IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 26, 2021, accepted August 11, 2021, date of publication August 25, 2021, date of current version September 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3107903

Malware Detection Inside App Stores Based on Lifespan Measurements

CARLOS CILLERUELO, ENRIQUE-LARRIBA, LUIS DE-MARCOS, AND JOSE-JAVIER MARTINEZ-HERRÁIZ

Computer Science Department, University of Alcalá, 28801 Alcalá de Henares, Spain

Corresponding author: Carlos Cilleruelo (carlos.cilleruelo@uah.es)

This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant 826284 (ProTego).

ABSTRACT Potentially Harmful Apps (PHAs), like any other type of malware, are a problem. Even though Google tries to maintain a clean app ecosystem, Google Play Store is still one of the main vectors for spreading PHAs. In this paper, we propose a solution based on machine learning algorithms to detect PHAs inside application markets. Being the application markets one of the main entry vectors, a solution capable of detecting PHAs submitted or in submission to those markets is needed. This solution is capable of detecting PHAs inside an application market and can be used as a filtering method, to automatically block the publishing of novel PHAs. The proposed solution is based on application static analysis, and even though several static analysis solutions have been developed, the innovation of this system is based on its training and the creation of its dataset. We have created a new dataset that uses as criteria the lifespan of an application inside Google Play, the shorter time an application is active inside an application market the higher the probability that this is a PHA. This criterion was added in order to avoid the usage and bias of antivirus engines for detecting malware. Involving the lifespan as criteria we created a new method of detection that does not replicate any existing antivirus engines. Experimental results have proved that this solution obtains a 90% accuracy score, using a dataset of 91,203 applications published on the Google Play Store. Despite showing a decrease in accuracy, compared with other machine learning models focused on detecting PHAs; it is necessary to take into account that this is a complementary and different method. The presented work can be combined with other static and dynamic machine learning models, since its training is drastically different, as it was based on lifespan measurements.

INDEX TERMS Machine learning, app stores, google play malware, android malware, malware detection, potentially harmful apps.

I. INTRODUCTION

Malware detection techniques are constantly evolving due to the necessity of detecting the presence of malware. Cybercriminals are constantly changing their techniques and novel methods of detection are needed to be developed. Moreover, Android has become one of the most popular operating systems in mobile devices. According to Statcounter, Android has a market share greater than 72% [1]. This situation has caused an increase in the malware ecosystem because of its popularity [2], [3]. All of this is related to the rise of smartphone users worldwide, more than 6 billion in 2021 [4]. Due to this situation, cybercriminals are increasing attacks against smartphones and the Android ecosystem in particular.

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

On top of that, we should take into account that even in the latest Android version 11, the system still allows installing applications from unverified sources. Several malware SMS campaigns, using SMiShing techniques [5], had exploited this possibility [6], [7] but the use of markets, third-party markets, and the official Google Play Store, is still the main distribution vector of infection for most Android malware [8]. Being Google Play Store the main distribution vector, novel techniques that control who published and which applications are published need to be developed. This evaluation is currently a challenge since there are around nearly 3 million applications in Google Play Store [9], making it difficult to evaluate all of them. A proposed solution should be applicable to all published applications and also have an acceptable evaluation and detection time. In 2017 Google tried to accomplish a solution to this problem by developing a system

called Google Play Protect [10]. Google Play Protect is a security measure that has managed to block, just in 2017, approximately 10 million harmful app installations [11]. However several studies [12], and the current situation of Android malware inside the Google Play Store [8], [13], has proved this technology inefficient. The incapacity or disregard from Google has been evidenced due to the number of malware campaigns in Google Play Store [14], [15]. And it is necessary to take into account that Android allows several alternative markets where there are even more malware applications [16]. Better detection rates are needed to fight malware inside application markets.

Furthermore, we should not forget the emergence of the Internet of Things (IoT) ecosystem and the use of Android as its operating system in these environments [17], [18]. These ecosystem also needs novel methods of malware detection [19], [20]. Those Android IoT devices can also incorporate the usage of application markets, for this a solution that grants better control over application markets will benefit, not only smartphones but also the full Android ecosystem.

Within this context, it is important to research and apply new methods of PHAs detection. Moreover, these new detection methods need to be applicable in the real world and take into account applications particularities. For example, there are devices, like Samsung, with a proprietary software development kit(SDK) that can use specific permissions like, samsung.accessory.permission.ACCESSORY_FRAMEWORK. These permissions can only be found in certain devices and have been normalized or removed to guarantee a multi-platform market solution. It is not real to create novel detection methods that do not take into account these peculiarities or are based on unique features like C&C domains or IP addresses.

In this paper, we present a novel method of detection based on lifespan measurements that can be used for detecting malware in application markets. This is a lightweight method that makes it possible to easily scan millions of applications in a feasible time. For example, newer applications submissions can be processed through this detection method, without significantly affecting the publication process. We developed an automatic solution based on static analysis techniques but taking into account the previous mentioned particularities of the Android application ecosystem. The features used by this system have been carefully normalized and can be present in any Android application. These features are divided into groups like Permissions, Hardware, or Google Play Store categories. This approach also generates a lightweight design, that contributes to its easier implementation. Only 601 application features are used in the training and evaluation methods. But the main difference with other detection methods is the PHAs dataset composition and its use in the creation of this detection system.

A dataset of 91,203 applications, published inside the Google Play Store has been utilized for this research. To split the applications into legitimate or PHAs a new classification criterion has been applied. Instead of just using know antivirus engines to classify the samples, the lifespan of mobile applications inside the Google Play Store has also been used as a selection criterion.

