Breaking the Structure of MaMaDroid

Harel Berger[a,*], Amit Dvir[b], Enrico Mariconti[c], Chen Hajaj[d,b]

[a]*Computer Science Department, Georgetown University, 37th and O Streets NW, Washington DC, USA.*
[b]*Computer Science Department, Ariel Cyber Innovation Center, Ariel University, 65 Ramat HaGolan st., Ariel 4076414, Israel.*
[c]*Jill Dando Institute, Department of Security and Crime Science, UCL, Gower Street, London WC1E 6BT,United Kingdom.*
[d]*Department of Industrial Engineering and Management, Data Science and Artificial Intelligence Research Center,65 Ramat HaGolan st., Ariel 4076414, Israel.*

Abstract

Android malware is a continuously expanding threat to billions of mobile users around the globe. Detection systems are updated constantly to address these threats. However, a backlash takes the form of *evasion attacks*, in which an adversary changes malicious samples in the wild such that they will be misclassified as benign. This paper comprehensively inspects a well-known Android malware detection system, MaMaDroid, which analyzes the control flow graph of the application. Changes in the portion of benign samples in the training set are considered to reveal their effect on the resulting classifier. These changes in the ratio between benign and malicious samples have a clear effect on each of the models, resulting in a decrease of more than 40% in their detection rate, model confidence, and reliability. Moreover, adopted Machine Learning models were implemented as well, including 5-NN, Decision Tree, and Adaboost. Exploration of the six models showed a typical behavior in different cases, of tree-based models and distance-based models. Moreover, three novel attacks that manipulate the Control Flow Graph (CFG)[1] are

*Corresponding author. Parts of this work were done when Berger was at Ariel University.

Email addresses: `bergerar0@gmail.com` (Harel Berger), `amitdv@g.ariel.ac.il` (Amit Dvir), `e.mariconti@ucl.ac.uk` (Enrico Mariconti), `chenha@ariel.ac.il` (Chen Hajaj)

[1]Non-standard abbreviations: Black Hole Statistical (BHS), Control Flow Graph (CFG), Defense Reciprocal Rate (DRR), Evasion Robustness (ER), Feature Access (FA),

described for each of the targeted models. The attacks decrease the detection rate of most models to less than 10%, with regards to different ratios of benign to malicious apps. As a result, a new version of MaMaDroid is engineered, which fuses the CFG of the app and static analysis of features of the app. This improved model is proven to be robust against evasion attacks targeting CFG-based models and static analysis models, achieving a detection rate of $\sim 80\%$.

Keywords: Machine Learning, Evasion Attacks, Android Malware Detection

1. Introduction

Malicious software, a.k.a malware, is defined as a file or program that tries to damage the normal activity of a digital device. Malware can take many forms, including viruses, worms, Trojans, ransomware, spyware, adware, and rootkits (Choo, 2011). Malware can be spread using various vectors, such as email attachments, downloads from the internet, or physical media like USB drives (Christodorescu et al., 2005). Recent research shows that malware can cause significant damage to computer systems, networks, and data, leading to financial losses and privacy breaches. For instance, ransomware attacks, a type of malware that encrypts victims' data and demands a ransom for its release, continue to be a major threat to businesses and individuals worldwide (Bekkers et al., 2023).

Most malware is specifically engineered according to their target operating system (OS), as OSs vary based on their hardware and functionalities. One of the popular targets for malware is the Android OS. Android application Packages (APKs), the popular executable files of Android OS, can be found in many Android markets around the world (e.g., Google play Store Google (2008)). Many malware apps in these markets target the attention of unsuspecting users to ensure that they download the malicious app. For example, Etinu malware leverages the fear of recent COVID-19 in Asia (WIRE, 2021). This malware stole information from incoming SMS messages, made purchases in the victim's name, and infected more than 700K users. As a safeguard for these users and many more, many researchers and

Full Statistical (FS), Manifest Based (MB), Model Reliability (MR), Statistics Access (SA), Structure Break (StB) cyber experts are seeking an efficient solution for the correct identification of malware applications (Aafer et al., 2013; Arp et al., 2014; Berger et al., 2021; Cai and Jenkins, 2018; Chen et al., 2016; Dini et al., 2012; Enck et al., 2009; Huynh et al., 2017; Kabakus, 2022; Onwuzurike et al., 2017; Shabtai et al., 2012, 2009, 2014; Shin et al., 2020; Sun et al., 2016; Talha et al., 2015; Treadwell and Zhou, 2009; Venugopal and Hu, 2008; Wang et al., 2014; Wu et al., 2012; Xu et al., 2013). Several studies have been conducted throughout the years to mitigate the threat of Android malware, implementing methods such as the use of heuristics of app structures and signatures (Treadwell and Zhou, 2009; Venugopal and Hu, 2008), permissions' analysis (Enck et al., 2009; Sun et al., 2016; Talha et al., 2015; Wang et al., 2014; Xu et al., 2013).

Machine learning (ML) algorithms have been commonly used for Android malware detection in recent years. The algorithms' ability to effectively classify and analyze large datasets of complex features extracted from Android apps increased their popularity. According to Aafer et al. (Aafer et al., 2013), traditional malware detection techniques using signature analysis are insufficient in detecting new and unknown types of malware. In comparison, ML algorithms can learn from past data to improve their accuracy, even in the presence of obfuscation techniques used by malware creators (Li et al., 2016; Tong et al., 2019). This has led to the development of different ML-based Android malware detection systems, such as MaMaDroid, Drebin, and AndroidAPIMiner, detection systems that demonstrated high detection rates of both known and unknown types of malware (Arp et al., 2014; Aafer et al., 2013; Onwuzurike et al., 2017). Therefore, ML algorithms have become an essential tool in the fight against Android malware. A strong ML classifier is based, among others, on the raw data that must represent the domain correctly. Specifically, for different subdomains of malware detection, there is a different real-world distribution of malicious and benign samples. For example, in the subdomain of URL malware detection, most Internet addresses are considered malicious (Maggi et al., 2013; Rahbarinia et al., 2017). In the domain considered in this paper, i.e., Android malware, most of the apps are considered benign (Google, 2017; Lindorfer et al., 2014; Pendlebury et al., 2019). A recent and popular study defined the recommended ratio as 90/10 between benign and malicious Android apps (Pendlebury et al., 2019). As a result, the challenge for Android malware classifiers is to analyze the important characteristics of malicious apps, despite the low volume of malicious apps in the dataset.

Even an ML classifier trained on the right amount of benign and malicious apps is not 100% accurate on any input sample. Goodfellow et al. (Goodfellow et al., 2014) proved that some ML classifiers are susceptible to manipulations. These manipulations are called *adversarial examples.* Adversarial examples are created when an attacker manipulates malicious samples so that the sample will be misclassified as benign and vice versa (Grosse et al., 2017; Kuppa et al., 2019; Yuan et al., 2019). In turn, *evasion attacks* are intelligent attacks in which the adversary manipulates malicious instances, such that they will be wrongly classified. Evasion attacks that change the physical malicious instance are called problem-space evasion attacks (e.g., (Berger et al., 2020, 2021; Chen et al., 2019; Pierazzi et al., 2019)). On the other hand, evasion attacks that manipulate the extracted feature vector of an instance are called feature-space evasion attacks (e.g., (Athalye et al., 2018; Demontis et al., 2017; Li and Li, 2020)).

To address the threats that arise from evasion attacks on APKs, this work follows one of the well-known Android malware detection systems, MaMaDroid (Onwuzurike et al., 2017). MaMaDroid is based on a control flow graph (CFG) (Allen, 1970), where a CFG is a representation of a program using graph notation. This representation depicts all paths that might be traversed through the program while executing it.

The contribution of this paper is threefold. First, it presents a full evaluation of MaMaDroid as a function of the portion of benign instances in the training set. Both clean data and evasion attacks are evaluated (compared to (Pendlebury et al., 2019), where the authors evaluated only clean data). This evaluation spans multiple models and focuses on the detection of malicious entities, the effect of attacks on the confidence of the model in its prediction, and the reliability of the model. Second, this paper introduces three innovative problem-space evasion attacks against MaMaDroid. The classifiers are evaluated on both clean data and data manipulated by various evasion attacks. These two cases are thoroughly evaluated based on multiple metrics (i.e., evasion robustness, defense reciprocal rate, and model reliability), and a diverse set of models. Finally, this paper provides a more robust version of MaMaDroid, i.e., MaMaDroid2.0. The new version incorporates an extended feature set. MaMaDroid2.0 was evaluated against the novel evasion attacks and other known evasion attacks. Compared to the MaMaDroid model in (Onwuzurike et al., 2017), which results in a detection rate of $\sim 0\%$, the novel version provides a recall of more than 60%.

