detect whether the software is suspicious or malicious is heuristics (proactive), which comprises observing behaviour or determining whether or not it is benign, using a machine learning system or machine learning [9].

Given that malware expands rapidly, machine learning offers a way to handle such threats, using the collection of known malware and automatically looking for patterns of behaviour, to detect new malware from families not yet classified [10], and thus constantly improve malware detection, without the need to update signatures.

Malware develops rapidly on any of the known platforms. The method of automatic learning or machine learning offers a way to handle these threats, using the collection of known malware families and looking for behaviour patterns automatically, to detect new malware from families not yet classified, and thus constantly improve in malware detection, with no need to update signatures. The steps that have led to the use of these techniques to analyse and predict malware behaviour are described below [11].

- Descriptive analysis: Knowledge of the past. It reports organisations about "what has happened" and how they can learn from their past actions to settle on better decisions later.
- Predictive analytics: It uses different static models and AI calculations to examine past information and anticipates future outcomes.
- Prescriptive analysis: Results-based solutions. It uses simulation and optimisation algorithms to guide organisations on a secure path by recommending useful solutions.

Android was attacked and threatened by malware in 2010. Not long after this date, the first malware designed specifically for this platform was found, particularly a Trojan (SMS.AndroidOS.FakePlayer [12]). From that time onwards, attackers have repeatedly targeted this platform as the main target of their attacks, mainly due to various reasons, such as its large market share.

Based on the Malware Bytes [13] threat catalogue, the different categories of malware most commonly discovered today are given as follows:

- Pre-installed. It is a type of built-in malware that can be found mostly in low-budget manufacturers. The case of the UMX mobile phone, financed by the United States, that was manufactured with pre-installed and immovable Trojans is known [13].
- HiddenAds. The second most identified and detected malware is an enormous group of Android Trojans that is classified as Android/Trojan.HiddenAds. It is based on a silent installation in which the only symptoms of HiddenAds are displaying ads aggressively, by any means necessary. This includes but is not limited to ads in notifications, full-screen pop-ups, and on the lock screen. It does not inform users who install HiddenAds applications in advance about advertising behaviour.
- Stalkware (Monitoring). The term can apply to any application that potentially allows it to be used to track the user or track others. It incorporates the gathering of the following data and information from others' devices without their consent: GPS location data, call logs, photos, emails, contact lists, text messages, non-public activities on social networks, and other personal information.

2.2. Malware Analysis

There are different methods used to perform malware analysis: dynamic analysis, static analysis, hybrid analysis, and memory analysis. Static analysis includes analysis of a given malware sample without executing it, while dynamic analysis is carried out systematically in a controlled environment [14], and hybrid analysis is a combination of both.

Methods that fall within the scope of static analysis allude to the extraction of useful data from the executable and do not involve running the specimen in question. This permits the building of efficient and effective patterns to detect malware. Notwithstanding, obfuscation techniques represent a major impediment to the success of this approach [15]. The "static analysis" incorporates the utilisation of reverse engineering methods to analyse the instruction set that characterises the functioning of the application [16]. In addition, and with a focus on the Android platform, a wide range of features can be discovered through this type of analysis. Data were collected from the Android manifest or assets that fall in this class.

Figure 1 shows the diagram of the various files and folders obtained once the APK has been decompressed. Because several of these files include encrypted data, it is necessary to utilise specific tools to extract the human-readable files. Files included in the several folders give various data that might be employed to categorise sample actions. For example, /META-INF/ includes certificates, designer data, or data to run the jar file. The resources arsc and res folders are linked to distinct methods for importing resources. The lib folder stores the aggregated libraries.

Finally, the two files that give the best applicable elements to handle a malware investigation assignment are presented in Figure 1; classes.dex specifies the application code in the way of Dalvik bytecode. From here, a catalogue of system commands, API calls, or collectors can be recovered. The other significant file is AndroidManifest.xml, which states the list of permissions, package names, or intent–filter relationships [17].

Dynamic analysis involves a method in which the specimen is run in a monitored environment, where the supervising service takes any events or actions that occur during execution [8]. This kind of investigation can provide insights not previously discovered by the static analysis workflow (mostly because of the utilisation of dynamic code methods). It is significantly more costly and less effective [18]. Notwithstanding, this is another down-

side, as existing documents show how malicious code can be detected in an application when a piece of code runs on a virtual platform.

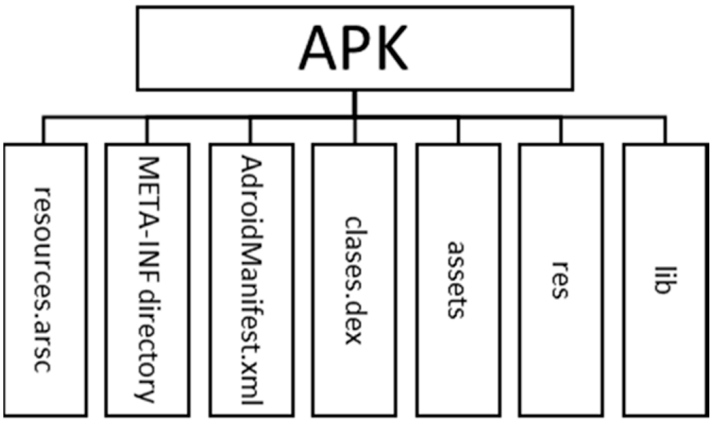


Figure 1. File and folder structure after unzipping an APK.

Another approach combining both dynamic and static analysis techniques is hybrid analysis. The advantages offered by each sort of analysis can create more robust classification, detection, and analysis models compared to others that select a single point of view. A hybrid analysis [19] represents the most efficient approach to use. The computational cost can be elevated. In most instances, the two-phase analysis process is the most suitable solution. Therefore, the first level deals with the static properties that define the nature of the specimen. But in situations where categorisation is not achieved through a particular level of accuracy, a dynamic analysis is needed.

The article presented by Chakkaravarthy et al. [20] proposed a hybrid analysis method to identify Advanced Persistent Threats (APTs). The suggested technique is called "Behavior-based Sandboxing (BbS)". It uses a mixture of memory, dynamic, static, and system state analysis procedures. In the conference article presented by authors Aslan and Samet [21], a method is proposed in which dynamic and static analysis tools are used to determine whether a sample is known malware. Using different tools results in increasing the detection rate of malware.

Finally, one type of analysis that provides very good results is that of the memory of the infected machine. As stated by Montes et al. in the reference article [22], "any process or object in an Operating System will have to pass through its RAM at some point. Some researchers have considered the RAM as an ideal place to perform their malware analysis". This analysis comprises analysing the capture of the computer's physical memory to analyse, identify, and obtain evidence of the malicious activity performed by the malware. In the article by Tien et al. [23], a sandbox solution is presented to observe live memory data and analyse system behaviour using memory forensics methods.

This technique is especially useful for the analysis of threats known as "fileless malware" or "memory-based attacks". In the article published by Gadgil et al. [24], they give an insight into how certain types of malware do not install files on the target's hard drive to execute malicious activities. Malware lives directly in memory and can take advantage of system tools to inject code into trusted and safe processes such as javaw.exe or iexplorer.exe.

2.3. Related Work

In preparing this work, an investigation has been carried out on existing methodologies that combine various ML techniques to develop malware classification tools for Android applications.

Malware analysis describes a set of methods and procedures that aims to discover the collection of actions that a suspicious specimen file can perform [25]. The above permits to us obtain important data to recognise malicious and corrupt payloads. The two different methods in which malware analysis methods can be organised are static and dynamic

analysis, but it is also possible to combine the two. Then there is talk of hybrid analysis, which is also possible. Every one of these methods shows various methods aiming to gather important data capable of describing the behaviour of the malicious code obtained from the dataset.

DroidMat [26] is a tool in which API calls have been utilised depending on the element they are associated with in runtime. The date associated with permissions, intention actions, or inter-component communications (ICCs) is contemplated. Clustering algorithms permit improved malware behaviour modelling, though Naive Bayes and k-NN run the learning procedure.

DroidMiner [27] and DroidAPIMiner [28] are additional instances of work carried out, suggesting API calls as the most important illustrative feature to train malware classifiers.

The initially generated Component Behaviour Charts (CBGs) correspond to the present links that connect the API resources and permissions with the actions made. Next, the algorithms Support Vector Machines (SVM), Bayesian Networks (Naive Bayes), Random Forest, and Decision Tree are trained. In DroidAPIMiner, special interest is given to threatening calls during training of the C4.5, ID5, SVM, and k-NN algorithms.

There is varied literature that focuses on the usage of ML methods to develop malware detection and classification methods [17]. The simple removal of static features beyond the complete description they deliver about application behaviour and intent is the reason for the significant quantity of research conducted, based on the following self-learning algorithms: Naive Bayes, SVM, Decision Tree, and Stochastic Gradient Descent (SGD) [29].

In DroidSIFT [30], API call dependency graphs and likeness metrics to classify and detect zero-day malware allow the training of a Bayes network classifier. Combined with permissions and other system events and calls, this provides an alternate forest model [31]. The creators demonstrate that this classifier, founded on the decision tree algorithm, offers better results compared to SVMs. A particular technique known as MOCDroid makes a malware classifier with a transformative method [32].

Another approach already studied previously involves the use of the MosBF framework [33] for analysing malware from a dataset as APK files for both benign or goodware applications or malicious Android applications. Another work like the past one carried out by Jianlin Xu et al. [34] proposes a mechanism for the security evaluation of mobile application (APK) applications using a prototype of a tool called MobSafe that combines static and dynamic analysis techniques to systematically evaluate an Android application.

In the work of Asaf Shabtai [35], an anomaly detection system is described that monitors the device frequently for suspicious events and performs machine learning to classify the results as benign or malicious based on the behaviour of the malware. However, this technique damages the device's battery by making multiple requests. In the TaintDriod framework, the device is monitored in real time, and the user is alerted when suspicious activity is presented by an application running on the device.

Similar work to the one presented in this article is performed by Agrawal and Trivedi [36], in which they analyse the various types of malware scanning tools for the Android operating system. The paper provides a comparison of the tools, revealing their advantages and disadvantages. It also concludes that most of the tools only perform static malware scanning and do not support bulk scanning of files.

In [37], a study of deep learning techniques, one of the groups of devices that typically use the Android operating system, which allows for the detection of malware in the IoT world, is carried out. Finally, Ashawa and Morris [38] conducted a systematic review of various papers on the different techniques for detecting malware on Android. Their main conclusion is that most detection techniques are not very effective in detecting obfuscated and zero-day malware.

After carrying out the above study and looking at the result of the comparison of existing techniques and studies carried out to date, it can be seen that there is no standardised use of tools for analysing malware in Android applications using frameworks or security frameworks [39] with a self-learning techniques engine. This is where it is intended to

contribute feasible research in this area, which is not yet covered or not with the necessary clarity and specification required by the technical community. The techniques or tools studied have limitations that must be considered when choosing the tool that best suits the needs of malware analysis.

3. Experimental Research

This paper explains the benefits of using ML tools as an analysis method to detect known families of malicious code for Android applications. It justifies the need for the use of more complete tools, which offer the indispensable basis to establish the realisation of a systematic process of malware analysis for Android applications.