In summary, a list of contributions of this paper are the following:

- *Unique dataset*. We systematically created and labelled a dataset based on applications lifespan inside the Google Play Store. This dataset is publicly accessible in GitLab, https://gitlab.com/ciberseg-uah/public/pha-android-dataset
- *New method of detection of PHAs*. Using the previously mentioned dataset, we created a new method of PHAs detection. This method is based on machine learning techniques and involves this lifespan measure as a selection criteria.
- *Antivirus Engines Bias Avoidance*. The use of our classified dataset avoids the bias produced by antivirus engines. We are not replicating the behaviour of an antivirus engine, but creating a new detection method.
- *Explainable Results*. The proposed method has been tested and trained using different machine learning models and techniques. These techniques are explained and analyzed in this research paper. All machine leanings algorithms in this research allow obtaining explainable results.
- *Lightweight training and Feature Selection*. In order to create a heterogeneous solution for the Android ecosystem, we reduce and normalized the application features to common ones. The system only uses 601 features for the training and evaluation of applications instead of thousands of features. Also, being a static analysis detection method, the model presents a Mean time to detect (MTTD) smaller than one second per application.

All these contributions follow practical use. This system, created using machine learning techniques, could be used for the early detection of malicious campaigns inside application markets. And for early detection of PHA before its publication inside the market. Multiple PHAs could be detected during the submission and validation process made by application markets.

II. BACKGROUND AND RELATED WORK

Day to day, malware evolves to pretend it is a legitimate program, increasing the complexity of the detection process. Furthermore, malware detection leads to another problem. There are cases where there is not a clear line to distinguish malware. Different antivirus engines have different criteria when classifying samples [21] Hurier *et al.* clearly state this lack of consensus in antivirus engines [22]. Antivirus used a threshold in order to consider a malware sample. Hurier *et al.* also stated that there is no public theory or golden rule behind the selection of this threshold. Finally, their work concludes by explaining the necessity of novel detection methods and the use of aggregated antivirus decisions for avoiding bias. Harmful applications may be detected by some antivirus and in other cases being classified as benign or detected as

malware but classified in different categories of malware. Each antivirus engine has different policies dealing with PHAs, occasionally more relaxed or restrictive, on malware analysis [23]. For example, some of them could have heavy policies against adware, and others tolerate this type of PHAs.

Malware analysis may involve different methodologies and techniques. Some of them base their classification on the recognition of known patterns on the program code [24]. If a certain code has been previously detected on security incidents, it will be remembered and detected by the antivirus engines, regardless of the machine in which it is executed. This analysis is usually done through static analysis, an analysis performed without actually executing programs.

Another technique used in malware analysis is dynamic analysis. When classifying new malware samples which have never been detected before, remembering patterns on the code does not improve the identification [24]. It is necessary to execute an analysis of the application while it is being executed to verify that its functionalities are the ones expected. Dynamic analysis has been used in malware detection [25], [26] and has been proven effective thanks to the information provided, which describes a program and its behaviour.

The classification of samples is a problem that requires the recognition of patterns on a data set. Static and dynamic malware analysis can provide a lot of features and patterns of PHAs. Due to its ability to recognize patterns, Machine Learning is a good technology for the implementation of novel detection methods. It offers several algorithms capable of find patterns and classify data based on certain features. Even so, it is needed to supervise the training by specifying the class to which samples belong.

The use of Machine Learning, for classifying mobile applications, has been involved in a multitude of studies [27], [28] [29]. Some researchers had used Support Vector Machines (i.e. SVM) [30]. Others have used algorithms like Random Forest Classifier (i.e. RFC) and Linear Regression [31] to classify applications.

There have been several studies applying machine learning to detect PHAs [30], [32] [33]. One of the most relevant studies is Drebin [30]. Drebin achieves a detection rate of 94% using 545,000 different features. But from our point of view, several features should not be considered (e.g., Network addresses, application-defined permissions, activities' names). Malware is easily mutable and training a machine learning model with unique characteristics will not improve the detection rate. It is necessary to generate a dataset with common characteristics that all the PHAs could have.

Another research related to Android malware classification that uses a more generic approach selecting features of a dataset [34] is APK Auditor. APK Auditor [34] uses Android permissions, common features between applications to create a permission-based model obtaining a detection rate of 88%. Even though the accuracy of APK Auditor is lesser than Drebin, APK Auditor is a better approach to malware detection. In a real-world environment, the accuracy of APK Auditor will be better than Drebin due to the evaluation of common features between samples. On the other hand, other recent studies have also proved the effectiveness of machine learning in dealing with Android IoT malware. [35], [36]

III. METHODOLOGY

A. AVOIDING ANTIVIRUS DETECTION BIAS

First of all, we created a novel dataset using Tacyt.[1] Tacyt is a cyber-intelligence tool developed by ElevenPaths, a cybersecurity company subsidiary of Telefónica Digital España, S.L. This tool performs a crawling of different mobile application markets, including the Google Play Store. This process downloads and then performs a static analysis of millions of mobile applications. Using this tool we were able to access a database of 273,662 applications, published inside the Google Play Store between 22th of January 2010, and 11th of July 2018. Those applications allowed us to create a labeled dataset of 91,203 applications. Tacyt provides information about each application, like the Android Package (APK) files and dates associated with the Google Play Store publication or its publishing category. This allows its subsequent analysis, even after the application has been deleted from the market.

There are several public PHAs datasets [37], [38] already accessible but to present an innovative way of detection we did not use any of them. Those datasets use antivirus engines as a classification method. Also none of the recently published datasets involve application store lifespan measurements to their building. AndroCT [39] and TraceDroid [40] present static and dynamic analysis data but do not give any metric related to lifespan. Moreover, recent works combined static and dynamic analysis, like Cai *et al.* [41], Cai and Ryder [42] which studied application structure and behaviours and then create an application classification approach base on that information. Cai *et al.* [43] also studied the evolution of benign and malign applications in the Android ecosystem to understand its behaviours. But again the previously mentioned approaches did not specifically study applications published in application markets and did not involve lifespan measurements. It is necessary to understand that the novelty of this work is not based on the application of static analysis, a technique widely tested and studied in numerous research papers [30], [44] [45].The novelty of the proposed machine learning model is based on its new method of dataset creation. First of all, the dataset creation takes into account the antivirus bias problem and involve a new selection method based on lifespan measures. As this method is specifically designed to be used in application stores, lifespan measures are taking based on the lifespan of applications inside those stores. Additionally to these unique characteristics, this method can be combined with previous methods, due to its different creation and behaviour. This novel method can be combined with previous ones as an ensemble learning method, improving previous detection rates.