2. Related Work

In Section 2.1, four main approaches in the field of Android malware detection systems are described. It is noteworthy that evasion attacks leverage weak spots in ML-based Android malware detection systems. These attacks take two forms: problem-space attacks and feature-space attacks. Problem-space attacks (Alzantot et al., 2018; Apruzzese and Colajanni, 2018; Dang et al., 2017; Demontis et al., 2017; Li et al., 2018a; Rastogi et al., 2013; Zheng et al., 2012) include modification of the samples. This is the type of attack implemented in this study. Feature-space attacks (Biggio et al., 2013; Carlini and Wagner, 2017; Chen et al., 2018a; Shahpasand et al., 2019; Shao et al., 2019; Xu et al., 2020; Zikratov et al., 2017) map the sample into a feature vector and modify the values of the features. Feature-space attacks are easier to implement than problem-space attacks. Also, Feature-space attacks can be generated automatically by an ML system (Aydogan and Sen, 2015; Chen et al., 2017; Hu and Tan, 2017; Ming et al., 2015; Zhao et al., 2019). Nonetheless, the correlated code in the sample that needs to be changed according to these attacks may severely damage the functionality of the sample (Berger et al., 2020; Chen et al., 2019; Pierazzi et al., 2020; Salem et al., 2018). Therefore, feature-space evasion attacks are less realizable. This study relates to problem-space evasion attacks. Therefore, a review of significant works on problem-space evasion attacks on Android malware detection systems is presented in Section 2.2.

2.1. Android Malware Detection Systems

This section surveys well-known Android malware detection systems following four main approaches. The first approach is static analysis, which gathers significant strings from the Smali code files and Manifest file. The second strategy is based on the control flow graph (CFG) of the application, which traces the order of the API calls used in the app. The behavior of the app is inspected in the third approach, which gathers information on the behavior of the Android OS during the run of the app, along with network packets sent and received, etc. The last approach analyses bytecode sequences. Several Hybrid systems are discussed as well.

Probably one of the most well-known Android malware detection systems using static analysis is Drebin (Arp et al., 2014). Drebin gathers eight types of static features, with a specification of two main components of the APKs. Drebin extracts permissions requests, software/hardware components, and intents from the Manifest file. From the Smali code, it extracts suspicious/restricted API calls, used permissions in the app's run and URL addresses. Sec-SVM (Demontis et al., 2017) is an improved version of Drebin, using a more evenly weighted learning model. Other versions of Drebin include a Factorization Machine (Li et al., 2019) and a DNN (Li and Li, 2020). The DroidAPIMiner (Aafer et al., 2013; ChenJunHero, 2018) is similar to Drebin, in regards to static analysis and feature types. The API calls and permissions are the feature set of this detection machine. The authors of this research performed a dataflow analysis for frequently used API values and package names. The static analysis detection systems look for several features from the APK. Enumeration of these features is easy and efficient. However, since most of the static features can be easily understood, they can also be automatically manipulated without much effort, like in (Demontis et al., 2017; Maiorca et al., 2015; Meng et al., 2016; Rastogi et al., 2013).

A more robust approach is MaMaDroid (Onwuzurike et al., 2017), which extracts the CFG of an application as a base for its feature set. MaMaDroid generates a tree of API calls based on family and package names. Then, the detection system analyzes the API call sequence performed by an app by each mode - family or package, to model its true nature. A similar approach to detect malicious third-party use in apps was employed by Backes et al. (Backes et al., 2016). Function and app profiling, such as types and parameters, were used to identify third-party libraries and different versions of the same library. Zhiwu et al. (Xu et al., 2018) used CFG along with data flow to characterize Android apps, along with a CNN model to predict the labels of new samples. An extension was suggested in (Zhiwu et al., 2019), where the same technique was employed using n-grams to identify malware families. Monitoring the sequence of functions and API calls inside the app seems like a great idea. Changing the flow of the app is more complicated, since the CFG may be complex and full of details. Furthermore, changing the order of API calls of the app may damage the malicious activity of the app. However, several evasion attacks manipulate the flow of the app and succeed in deceiving this kind of detection system, such as (Chen et al., 2019; Ikram et al., 2019; Maiorca et al., 2015; Piao et al., 2016; Sun and Tan, 2014).

The third approach tries to map the traces of a malicious app, thus acknowledging the behavior of such apps, using several operating systems and communication features, as was introduced in Andromaly (Shabtai et al., 2012). A similar approach was taken by Shabtai et al. (Shabtai et al., 2014).

These detection systems focused on network usage by measuring the network traffic patterns in a host device running an app. The authors learned the statistics of network packets a user sent and received, the RTT, etc. Another research using a similar strategy was conducted by Shabtai et al. (Shabtai et al., 2010). The authors aggregated data during an app run, including user interactions like clicks and touches and OS behavior such as CPU and network usage. Saracino et al. (Saracino et al., 2016) found the correlations between features at four levels: kernel, application, user, and package, to identify malicious applications. Wang et al. (Wang and Li, 2021) explored the number of pages on virtual machines, the change of states between tasks, and so forth. Behavioral detection methods are based on the nature of the Android device while running various apps. These behaviors may depend on the app and are therefore hard to generalize. Therefore, they are not a complete solution to malware detection.

Bytecode inspection is the last approach to Android malware detection. Dalvik Bytecode Frequency Analysis (Kang et al., 2013) is one example, which looks for popular Dalvik Bytecode instructions of malware apps. Sequences of Dalvik Bytecode instructions were also explored by TinyDroid (Chen et al., 2018b). The authors of this system gathered families of Bytecode instructions under a single symbol and used n-grams (Damashek, 1995) to create the feature set. Yang et al. (Yang et al., 2015) extracted Bytecode instructions from the APK and converted them to a matrix. This matrix was analyzed by a CNN. Bytecode inspection is a heavy method and ambiguous for the human eye, as it is not a convenient programming language.

A hybrid detection system was suggested by Martín et al. (Martín et al., 2019). This system fused the static and dynamic analysis of Android apps. The authors combined the transitions between execution states (dynamic) and the inspection of API calls (static). A combination of static analysis of permission requests and dynamic testing of their derived API calls was introduced in (Wang et al., 2015). MARVIN (Lindorfer et al., 2015) inspects the nature of Android apps through static analysis of their design, certificates, etc. In addition, a dynamic analysis of behavioral activity is processed. The combination of static analysis of permissions and intents and dynamic analysis of network traffic was introduced by Ding et al. (Ding et al., 2021). A hybrid solution seems the best option to identify malicious apps. However, each method added as another layer of processing consumes time and resources from the host device. Table 1 summarizes the advantages of each paper in this section.

Paper	**Type**	**Robust to to Static Changes**	**Robust to to Flow Change**	**Easily Generalized**	**Light Process**
Drebin (Arp et al., 2014)	Static Analysis		✓	✓	✓
Sec-SVM (Demontis et al., 2017)	Static Analysis		✓	✓	✓
FM (Li et al., 2019)	Static Analysis		✓	✓	✓
DNN (Li and Li, 2020)	Static Analysis		✓	✓	✓
Droid-API-Miner (Aafer et al., 2013)	Static Analysis		✓	✓	✓
MaMaDroid (Onwuzurike et al., 2017)	CFG Analysis	✓		✓	✓
Backes et al. (Backes et al., 2016)	CFG Analysis	✓		✓	✓
Zhiwu et al. (Xu et al., 2018)	CFG Analysis	✓		✓	✓
Zhiwu et al. (Zhiwu et al., 2019)	CFG Analysis	✓		✓	✓
Andromaly (Shabtai et al., 2012)	Behavioral Analysis	✓	✓		
Shabtai et al. (Shabtai et al., 2014)	Behavioral Analysis	✓	✓		
Shabtai et al.	Behavioral	✓	✓		

Paper	**Type**	**Robust to to Static Changes**	**Robust to to Flow Change**	**Easily Generalized**	**Light Process**
(Shabtai et al., 2010)	Analysis				
Saracino et al. (Saracino et al., 2016)	Behavioral Analysis	✓	✓		
Wang et al. (Wang and Li, 2021)	Behavioral Analysis	✓	✓		
Kan et al. (Kang et al., 2013)	Bytecode Inspection	✓	✓	✓	
TinyDroid (Chen et al., 2018b)	Bytecode Inspection	✓	✓	✓	
Yang et al. (Yang et al., 2015)	Bytecode Inspection	✓	✓	✓	
Martín et al. (Martín et al., 2019)	Hybrid	✓	✓	✓	
Wang et al. (Wang et al., 2015)	Hybrid	✓	✓	✓	
MARVIN (Lindorfer et al., 2015)	Hybrid	✓	✓	✓	
Ding et al. (Ding et al., 2021)	Hybrid	✓	✓	✓	

Table 1: Android malware detection systems. This tables summarizes the advantages of each Android malware detection system of Section 2.1, including the robustness to static analysis and flow change, and also the generalization and processing of the detection system.

2.2. Problem-Space Evasion Attacks

This section targets the problem-space evasion attacks against malware detection systems. Problem-space evasion attacks are categorized into three forms of attacks. The first uses camouflage to conceal incriminating strings and values contained in the app, by encryption and obfuscation. Next, the second incorporates noises into the app; e.g., uncalled functions. At last, the third form tries to alter the behavior of the app. It reviews the flow of the original app and manipulates the code of several function calls.

The first course of action in evasion attacks is to conceal specific suspicious components of the app. One well-known example of a concealment effort is with the help of encryption or obfuscation. Demontis et al. (Demontis et al., 2017) obfuscated suspicious strings, packages, and API calls. Another example of concealment is packing an app inside a fellow app. DaDidroid (Ikram et al., 2019) explored an approach similar to that of Demontis et al. (Demontis et al., 2017) using packing and obfuscation.