The purpose of this article is to explain the benefits of using artificial intelligence. It can automatically learn from studying data, identify patterns, and make decisions with little human intervention, thus automating the construction of analytical models for detecting and analysing malware in specific applications for Android environments. As can be seen from the state of the art, there are various techniques for doing this. However, there is no standardised way to bring all these techniques together and form a procedure or working strategy that efficiently facilitates all the steps to be followed in the event of a malicious event generated by these applications.

3.1. Methodological Design

The following paragraphs describe how the experimental pilot was conducted, where data have been obtained from the analysis brought by the chosen tool, and in return, they have been used to draw conclusions and possible future work.

The following research hypothesis is established: "Are tools that use existing machine learning techniques more effective than tools that do not use Artificial Intelligence engines?".

This implies that one of the most important objectives of this research work is to establish the benefits of using machine learning tools as an analysis method for detecting known malicious code families for Android applications. Another purpose of this work is to evaluate and test how the utilisation of malware analysis tools for Android devices speeds up obtaining results, analysing, and classifying malicious applications. A study of various tools will be carried out and compared from a functional and performance point of view based on a series of defined metrics. About this experimental pilot, a work program has been developed with the following steps:

1. Choice of the different tools to be analysed. At least one of them must use machine learning techniques.
2. Implementation and configuration of the virtual analysis environment.
3. Dataset construction. Depending on the characteristics of the tools to be analysed, different datasets will be constructed.
4. Selection and definition of metrics to analyse the performance of the tools.
5. Functional analysis of selected tools.
6. Running the different experiments. Execution of the different tools against the different datasets and carrying out the different analyses and assessments of the performance of the tools based on the established metrics.
    - Experiment 1: Analysis of the AndroPyTool application.
    - Experiment 2: Analysis of the MobSf application.
    - Experiment 3: Analysis of the online applications.
7. Discussion and lessons learned.

The above procedure is shown in Figure 2.

The method used to carry out the analysis of the APKs in the different tools selected for the experimental pilot is shown in Figure 3.

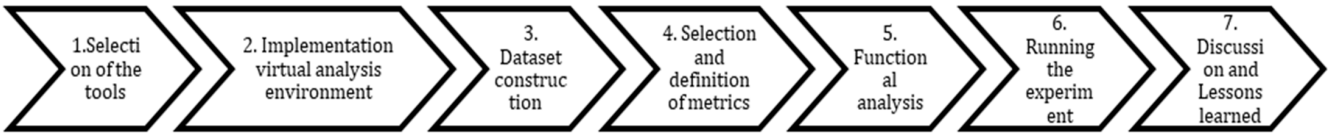


Figure 2. Work method.

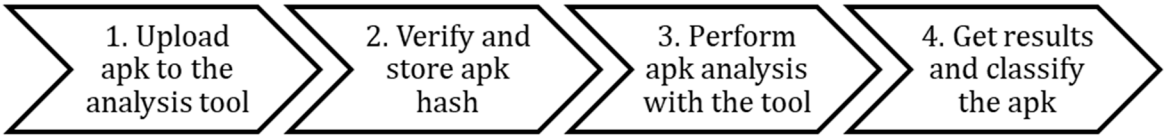


Figure 3. Method used to carry out the analysis of the APKs.

3.2. Different Tools to Be Analysed

Research has been carried out on the different malware analysis tools that currently exist for Android operating systems, considering the research hypothesis set out in the previous paragraph. For this purpose, a tool that uses machine learning techniques has been selected, AndroPyTool, whose performance will be compared to all the others selected.

A multitude of APK analysis tools can be found on the internet. Many of them have limits to avoid bugs on the platform or indiscriminate use of it. Others require a previous registration to make more extensive use of it.

As a general feature, all the tools have an input interface that allows the loading of the malware to be analysed; the big difference is that some of them have an API that allows the automatic loading of the files to be analysed through a script, and others do not. This automatic upload allows bulk or non-bulk scanning. Some are available for online use or can be installed locally in a laboratory.

Based on the above, a specific set of tools was selected for this research based on their functionality, user-friendly approach, use of the different types of analysis methods, whether they are free to use or not, and their available online option. The selection of online tools was based on the work of Agrawal and Trivedi [36]. The tools selected are:

- AndroPyTool.
- Mobile Security Framework (MobSf).
- Online tools: Virustotal, Intezer Analyzer, Hybrid Analysis, Joe Sandbox, Metadefender, Jotti, and Pithus.

Once this work has been completed, an attempt will be made to adapt the possible test scenarios of the selected Android malware scanning tools to their characteristics. In addition, the operation of the tool using machine learning techniques (AndroPyTool), on which this work is based, will be described.

In the following sections, the above tools are compared and analysed. To carry out this experimental pilot and to ratify the use of tools with self-learning mechanisms, the results obtained have been compared with the other selected tools.

3.2.1. AndroPyTool

The AndroPyTool [40] security framework will be used as a reference for this entire study. This tool was designed and applied to automate the method of analysing APKs by extracting representative behaviour, using static and dynamic analysis, to distinguish between infected and benign applications. Using this tool makes it possible to effectively gain diverse behavioural data that would otherwise require a considerable investment of time and personnel resources.

AndroPyTool is an open-source Python tool where several scripts are executed sequentially, using machine learning techniques. The data collected during this procedure might be categorised into three distinct types: pre-static, static, and dynamic characteristics [41].

The last phase of the analysis performed by this tool consists of the extraction and processing of features using machine learning techniques, as shown in Figure 4, such as

random forest and bagging classifier. It processes all the data collected in the previous stages to obtain the main features of the APK and proceed to a final classification [42].

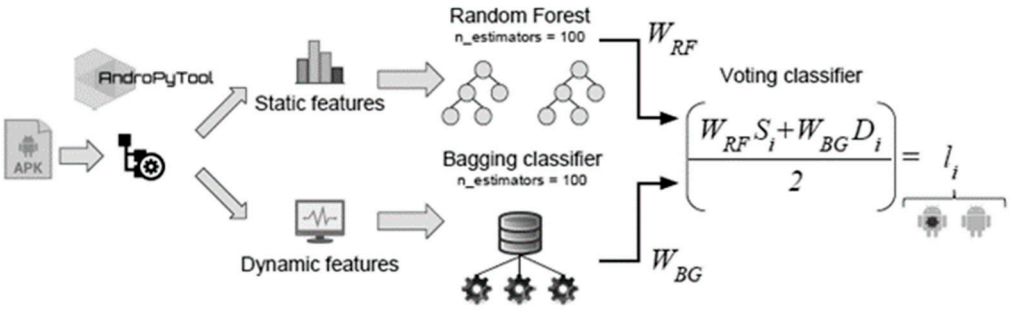


Figure 4. Schematic of the operation approach of AndroPyTool [41].

3.2.2. MobSf

MobSf (https://github.com/MobSF/Mobile-Security-Framework-MobSF) (accessed on 18 February 2024) is an open-source framework [34] that combines static and dynamic analysis methods to comprehensively evaluate an Android OS application. It also allows for cloud-based analysis and data mining in a significant amount of time. It is worth mentioning that, although it is a good framework, no system can solve all the difficulties that malware can generate.

It contains a set of tools to decode, debug, review code, and comprehensively perform a penetration test, aiming to minimise analysis time with a single tool in a few steps. The most important of these tools is the Static Android Analysis Framework (SAAF) for static analysis and the Security Evaluation Framework (ASEF) for dynamic or behavioural analysis. This tool supports binary files (APK, IPA, and APPX) as well as compressed source code.

It is scalable and allows for the easy addition of custom rules. YARA (Yara is a tool designed to identify and classify malware by creating rules to detect strings, instruction sequences, regular expressions, and other patterns within malicious files) rules can be added to classify malicious code based on the characteristics of each sample. From text strings, the rules can identify instruction sequences, regular expressions, or other patterns within the application, for example, if the code contains information to connect to a specific URL. This can be used to find malware variants that are spreading as a type of targeted attack.

3.2.3. Virustotal

It is a well-known open-access online tool (https://www.virustotal.com/gui/home/upload) (accessed on 18 February 2024) that permits checking suspicious files, hash, APKs, URLs, etc. It includes over 70 antivirus engines. For every one of them, if positive, a label identifying the sort or group of malware found is generated. It allows for rapid detection of many categories of malware. The uploaded file is up to 150 MB in size.

3.2.4. Intezer Analyzer

It is a malware analysis tool (https://analyze.intezer.com/) (accessed on 18 February 2024) available online [43] with different licensing options, in which it is possible to analyse malicious files in multiple formats (exe, .dll, .sys, ELF, Zip, RAR, TAR, 7-Zip, APK, msi, doc, xls, ppt, PDF, PowerShell scripts, vbs, and js). It uses the technique "genetic analysis of malware", and the basic premise is that "all software, whether legitimate or malicious, is composed of previously written code" which allows identifying new types of malware by comparing the code with previously found threats.

3.2.5. Hybrid Analysis

This is an advanced security tool (https://hybrid-analysis.com/) (accessed on 18 February 2024) developed by Payload Security that classifies, detects, and analyses uniden-

tified threats using distinctive hybrid scanning technology. There are suspicious files and URLs that it scans. It also provides an in-depth analysis of code and programs for Windows systems. It performs both a hybrid analysis using Falcon Sandbox that combines runtime data, static analysis, and memory dump and a multi-scan analysis that uploads the sample to Metadefender and Virustotal. This tool gives a threat score that can be taken as a metric. In addition, incident response and risk assessment reports are provided.

3.2.6. Joe Sandbox

Another online sandbox (https://www.joesandbox.com/#windows) (accessed on 18 February 2024) available, but which does require registration with a professional email address, is Joe Sandbox [44]. It has a Web API, but only for those who have a Cloud Pro account, which comes at a cost. The cloud sandbox offered by Joe Sandbox detects and analyses malicious files, URLs on Windows OS, and the hash value on different platforms such as MacOS, Android, Linux, and iOS for suspicious events. It carries out profound malware analysis and creates exhaustive and point-by-point investigation reports. It only allows running a maximum of 15 scans/month, 5 scans/day on Linux, Windows, and Android with limited scan results. This tool has an upload limit of 25 MB, making it ineffective if our purpose is to analyse a dataset with thousands of files (APKs).

3.2.7. Metadefender Cloud

It is an online malware-scanning utility (https://metadefender.opswat.com/) (accessed on 18 February 2024) that provides the ability to upload and scan files up to 140 MB in size. It performs two types of analysis: static, in which a multiscan analysis is performed with up to 35 different antivirus engines including McAfee, Kaspersky, AVG, etc., and an analysis of the metadata of the APK, mainly of the dangerousness of the permissions; it requires running and dynamic analysis with a sandbox that does not work for the case of APKs.

3.2.8. Jotti

Jotti's malware scan (https://virusscan.jotti.org/es-ES/scan-file) (accessed on 18 February 2024) is a free service that scans a file against more than 13 antivirus engines, including Avast, F-Secure, Sophos, etc. It permits the upload of up to 5 files simultaneously, up to a limit of 250 MB for the 5 files and 25 MB for each one. It also permits the download and use of a client to upload files without using the browser. Finally, it is worth mentioning that it has an API for bulk file scanning.