[1] https://www.elevenpaths.com/es/tecnologia/tacyt/index.html

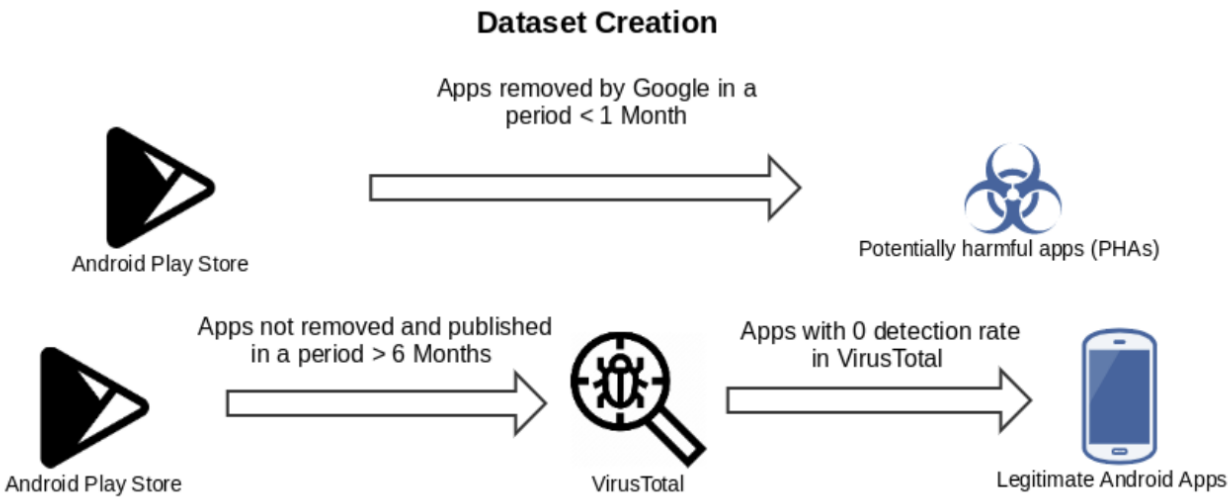

FIGURE 1. Method used in order to select applications for the dataset.

In our use case, since we are using supervised learning algorithms, we also need to define which applications can be considered PHAs and which ones are not. As previously mentioned, distinguish between PHAs and legitimate applications is a problem. In multiple cases, there is not a clear line that differentiates between benign and malicious apps. This also happens using antivirus decision engines, several antivirus engines can provide different results analyzing the same application. Moreover, if we classify our samples based on antivirus decision engines we will be only replicating their judgment and not creating a new one.

The proposed approach is based on considering as PHAs android applications that have been banned from the Google Play Store market. Those applications could be considered harmful to the user, so they should be detected by the machine learning model. At the moment of writing this article, Google does not publicly share any information of banned applications from the Google Play Store. The strategy followed by this research uses Tacyt to access the publishing and removal dates of each application, inside Google Play. Each application that Google removes in less than a month is considered PHAs.

This new approach allows us to create a new way of detection that avoids imitating the criteria and bias of antivirus engines and creates a unique machine learning model able to identify PHAs inside application markets. This machine learning model is able to detect applications whose characteristics are similar to PHAs that have been previously removed and present a lifespan of less than a month inside the Google Play Store Market.

On the other hand, the samples of non-malicious applications need to be filtered before their inclusion on the dataset. The reason for this is due to the fact that a large number of malware applications are not removed from the Google Play Store. This make necessary to verify that the samples collected are benign. All possible non-malicious applications have been verified through VirusTotal [46]. This service allows scanning each sample by 67 different antivirus engines, ensuring that all applications are harmless. Only applications with 0 detection rate and a period of life greater than six months have been included as non-malicious applications. The rest of the possible non-malicious applications were not included inside our dataset. Even though some of them present a low detection rate in VirusTotal we preferred to avoid those cases. The usage of VirusTotal service can seem counterproductive because we are involving antivirus detection bias in our dataset creation process. But like other security solutions, this new detection method need to maintain a False Negative (FN) ratio to the minimum. The usage of VirusTotal is only used to successfully guaranteed a clean dataset of legitimate applications. VirusTotal allow us the possibility of removing PHAs from the dataset, malicious applications with more than six months of life inside the Google Play Store. We look up to obtain a significant sample of legitimate applications and their lifespan, and if we only base on the number of downloads we could end tainting that sample. Several malicious campaigns have been known for being able to accomplish millions of downloads inside official markets during extended periods of time [47], [48]. We would have preferred not to depend on existing antivirus solutions in any part of the process. But this was the only option that allows us to obtain a legitimate dataset of applications published in the application markets. The usage of Tacyt guaranteed that these applications were published in the application market and the number of downloads but not its legitimacy. This classification process, previous to the training, is shown in Figure 1. In total, the dataset contains 91,203 applications, divided as shown in Table 1.

B. DATASET COVERAGE

Tacyt allows to query a database with more than seven million applications published inside the Google Play Store, and it

TABLE 1. Classification of application samples.