Reflection allows a program to change its nature at runtime. It is another classic evasion technique. Rastogi et al. (Rastogi et al., 2013) presented an attack, which mixes the Demontis et al. (Demontis et al., 2017) obfuscation method with the addition of the reflection approach.

Another form of problem-space evasion attack includes adding noise to the app. These noises mislead the classifier's labeling process. An example of noise addition can be a stub function/code injection. A stub function is a non-operational function, that does not do anything. However, it changes the original flow of an app. An example of a stub function addition is Android HIV (Chen et al., 2019), where the authors implemented non-invoked suspicious functions against the Drebin classifier and a stub function injection against the MaMaDroid classifier. A recent example of this kind of attack is in (Pierazzi et al., 2020), where the authors implanted benign snippets of a code in malicious apps to evade the Drebin and Sec-SVM (Demontis et al., 2017) classifiers. Rosenberg et al. (Rosenberg et al., 2018) generated an evasion attack against Android malware detection systems using API call manipulation. Three methods were used: The addition of non-operational functions to the application, obfuscation of strings, and encoding of short API calls. Cara et al. (Cara et al., 2020) added non-invoked classes to the end of functions to evade the classification of Android malware.

The last approach changes the app flow. One of the ways to implement this approach is by function outlining/inlining. In function outlining, the attacker breaks a function into smaller code snippets. In function inlining, the adversary replaces a function call with the entire function body. This technique was implemented in Droidchameleon (Rastogi et al., 2013), which incorporated function outlining in its evasion attacks. Another option to break the app flow is the stub function, as in (Chen et al., 2019; Piao et al., 2016; Sun and Tan, 2014), resulting in an ML misclassification of a malicious app.

3. MaMaDroid

The targeted system, namely MaMaDroid (Onwuzurike et al., 2017, 2019), is presented in this section. MaMadroid is an Android malware detection system introduced in 2017 by Onwuzurike et al. (Onwuzurike et al., 2019). It extracts features from the CFG of an APK sample. It enumerates abstracted API calls to capture the behavioral model of the app. Analyzing every API call used in the app to create the feature set is better, of course, because it captures the flow of the app in a more precise way. However, there are numerous API calls. The analysis of a single application can take weeks. Therefore, an abstraction of API calls is a suitable solution. The intuition behind analyzing the sequence of API calls or their abstraction is that benign and malicious apps conduct different sets of operations. For example, the recording of media by the android.media.MediaRecorder class is usually initiated after running getRunningTasks(). These API calls allow the recording of conversations (Zhang et al., 2015). This may indicate malicious activity. In contrast, benign apps may utilize calls to the MediaRecorder class and its API calls in any other order. MaMaDroid operates in two modes: family mode and package mode. For example, the API call **android.media.MediaRecorder->starts()** is abstracted as **android.** in the family mode, and **android.media.** in the package mode. Packages or families defined by the app's developer or obfuscated are abstracted as *self-defined* and *obfuscated*, respectively. In light of the fact that the structure of the evasion attack is similar for both modes, and the family mode results in lower processing time and memory, the family mode was chosen for this analysis. The structure of the evasion attack is similar for both modes.

MaMadroid creates the features for the learning algorithm in the following manner: First, it extracts the CFG from the app. Then, it gets the sequences of API calls. The APIs are then abstracted using one of the modes. Finally, it constructs a Markov chain (Brooks et al., 2011; Geyer, 1992; Marjoram et al., 2003), with the probabilities of transition between any family/package.

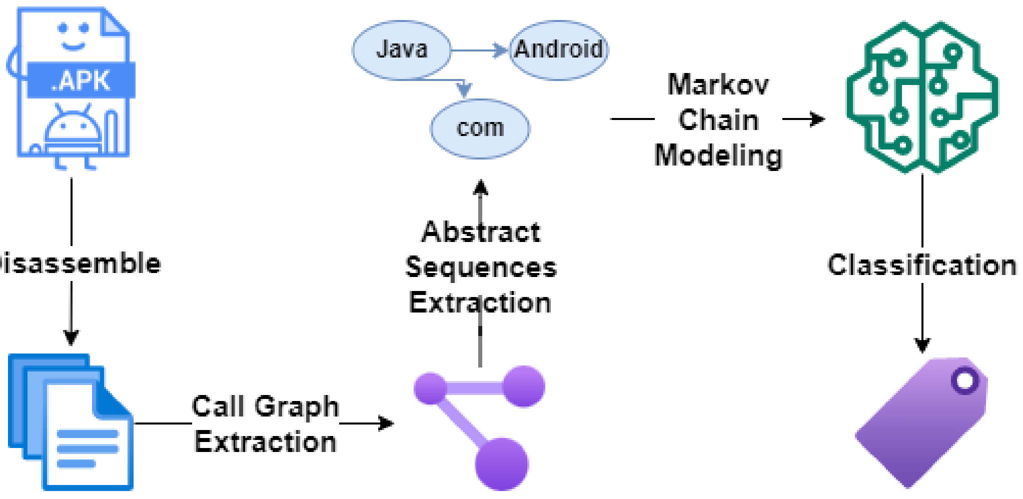


Figure 1: Overview of MaMaDroid Operation. First, the compressed APK file is disassembled to code files. Then, from the files, the API call graph is extracted. Due to the great volume of API calls, MaMaDroid abstracts the API calls as sequences of abstract calls. Then, a Markov Chain model is built based on the sequences, and a feature vector is created. At last, the app is classified as benign or malicious.

These probabilities are used as the features. For example, **androidTojava** is the feature that resembles the probability of transition between the android family to the java family. For a full description of the MaMaDroid classifier, see (Onwuzurike et al., 2017) (implementation is available at (Mariconti et al., 2018)). A graphic description of MaMaDroid is presented in Fig. 1. As an illustration, $\mathcal{A}$ is an imaginary malicious APK. It composes a handful of code files used to generate the call graph. In these code files, there are 10 files where the API call android.media.MediaRecorder->starts() is followed by java.io.FileWriter->write(). This happens in 80% of the cases where an API call that starts with Android is used in $\mathcal{A}$. When the graph is abstracted, the android->java transition has a high probability of 80%. Using this transition and others, $\mathcal{A}$ is labeled as malicious.

4. Evasion Attacks

This section describes the evasion attacks of this paper. The idea behind these evasion attacks is to break and change the structure of the sample so that the detection system would not recognize the manipulated samples as malicious. Each evasion attack is based on an abstract attacker model, that defines the knowledge the attacker holds. The attacker models are defined in Section 4.1. Using the embedded knowledge of the attacker models, three evasion attacks are devised. These attacks are variants of the general Structure Break (StB) evasion attack. The algorithm is presented in Section 4.2. Thereafter, the variants are described in Section 4.3.

4.1. Attacker Models

This section describes two attacker models, which depict the embedded knowledge each attack holds. The first model, named **Feature set Access (FA)**, depicts a gray-box attacker who knows the set of characteristics of the targeted system (as the attacker model in (Spooren et al., 2019) and the attack scenario F in (Chen et al., 2019)). The second model is the **Statistics Access (SA)**, which is a white-box attacker, and can access the feature set and the training data (as attack scenario FB in (Chen et al., 2019)).

4.2. Structure Break (StB) Algorithm

Given a malicious APK, the attacker manipulates the structure of the APK with StB evasion attacks. The general algorithm is presented in Algo. 1. This algorithm creates different manipulated APKs according to the use of three arguments (out of a total of four arguments, which includes the APK and the three mentioned arguments). Each attack variant uses a slightly different set of arguments. The arguments are *Mode Elements*, *L_func* and *P_func*.

4.2.1. Mode Elements

As mentioned in Section 3, MaMaDroid analyzes the transitions between families/packages. For example, the androidTojava transition depicts the transition from the android.* family to the java.* family. In this work, the family mode was chosen. There are 11 families in the family mode of MaMaDroid, e.g., android.*, java.*, json.*, etc. Therefore, let *Mode Elements* be a subset of the families where each item in the Mode Elements will be termed an *element*. The subset will be engineered in one of three ways:

1. **Randomly** - Randomly choosing a subset of families. Specifically, several families such as android.* java.* and xml.* can be chosen.
2. Statistically - This method uses the statistics of the given data. Given data can be the training and test data, or just the test data. This method is split into two sub-methods:

(a) **Full Statistical** method - In this sub-method, the training and test data are given. The elements that are chosen for the Mode Elements are the ones that hold high values in the benign data, and low values in the malicious data. For example, suppose the java family's transitions (javaTojava,javaToandroid, etc.) in the benign data are high values, and also low values in the malicious data. In that case, java will be selected for the Mode Elements. The idea is to try and mimic the behavior of the benign samples.

(b) **Black Hole** method- If only the test data is given, the least popular family is chosen for the Mode Elements. For example, if the java family's transitions (javaTojava,javaToandroid, etc.) in the test data are in low volume, then java will be selected for the mode elements. The idea behind this choice is to attempt to move as many transitions/feature values to the chosen element, which the attacker perceives as less popular. The transitions that belong to this element are supposed to weigh less and therefore are neglected by MaMaDroid.