3.2.9. Pithus

Pithus (https://beta.pithus.org/) (accessed on 18 February 2024) is a free and exclusive open-source malware analysis platform specially developed for the analysis of APKs. It has been recently developed and its current version is in beta. It performs several types of analysis, such as fingerprint, control flow analysis, and threat intelligence; basically, it submits the sample to Virustotal, code analysis, behaviour analysis, and network analysis. It also has a fuzzy tool to verify if the sample belongs to a known malicious family.

3.3. Implementation and Configuration of the Virtual Analysis Environment

The experiment with the AndroPyTool and MobSafe tools has required the implementation of a virtual analysis environment. For the online tools, this was unnecessary. For implementing the virtual analysis environment of the AndroPyToll tool, the Docker tool [45] has been used, as it does not require installing dependencies, downloading the several necessary repositories, or configuring the Android emulator for the dynamic analysis phase.

On the Ubuntu machine mentioned above, the MobSf tool is also installed. The aforementioned lab machine will allow the server to run together with the MobSf application to perform static analysis of the APK file, e.g., analysing the source code and the permis-

sions that the application has on the device, along with the dangers that each of these can generate.

To perform dynamic analysis, the MobSf tool provides several ways to emulate an Android system, either using a virtual machine in Virtual Box, an ARM emulator, or finally by a physical device; the latter option is the least ideal, as it will infect the device to be tested, so the first option is implemented.

In this phase, it is also necessary to guarantee that the analysis can be carried out in its entirety so that the state of the virtual machine can be returned with a snapshot, or that the emulator can return to its previous state so that it does not interfere in each analysis that is carried out. In this sense, the MobSf tool always takes the main snapshot to be able to return to that state so that no problem arises when the analysis of an application is restarted.

Once the lab machine has been configured with the application server on which MobSf will run, the applications are uploaded for analysis.

3.4. Dataset Construction

Given the different malware analysis tools selected in this research, different datasets have been constructed according to their capabilities for automated bulk file scanning of the tools through the execution of scripts.

For the construction of the datasets, it is started from the one provided by AndroZoo [46] for testing. This dataset contains over 17,000 different APKs in total, where 7002 APKs have been used to carry out this work. The way to obtain this dataset has been stated on one GitHub page, where the use and installation are described [47]. Once the required API key provided by the University of Luxembourg (AndroZoo) is available, it requires a manual download, using a CSV file of as many entries as there are APKs in the dataset, for example:

```
curl -O --remote-header-name -G -d apikey=${APIKEY} -d
                    sha256=${SHA256} \
            https://androzoo.uni.lu/api/download
```

To avoid performing this task manually, a script has been designed based on a plain text file (CSV AndroZoo), from which all fields that are not relevant have been discarded, redirecting the output to this text file, from where the extraction and download have been carried out in a fully automated way. The CSV file contains the sha256 key, which will be necessary to download beforehand.

Once enough APKs have been downloaded to carry out a reliable study of them, it is necessary to make sure that benign files are present in all the experiments to be carried out. A specific dataset of the Canadian Institute for Cybersecurity has been used [48], where 1602 non-malicious application files have been selected. With all these files and the malicious ones downloaded, a total of 7002 APKs are obtained, to carry out the different experiments of this research.

Once the base dataset was obtained, specific datasets were designed for the different experiments that were intended to be carried out in this research:

- Experiment 1: Analysis of the AndroPyTool application. The dataset built for the analysis of this tool is composed of 7002 APKs, 1602 of them benign (goodware) and 5400 malicious (malware). This experiment includes many APKs due to the tool's ability to perform mass scans using a script.
- Experiment 2: Analysis with AndroPyTool, MobSf, and online application tool. The dataset built for the analysis of these tools consists of 53 goodware and 53 malware applications. This dataset is smaller than the previous one given that some of the tools do not allow automated bulk file scans, its high manual interaction, and the high time consumption that other tools require.

3.5. Metrics to Analyse the Performance of the Tools

The selection of the metrics to be applied in the experiment was based on the articles by Surera et al. [49] and Antunes et al. [50], which propose a series of metrics to measure the effectiveness of different tools based on the data gathered when running them against a series of benchmarks. In the following paragraphs, the reasoning and decisions made in selecting the most appropriate metrics for the experiment are explained.

About detecting and classifying malicious APKs, the tools can be considered binary classifiers, as they usually classify the target APK into one of the following two classes: APKs that are "clean or trusted" and "malicious". In such a case, the best-performing tools are those with a maximum of True Positives (TP), as they detect the most malicious APKs, and a minimum of False Negatives (FN) and False Positives (FP). In the above, it should be noted that it is more important to have a minimum of FN than a minimum of FP, because a malicious APK that has not been detected as malicious will cause the user a false sense of security. After all, if the user believes that the APK is not malicious and uses it, when in fact it is not, this could cause problems.

The confusion matrix used in the experiment and the meaning of the abbreviations TP, TN, FP, and FN are presented in Table 1.

Table 1. Types of diagnosis.

Types of Diagnosis		**Malicious APK**	
		Absent	**Present**
Diagnostic Test	**Negative**	TN (True negative). Files (APKs) that the tool has correctly classified as negative or goodware.	FN (False Negative). Files (APKs) that are extracted from a malicious dataset but have been classified as non-malicious by the tool.
	Positive	FP (False Positive). Files (APKs) that the tool classifies as infected that are benign applications.	TP (True Positive). Files (APKs) analysed by the tool as infected or malicious that are malware.

Based on the above reasoning, Accuracy (ACC) was chosen as the main metric to be used in this experiment, since it considers the TP, FN, and FP, and as a secondary metric Recall (RE), since it allows following the proportion of correctly classified TP, False Negative Rate (FNR) for the proportion of FN, or failure rate, and finally, False Positive Rate (FPR), for the proportion of goodware APKs that are erroneously classified as positive.

- Accuracy (ACC): The ratio of correctly identified APKs, divided by the total number of files analysed. This metric will allow us to evaluate the total number of correct predictions over the total amount of test cases.

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

- Recall (RE): Also known as True Positive Rate, it determines the quality of the capacity detection and shows the proportion of infected APKs. In terms of our research, this indicator will determine the ability of the tool to predict malware within the group of infected APKs. It measures the proportion of True Positives that are correctly identified.

$$\mathrm{RE} = \frac{\mathrm{TP}}{\mathrm{FN} + \mathrm{TP}}$$

- False Negative Rate (FNR): Also known as failure rate, it indicates the proportion of all malicious APKs incorrectly classified as negative or trusted.

$$FNR = \frac{FN}{FN + TP}$$

- False Positive Rate (FPR). Represents the quantity of goodware APKs that are incorrectly classified as positive, i.e., malicious. It is also called "fall-out".

$$FPR = \frac{FP}{FP + TN}$$

Some of the online analysis tools do not perform a binary classification but add some more cases, such as "suspicious" or "unknown". In this case, an aggregation of the mentioned classification to one of the two main cases of APK, "clean or trusted" and "malicious", will be performed.

3.6. Functional Analysis

The functional analysis of the tools is based on the work carried out in the reference article [36] in which a comparison of several malware analysis tools available on the Internet is performed. Some tools that are no longer available for APK analysis, such as AVC Android, NVISO, and VirSCAN, have been discarded, and other new tools are added in the comparison, including AndroPytool, MobSf, Jose Sandbox, Metadefender, Jotti, and Pithus. In addition, other comparison parameters are added, such as type of application, limitations, options, advantages, and disadvantages. The result is shown in Tables 2 and 3.

Table 2. Malware tool functional analysis 1. Adapted from [36].

Parameter	AndroPytool	Joe Sandbox	Metadefender	Intezer Analyzer
Type of application	Desktop	Online	Online	Online
No of Scanners	Not applicable	Not applicable	35	Not applicable
Inputs	Inputs	Files, APK	Files (PE, ELF, IOS, document), URL, hash and APK,	Files (PE, ELF, document), URL, domain, hash, and APK
Limits	Limits	None	Upload size limit, Limit on the number of 5 scans per day and 25 per month and per account	Files up to 140 MB. The sandbox does not support APKs.
Scanning Time	5–10 min	5–15 min	1–2 min	4 min
Bulk File Scanning	Yes	Yes (RESTful API) but only in the paid version	Yes	Yes (RESTful API) and integrates with different SIEM and malware analysis tools.
Analysis Technique	Static, dynamic, and hybrid analysis, machine learning	Dynamic or Behavioural approach	Static Multiscan antivirus engine, APK metadata analysis, and dynamic or behavioural analysis but it does not work for APKs.	Not a "true" sandbox, Dynamic Execution, Static and Unpacking
Output	A JSON file and CSV format. It also allows direct export of all data collected from the sample set to the Mongo database.	Static File Info (type, entr0py, SHA256, SHA1, MD5, File Size name, etc.), permission, network traffic and behaviour (URL, domains, etc.), AV Detections, IOC, MITRE ATT&CK matrix, etc.	Static File Info (SHA256, SHA1, MD5, File Size, etc.), AV Detections, network traffic, filesystem activity, mutex, MITRE ATT&CK matrix, etc.	Static File Info (Size, SHA256, MD5, etc.) Company, File Type, Ssdeep result, Virus Total detection, Report, Timestamp, strings, MITRE ATT&CK, IOC, Behaviour
Summary Report	Yes (CSV format)	Yes	Yes	Yes

Table 3. Malware tool functional analysis 2. Adapted from [36].

Parameter	MobSf	VirusTotal	Jotti	Pithus	Hybrid Analysis
Type of application	Desktop	Online	Online	Online	Online
No of Scanners	Not applicable	59	13	Not applicable	59
Inputs	Files, APK	Files, APK, URL, IP Address, Domain, hash File	Files, APK, and hash File	APK, hash, fuzzy hash, domain,	Files, APK, URL, IP Address,
Limits	Not applicable	Not mentioned	5 files up to a limit of 250 MB and 25 MB for each one.	The number of returned results is limited to 50	Not mentioned
Scanning Time	1–5 min	5 min	<1 min	<1 min	3–5 min
Bulk File Scanning	No	Yes	No	No	Yes
Analysis Technique	Static, Dynamic	Static Multiscan antivirus engine, Signature, and Heuristic-based, Static, Dynamic	Static Multiscan antivirus engine, Signature and Heuristic	Fingerprint, control flow analysis, threat intelligence, code analysis, behaviour analysis, and network analysis	Static Multiscan antivirus engine, Hybrid analysis
Output	File (size, sha, etc.), APK information and components, certificate, APK permissions, browsable activities, manifest analysis, code analysis, domain malware check, URL, and trackers	SHA1, MD5, File type, file size, certificate details, Permissions, Activities, Services, Receivers, Providers, Intent Filters, Bundle Information, file system mechanism and process actions, service actions, Synchronisation signals, Modules loaded, Highlighted actions and Dataset Actions	SHA1, MD5, File type, Found Malware	SHA1, MD5, file size, APKid, certificate details AV detection, Code analysis, Permissions, activities, malware family, domains, URL	SHA1, MD5, file size, Risk Assessment, certificate details, IOC, MITRE ATT&CK, Suspicious Indicators, General
Summary Report	Yes	Yes	Yes	Yes	Yes

As a conclusion of the functional analysis carried out, it is indicated that the most important features that a malware analysis tool for Android operating systems should have would be the ability to perform bulk file scanning to perform automatic scans of multiple applications, adequate processing times between 1 and 5 min, different analysis techniques that include machine learning to improve the results, and the possibility of providing several output formats, such as JSON, CSV, etc.