	Applications	Percentage
Malicious	36,539	40.06%
Benign	54,664	59.94%

also offers valuable metadata like the number of downloads of each application, publisher or market category amongst others. Based on that information a dataset has been created. The distribution based on downloads of Tacyt database can be found in Table 2. A distribution based on the number of downloads of each application ensures the representativity and coverage of the dataset. As mentioned before, our dataset is composed of [91,203] applications, a distribution based on the number of downloads of our dataset that is presented in Table 3.

If we compare Table 2 and Table 3 several differences can be appreciated. There is a larger number of applications in our dataset within the range of 201, 1000 downloads and less number of apps within the range of 10001, 100000 and 100001, 500000. To not unbalance the dataset we maintained a similar number of malicious and benign applications in our dataset. Due to our criteria for selecting malicious applications, an application with a lifespan less than a month inside the Google Play Store, it is not possible to find a lot of results with more than 10001 downloads. Because of this reason, we increased the range 201, 1000 to have more malicious applications.

TABLE 2. Distribution inside Tacyt database based on the number of downloads.

Download range	Population - Play Store Applications	Percentage
[0, 200]	2,686,557	39.40
[201, 1000]	1,089,295	15.98
[1001, 10000]	1,529,846	22.44
[10001, 100000]	980,316	14.38
[100001, 500000]	345,425	5.07
[500001, ∞]	186,832	2.73

TABLE 3. Distribution inside our dataset based on the number of downloads.

Download range	Population - Play Store Applications	Percentage
[0, 200]	35,935	39.40
[201, 1000]	31,016	34.00
[1001, 10000]	13,820	15.15
[10001, 100000]	7942	8.7
[100001, 500000]	505	0.55
[500001, ∞]	1985	2.17

Altogether, 601 features have been extracted from each application. These include the permissions requested by the application, the hardware resources that the application it is trying to access, and the information published on the Google Play Store. The entire dataset, that is composed of these features, is used to train and test the effectiveness of the learning model. Moreover, authors also considered Android run time permissions too. Starting with Android 6.0, Marshmallow, developers can ask for permissions on runtime. But those permissions need to be specified in the app's manifest file, like any other permission [49]. Tacyt extracts permissions using different techniques. One of them is the analysis of the app's manifest files, which guarantee the extraction of runtime and non-runtime permissions. This dataset is the one that allows us to create a new method for PHA detection that avoids the replication of existing antivirus engines and uses the lifespan as a feature. This novel method can detect with a 90% accuracy when an application is going to be removed, in a period less than a month, from the Google Play Store. Other research methods have presented better accuracy measurements but it is necessary to bring out again the differential characteristics of this novel method of detection. To the best of our knowledge, this is the first malware detection method that uses the lifespan of applications inside a market as selection and detection criteria.

To assure a representative and coverage of our dataset, the number of downloads is not the only metric that we checked. On one hand, we added and reviewed the number of android permissions used in our dataset. There are 455 permissions, which identify the data and system features that the applications may access.

Most declared permissions, by the applications stored in the dataset, are shown in Table 4.

TABLE 4. Most popular permissions in Android applications.

Permission	Applications	Percentage
INTERNET	88,850	0.97
ACCESS_NETWORK	84,848	0.93
READ_EXTERNAL_STORAGE	55,023	0.60
WRITE_EXTERNAL_STORAGE	54,576	0.41
WAKE_LOCK	37,391	0.41
ACCESS_WIFI_STATE	37,226	0.33
Others	Less than 30,000	-

On the other hand, hardware access permissions were included and then tested as representative attributes of the sample. These android permissions refer to the use of some hardware components, like the camera. All those hardware components of the Android operating system have been included in the dataset as characteristics of each application.

In total, 105 hardware declarations have been included in the dataset. Those declarations are distributed, as shown in Table 5, across the dataset.

Finally, data application size, price, minimum Android SDK version, and developer have been included as features.

TABLE 5. Most popular hardware components in Google Play Store applications.

Hardware	Applications	Percentage
Android.hardware.camera	762	0.8%
Android.software.live_wallpaper	326	0.4%
Android.hardware.screen.portrait	275	0.3%
Android.hardware.touchscreen	195	0.2%
Android.hardware.sensor.accelerometer	111	0.1%
Android.hardware.screen.landscape	100	0.1%

We considered this information useful in the detection of PHAs and also allowed us to get some insights about popular categories, like games or education applications. As an example of this, malware designed to act like small-sized video games may have different permissions and features than malware designed to act like medium-sized social applications. The category and type of each application, which determine how it is classified in the Google Play Store, is presented in Table 6.

TABLE 6. Most common categories on the Google Play Store.

Category	Applications	Percentage
ENTERTAINMENT	10,604	11.63%
GAME	7,399	8.11%
MUSIC_AND_AUDIO	6,164	6.76%
TOOLS	5,579	6.12%
BOOKS_AND_REFERENCE	5,556	6.09%
EDUCATION	5,131	5.63%
LIFESTYLE	5,080	5.57%

C. FEATURES SETS AND NORMALIZATION

This research has always taken into account the current situation and techniques of PHA development and distribution. This knowledge has been applied in the creation of the dataset and the following feature selection. The proposed solution is intended to be a real solution that can be applied to all possible Android applications, independently of the device manufacturer. As previously mentioned, the features selected for this training have been specifically studied and every feature that did not represent a common characteristic between applications has been removed. The solution proposed in this paper does not use network addresses, activity names or specific permissions associated with a manufacturer. Different PHAs will only share those features in the case that they are from the same family or developer. For example, in the case of botnets different PHAs will not share the same network addresses because they will have different command-and-control (C&C) servers. Involving these features will grant further detection rates in our test dataset but will not be representative of a real case scenario.