4.2.2. L_func

L_func is a function that given the range of 0 to the directories tree's height (of the application) chooses a specific height for the manipulation process. This function randomly selects a value in the given range or returns a specific value. The type of L_func determines the return value.

4.2.3. P_func

P_func is a function that chooses a ratio of change (for the specific level chosen by L_func) given a range of change (the default is [0,1]). This function randomly selects a value in the given range or returns a specific value. The type of P_func determines the return value. For simplicity, two functions were defined, although their functionality is similar.

The algorithm implements the following steps: (1) Depackage the APK into the Manifest file, Smali code files, and other subordinate files (line 2); (2) Get the structure of the Smali code files as a tree (*APK_Tree*). In other words, traverse the root directory recursively to get the full structure of the directories and files. Store the output in *APK_Tree*. Also, get the height of the tree as the lowest file/directory of the tree by this traverse. Store it in *Height* (line 3); (3) Run L_func and store the result in L (line 4); (4) Run P_func and store the result in P (line 5); (5) Get a random item from the

Algorithm 1 Structure Break Attack - StB General Algorithm

```
1: procedure STRUCTURE BREAK ATTACK($APK, Mode\_elements, L\_func, P\_func$)
2:     $Manifest, Smali, Layouts \leftarrow depackage(APK)$
3:     $APK\_Tree, Height \leftarrow apk\_structure(Smali)$
4:     $L \leftarrow L\_func(0, Height)$
5:     $P \leftarrow P\_func(0, 1)$
6:     $f \leftarrow random(Mode\_elements)$
7:     $mkdir(f, APK\_Tree)$
8:     $Roots \leftarrow get\_roots\_random(APK\_Tree, P, L)$
9:     for each $r \in Roots$ do
10:        $r\_new \leftarrow f + r$
11:        $S\_roots \leftarrow get\_Smali\_files(r)$
12:        for each $file \in S\_roots$ do
13:            $file \leftarrow change\_oc(file, r, r\_new)$
14:            $move(r + file, r\_new + file)$
15:        for each $file \in Manifest, Layouts$ do
16:            $file \leftarrow change\_oc(file, r, r\_new)$
17:    $APK \leftarrow Repackage(Manifest, Smali...)$
18:    return $APK$
19: procedure CHANGE_OC($file, r, r\_new$)
20:    for each $line \in file$ do
21:        $file[line] \leftarrow file[line].replace(r, r\_new)$
     return $file$
```

set of Mode_elements (line 6) as f; (6) Create a directory whose name is f at the top of *APK_Tree*, alongside the former root of the tree (line 7); (7) According to steps 4-5, get P of the directories in level L of the *APK_Tree*. Store them as *Roots* (line 8); (8) For each directory r in the *Roots* set, run lines 10-16 (line 9); (9) Concatenate f as a prefix of the former directory name r. Call it *r_new* (line 10); (10) Store all the subdirectories and files of r in *S_roots* (line 11); (11) For each file/directory in *S_roots*, run lines 13-14 (line 12); (12) Run *change_oc* on the file/directory, r and *r_new* (line 13). The *change_oc* function (lines 20-21) replaces any occurrence of a line in a file/set of files with a replacement. In this case, change the occurrence of r with *r_new*; (13) Move the file from the previous directory r to the new directory *r_new* (line 14); (14) For each file in the set of Manifest and layout files, run line 16 (line 15); (15) Run the *change_oc* function on the file (line 16); (16) Repackage the APK (line 17), and return it as an output (line 18).

The attacker creates a new full functional APK, as it changes the structure of the Smali code files, the Manifest file, and the layout files (lines 12-16). As all of these files might include some occurrences of the part that changed (in other words, references to files that moved from their original places), these occurrences need to be changed accordingly. For example, a case where the changed family is android and the new family is xml. In this case, the android folder becomes a sub-directory of a new xml folder. In other words, **android.** becomes **xml.android.**. Each of the files of the depackeged application are changed by the change_oc function, so that each occurrence of the **android.** family is replaced by the **xml.android.**. The repackeged application created after these modifications take place differs in the structure of the folders and the strings that were changed. The basic functionality is preserved.

4.3. Structure Break Attack Variants

The previous section (4.2) described the general StB algorithm. This section describes three novel evasion attacks, which are variants of the StB general algorithm. Each variant is described by its attacker model, $Mode\_elements$, L_func, and P_func. In other words, while the algorithm is the same for the three evasion attacks, the inputs are not.

1. **Random StB Attack:** This attack variant is based on the FA attacker model, as it only uses the knowledge of the features' names. A random set of $Mode\_elements$ is taken from the family mode. The L_func and P_func randomly pick a level in the range of [0,$Height$],

and a change ratio between 0 and 1, respectively. This variant is called Random StB Attack, as the $Mode\_elements$ are chosen randomly. Following the example, in the Random StB attack, $\mathcal{A}$ is modified into $\mathcal{A}_{\mathcal{R}}$. Random L_func, P_func and *Mode elements* are chosen (e.g., 0, 50% and Json) for this modification. In the transitions of $\mathcal{A}_{\mathcal{R}}$, half of the occurrences of android.media.MediaRecorder are changed to json.media.MediaRecorder (with all their correlative uses and additional changes needed). Therefore, the android->java transitions that were 80% are now changed to only 40%.

2. **Full Statistical StB Attack:** This attack variant is based on the SA attacker model, which incorporates the knowledge of the features' names and the training data. As the test data are the given samples the attacker holds, it has access to the whole dataset. The $Mode\_elements$ are chosen statistically (for more information, see Section 4.1). The attacker chooses the element that has the highest values of transitions between families in the benign apps and low values in the malicious apps. Then, it stores it in $Mode\_elements$. The L_func and P_func are not random functions. They are both set to maximize the effect of the change. In other words, L_func chooses 0 to include the whole Smali directory tree in the change of the app. In addition, P_func is set to 1 to include the maximum number of files and directories in the manipulation. This attack variant is called Full Statistical StB Attack, as it leverages the full knowledge of statistics of the training data.
   Utilizing the Full Statistical StB attack on $\mathcal{A}$, $\mathcal{A}$ is modified into $\mathcal{A}_{\mathcal{FS}}$. L_func and P_func are constant (i.e., 0 and 100%) for this modification. The *Mode elements* is engineered using the additional data of correlations between transitions of benign and malicious apps. The element is picked as com. In the transitions of $\mathcal{A}_{\mathcal{FS}}$, all occurrences of android.media.MediaRecorder are changed to com.media.MediaRecorder (with all correlative uses and additional changes necessary). Therefore, the value of the android->java transitions decreased from 80% to 0%.
3. **Black Hole Statistical StB Attack:** This attack variant is based on the FA attacker model, which uses only the knowledge of the features' names. The element in $Mode\_elements$ is chosen statistically only by the test data. The attacker chooses the element that holds the lowest transition values between families in the test data it obtains (as explained in Section 4.1). The selected element is stored in $Mode\_elements$. The L_func and P_func are identical functions to

those used in the Full Statistical StB Attack, to maximize the effect of the attack. The idea of this attack variant is to move the maximum feature values to the features belonging to the chosen elements, as it initially does not hold many values. The features that belong to this family/element are assumed by the attacker to be neglected/have low weight by the detection system. This element gathers the values and makes them "disappear". Therefore, this variant is called the Black hole Statistical StB attack. In the third modification of $\mathcal{A}$ using the Black Hole Statistical StB attack, $\mathcal{A}$ is transformed into $\mathcal{A}_{\mathcal{BH}}$. L_func and P_func are constant (i.e., 0 and 100%) for this modification. The *Mode elements* is engineered using the additional data of zero values in the feature vector. The element is chosen as xml. In the transitions of $\mathcal{A}_{\mathcal{BH}}$, all occurrences of android.media.MediaRecorder are changed to xml.media.MediaRecorder (with all their correlative uses and additional changes necessary). Therefore, the android->java transitions which were 80% are now changed to 0%.

5. Metrics and Evaluations

A set of experiments was conducted to evaluate the effects of the evasion attacks suggested in Section 4.3. First, Section 5.1 describes the design of the experiments. Then, evaluation metrics are reported in Section 5.2, followed by their evaluation of the evasion attacks. Section 5.3 describes the evaluation of the evasion robustness (ER) metric. Section 5.4 presents the results of the defense reciprocal rate (DRR) metric and lastly, Section 5.5 describes the evaluation of the model reliability (MR).

5.1. Experimental Design

The experiments include an analysis of an extended set of MaMaDroid models. First, the set of distance-based models of MaMaDroid is used (1-NN and 3-NN) and extended by a 5-NN model. Second, the tree-based models are tackled in a parallel way. RF is the only tree-based model that was originally used. As RF is an ensemble of decision trees (DTs), the basic DT model is included. On the other hand, a boosting model (i.e., AdaBoost), a more sophisticated ensemble of DTs, is added. In contrast to the RF, where the final decision is based on the decision of each DT independently, in AdaBoost, each DT focuses on the wrong classification of the prior ones. In total, six models were explored.