3.7. Running the Experiments

3.7.1. Experiment 1 with AndroPytool

To study IA techniques such as ML in tasks of analysis, classification, and detection of possible files (APKs) infected with malicious code, an experiment has been carried out with the AndroPyTool against the two datasets constructed: one based on 7002 APKs, 1602 of them goodware and 5400 malware, and the other one based on 106 APKs, 53 of them goodware and 53 malware. Many APKs are included in this experiment due to the tool's ability to perform bulk scans via scripting. The execution of the experiment is shown below.

The first step is to run the tool against the dataset described above, using the following command:

```
$ docker run --volume=</PATH/TO/FOLDER/WITH/APKS/>:/apks
alexmyg/andropytool -s /apks/ <ARGUMENTS> --single --filter -vt
(VirustTotal API Key) -cl -csv EXPORTCSV -colour -all
```

where "volume=" is equal to the path where the APK files to be analysed are stored. The argument chosen for this analysis is the one covered by the "-all" parameter, which makes use of all the analyses.

According to the analysis phases of this tool, explained in Section 3.2.1, it first filters the files found for further analysis, depending on whether or not they are considered valid. To do so, it renames the folder containing the files to be analysed and creates two other folders to filter between infected and benign applications (malware and benignware). The next step is to analyse the applications with the available VirusTotal reports (Figure 5).

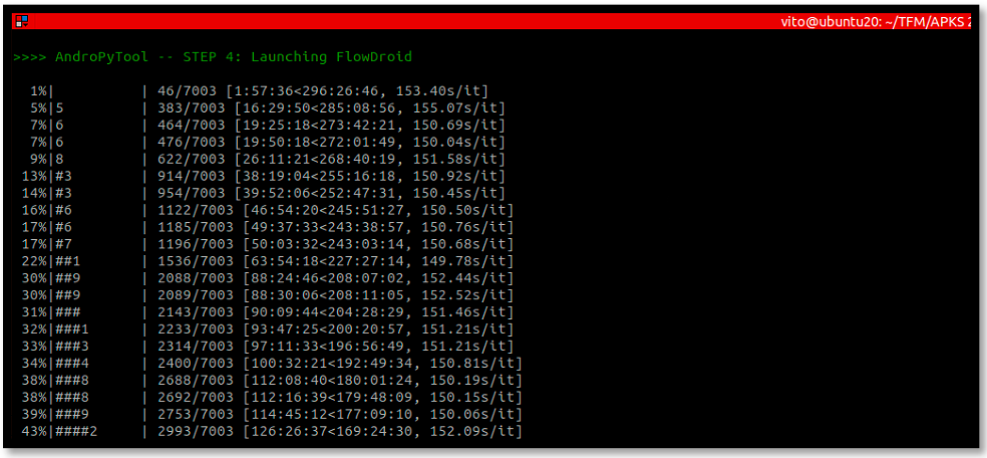


Figure 5. Comparison with Virus Total database.

The third step is an internal classification to discriminate malicious APKs from benign ones. After this classification, the program runs the built-in FlowDroid tool, explained in Section 3.2.1 as the compendium of tools offered by this hybrid analysis system (Figure 6).

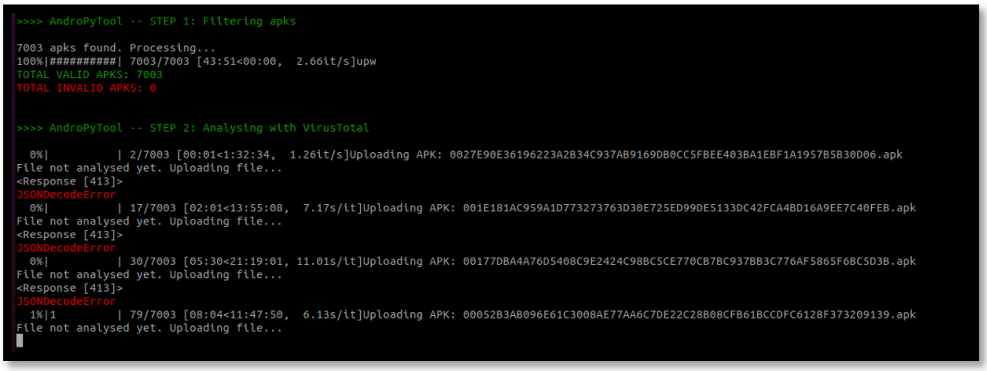


Figure 6. AndroPytool tool progress.

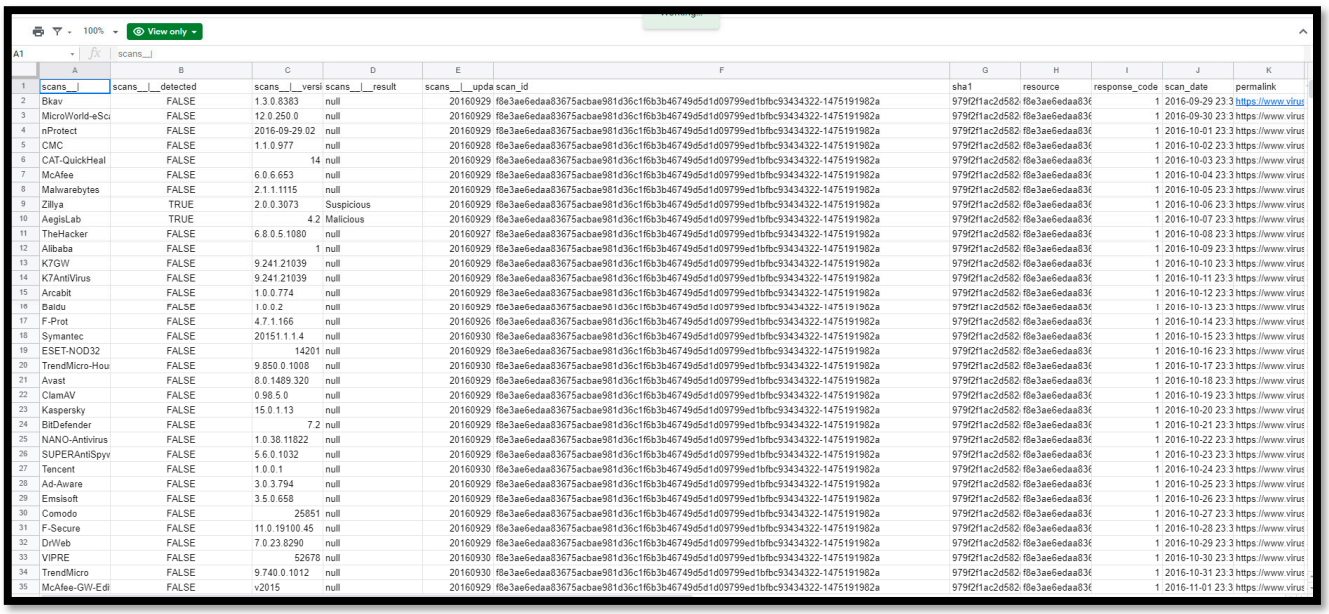
In this phase, the tool installs the application to see its live operation, using an internal sandbox, and discover its dynamic behaviour. This whole process can take several days. The most time-consuming step in this experiment was the analysis with FloidDroid (Figure 7). If the time required to download the dataset is included, it has been about two months, 24/7, to extract all the necessary information for its study.

Figure 7. Development of internal DroidBox analysis.

The data received need external analysis with the help of tools that facilitate their visualisation, for which the data have been exported to a CSV file and subsequently transformed into the data that will be displayed through Google DataStudio.

Once all the data provided by this tool have been obtained, the JSON files have to be formatted for further analysis. The 7000 JSON files (Figure 8) have been converted to Excel format and given a more intuitive format from a user's point of view, for later handling (Figure 9).

Figure 8. JSON file structure.



scans__|	scans__|__detected	scans__|__versi	scans__|__result	scans__|__upda	scan_id	sha1	resource	response_code	scan_date	permalink
Bkav	FALSE	1.3.0.8383	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-09-29 23:3	https://www.virus
MicroWorld-eSc	FALSE	12.0.250.0	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-09-30 23:3	https://www.virus
nProtect	FALSE	2016-09-29.02	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-01 23:3	https://www.virus
CMC	FALSE	1.1.0.977	null	20160928	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-02 23:3	https://www.virus
CAT-QuickHeal	FALSE	14	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-03 23:3	https://www.virus
McAfee	FALSE	6.0.6.653	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-04 23:3	https://www.virus
Malwarebytes	FALSE	2.1.1.1115	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-05 23:3	https://www.virus
Zillya	TRUE	2.0.0.3073	Suspicious	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-06 23:3	https://www.virus
AegisLab	TRUE	4.2	Malicious	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-07 23:3	https://www.virus
TheHacker	FALSE	6.8.0.5.1080	null	20160927	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-08 23:3	https://www.virus
Alibaba	FALSE	1	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-09 23:3	https://www.virus
K7GW	FALSE	9.241.21039	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-10 23:3	https://www.virus
K7AntiVirus	FALSE	9.241.21039	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-11 23:3	https://www.virus
Arcabit	FALSE	1.0.0.774	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-12 23:3	https://www.virus
Baidu	FALSE	1.0.0.2	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-13 23:3	https://www.virus
F-Prot	FALSE	4.7.1.166	null	20160926	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-14 23:3	https://www.virus
Symantec	FALSE	20151.1.1.4	null	20160930	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-15 23:3	https://www.virus
ESET-NOD32	FALSE	14201	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-16 23:3	https://www.virus
TrendMicro-Hou	FALSE	9.850.0.1008	null	20160930	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-17 23:3	https://www.virus
Avast	FALSE	8.0.1489.320	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-18 23:3	https://www.virus
ClamAV	FALSE	0.98.5.0	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-19 23:3	https://www.virus
Kaspersky	FALSE	15.0.1.13	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-20 23:3	https://www.virus
BitDefender	FALSE	7.2	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-21 23:3	https://www.virus
NANO-Antivirus	FALSE	1.0.38.11822	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-22 23:3	https://www.virus
SUPERAntiSpyv	FALSE	5.6.0.1032	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-23 23:3	https://www.virus
Tencent	FALSE	1.0.0.1	null	20160930	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-24 23:3	https://www.virus
Ad-Aware	FALSE	3.0.3.794	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-25 23:3	https://www.virus
Emsisoft	FALSE	3.5.0.658	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-26 23:3	https://www.virus
Comodo	FALSE	25851	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-27 23:3	https://www.virus
F-Secure	FALSE	11.0.19100.45	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-28 23:3	https://www.virus
DrWeb	FALSE	7.0.23.8290	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-29 23:3	https://www.virus
VIPRE	FALSE	52678	null	20160930	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-30 23:3	https://www.virus
TrendMicro	FALSE	9.740.0.1012	null	20160930	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-10-31 23:3	https://www.virus
McAfee-GW-Edi	FALSE	v2015	null	20160929	f0e3ae6edaa83675acbae981d36c1f6b3b46749d5d1d09799ed1bfbc93434322-1475191982a	979f2f1ac2d582	f0e3ae6edaa83€	1	2016-11-01 23:3	https://www.virus

Figure 9. Conversion of JSON file into CSV to be treated.