Moreover, multiple permissions may be the same but present differences based on the application package. Table 7 presents some Android permissions used in Drebin [37] and then normalized in our experiments. This process is the one that allows us to only use 601 features instead of thousands. It is necessary to make clear that this process was not designed to create a lightweight system but to create a generic solution that can behave well in real environments. The lightweights of the system is a consequence of this feature selection process. Regarding this topic, recent works have also tried to find the best features for machine learning training. Surendran *et al.*, used a system call sequence generated by malware applications to identify common patterns and create detection features [50]. The mentioned solution is a dynamic analysis solution, but any real applicable solution needs to involve a feature selection process and involve common features across multiple applications.

D. MACHINE LEARNING CLASSIFIERS

PHA automatic detection, like any other malware detection, is a binary classification problem. And supervised machine learning algorithms have proven to be successful detecting PHAs in numerous studies [27], [29], [32]. The use of mathematical algorithms oriented to classification problems allow the creation of trained models that sort out different applications based on their features. But it is necessary to take into account that the accuracy of these algorithms are heavily related to the training dataset. Using supervised training, all the data must be chosen carefully to obtain good performances. Our approach follows these ideas but with substantial changes like the selection and normalization of features and the dataset creation process.

Machine Learning algorithms search for a mathematical function that is able to distinguish effectively between different types of samples. Since this is a binary classification problem, and taking into account the current state of the art, the following algorithms have been chosen: Support Vector Machines(SVM), Stochastic gradient descent(SGD), Random Forest Classification(RFC) y eXtreme Gradient Boosting(XGB). Most research work uses SVM [30] or One-Class SVM algorithms [27] but we extended the test set involving RFC and a modern classifying algorithm like XGB. Those algorithms are the ones that have been used to train a model using our custom dataset. And later on, their effectiveness and accuracy in classifying PHAs have been exhaustively evaluated.

The training has been done gathering 70% of the samples randomly, 63,842 of 91,203. The remaining 30% is used as the test dataset, 27,361 of 91,203. Thus, the test dataset allows establishing the effectiveness of the generated model through the use of the metrics precision, recall, and f1-score.

The algorithms used are implemented on the Scikit-learn[2] and DMLC-XGBoost[3] libraries, both written in Python. Scikit-learn allows using a training method known as grid search. It searches for the most optimum parameters, of each Machine Learning algorithm, during the model training. The grid search looks for the parameters that achieve a better f1-score. We used f1-score as a measure of a test's accuracy because considers both the precision and the recall of the test to compute the score.

A more detailed and formal description of the machine learning process used during the training of the model are the following:

- Stochastic gradient descent (SGD) [51] algorithm used tries to minimize the value returned by the softmax function. It searches for a hyperplane that divides the dataset into two classes. The effectiveness of this model depends on whether the classes of the problem are linearly separable.
- Support vector machine (SVM) [52] algorithm has been widely used in classification problems.

[2]https://scikit-learn.org/stable/
[3]https://github.com/dmlc/xgboost

TABLE 7. Example of specific application permissions.

Permission	Normalized permission
com.samson.samsonproductions.**permission.C2D_MESSAGE**	permission.C2D_MESSAGE
com.dreamstep.wsmsanonymes.**permission.C2D_MESSAGE**	permission.C2D_MESSAGE
com.google.android.apps.chrometophone.**permission.C2D_MESSAGE**	permission.C2D_MESSAGE
com.android.samsung.rmt_exercise.**permission.KEYSTRING**	permission.KEYSTRING
com.sec.modem.settings.**permission.KEYSTRING**	permission.KEYSTRING

The implementation used in this research base its classification function on a Gaussian kernel. Thus, it is able to effectively distinguish radially separable problems.

- Random Forest Classification (RFC) [53] algorithm creates different sets of random decision trees. Through the training, it chooses the set whose decision trees make better decisions on average.
- eXtreme Gradient Boosting [54], [55] algorithms are used in classification [56] to create prediction models based on an ensemble of weak prediction models. Those weak models are decision trees that, through the training, discard the less valuable features of a certain data class. The most valuable features create decision trees with an associated weight. The total sum of these weights is the output value that identifies each class. This behaviour allows an application, depending on whether its category is Tools or Education, to be classified in a different way even when their features are similar.

IV. RESULTS

A. MACHINE LEARNING MODELS TRAINING AND COMPARISON

Due to the balanced dataset, distribution and the normalized features, we expected that novel optimized gradient boosting classifiers like XGBoost outperform other traditional classifiers like SVM. Results for the SGD are presented on Table 8. SGD returned 13,809 as true negatives samples, 8,929 as true positive, 2,183 as false negatives and 2,440 false positive samples, resulting in an overall f1-score score of 83%.

TABLE 8. SGD Classification Report.

Dataset	Precision	Recall	f1-score
Goodware	86%	85%	86%
Malware	79%	80%	79%
Average	83%	83%	83%

Results for the SVM are presented on Table 9. SVM returned 13,748 as true negatives samples, 8,816 as true positive, 2,101 as false negative and 2,696 false positive samples, resulting in an overall f1-score score of 83%.

Results for the RFC are presented on Table 10. RFC returned 15,311 as true negatives samples, 9,222 as true positive, 1,487 as false negative and 1,341 false positive samples, resulting in an overall f1-score score of 90%.

Results for the XGB are presented on Table 11. XGB returned 15,150 as true negatives samples, 9,316 as true positive, 1,648 as false negative and 1,247 false positive samples, resulting in an overall f1-score score of 89%.

TABLE 9. SVM Classification Report.

Dataset	Precision	Recall	f1-score
Goodware	87%	84%	85%
Malware	77%	81%	79%
Average	83%	82%	83%

TABLE 10. RFC Classification Report.

Dataset	Precision	Recall	f1-score
Goodware	91%	92%	92%
Malware	87%	86%	87%
Average	90%	90%	90%

TABLE 11. XGB Classification Report.