An experiment was conducted for each evasion attack using these six models (1-NN, 3-NN, 5-NN, DT, RF, Adaboost). The experiments were run on an Intel(R) Xeon(R) CPU E5-2683 v4 2.1 GHz with 64 GB RAM with GeForce RTX 2080 TI GPU. The dataset for the experiments consisted of $\sim$ 73K benign apps from the Google Play Market (Google, 2008) (obtained from Androzoo (Allix et al., 2016)) and $\sim$6K malicious apps from the Drebin dataset (Ali-Gombe et al., 2016; Arp et al., 2014; Berger et al., 2021; Frenklach et al., 2021; Elish et al., 2022; Maiorca et al., 2017; Yuan et al., 2020; Zulkifli et al., 2018).[1] To account for variations, the experiments were run 5 times, where each time the order of benign applications, and the split of malicious applications to train and test were random. The STD was less than 5%. Different ratios of benign and malicious apps were used during the experiments, to test the changes in the ER of each of the models. The data were split into train and test sets such that 80% of the data was used for the train set, while the remaining 20% were assigned to the test set. However, as explained, the number of benign apps in the train data changed during the run of the experiments, between 10% of the benign train data and 100% of the benign train data. The test data without any manipulation is termed *clean data*, while the post-manipulation test data is termed *manipulated data*. To fully evaluate evasion attacks, an additional experiment on clean data was used as a baseline. Also, as the effect of the amount of benign data on the decision function of each model is tested, the malicious apps are fixed for each experiment, while the benign ratio changes (10%,20%,. . . ,100%).

The modifications of the apps caused by the StB attacks did not change any functionality of the manipulated samples. This is straightforward as every occurrence of the family that was picked, was changed accordingly. In other words, any reference to the old name of the chosen family was changed (as explained in Section 4). Nevertheless, a random subset of 90 apps was tested from each StB attack variant using DroidBot (Li et al., 2017). Each

[1]The benign dataset was obtained from Androzoo and represent apps from 2020 and 2021. This difference between the origin of each part of the dataset may raise a bias in the dataset, as was described by (Pendlebury et al., 2019). However, as proven in (Berger et al., 2022), there is no agreed dataset in the Android malware detection domain. Moreover, the benign and malicious sources for Android applications that are used in this paper are the most popular among APK sources. At last, even if there is a bias toward benign applications, this paper shows that some algorithms are less affected by this bias than others (e.g., DT).

app was installed and run on an emulator. Using the DFS policy of DroidBot, a subset of 5 actions were tested on the original and manipulated apps. One hundred percent of the sample apps retain the basic functionality of the original malicious apps.

5.2. Metrics

Evasion Robustness (ER): To evaluate robustness, the portion of malicious instances that was incorrectly classified was calculated (similar to the analysis provided in (Tong et al., 2019)). The true positive rate (TPR) of malicious apps was used as this evaluation metric, both for manipulated apps and clean malicious data.

Defense Reciprocal Rank (DRR): A new metric was presented to assess the strength of an evasion attack in (Brama et al., 2022). The intuition behind the defense reciprocal rank (DRR) is that a correct classification is not the only factor in an evasion attack. The confidence of the classifier matters as well. For example, lower confidence may indicate that the attack, although failing to fully deceive the classifier, gained some effect on the classifier and could be improved more easily to become a successful attack.

Each of the classes (i.e., benign and malicious) is given a rank. The ranking can be based on several parameters. For example, the popularity among the data or target classes vs other classes. In this work, the target class, which is the malicious class, is ranked 1. The benign class is given the rank of 2. $\bar{P}_i$ represents the probability assigned to the true class i, and R_i is the rank of that class within the ordered predictions list. For the class chosen by the classifier, the $\bar{P}_i$'s range is between 0-1. For the second-best class, $\bar{P}_i$'s range is between 0-0.5, because if it was greater than 0.5, it would have been the class selected. The other classes follow a similar rule. The ranks in a binary classification are 1 and 2, where 1 is the highest rank. The ranks in this paper are selected such that 1 is the right class of the sample, and 2 is the wrong class. Each rank has a range. In light of the fact that $\bar{P}_i$ will be mapped to a R_i's range, the ranges should not overlap. The range of the first rank is between 0.5-1, the second rank is between 0.33-0.5, and so forth. The DRR is calculated in Eq. 1, for a sample x:

$$DRR(x) = \frac{\bar{P}_i}{R_i + 1} + \frac{1}{R_i + 1} \tag{1}$$

The first element maps the range of $\bar{P}_i$ to a range the size of R_i's range. The second element adds the lower bound of the R_i's range. A sum of both maps

$\bar{P}_i$ to the actual range of R_i. Therefore, the overall DRR of a classifier CL on a set of samples X is defined in Eq. 2:

$$Overall\_DRR(CL, X) = \frac{\Sigma_{x \in X} DRR(x)}{|X|} \tag{2}$$

Model Reliability (MR): As the process of a binary classification goes, for each test sample, the model outputs probabilities of the positive class and the negative class, where they complement 1. For each test sample, the entropy of the positive class and negative class is denoted as s. In other words, $s = E(P, \overline{P})$, where E symbolizes the entropy function, and P,$\overline{P}$ are the probabilities of each class produced by the classifier. To fully evaluate the magnitude of each model, the Shannon Entropy of the probabilities is used, to generate a number that represents the reliability of the model, similarly to the approach suggested in (Nguyen et al., 2020). Reliability is based on the complement of the average entropy of a test sample. As entropy usually measures uncertainty and disorder, reliability is calculated as the complement of the average entropy. That way, the model reliability (MR) score is translated as the measurement of disorder and uncertainty of the classifier. The MR is defined with regards to the classifier CL, the set of test samples X and the set of entropy S for the test set X using Eq. 3:

$$Re(CL, X, S) = 1 - \frac{S}{|X|} \tag{3}$$

Combining the three metrics results in a comprehensive analysis of attacks from different angles. The ER metric depicts the gap in detection rates of malicious samples, between clean and manipulated apps. The DRR metric describes the effect that the samples have on the certainty of the prediction function of each detection model. Finally, the MR metric concludes the analysis with the overall reliability of each detection model. As demonstrated in the next section, the following scenario is feasible for two arbitrary models: The first model is highly affected by the evasion attacks in terms of DRR compared to the second. However, the first model is more reliable in tackling evasion attacks. Therefore, DRR and MR metrics complete each other in terms of a thorough analysis of the effects of evasion attacks.

5.3. Results - Evasion Robustness

The first evaluation analyzes the effect of the benign ratio on the ER. Naturally, as one can expect, as the ratio of benign samples in the training

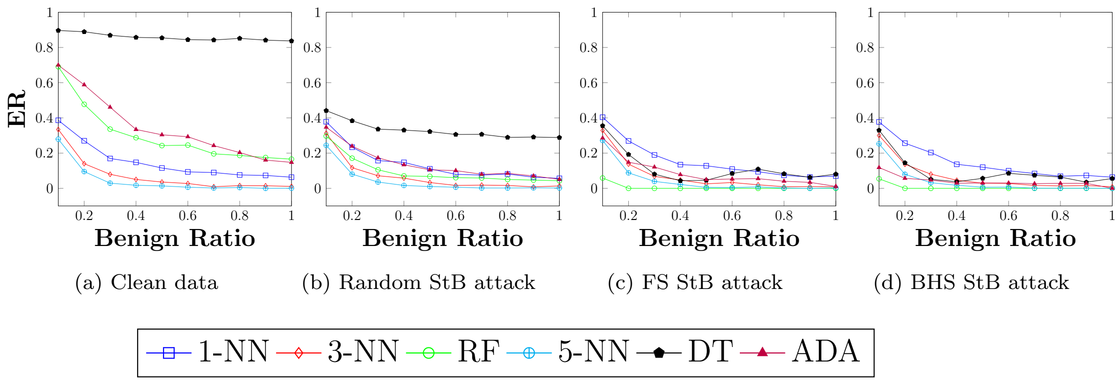


Figure 2: ER results: Distance-based models (1-NN, 3-NN, 5-NN) and tree-based models (RF, DT, Adaboost). The models were tested with clean data (a), and three types of manipulated data by evasion attacks (b-d) - Random, Full Statistical (FS), and Black Hole Statistical (BHS) StB attacks. The ER represents the evasion robustness of each model.

set increases, the classifier will tend to focus on the benign samples, in order to minimize the loss function, resulting in a monotonically decreasing ER.

The ER results are divided into four parts[2], depicted in Fig. 2. First, the ER of the clean data is presented as a baseline in Fig. 2a. Second, the ER of the Random StB attacks is described in Fig. 2b. Then, the ER of the Full Statistical StB attack is presented in Fig. 2c. At last, the ER of the Black Hole Statistical StB attack is depicted in Fig. 2d.

a) **Clean data:** The results of the ER using the clean data can be seen in Fig. 2a. The 1-NN (blue line) and 3-NN (red line) models' starting points[3] (blue and red lines) are lower than the RF (green line), by ∼30%. Their ending point reaches almost 0%. In other words, the ER of the RF (green line) decreases by ∼ 50%, and the KNNs by ∼ 40%. The starting point of the DT (black line) is an ER of ∼90%. Moreover, this model sustains a stable ER. The Adaboost model (purple line) shows an ER similar to the RF model (green line), 1-NN (blue line), and 3-NN (red line), with a slight superiority over the RF model. The 5-NN model (cyan line) is less

[2]An extended version of these results and results of the evasion attacks can be found in Github:https://github.com/harelber/MaMaDroid2.0.