After dissecting each JSON file individually, the following scale is used to determine which files are infected, suspicious, goodware, or, conversely, if they have not been analysed because they are categorised as unknown or in unknown status. Several examples are shown below:

```
 verbose_msg'': ''Scan finished, information embedded'', ''total'': 61,
''positives'': 0, ''sha256'':
''001f91177291bb5fe2b23d43674c85b76f56de677da13ab9a73eb996662e705b'',
''md5'': ''5cb3d50e80f74a526d9d59de7db26113''
 verbose_msg'': ''Scan finished, information embedded'', ''total'': 64,
''positives'': 23, ''sha256'':
''000a69d61dc389579b9b931c3c04bbe287b37e471f1c97c4326143665f34c3a6'',
''md5'': ''5a322ac4862e8521ae844dd95327c705''}
```

After filtering all the data and offering only those that are conclusive for the present work, with the Google DataStudio tool, the results are shown graphically. This representation has two different views, a more general one, as can be seen in Figure 10, and a more generic one, as can be seen in Figure 11. It is also worth noting that it can be filtered by different fields, thus offering an interactive way of visualising the data.

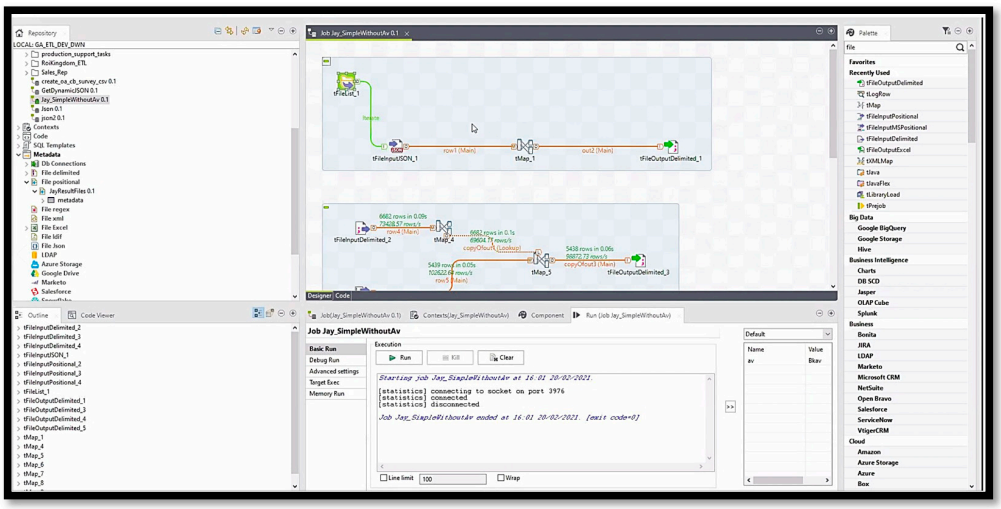


Figure 10. Filtering of JSON files for conversion to CSV to be processed.

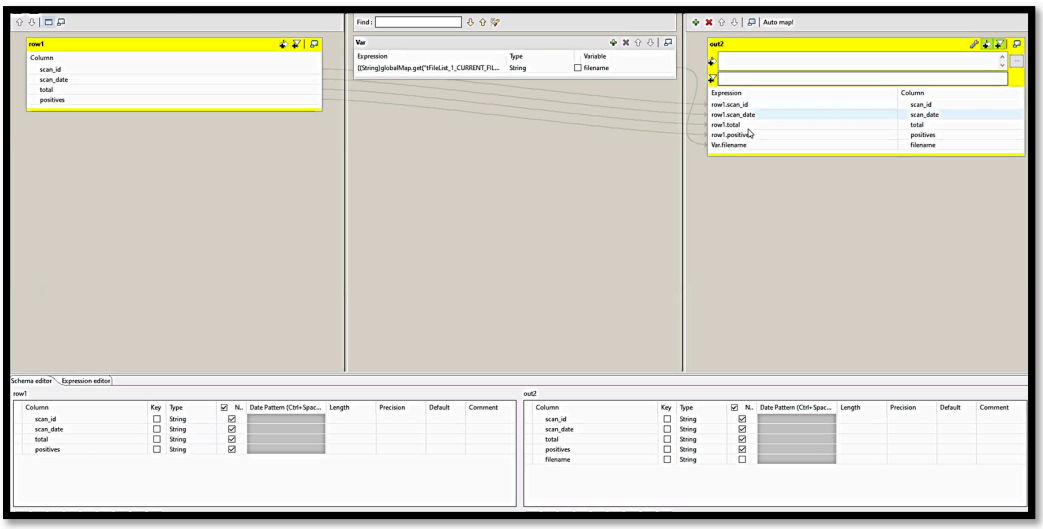


Figure 11. Data visualisation dataset APKs.

Figure 10 shows the process performed for the transformation of the JSON files, obtained from the analysis performed by AndroPytool, to provide a classification between goodware, suspicious, and malware.

Figures 11 and 12, respectively, show the data in graphical format, where it can be filtered by various fields, such as scan_id, classification, or status of the analysed files. In addition, the result of the analysis can be compared with the source dataset imported as "is_Benign". To address this, pre-processing has been necessary, where it has been filtered by fields determined for this output.

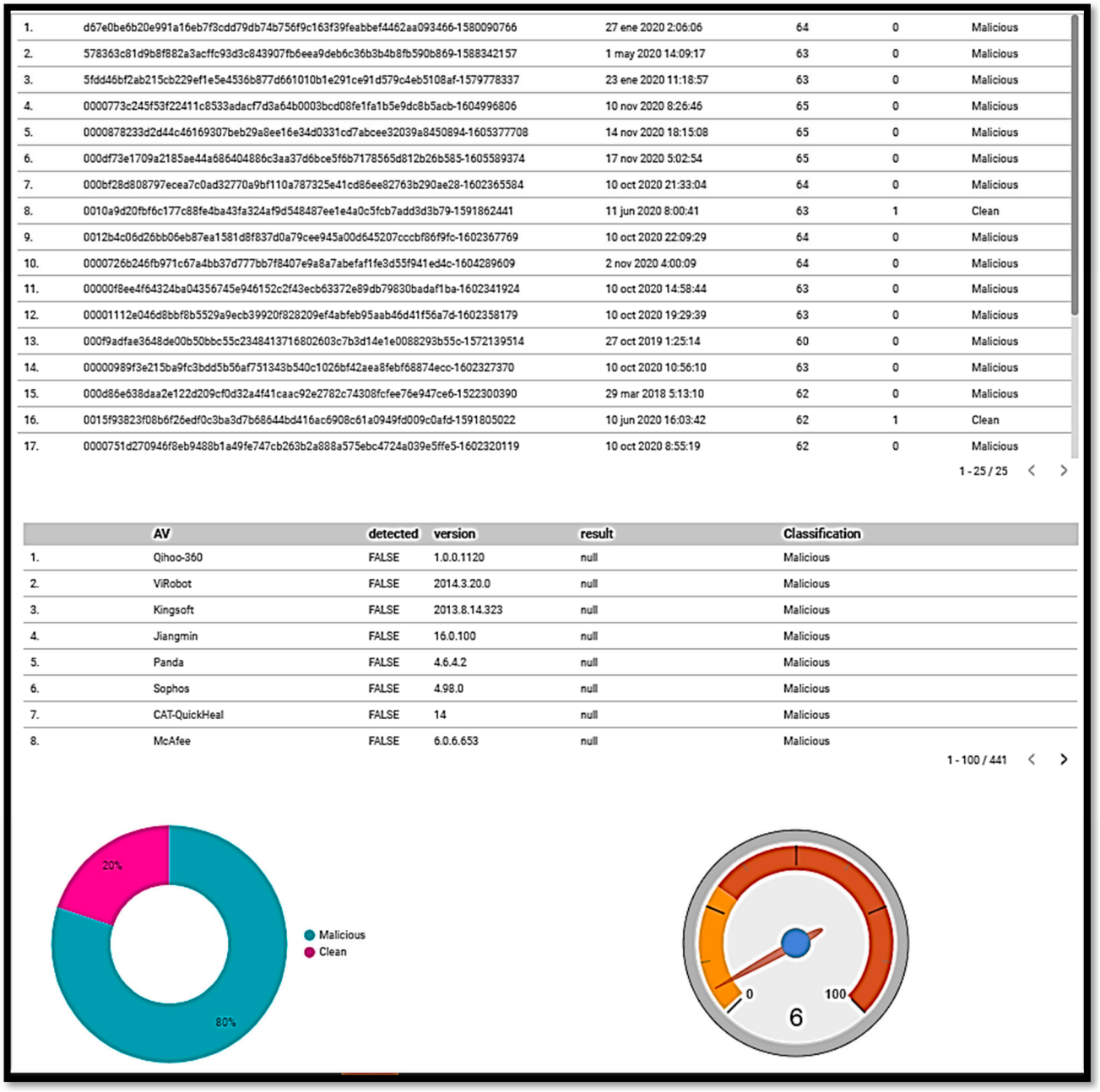


Figure 12. Data visualisation dataset APKs.

The following Figure 13 shows the process and transformation of the data before obtaining the data required for this work. As seen, the filtering and classification of the data, together with the understanding of the classification analysed by the tool itself, represent a substantial contribution.

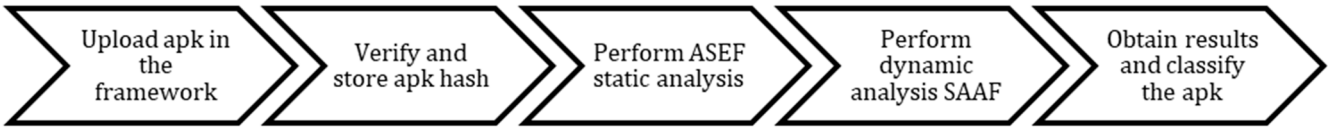


Figure 13. Methodology.

We present a sample catalogue that is faithful to the data captured. The results are available in the Google DataStudio Dashboard at the URL (https://datastudio.google.com/u/0/reporting/c45b67e1-8797-4f05-a10c-6aff59db6827/page/2531B, accessed on 18 February 2024).

Later, the same analysis process is performed against the 106 APKs dataset. The experiment against the first dataset provides a more precise measurement of the selected metrics, while the second one provides a set of measurements for comparison against other tools. Finally, the calculation of the metrics defined in Section 3.5 is carried out and the results are presented in Table 4.

Table 4. Results obtained with the AndroPytool.