Dataset	Precision	Recall	f1-score
Goodware	90%	92%	91%
Malware	88%	85%	87%
Average	89%	89%	89%

Table 12 presents the results of different machine learning algorithms applied to our dataset. Even though the model trained with the XGB algorithm reaches 89% accuracy, the RFC model achieves 90% accuracy with a false positive rate of 5.43%. It is a small difference, but it made RFC the suited algorithm for this problem. A greater difference was found in the models trained with SVM and SGD, achieving an 82% and an 83% of f1-score respectively. This denotes the fact that algorithms based on ensemble learning are the best ones facing this type of problems.

TABLE 12. Comparison of different classification algorithms.

Algorithm	Precision	f1-score	False negative	False positive
SVM	83%	83%	7.98%%	8.92%
SGD	83%	83%	7.68%	9.85%
RFC	90%	90%	5.43%	4.01%
XGB	89%	89%	6.03%	4.56%

V. DISCUSSION

After the training, the results obtained seem to be promising. On one hand, we have XGB with an accuracy of 89% and on the other hand, we have RFC with a 90% accuracy.

SVM and SGD did not behave that well with our dataset in comparison with RFC. This has some explanation, since the classification of malware and PHAs are not always a simple task. It is difficult to draw a line between the different sets of applications. Like we mentioned before in this paper this

is also shown in the antivirus market [21]. Some applications could be PHAs to some antivirus engines and others could be considered non-malicious applications by other antivirus engines. In a malware classification problem, we will always encounter a lot of grey areas.

Because of that difficult classification, a random forest approach behaves better selecting malware and PHAs applications. Also, the RFC algorithm allows us to identify which features are more important when classifying samples. Using the 601 features, which constitute the dataset, those with more weight are shown in Table 13.

TABLE 13. Most important features.

Feature	Importance
applicationType (APPLICATION)	0.159357
targetSdkVersion	0.090245
nFiles	0.083069
size	0.078786
minSdkVersion	0.066245
nActivities	0.050453
nImages	0.049002
nPermissions	0.032027
Other features	0.388817

Previous research like Drebin had achieved an accuracy of 93.90% [30], but they do not present other values like recall, precision, or f1-score. Furthermore from our point of view, Drebin did not apply a generic approach to features selection. They present a solution with 545,000 different features. Like mentioned before Drebin uses network addresses, activities' names, and other unique features. Because of the large quantity and type of features selected it will not behave well in a real world environment. These unique features has been taken into account, features that will only exist in an application of a group of applications developed by the same developer or group of developers.

If you use network addresses as a feature you will detect some PHAs but you will discard others inside the model because that feature will not be present in all cases. To detect the error inserted by these features, it will be necessary to evaluate the weight of each feature inside the model.

The proposed model uses 601 features. This is a great difference and it generates a lightweight machine learning system, in comparison with other works like Drebin that instead has used 545,000 [37] features. Sometimes the reduction of features could have an impact on the accuracy. But research works like Cai *et al.* has shown that a specific set of selected features, in their work they only used 70 features, can obtain promising results in PHAs detection [41]. Moreover, it is necessary to take into account that all of the 601 features selected could exist in any Android application. Because of this selection of features, we consider that our approach will present better results in a real world environment. Additionally the lightweight of the system directly affect the MTTD of the system. A PHA can be identified in less than a second by the system. Like any other machine learning system, the system will need to periodically be retrained but this factor also affects this timing. The lightweight of the system reduces the amount of time and hardware needed to train this solution.

All these results can be summarized in two main contributions. First, that it is possible to use methods based on the lifespan of applications inside Google Play Store, for creating PHAs datasets. And second, the number of features required for training these machine leanings models have been drastically reduced, 601 versus other machine learning systems that used thousands of features [37], [44].

Moreover, it could be interesting to compare these results with other industries and research solutions, not only previous research papers. The industry average is around 98%, according to the studies of AV-TEST - The Independent IT-Security Institute [57]. But it is necessary to take into account that these solutions also perform dynamic analysis of the applications, the presented solution is only based on static analysis features. Another comparison can be made against Google Play Protect, the mobile malware detection solution offered by Google. This solution has an accuracy of around 70% [10] analyzing applications published in the Google Play Store.

A. REAL-TIME USE CASE SCENARIO

Through the different sections of this paper, we have mentioned how this system has been designed to be applied in a real use case scenario. The design and test have always taken this into account. The proposed way to use this machine learning detection system in the real world could be to use it as an application validation process.

Before the publishing of an Android application into any store, this application can be scanned by the machine learning system presented in this paper. Being an automatic process it will not severely impact the application validation process. Thus, this validation can be used as an indicator of PHA. The current machine learning model present a MTTD (Mean Time To Detect) smaller than one second per application. Being a static analysis method, it does not need to study the application behaviour for a specified amount of time inside a sandbox. This MTTD could be increased if the feature extraction time is taken into account. In order to evaluate the application through this system, the application's permissions, accessed hardware components and categories to publish need to be known. Most of this information is available in the Android application manifest, so collection time must be considered. And even though that the process of obtaining and parse an Android application manifest can be done in a matter of seconds, any official application store can ask for this information. During the upload and submission process of an application, the application market can ask for a separate manifest file, corresponding to the application, in the submission form. In conclusion, the small MTTD would not affect the submission performance of applications and will end blocking several PHA along the process.

On the other hand, one of the problems of this system will be with the False Negative rate, 5,43%, but further work and data could improve this detection rate.

VI. CONCLUSION AND FUTURE WORK

This paper presents a new way for training and detecting PHAs inside the Android ecosystem. The objective is to detect mobile applications that will be removed by Google in a period shorter than one month, where applications removed by Google in short periods from the store are, in most cases, PHAs or malware. To achieve this goal, a new dataset has been created and several classification algorithms have been used, SGD, SVM, RFC, and XGB. The dataset creation uses as criteria the lifespan of an application inside Google Play instead of antivirus decision engines, for identifying PHAs. Training with this dataset a Random Forest Classifier machine learning, a 90% of effectiveness can be reached.