[3]Note that the starting point is the result at 10% of the benign apps. The ending point is the result at 100% of the benign apps.

accurate than all models. It seems that the DT model (black line) is the best out of the six models.

b) **Random StB attack:** The results of the ER evaluation of the manipulated data using the Random StB attack are depicted in Fig. 2b. The distance-based models (1-NN, 3-NN, and 5-NN) (blue, red, and cyan lines) show an interesting behavior. They have a similar ER along the benign ratio values between the clean data and the manipulated data. In other words, the attack did not greatly affect them. However, the RF model (green line) suffers from an immense loss of ∼40% along the benign ratio values. In other words, the attack dramatically affected the RF model (green line). This is a change of course, because in the case of clean data, the RF model (green line) was more accurate than the 1-NN (blue line) and 3-NN (red line) models. The DT (black line) and Adaboost (purple line) models show a similar distribution as with the clean data. However, the starting and ending points of each model in both are lower than the correlative points in the clean data's results. The DT model (black line) is stable and demonstrates the most effective results against the Random StB attack. However, it has an ER of less than 50% at its starting point and falls below 40% at its ending point. The Adaboost model (purple line) is now as effective as the 1-NN model (blue line).

c) **Full Statistical StB attack:** Fig. 2c, presents the ER of the manipulated data using the Full Statistical StB attack. The distance-based models (blue, red, and cyan lines) remained the same as in previous cases. Their ER remained stable, despite their low rates. The ER of the RF model (green line) continued its decrease in ER, and in this case, to ∼ 0% for most benign ratio values. The ER of the DT (black line) and Adaboost (purple line) models also decreased by 20%-30%.

d) **Black Hole Statistical StB attack:** The ER results of the manipulated data using the Black Hole Statistical StB attack are depicted in Fig. 2d. In this case, the distance-based models (1-NN, 3-NN, and 5-NN, which are represented by the blue, red, and cyan lines) remained the same as before. The ER of the tree-based models (DT, RF, Adaboost which correlate to the black, green, and purple lines) continued to decrease, and in this case - less than 10% for most of the benign ratio values.

In conclusion, the ER results presented above show that the distance-based models (1-NN, 3-NN, and 5-NN which are represented by the blue, red, and cyan lines) show poor results with clean data and also against evasion attacks. In comparison, the tree-based models (DT, RF, and Adaboost, represented by the black, green, and purple lines) show high ER with clean data. Furthermore, tree-based models were proven to be more susceptible to attacks than distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines). An explanation for these findings is based on the node-split function of the tree models. The node split function is effective in means of identification of malicious activity, without any manipulation. The more features it processes, the more accurate it becomes. This is true for the DT model (black line), as the amount of benign data did not dramatically affect its ER. However, splitting the data between multiple weaker modules, as demonstrated in the Adaboost (purple line) and RF (green line) models, results in a decrease in the ER. Also, the tree-based models are based on features that are often manipulated by the attack. Therefore, the ER changes according to the attacks. However, as most features remain similar to the clean data, the distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines) show a similar ER between the baseline and the evasion attacks.

Another interesting insight involves the DT model. The results show it is the best model with clean data. However, with evasion attacks, its ER decreases dramatically. Other models, such as the RF or Adaboost models, that were less effective with clean data, are less affected by the evasion attacks. There are a couple of reasons for this phenomenon. First, DT is prone to overfitting. In other words, DT can become too closely tailored to the training data. This can make DT vulnerable to evasion attacks. Also, DTs are sensitive to the ordering and scaling of the feature vectors. Small changes in the input data can result in large changes in the output. This makes them more vulnerable to manipulations of the feature vectors. On the contrary, RF and AdaBoost use an ensemble approach, combining multiple DTs to make a final prediction. This can make them more robust to evasion attacks, as the combined predictions of multiple trees can help to counteract the effect of any individual tree. Furthermore, the use of randomized feature selection in RFs can help reduce the impact of small changes in input features.

Overall, while the DT model is effective with clean data, its decrease is dramatic compared to other models. However, as will be discussed in the next sections, its reliability is high, and with a little modification of the feature set, its decrease stops.

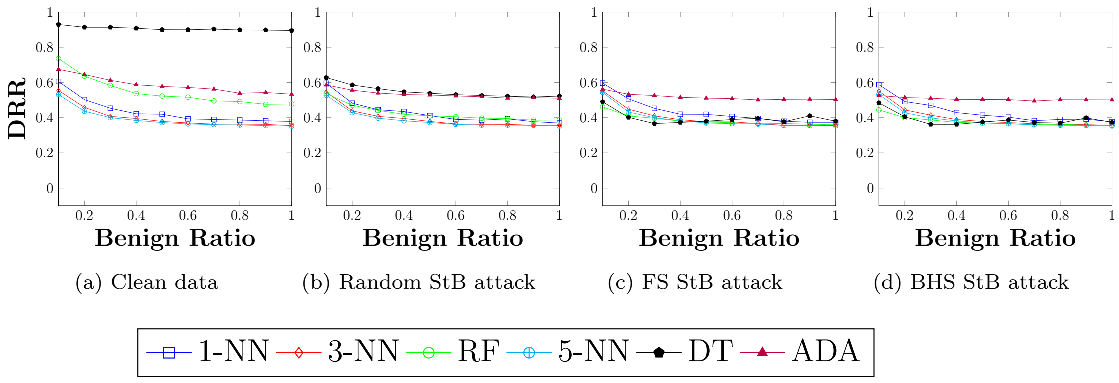


Figure 3: DRR results: Distance-based models (1-NN, 3-NN, 5-NN) and tree-based models (RF, DT, Adaboost). The models were tested with clean data (a), and three types of manipulated data (b-d) - Random, Full Statistical (FS), and Black Hole Statistical (BHS) StB attacks. The DRR represents the DRR results of each model.

5.4. Results - Defense Reciprocal Rate

The DRR measures the effectiveness of an evasion attack, where a higher DRR means a stronger classifier. In addition to the ER, this metric reflects the gap between the predictions on clean data and the predictions on evasion attacks. The ranks chosen for this experiment were 1 for malicious and 2 for benign, as the test data consisted of only malicious data. Therefore, a rank of 1 represents the correct label, and a rank of 2 represents a mistake. A review of the results of the models tested on clean data and the three evasion attacks is presented. Clean data results are depicted in Fig. 3a. The DRR of the evasion attacks is presented in Fig. 3b-3d.

1. **Clean data:** The results of the DRR evaluation of clean data are shown in Fig. 3a. As with the case of the ER, the tree-based models (DT, RF, and Adaboost represented by the black, green, and purple lines) outperform the distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines), considering the data without any evasion attack. Also, the DT model (black line) holds the highest DRR, which depicts the most resilient model. Its DRR is greater than 90% along the values of the benign ratio.
2. **Random StB attack:** In Fig. 3b, the DRR evaluation of the manipulated data is depicted using the Random StB attack. The DRR of the distance-based models remained the same as in the clean data. The

RF model's (green line) DRR suffered from a decrease of ∼20% compared to the clean data. The DT model's (black line) DRR decreased by ∼30% in comparison to the clean data. The decrease for the Adaboost model (purple line) was ∼20% compared to the clean data. The DT (black line), and Adaboost (purple line) models show close results along the benign ratio values.

3. **Full Statistical StB attack:** The DRR of the data manipulated by the Full Statistical StB attack is presented in Fig. 3c. The distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines) maintained a steady DRR as before. The Adaboost model (purple line) shows surprising results, similar to the DRR of the Random StB attack. In other words, the attacks had a similar effect on the DRR of the Adaboost model (purple line), in contrast to the DT model (black line), which decreased by an additional 10% compared to the Random StB attack. This might raise the question as to whether or not the DT model (black line) is less reliable than the Adaboost model (purple line). However, the next section (on the MR metric) will answer this question. The RF model (green line) suffers another decrease in DRR of ∼10% compared to the Random StB attack.
4. **Black Hole Statistical StB attack:** The DRR evaluation of the manipulated data by the Black Hole Statistical StB attack is depicted in Fig. 3d. The distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines), the DT (black line), and Adaboost (purple line) models present identical DRR to the Full Statistical StB attack. The starting point of the RF (green line) is lower than its correlative starting point in the Full Statistical StB attack.