Tool	Nº APK	TN	FP	FN	TP	FPR	FNR	RE	ACC
AndroPytool	7002	1507	95	4	5396	0.059	0.001	0.999	0.986
	106	51	2	1	52	0.038	0.019	0.981	0.972

3.7.2. Experiment 2 with the MobSF Tool

As stated in Section 3.4, an analysis of the MobSF tool with a dataset of 53 benign and 53 malicious applications is performed in this experiment. The reason for using a dataset with a smaller number of APKs than the one used in experiment 1 is that in this tool, certain manual operations must be performed in the dynamic analysis and the file loading.

A specific methodology has been designed for conducting this experiment to cover all possible surface attacks on a mobile device with an Android operating system running. Figure 14 shows the process diagram of the proposed methodology.

```
public Location getLocation() {
    if (this.i == null) {
        return null;
    }
    Context context = (Context)this.i.get();
    if (context == null) {
        return null;
    }
    if (context.checkCallingOrSelfPermission("android.permission.ACCESS_FINE_LOCATION") == 0) {
        return ((LocationManager)context.getSystemService("location")).getLastKnownLocation("gps");
    }
    if (context.checkCallingOrSelfPermission("android.permission.ACCESS_COARSE_LOCATION") == 0) {
        return ((LocationManager)context.getSystemService("location")).getLastKnownLocation("network");
    }
    return null;
}
```

Figure 14. Access to DroidKungFu app location permissions.

As shown in Figure 13, the first task is to load the APK under analysis to the Mobile-Security-Framework (MobSf) tool that has been installed in an isolated environment.

As a second step, the hash of the application under analysis must be compared against a database that has, as a record, other analysed and tested applications just to classify it and define it as malicious or not. If the hash of the application exists in the application database, it will provide us with information from other previous analyses to determine and classify the malware without having to perform another analysis after this step.

If the hash of the application does not exist in the database, a static analysis (ASEF) can be performed, using the MobSf tool to obtain the application's permissions from the global manifest configuration file, or another Android tool called Apktool, which extracts the manifest.xml file from any application. From this file, it is possible to see the permissions that the application has and categorise the risk of each permission, as many permissions can access sensitive information that should not be accessible. This step will give an insight into the possible exploitation points of the application.

After performing the static analysis, a dynamic or behavioural analysis (SAAF) is performed where the MobSf tool will run the application in an Android virtual machine or on a device configured with the tool to detect runtime problems. Within this type of analysis, captured network packet logs will be analysed by decrypting HTTPS traffic, log reports, error logs, debugging information, and memory stack tracing.

After these analyses, the information obtained will classify the malicious application, and the hash of the application will be stored in the MobSf tool database so that in a subsequent analysis, this application can be confirmed as malicious at the beginning of its analysis.

The following is an example of the analysis of one application infected with malware, to virtually show the proposed methodology.

DroidKungFu. This application was the first Android malware to bypass antivirus software and take control of the phone by creating a backdoor. This malware is considered an evolution of DroidDream, the first large-scale Android virus, with the difference that DroidKungfu can avoid detection by security or antivirus software.

- Size: 0.21 MB
- MD5: 5e26403a5f7ec59479fb1009d875bcdb
- SHA1: 5e2fb0bef9048f56e461c746b6a644762f0b0b54
- SHA256: 7d382faafcaad6f8bf5da383cb8703b7094a045aeac5e13b5f4225c6272a615
- 13 ACTIVITIES, 0 SERVICES, 0 RECEIVERS, 0 PROVIDERS.

The first step of the methodology is to collect and perform a search of the hashes of the application, to classify the sample in case previous research already exists. Although this application is known, in this case, the application is analysed with MobSf. From the manifest file, it was possible to retrieve the following permissions used by this application:

- android.permission.GET_TASKS (Dangerous): This may allow malicious applications to discover private information about other applications.
- android.permission.INTERNET (Dangerous): This allows an application to create network sockets.
- android.permission.WRITE_EXTERNAL_STORAGE (Dangerous): Read/modify/delete content from the SD card.
- android.permission.ACCESS_NETWORK_STATE (Normal): Allow an application to see the status of all networks.
- android.permission.READ_PHONE_STATE (Dangerous): Read phone status and identity.

When analysing the application code, the MobSf framework shows us as a summary that the application has many classes with unsafe random codes, to make recursive calls to instances of the entire application. It also shows us that the application has a method to obtain the location of the device by GPS and network. Then, it makes an HTTP connection to the following URL: http://app.waps.cn/action/account/offerlist (accessed on 18 February 2024) to send information about the device and its location, as shown in Figure 14.

This application interacts directly with the user, making use of the activities made in the analysis since it does not have any service running in the background.

In the next phase, by performing a behavioural analysis of the sample, the MobSf framework allows running an automatic interactive analysis to obtain relevant information, in which information is sent from the device to the URL described in the previous phase and as a result, commands are brought to perform certain actions, as shown in Figure 15.

Figure 15. Response when sending device data from the DroidKungfu application.

When performing the static analysis of the permissions found in the application samples of the dataset, it was found that infected applications access more permissions than healthy ones. Malicious applications access over 30 different permissions of eight types, while benign applications access around 16 permissions. Finally, the calculation of the metrics defined in Section 3.5 is carried out and the results are presented in Table 5.

Table 5. Results obtained with MobSf.

Tool	Nº APK	TN	FP	FN	TP	FPR	FNR	RE	ACC
MobSf	106	50	3	1	52	0.056	0.018	0.981	0.962

3.7.3. Experiment 3 with Online Tools

Finally, a study of different online analysis tools for APKs of Android systems has been carried out. The dataset of 53 benign and 53 malicious APKs has been used for this experiment, since manual operations are also required to load the file under analysis. Specifically, the following online tools have been analysed: VirusTotal, Hybrid Analysis, Joe Sandbox, Intezer Analyzer, Metadefender, Jotti, and Phitus. The experiment theoretically involves loading the 106 APKs from the dataset and classifying them into four categories: TP, FP, TN, or FN. The results are shown in Table 6.

Table 6. Results obtained with the online tools.

Tool	Nº APK	TN	FP	FN	TP	FPR	FNR	RE	ACC
VirusTotal	106	48	5	1	52	0.094	0.019	0.981	0.943
Inteze Analize	106	48	5	20	33	0.094	0.377	0.623	0.764
Hybrid Analysis	106	19	34	3	50	0.642	0.057	0.943	0.651
Joe Sandbox	106	27	26	6	47	0.491	0.113	0.887	0.698
Metadefender	106	53	0	18	35	0.000	0.340	0.660	0.830
Jotti	106	52	1	15	38	0.019	0.283	0.717	0.849
Pithus	106	47	6	2	51	0.113	0.038	0.962	0.925

As outlined in Section 3.5, some of the online analysis tools (Intezer Analize, Hybrid Analysis, and Jose SandBox) do not perform binary classification but add some more cases, such as “suspicious” or “unknown”; to have a binary classification, the following aggregations have been performed:

- Intezer Analyze: Made up according to the classification made by the tool in the group of malware APKs: seven trusted, thirteen unknown, thirty-one malicious,

and two suspicious, and in the group of goodware APKs: seventeen trusted, thirty-one unknown, zero malicious, and five suspicious. Those classified as trusted and unknown and malicious and suspicious are added, thus obtaining 32 TP, 20 FN, 48 TN, and 5 FP.

- Hybrid Analysis: Taken according to the classification made by the tool in the group of malware APKs: three not specific threats, forty-two malicious, and seven suspicious, and in the group of goodware APKs: nineteen not specific threats, six malicious, and twenty-eight suspicious. Those classified as malicious and suspicious are added, thus obtaining 49 TP, 3 FN, 19 TN, and 34 FP.
- Jose SandBox: According to the classification made by the tool in the group of malware APKs, six were clean, five suspicious, and thirty-two were malicious, and in the group of goodware APKs, seventeen were clean, twelve malicious, and fourteen were suspicious. Those classified as malicious and suspicious total 47 TP, 6 FN, 27 TN, and 26 FP.

It has also been considered in all tools using the Static Multiscan antivirus engine (Virustotal, Jotti, and Pithus) technique that it only shows one positive detection by one of the engines in the group's case of malware APKs, such as TP and FN in the case of goodware APKs.

With the online tool, one of its weaknesses is the daily and monthly limit, along with the need to create a paid account, so that the results of the analysis are not made public. To carry out a more extensive analysis with more capabilities and eliminate any type of limitation, apart from the financial outlay, the tool administrator must approve that the intentions are legitimate, to give access to all the tools and reports available in the tool. This factor is a process that can be delayed in time, because of poor maintenance and non-existent communication with the support staff.

3.8. Discussion and Lessons Learned

The following table shows the results of all the experiments carried out.

An important aspect to consider when analysing the results of all the tools shown in Table 7 is that the results gathered with the AndroyPytool tool are more accurate than those obtained with the other tools, since the dataset used is much larger: 7002 versus 106 APKs. As already explained in previous sections, this was possible because of the automatic bulk loading of the APKs to be analysed. However, to be able to make a comparison against the other tools under investigation, the tool has also been run against the 106 APK dataset.

Table 7. Total results sorted by tool accuracy.

Tool	N° APK	TN	FP	FN	TP	FPR	FNR	RE	ACC
AndroPytool	7002	1507	95	4	5396	0.059	0.001	0.999	0.986
AndroPytool	106	51	2	1	52	0.038	0.019	0.981	0.972
MobSf	106	50	3	1	52	0.057	0.019	0.981	0.962
VirusTotal	106	48	5	1	52	0.094	0.019	0.981	0.943
Pithus	106	47	6	2	51	0.113	0.038	0.962	0.925
Jotti	106	52	1	15	38	0.019	0.283	0.717	0.849
Metadefender	106	53	0	18	35	0.000	0.340	0.660	0.830
Inteze Analyzer	106	48	5	20	33	0.094	0.377	0.623	0.764
Joe Sandbox	106	27	26	6	47	0.491	0.113	0.887	0.698
Hybrid Analysis	106	19	34	3	50	0.642	0.057	0.943	0.651

The results shown in Tables 7 and 8 and Figures 16 and 17 show that the AndroPytool tool obtained the best performance in all the metrics. The value of these was as follows.

Table 8. Total results are sorted by the detection capability of the tool.