One of the main limitations of this approach is its accuracy. Future work can be done in this aspect and for example, the combination of several algorithms through ensemble learning techniques could obtain better results. Also, like any other machine learning model, it is necessary to periodically retrain this detection model with new data to detect new threats.

Another possible limitation is the way that PHAs are selected in our dataset. The proposed approach considered PHAs based on the lifespan of applications inside the Google Play Store. Our selected PHAs are applications that Google banned or removed from the Google Play Store. But it is not possible to know how many of them were PHAs or applications infringing Google Play Store policies. Applications could not be a PHA but Google could consider that it is infringing publishing policies. Moreover, a lot of PHAs are not banned or retired by Google in a short period. Our machine learning model is detecting the most common and aggressive campaigns but most elaborated ones could evade our system.

Finally, this approach has proved that is possible to create an automated analysis solution for detecting PHAs based on the lifespan of the application inside markets. On one hand, Google could use this system to detect if a new application is going to be removed from the Google Play Store and use it as a filter for newly published applications. On the other hand, any security research could use this model for detecting aggressive mobile malware campaigns. Finally, we also prove that a limited set of generic features can be used for detecting PHAs.

These results also evidence the necessity of identifying and conceiving new detection methods that avoid the usage of antivirus commercial models. To increase detection rates new methods that do not try to emulate actual commercial tools need to be developed.

REFERENCES

[1] *Mobile Operating System Market Share Worldwide*. Accessed: Jun. 11, 2021. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] G. Kelly. (2014). *Report: 97% of Mobile Malware is on Android. This is the Easy Way You Stay Safe*. [Online]. Available: https://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile%-malware-is-on-android-this-is-the-easy-way-you-stay-safe

[3] C. Lueg. (Jun. 2017). *8, 400 New Android Malware Samples Every Day*. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malw%are-samples-every-day

[4] (2020). *Smartphone Users*. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-w%orldwide/

[5] J. H. Says. (Jan. 2020). *SMiShing: About the FedEx SMS Phishing Scam | McAfee*. [Online]. Available: /blogs/consumer/consumer-threat-notices/fedex-sms-phishing-scam/

[6] J. H. Says. (Jan. 2020). *SMiShing: About the FedEx SMS Phishing Scam | McAfee*. [Online]. Available: /blogs/consumer/consumer-threat-notices/fedex-sms-phishing-scam/

[7] *FakeSpy Android Malware Spread Via Postal-Service Apps*. Accessed: Jun. 11, 2021. [Online]. Available: https://threatpost.com/fakespy-android-malware-spread-via-postal-servic%e-apps/157102/

[8] P. Kotzias, J. Caballero, and L. Bilge, "How did that get in my phone? Unwanted app distribution on Android devices," 2020, *arXiv:2010.10088*. [Online]. Available: http://arxiv.org/abs/2010.10088

[9] Statista. (2017). *Number of Available Applications in the Google Play Store From December 2009 to December 2020*. [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applicat%ions-in-the-google-play-store/

[10] *Test Google Play Protect 24.3 for Android (213208)*. Accessed: Jun. 11, 2021. [Online]. Available: /en/antivirus/mobile-devices/android/march-2021/google-play-protect-24.%3-213208/

[11] MalwareBytes. (Mar. 2018). *Android Security 2017 Year in Review*. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf

[12] S. Hutchinson, B. Zhou, and U. Karabiyik, "Are we really protected? An investigation into the play protect service," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 4997–5004.

[13] D. Maier, T. Müller, and M. Protsenko, "Divide-and-conquer: Why Android malware cannot be stopped," in *Proc. 9th Int. Conf. Availability, Rel. Secur.*, Sep. 2014, pp. 30–39.

[14] BleepingComputer. (Nov. 2017). *Google Play Store Sees Sudden Surge of Malicious Apps*. [Online]. Available: https://www.bleepingcomputer.com/news/security/google-play-store-sees-s%udden-surge-of-malicious-apps/

[15] MalwareBytes. (Nov. 2017). *New Android Trojan Malware Discovered in Google Play*. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2017/11/new-trojan-malware-dis%covered-google-play/

[16] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *Proc. NDSS*, vol. 25, no. 4, Feb. 2012, pp. 50–52.

[17] Techcrunch. (May 2018). *Google's Android Things IoT Platform Comes Out of Beta*. [Online]. Available: https://techcrunch.com/2018/05/07/googles-android-things-iot-platform-c%omes-out-of-beta/

[18] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl.*, Nov. 2014, pp. 230–234.

[19] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A multimodal malware detection technique for Android IoT devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[20] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "End-to-end malware detection for Android IoT devices using deep learning," *Ad Hoc Netw.*, vol. 101, Apr. 2020, Art. no. 102098. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870519310984

[21] I. Gashi, V. Stankovic, C. Leita, and O. Thonnard, "An experimental study of diversity with off-the-shelf AntiVirus engines," in *Proc. 8th IEEE Int. Symp. Netw. Comput. Appl.*, Jul. 2009, pp. 4–11.

[22] M. Hurier, K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of Android malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (Lecture Notes in Computer Science), J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham, Switzerland: Springer, 2016, pp. 142–162.

[23] A. Mohaisen, O. Alrawi, M. Larson, and D. McPherson, "Towards a methodical evaluation of antivirus scans and labels," in *Proc. Int. Workshop Inf. Secur. Appl.*, USA. Cham, Switzerland: Springer, Aug. 2013, pp. 231–241.

[24] D. J. Sanok, Jr., "An analysis of how antivirus methodologies are utilized in protecting computers from malicious code," in *Proc. 2nd Annu. Conf. Inf. Secur. Curriculum Develop.* Kennesaw, GA, USA: Kennesaw State Univ., Sep. 2005, pp. 142–144.