To summarize, the distance-based models (1-NN, 3-NN, and 5-NN represented by the blue, red, and cyan lines) showed similar DRRs in each case explored, as in the ER metric. The evasion attacks had a similar effect on the DRR of the Adaboost (purple line) model, in contrast to the DT model, which decreased by 50% in the case of the Random StB attack, and by an additional 20% in the Full and Black Hole Statistical attacks. These findings show the need to investigate whether the Adaboost model (purple line) is more effective than the DT model (black line) and whether the Adaboost model (purple line) is more reliable. A similar phenomenon to Section 5.3 regarding the decrease of DT model ER with evasion attacks was found in this section as well. As mentioned before, the reliability of the DT model and

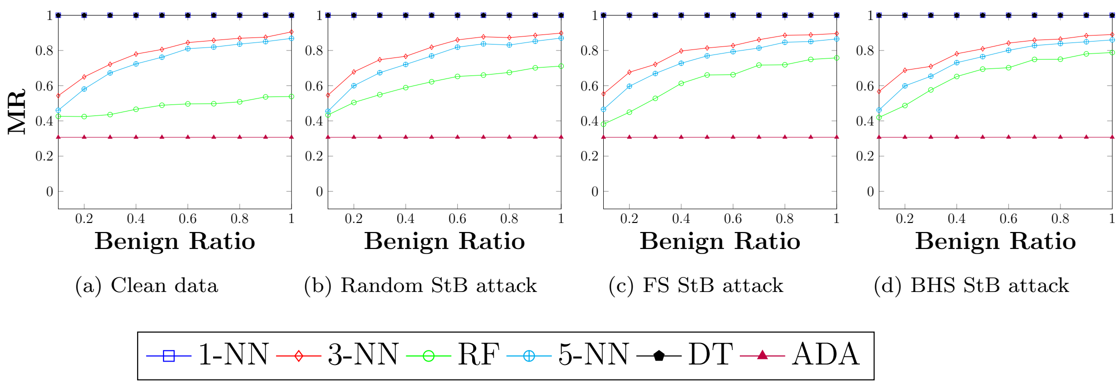


Figure 4: Model Reliability results: Distance-based models (1-NN, 3-NN, 5-NN) and tree-based models (RF, DT, Adaboost). The models were tested with clean data (a), and three types of manipulated data (b-d) - Random, Full Statistical (FS), and Black Hole Statistical (BHS) StB attacks. The Model Reliability of each model is presented.

its extension with a few more features cover this problem. The next section on the MR metric addresses these inquiries.

5.5. Results - Model Reliability

The reliability of each model was inspected to understand the confidence of each model in the predictions. High reliability means that the gap between the two classes is wide. A low value means that the classes are close to being indistinguishable. This does not influence the ER, since a classifier with a low ER can be "confident" in its predictions. Also, a "lucky" classifier can have a high ER, but the probabilities of each class may be close. The MR assessment was done on four cases - clean data and manipulated data by the three evasion attacks - to compare the changes the attacks have on the classifiers. The assessment is depicted in Fig. 4. As the MR results are similar for all four cases - clean data and manipulated data - they will be discussed without specific references to each of the subfigures.

It can be seen that the 1NN model (blue line) is reliable, as its MR is 1 on the benign ratio values, for each one of the cases. The MR of the DT model (black line) is also found to be very high, as it receives the value of 1 along the benign ratio values for each one of the cases. The MR of the Adaboost model (purple line) was found to be a constant value of 0.3 along the benign ratio values, for each one of the cases. The MRs of the other models, 3NN (red line), 5NN (cyan line), and RF (green line), increased throughout the benign ratio values. The increased reliability of the 3NN (red line) and 5NN (cyan line) models is similar in the four cases. Their reliability was different from the 1-NN (blue line) because finding the closest neighbor and deciding by its labels is binary in the case of 1-NN (blue line). However, 3-NN (red line) or 5-NN (cyan line) allows more flexibility in terms of the model's confidence, based on the majority of the neighbors. Therefore, it is not a binary decision. The RF model (green line) was found to be more reliable in the cases of evasion attacks than the clean data. In other words, it was more "confident" in its labels of evasion attacks than the clean data.

Both the 1NN (blue line) and DT (black line) models show a constant high value of model reliability, with a value of 1. In contrast, the Adaboost model (purple line) provided a constant value of 0.3. As these phenomena were surprising, a close examination of the probabilities of each class for these models was done. The findings show that for the 1NN (blue line) and DT (black line) models, the probabilities were binary for each sample, without any regard for the benign ratio of the apps. For the Adaboost model (purple line), both probabilities on each sample were close to 0.5. Therefore, the entropy was high in the results of the Adaboost model (purple line), and low in the results of the 1NN (blue line) and DT (black line) models, which outcomes in correlative complement MR rates. Therefore, although the Adaboost model (purple line) had better DRR than the DT (black line) model against the evasion attacks, its MR was found to be far worse. The DT model (black line) was found to be the most promising of the six models, based on the combined insights of the ER, DRR, and MR metrics.

In summary, four important insights can be derived from the combination of the results of ER, DRR, and MR:

1. The most promising model tested on clean data and evasion attacks is DT. However, this model is susceptible to evasion attacks.
2. The distance-based models were not greatly affected by the evasion attacks. However, their ER was low at baseline and also in any evasion attack.
3. The ER and DRR results proved that the Random StB attack is less effective than the Full Statistical and the Black Hole Statistical attacks. For example, for the DT model, the DRR of the clean data was 80%-90%. On the Random StB attack, the DRR was 50%-60%. However, the DRR decreased to 40% and less in the cases of the Statistical StB attack and the Black Hole Statistical attack. Also, the ER results sup-

port this conclusion. This insight is pretty intuitive as a random attack does not specifically target the data upon which the model is built. In comparison, the Full Statistical StB attack requires the training data to generate the attack. However, it was important to establish this insight by viewing the results. In addition, the Black Hole Statistical StB attack achieved results similar to the Full Statistical StB attack but needed less information, only the test data. Therefore, it can be considered the best attack of the three.

4. A lower DRR does not necessarily mean a less effective model. As proven, the DT model had a lower DRR in the Full and Black Hole Statistical StB attacks than the Adaboost model, which had a similar DRR for each attack. However, the ER and MR revealed the full picture, in which the Adaboost model is less reliable and robust than the DT model.

A more robust version of MaMaDroid can be suggested, due to the effect of the ratio of benign apps on the detection of malicious apps and evasion attacks. This version should use the most effective model that was found - DT. However, using the existing DT model is insufficient, as it is susceptible to StB attacks. The next section will demonstrate a suitable solution for the weaknesses in the model.

6. MaMaDroid2.0

As suggested in the previous section, all models were found to be vulnerable to StB attacks. The distance-based models were found to be robust against evasion attacks, but their original ER was low. A stronger version of MaMaDroid is suggested as a result of these findings. In light of the fact that even the strongest model, i.e., DT, was evaded by the StB attacks, there is still room for improvement.

Different approaches can be considered to upgrade MaMaDroid. For example, APIGraph (Zhang et al., 2020) analyzes relations between API calls (and permissions) based on the different versions of the Android OS, which is a more advanced detection system based on the APIs CFG. However, APIGraph is constructed by security experts to analyze the basic relations for a couple of days and then extended by the detection system if there were not many changes between the Android OS versions. Future human interactions may be needed if the changes between the versions of Android OS.

MaMaDroid2.0 is constructed without manual analysis. Also, the analysis of APIGraph is complex, in comparison to the work of the suggested MaMaDroid2.0 detection system. Therefore, in the case of low resources and when full automation is searched, MaMaDroid2.0 is a better option for a classifier.

An extension of the feature set might help mitigate the attacks, thus creating a hybrid approach. One of the influential works on Android malware detection is Drebin (Arp et al., 2014). This work laid the foundations for ML Android malware detection which is based on static analysis. This system is based on eight feature sets. One of the feature sets, i.e., the required permissions set, was selected for the extension of MaMadroid. Permissions have been a great candidate for a feature set of Android malware detection systems over the years, either as a complete or partial set of features for a detection machine (i.e., (Arora et al., 2019; Aswini and Vinod, 2014; Ding et al., 2021; Enck et al., 2009; Li et al., 2018b; Pierazzi et al., 2020; Sanz et al., 2013; Talha et al., 2015)). The exploration of different aspects of the APK sample covers more ground in terms of malware detection. Permission requests govern the accessibility to resources that the developer requires. The features of MaMaDroid1.0 depict the flow of the application and, therefore, may lead to understanding how the application operates. In each one, an indication of malicious activity can be achieved. For example, an application that steals information from the user needs a risky permission request P. On the one hand, a set of attacks called MB attacks (Berger et al., 2021) conceal the appearance of P. However, these attacks do not alter the flow of the application. On the other hand, the StB attacks change the flow of the application but do not change the permission requests. Consequently, the combination of both feature sets describes two different aspects of the application. This extended feature set creates the potential for a more robust classifier that analyzes different aspects of malicious activity. Therefore, the permissions set was chosen to enhance MaMaDroid and is hereinafter termed *permission set*.