	N° APK	TN	FP	FN	TP	FPR	FNR	RE	ACC
AndroPytool	7002	1507	95	4	5396	0.059	0.001	0.999	0.986
AndroPytool	106	51	2	1	52	0.038	0.019	0.981	0.972
MobSf	106	50	3	1	52	0.057	0.019	0.981	0.962
VirusTotal	106	48	5	1	52	0.094	0.019	0.981	0.943
Pithus	106	47	6	2	51	0.113	0.038	0.962	0.925
Hybrid Analysis	106	19	34	3	50	0.642	0.057	0.943	0.651
Joe Sandbox	106	27	26	6	47	0.491	0.113	0.887	0.698
Jotti	106	52	1	15	38	0.019	0.283	0.717	0.849
Metadefender	106	53	0	18	35	0.000	0.340	0.660	0.830
Inteze Analyzer	106	48	5	20	33	0.094	0.377	0.623	0.764

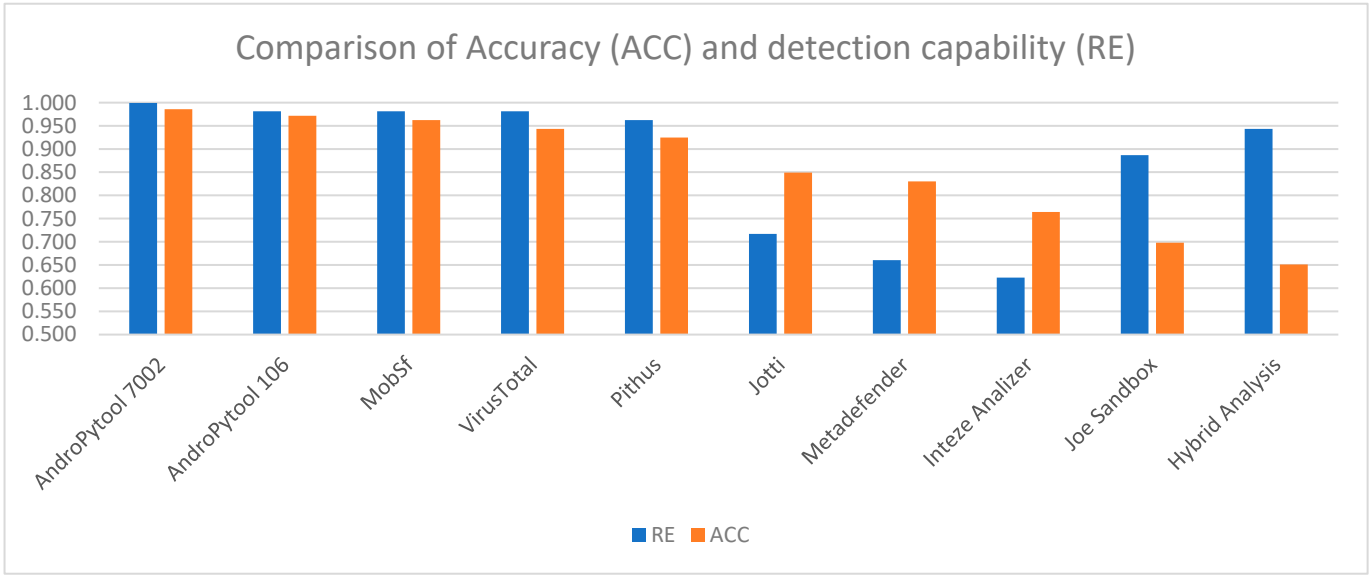


Figure 16. Graph showing accuracy (ACC) and detection capability (RE).

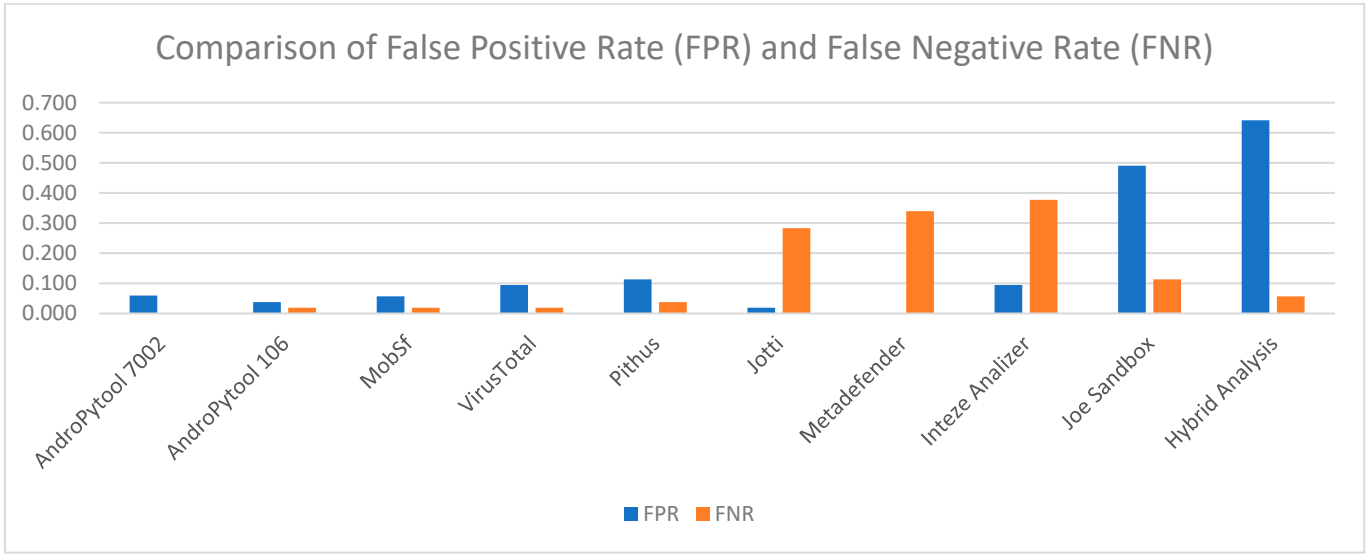


Figure 17. Comparative graph of False Positive Rate (FPR) and False Negative Rate (FNR).

- Total successful predictions over the total number of test cases with an accuracy of 0.986 with 7002 APKs dataset and 0.972 with 106 APKs dataset. The best accuracy of all tools.

- The tool shows a detection capability or recall of 0.999 cases on the 7002 APK dataset, so 99.9% of the time it will hit the positive cases of infected APKs, within the group of infected APKs. With the 106 APK dataset, it shows 0.981.
- A value of 0.001 in the false negative rate with the 7002 APK dataset and 0.019 with the 106 APK dataset. The tool has almost no false negatives. It means that the tool has detected almost all the malware APKs.
- The value of the fall-out (false positive rate) is 0.059 with the 7002 APK dataset and 0.038 with the 106 APK dataset, proving to be the best tool being behind the online analysis tools Metadefender and Jotti.

Although the AndroPytool tool presented the best results, it is worth mentioning the results of the MobSF tool, which has the second-best prediction and the same detection capability (Recall) as the mentioned tool. Furthermore, in some circumstances, the tool cannot replace the analysis and observations to be performed by malware analysts. In this sense, MobSF is a tool that presents a complete framework for analysts to perform APK analysis manually.

Table 9 shows a comparison of the results obtained in this research with the AndroPytool concerning other tools of similar works obtained from the review of the state of the art. The metric included for comparison is the one that all have used in common, namely Accuracy.

Table 9. Comparison of the results obtained with the AndroPytool concerning other state-of-the-art works.

Artículo	Tool	Accuracy
[26]	Androguard	0.9304
[26]	DroidMat	0.9787
[27]	DroidMiner (Random Forest)	0.953
[31]	MOCDroid	0.951
[38]	DroidDetector	0.946
[38]	HEMD	0.899
This article	AndroPytool	0.986
This article	MobSf	0.962

In conclusion, it can be seen that AdroPytool is the tool with the best results, followed by Droidmat.

Concerning online tools, it should be noted that the results of the metrics for this type of tool are not very good, leading sometimes to confusion with states such as unknown or suspicious. Virustotal and Pithus obtained the best results in terms of accuracy and detection capability (Virustotal with an ACC = 0.943 and RE = 0.981 and Pithus with an ACC = 0.925 y RE = 0.962). Virustotal has the same detection capacity as the two best tools. In the specific case of Pithus, the tool is a recent creation (beta version), so it is considered that the margin for improvement of this tool is wide, so it can be assumed that in the future, its results will be better.

As stated in the previous paragraph, Metadefender and Jotti have the lowest fall-out or false positive rates. The tools that use techniques such as hybrid and behavioural or dynamic analysis (Joe Sandbox, Hybrid Analysis, and Intezer analyzer) generate many false positives within the group of goodware APKs so they do not obtain good results, unlike tools that perform mainly Static Multiscan Antivirus analysis, if they obtain them. On the other hand, Jotti, Metadefender, and Inteze Analyzer have the worst false negative rate, which means they are the worst at detecting and classifying malware APKs.

Regarding the hypothesis established for this research, "Are tools that use existing machine learning techniques more effective than tools that do not use Artificial Intelligence engines?" it can be affirmed that the tool that uses machine learning techniques, AndroPytool, is more effective than the others analysed that do not use artificial intelligence techniques.

Finally, the main limitation of the research was that several of the tools available online could not perform bulk scanning of files. As a consequence, three types of experiments (experiment 1: Analysis of the AndroPyTool application, experiment 2: Analysis of the MobSf application and experiment 3: Analysis of the online applications) had to be carried out with different smaller datasets in experiments 2 and 3, when the ideal would have been to carry it out with only one, experiment 1, which had 7200 APKs. Another concern has been the amount of time it has taken to perform scans on applications that did not allow bulk uploading of files.

4. Conclusions

Proper malware detection is a very important aspect of today's mobile technology. With the increase in malware daily, there is also a need for a suitable malware detection scheme with a robust malware detection scanning tool. Based on the limitations of existing Android malware scanning tools, it can be concluded that most of the tools only perform static malware scanning.

Most of the tools only provide file upload as input and support small file upload to perform malware scanning. The tools also do not support bulk scanning of files. The time taken by the tools to scan a single file is also high.

There is a need for a robust malware scanning tool that overcomes all the limitations of existing scanning tools, performs hybrid scanning, and can be deployed as a service. In addition, the results of existing scanning tools can be combined and provide a more detailed and appropriate summary report.

Throughout this paper, the problems associated with Android malware classification and detection have been shown and developed from several viewpoints, all of them within the framework of using ML techniques.

Because of the present research, it has been demonstrated how the use of ML tools, such as AndroPytool, improves the detection and classification of malicious APKs compared to those obtained using other types of tools already discussed during this work. Furthermore, as shown in Table 9, it is the tool that obtains the best results compared to others analysed in various studies in the literature.

The detection and classification of malicious APKs (malware or malicious software) is an important technique that allows the assignment of a given specimen of malware to its corresponding family. This allows improved tracking of the several current families, improved detection of zero-day specimens, and detection of different variations of known malware families. These techniques are fundamental in the fight against malicious software, given the different types of economic damage and data confidentiality that can occur if a device is successfully infected. Knowing the malware family, therefore, facilitates the adoption of practical measures to prevent its spread and minimise its impact.

Although the results achieved have revealed a high-level rate of accuracy and detection capability, it is considered that there is still some room for improvement, so it is necessary to analyse new functionalities, representations, learning algorithms, and data processing techniques. Further research of other features should be carried out, building with more files, obtaining data from compiled libraries, and in the presence of functions that load dynamic code. Other research that can be carried out is the improvement of existing malware classifiers.

Finally, it is also worth highlighting the functionalities and capabilities of the MobSf framework, since it allows the analyst to reduce detection time by having multiple tools in one, which would otherwise require separate work: decoding, debugging, code review, and penetration testing. Therefore, the framework will also allow the automation of repetitive tasks.

5. Future Work

From the results scored in this research, it is considered that the application and integration in AndroPytool of new ML algorithms and tools would allow extracting a

broader set of features that could allow the improvement of the classification and detection capabilities of the tool. Other improvements to be highlighted include the following:

- Growing the number of both malicious and goodware APKs.
- Improving the data characterising the different malware families' classification.
- Performing a detailed study of the features to be extracted with the different analysis methods to improve the results received in the detection and classification of malware.
- Improve the processing of data gained.