[25] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Secur. Privacy*, vol. 5, no. 2, pp. 32–39, Mar./Apr. 2007.
[26] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *Proc. 21st USENIX Secur. Symp. (USENIX Secur.)*, 2012, pp. 569–584.
[27] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," in *Proc. Eur. Intell. Secur. Informat. Conf.*, Aug. 2012, pp. 141–147.
[28] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in Android malware detection," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 371–372, 2014.
[29] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.
[30] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.
[31] I. Martín, J. A. Hernández, A. Muñoz, and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," *Secur. Commun. Netw.*, vol. 2018, pp. 1–11, Jul. 2018.
[32] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proc. IEEE 25th Int. Conf. Tools With Artif. Intell.*, Nov. 2013, pp. 300–305.
[33] B. Baskaran and A. Ralescu, "A study of Android malware detection techniques and machine learning," presented at the Mod. Artif. Intell. Cogn. Sci. Conf. Dayton, OH, USA: Univ. Dayton, Apr. 2016, p. 9. [Online]. Available: https://ecommons.udayton.edu/maics/2016/Saturday/3/
[34] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digit. Invest.*, vol. 13, pp. 1–14, Jun. 2015.
[35] R. Kumar, X. Zhang, R. U. Khan, and A. Sharif, "Research on data mining of permission-induced risk for Android IoT devices," *Appl. Sci.*, vol. 9, no. 2, p. 277, Jan. 2019. [Online]. Available:https://www.mdpi.com/2076-3417/9/2/277
[36] R. Kumar, W. Wang, J. Kumar, T. Yang, and W. Ali, "Collective intelligence: Decentralized learning for Android malware detection in IoT with blockchain," 2021, *arXiv:2102.13376*. [Online]. Available: http://arxiv.org/abs/2102.13376
[37] Drebin. *Drebin Dataset*. Accessed: Nov. 9, 2017. [Online]. Available: https://www.sec.cs.tu-bs.de/~danarp/drebin/
[38] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment (DIMVA)*. Bonn, Germany: Springer, 2017, pp. 252–276.
[39] W. Li, X. Fu, and H. Cai. (Jan. 2021). *AndroCT: Ten Years of App Call Traces in Android*. Type: Dataset. [Online]. Available: https://zenodo.org/record/5010831
[40] H. Cai. (Jan. 2020). *TraceDroid: Eight-Year Behavioral Profiles of Android Apps*. Type: Dataset. [Online]. Available: https://zenodo.org/record/3665877
[41] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019
[42] H. Cai and B. G. Ryder, "A longitudinal study of application structure and behaviors in Android," *IEEE Trans. Softw. Eng.*, early access, Feb. 19, 2020, doi: 10.1109/TSE.2020.2975176.
[43] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in Android," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106291. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300410
[44] Q. Han, V. S. Subrahmanian, and Y. Xiong, "Android malware detection via (somewhat) robust irreversible feature transformations," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3511–3525, 2020.
[45] Q. Wu, M. Li, X. Zhu, and B. Liu, "MVIIDroid: A multiple view information integration approach for Android malware detection and family identification," *IEEE MultimediaMag.*, vol. 27, no. 4, pp. 48–57, Oct. 2020.
[46] VirusTotal. *Virustotal. Free Online Virus, Malware and URL Scanner*. Accessed: Oct. 5, 2019. [Online]. Available: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-use%rs-worldwide/
[47] (Oct. 2019). *Google Play Store Malware Hits 42 Apps With 8 Million Downloads*. [Online]. Available: https://www.digitaltrends.com/mobile/google-play-store-malware-hits-42-%apps-with-8-million-downloads/
[48] D. Winder. *New Android App Malware Infects 250 Million Downloads–Here's What You Need to Know*. Section: Cybersecurity. Accessed: Oct. 5, 2019. [Online]. Available: https://www.forbes.com/sites/daveywinder/2019/03/13/new-android-app-mal%ware-infects-250-million-downloads-heres-what-you-need-to-know/
[49] *Request App Permissions*. Accessed: Jul. 19, 2021. [Online]. Available: https://developer.android.com/training/permissions/requesting
[50] R. Surendran, T. Thomas, and S. Emmanuel, "On existence of common malicious system call codes in Android malware families," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 248–260, Mar. 2021.
[51] H. Robbins and S. Monro, "A stochastic approximation method," in *The Annals of Mathematical Statistics*. USA, 1951, pp. 400–407.
[52] T. Fletcher, "Support vector machines explained," Tutorial Paper, 2009, pp. 1–19. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/43282568/SVM_Explained-with-cover-page-v2.pdf?Expires=1629968616&Signature=LRvcm2bJ4ipySw~14j4sls6wz-kCVwLIGAq2CiUMryiUE30Xe8waIIAckZHGVEXVUBPcSaSJO4eHyFwIxbbv2SzFNcqdYlqtZLHHaaYhiQjlAJXQRc7MTRC8pmFibCBNvbNaGmnHsiOLn-m9QDFXQya3SXufPq5CJ9kqE6eC5Jtu4gaTuqSfga1RgSgtgprkpRRd6V9eTLr9kjlcQMzzN1Y2vvajIN5i6Fov-buE8CxRWIVt59e8c6zhIst wA1mXYJFiUHHbR2vayaP2N4mu7KmvAt7xdugozURHtLlKq9zU3WxKL28lacQEVVFWYr20w6MnqjgFmbNh-yspFDFcg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
[53] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
[54] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.
[55] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.
[56] I. B. Mustapha and F. Saeed, "Bioactive molecule prediction using extreme gradient boosting," *Molecules*, vol. 21, no. 8, p. 983, 2016.
[57] (2021). *Test Antivirus Software for Android*. Accessed: Jul. 19, 2021. [Online]. Available: https://www.av-test.org/en/antivirus/mobile-devices/