Another area for improvement could be to take advantage of the JSON files obtained during the analysis to display an HTML-based graph of the results obtained. Another plausible improvement could be a real-time estimation that, according to the number of APKs as well as the size of them, gives an approximation of the estimated waiting time that the tool would need to finish the whole process.

Author Contributions: Conceptualization, J.B.H. and J.M.M.; Methodology, J.R.B.H., T.M.S.R. and J.A.S.M.; Software, J.M.M. and G.J.B.M.; Validation, J.R.B.H., T.M.S.R., J.A.S.M. and G.J.B.M.; Formal analysis, J.R.B.H., J.A.S.M. and J.B.H.; Investigation, J.M.M., J.B.H. and G.J.B.M.; Writing—original draft, J.M.M. and J.R.B.H.; Writing—review and editing, J.A.S.M., T.M.S.R. and J.R.B.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Correa, R.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Sicilia Montalvo, J.A.; Sanchez Rubio, M.; Magreñan, A. Hybrid Security Assessment Methodology for Web Applications in Computer Modeling. *Eng. Sci.* **2021**, *126*, 89–124.
2. McGraw, G.; Morrisett, G. Attacking malicious code: A report to the Infosec Research Council. *IEEE Softw.* **2000**, *5*, 33–41. [CrossRef]
3. Murphy, K. *Machine Learning: A Probabilistic Perspective*; MIT Press: Cambridge, MA, USA, 2012.
4. Android Security Team. Android Open Source Project. Application Security. Available online: https://source.android.com/security/overview/app-security (accessed on 18 February 2024).
5. International Data Corporation (IDC). Smartphone Market Share. Available online: https://www.idc.com/promo/smartphone-market-share/os (accessed on 18 February 2024).
6. Mateo Tudela, F.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Sicilia Montalvo, J.A.; Argyros, M.I. On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *Appl. Sci.* **2020**, *10*, 9119. [CrossRef]
7. Callahan, J. The History of Android: The Evolution of the Biggest Mobile OS in the World. Android Authority. Available online: https://www.androidauthority.com/history-android-os-name-789433/ (accessed on 18 February 2024).
8. Vicente Mohino, J.V.; Bermejo Higuera, J.; Bermejo Higuera, J.R.; Sicilia Montalvo, J.A.; Sanchez Rubio, M.; Martinez Herraiz, J.J. MMALE-A Methodology for Malware Analysis in Linux Environments. *CMC-Comput. Mater. Contin.* **2021**, *67*, 1447–1469.
9. Baker, K. Crowdstirke. Malware Analysis. Available online: https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/ (accessed on 18 February 2024).
10. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
11. Cuadrado, A.M. *Utilización del Machine Learning en la Industria 4.0*; Valladolid University: Valladolid, Spain, 2019.
12. Maslennikov, D. Karpesky Securelist. First SMS Trojan for Android. Available online: https://securelist.com/first-sms-trojan-for-android/29731/ (accessed on 18 February 2024).
13. Malwarebytes Labs. 2020 State of Malware Report. Available online: https://www.malwarebytes.com/resource/2020-state-of-malware-report (accessed on 18 February 2024).
14. Riadi, I. Implementation of Malware Analysis using Static and Dynamic Analysis Method. *Int. J. Comput. Appl.* **2015**, *975*, 8887.
15. GuardSquare. Available online: https://www.guardsquare.com/es/hardening-del-c%C3%B3digo-encriptaci%C3%B3n-y-ofuscaci%C3%B3n (accessed on 18 February 2024).
16. Bermejo Higuera, J.; Abad Aramburu, C.; Bermejo Higuera, J.R.; Sicilia Urban, M.A.; Sicilia Montalvo, J.A. Systematic Approach to Malware Analysis (SAMA). *Appl. Sci.* **2020**, *10*, 1360. [CrossRef]
17. Agarwal, D. Mobile App | File Extensions. Available online: https://testingmobileapps.wordpress.com/2016/03/01/mobile-app-file-extensions-apk-ipa-appx-xap/ (accessed on 18 February 2024).

18. Srinivas. Android Malware Analysis. 2016. Available online: https://www.infosecinstitute.com/resources/malware-analysis/android-malware-analysis-2/ (accessed on 18 February 2024).
19. Ka1d0. Android Malware Analysis-DroidDream. 2019. Available online: https://medium.com/@nikhilh20/android-malware-analysis-droiddream-d06fc0d87bd2 (accessed on 18 February 2024).
20. Chakkaravarthy, S.S.; Vaidehi, V.; Rajesh, P. Hybrid Analysis Technique to detect Advanced Persistent Threats. *Int. J. Intell. Inf. Technol. (IJIIT)* **2018**, *14*, 59–76. [CrossRef]
21. Aslan, Ö.; Samet, R. Investigation of Possibilities to Detect Malware Using Existing Tools. In Proceedings of the IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 30 October–3 November 2017; pp. 1277–1284.
22. Montes García, F.; Bermejo Higuera, J.; Sanchez Crespo, L.E.; Bermejo Higuera, J.R.; Sicilia Montalvo, J.A. Detecting Malware in Cyberphysical Systems Using Machine Learning: A Survey. *KSII Trans. Internet Inf. Syst.* **2021**, *15*, 1119–1139.
23. Tien, C.-W.; Liao, J.-W.; Chang, S.-C.; Kuo, S.-Y. Memory forensics using virtual machine introspection for Malware analysis. In Proceedings of the 2017 IEEE Conference on Dependable and Secure Computing, Taipei, Taiwan, 7–10 August 2017; pp. 518–519.
24. Gadgil, P.; Sangeeta, N. Analysis of Advanced Volatile Threats Using Memory Forensics. In Proceedings of the Conference on Technologies for Future Cities (CTFC), Navi Mumbai, India, 8–9 January 2019.
25. ForcePoint. What Is Malware. 2018. Available online: https://www.forcepoint.com/cyber-edu/malware#:~:text=Malware%20is%20the%20collective%20name (accessed on 18 February 2024).
26. Wu, D.J.; Mao, C.H.; Wei, T.E.; Lee, H.M.; Wu, K.P. DroidMat: Android Malware Detection through Manifest and API Calls Tracing. In Proceedings of the 2012 Seventh Asia Joint Conference on Information Security, Tokyo, Japan, 9–10 August 2012; pp. 62–69.
27. Yang, C.; Xu, Z.; Gu, G.; Yegneswaran, V.; Porras, P. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, 7–11 September 2014. Proceedings, Part I 19*; Springer International Publishing: Cham, Switzerland, 2014; pp. 163–182.
28. Aafer, Y.; Du, W.; Yin, H. Droidapiminer: Mining API-level features for robust malware detection in Android. In *Security and Privacy in Communication Networks: 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, 25–28 September 2013, Revised Selected Papers 9*; Springer International Publishing: Cham, Switzerland, 2013; pp. 86–103.
29. Huang, J.; Lu, J.; Ling, C.X. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. In Proceedings of the Third IEEE International Conference on Data Mining, Melbourne, FL, USA, 22–22 November 2003; pp. 553–556.
30. Zhang, M.; Duan, Y.; Yin, H.; Zhao, Z. Semantics-aware android malware classification using weighted contextual API dependency graphs. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 3–7 November 2014; pp. 1105–1116.
31. Rodriguez, J.J.; Kuncheva, L.I.; Alonso, C.J. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 1619–1630. [CrossRef] [PubMed]
32. Martín, A.; Menéndez, H.D.; Camacho, D. MOCDroid: Multi-objective evolutionary classifier for Android malware detection. *Soft Comput.* **2017**, *21*, 7405–7415. [CrossRef]
33. Martillo, G.J.B. *Análisis de Vulnerabilidades de Malware en Aplicaciones*; Android: Quito, Ecuador, 2017.
34. Xu, J.; Yu, Y.; Chen, Z.; Cao, B.; Dong, W.; Guo, Y.; Cao, J. MobSafe: Cloud computing based forensic analysis for massive mobile applications using data mining. *Tsinghua Sci. Technol.* **2013**, *18*, 418–427. [CrossRef]
35. Shabtai, A.; Kanonov, U.; Elovici, Y.; Glezer, C.; Weiss, Y. Andromaly: A behavioural malware detection framework for Android devices. *J. Intell. Inf. Syst.* **1012**, *38*, 161–190. [CrossRef]
36. Agrawal, P.; Bhushan, T. Analysis of Android malware scanning tools. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 807–810. [CrossRef]
37. Elhanashi, A.; Dini, P.; Saponara, S.; Zheng, Q. Integration of Deep Learning into the IoT: A Survey of Techniques and Challenges for Real-World Applications. *Electronics* **2023**, *12*, 4925. [CrossRef]
38. Ashawa, M.A.; Morris, S. Analysis of Android malware detection techniques: A systematic review. *Int. J. Cyber-Secur. Digit. Forensics* **2019**, *8*, 177–187. [CrossRef]
39. Team, H. Andro4lab. 2017. Available online: https://hydrasky.com/mobile-security/android-malware-analysis-tool-dynamic-analysis-tools/ (accessed on 18 February 2024).
40. alexmYg, AndroPyTool. 2020. Available online: https://github.com/alexMyG/AndroPyTool (accessed on 18 February 2024).
41. Martin Garcia, A. Machine Learning Techniques for Android Malware Detection and Classification. Ph.D. Thesis, Universidad Autónoma de Madrid, Madrid, Spain, 2019.
42. Arzt, S. FlowDroid Data Flow Analysis Tool, GitHub, Inc 2021. Available online: https://github.com/secure-software-engineering/FlowDroid (accessed on 18 February 2024).
43. Arkajit, D.; Kakelli, A.K.; Aju, D. An Emerging Malware Analysis Techniques and Tools: A Comparative Analysis. *Int. J. Eng. Res. Technol. (IJERT)* **2021**, *10*, 112–116.
44. Sandbox, J. Android Sandbox Cloud. Joe Security LLC. 2021. Available online: https://www.joesandbox.com/#windows (accessed on 18 February 2024).
45. Docker, What is a Container? A Standardized Unit of Software. Available online: https://www.docker.com/resources/what-container (accessed on 18 February 2024).

46. Allix, K.; Bissyandé, T.; Klein, J.; Le Traon, Y. Androzoo: Collecting millions of Android apps for the research community. In Proceedings of the 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), Austin, TX, USA, 14–15 May 2016; pp. 468–471.
47. Université du Luxembourg. AndrpZoo API Documentation. Available online: https://androzoo.uni.lu/api_doc (accessed on 18 February 2024).
48. Brusnwick, U. CICMalDroid. 2020. Available online: https://www.unb.ca/cic/datasets/maldroid-2020.html (accessed on 18 February 2024).
49. Surera Riera, T.; Bermejo Higuera, J.R.; Bermejo Higuera, J.; Martinez Herraiz, J.J.; Sicilia Montalvo, J.A. Prevention and Fighting against Web Attacks through. *Sustainability* **2020**, *12*, 4945. [CrossRef]
50. Antunes, N.; Vieira, M. On the Metrics for Benchmarking Vulnerability Detection Tools. In Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Rio de Janeiro, Brazil, 22–25 June 2015; pp. 505–516.