Different approaches can be followed for the permission set. First, all permissions in the dataset can be used to generate the *permission set*. However, the size of the original MaMaDroid feature set is 121. The applications in this study's dataset use more than 500 permissions in total. Using the full permission set cannot be considered for the extension of MaMaDroid. Another option is to use the most popular X permissions in the dataset. However, the question of how to choose a justifiable X is very hard and should be considered a topic for an additional study. Therefore, the permissions that were chosen for the *permission set* are the permissions that were gathered in permission groups by the Android Open Source Project (AOSP) (Project, 2006). This set comprises 50 popular permission requests, which were gathered by AOSP and categorized into 13 categories. Examining both the groups of permissions and the permissions themselves as the *permission set* led to the understanding that the groups are too abstract to use[4] MaMaDroid 2.0, the enhanced version of MaMaDroid, follows the following steps:

1. Run MaMaDroid's feature extraction on the app to get the feature set, as suggested in (Onwuzurike et al., 2017).
2. Extract the permission set from each app by exploring its Manifest file. Choose only the permissions of the *permission set*.
3. Merge the two feature sets.
4. Run the model on the merged feature set and get a classification of the app.

In light of the fact that MaMaDroid2.0 incorporates the permission set as part of its feature set (as in Drebin (Arp et al., 2014)), attacks against Drebin (Berger et al., 2021) were also tested, to assess whether the updated detection system is also robust against these attacks. The attacks against Drebin termed manifest-based (MB) attacks, conceal the appearance of permission requests. MB attacks were proven (in their original paper) to decrease the detection rate of Drebin. The attacker model may be different in MB attacks. For example, in the MB1 attack, the attacker had access to the weights of the classifier and its labels. The StB attacks did not include a similar attacker model. In addition, the MB and StB attacks were different in their nature. The StB attacks changed the content of the Smali code files. The MB attacks manipulated the manifest file. Therefore, these two types of attacks were excellent candidates to test the effectiveness of MaMaDroid2.0.

The same dataset and training/test splits used in the previous experiments were used in these experiments. Each experiment is described by the test data and feature set since the training data remain the same. The settings of each experiment of MaMaDroid2.0 are described in Table 2. In light

[4]The results of using both the groups and the permissions themselves can be viewed in this study's repository (Berger, 2022).. Therefore, 50 permission requests were used as the *permission set*.

Exp	Clean	Manipulated	StB	MB (Berger et al., 2021)	Feature Set
1	✓	✓	✓		Base
2	✓	✓	✓		Ext
3	✓	✓	✓		Perm
4	✓	✓		✓	Ext
5	✓	✓		✓	Perm

Table 2: MaMaDroid2.0 experiments settings. Each experiment is described by the data it incorporates - clean and manipulated data, the evasion attack that created the manipulated data (StB/MB (Berger et al., 2021)). Also, the feature set that was used is presented - the basic MaMaDroid1.0 feature set (Base), the extended feature set of MaMaDroid2.0 (Ext), and the permission set (Perm).

of the high volume of attacks and metrics involved in each set of attacks (the StB and the MB attacks), the results of only one of the attacks are presented in the next section, analyzed by the three metrics: ER, DRR, and MR. The rest of the results are presented in Appendix Appendix A.

6.1. Results - Evasion Robustness (ER)

The first evaluation is the ER metric, as in Section 5. Like before, as the ratio of benign samples in the training set increased, the classifier tended to focus on the benign samples, in order to minimize the loss function, thus resulting in a monotonically decreasing classifier. The results of the ER are divided into three parts as depicted in Fig. 5. First, the ER of the clean data is presented as a baseline in Fig. 5a. Second, the ER of the Full Statistical StB attack is described in Fig. 5b. Then, the ER of the MB2 attack is shown in Fig. 5c.

The results in Fig. 5a show that the ER of the model using the base features (Base), with clean data was $\sim$ 0.9 at each point. When adding the permission set (i.e., experiment 2) to the feature set, namely the model with the enhanced feature set (Ext), the ER at each point decreased until the ending point was 0.8. The model that analyzed only the permission set (Perm) decreased even more, to 0.6 at the ending point. The Perm classifier is less effective than the original MaMaDroid and MaMaDroid2.0. The results demonstrate that permission requests themselves are less effective as a feature set.

Fig. 5b shows that the Perm and Ext models have similar results in the detection of the Full Statistical StB attack. In some points, one of the models is slightly better, and in others, the second model is better. However, the Base model, as was proven in Section 5, trails the other two models. The first two models start at an ER of 0.8 and decrease to an ER ~ 0.6 at the ending point. In comparison, the Base model starts at an ER of 0.4 and decreases to an ER of ~ 0. The difference between this model and the Ext and Perm models is ~ 0.6. The Perm classifier was not affected because the StB attack changed the transition probabilities that this classifier did not analyze. The results show that the addition of the *permission set* to the feature set enhances the Base classifier (which results in the Ext classifier) against this StB attack.

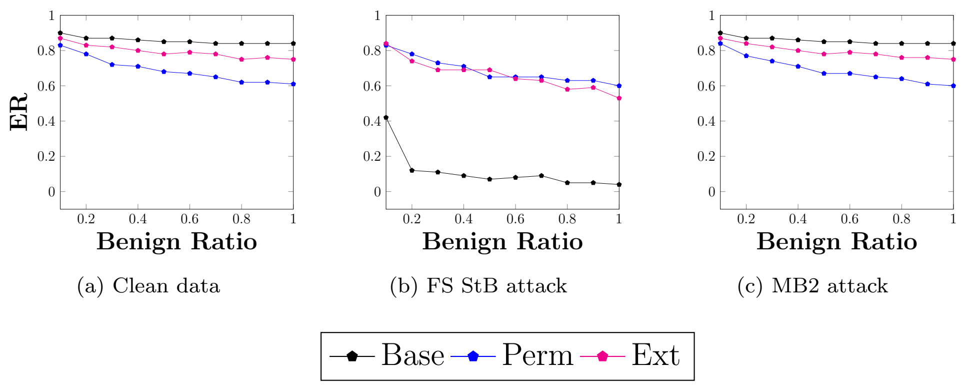
Fig. 5c depicts the results of the MB2 attack. Against this attack, the models demonstrated less distinguishable achievements than the Full Statistical StB attack. However, the Perm model was the less effective model against this attack. The starting point for the three models was an ER of ~ 0.9. The Perm model ended with an ER of 0.6. The Ext model was finished with an ER of ~ 0.8. The ER of the Base model was slightly better than that of the Ext model, by 0.05. The Base model was not affected by this attack, as the MB attack changed the permission requests, which are not analyzed by the Base classifier. The other two models were affected, whereas the Ext model was more advantageous.

These results are interesting, as both the Base and Perm models were weak against the attacks that target their initiative - the StB attacks against MaMaDroid, and the MB attacks against the Perm classifier. In comparison, the extended version, MaMaDroid2.0, is a classifier that presents a meaningful defense against both attacks and makes a reasonable alternative to both classifiers.

6.2. Results - Defense Reciprocal Rank (DRR) and Model Reliability (MR)

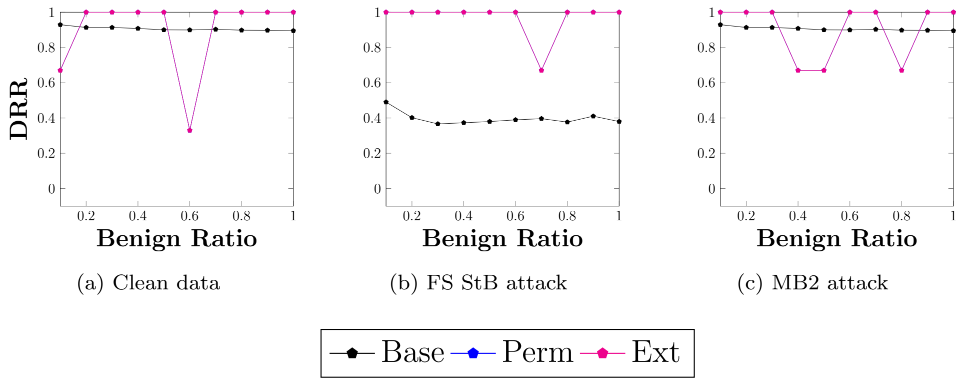
Fig. 6 presents the results of the DRR evaluation of the Base, Ext, and Perm models. This metric reflects the effect of evasion attacks on the classifiers. The lower DRR indicates that the attack gained some effect on the classifier. Even if the attack gained a high ER, it could be improved more easily to become a successful attack. The clean data results are shown in Fig. 6a. The DRR of the Full Statistical StB attack is presented in Fig. 6b. The DRR of the MB2 attack is depicted in Fig. 6c.

Fig. 6a shows that on clean data, the Base classifier had a DRR rate of ~ 0.9. The Perm and Ext classifiers had a DRR rate of 1 in most points. Some of the points have a DRR of lower DRRs of 0.7 and 0.4. Thus, the



(a) Clean data (b) FS StB attack (c) MB2 attack

Figure 5: ER results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with clean data (a), and two types of manipulated data (b-c) - Full Statistical (FS) StB and MB2 attacks. The ER represents the evasion robustness of each model.



(a) Clean data (b) FS StB attack (c) MB2 attack

Figure 6: DRR results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with clean data (a), and two types of manipulated data (b-c) - Full Statistical (FS) StB and MB2 attacks. The DRR represents the Defense Reciprocal Rate of each model.

Base classifier was more confident than the other two classifiers at some of the points. Still, in most of the points, the two other classifiers were more confident. Fig. 6b shows a different picture. The DRR of the Base classifier was between 0.4-0.5. The DRR of the other two classifiers was between 0.75-1. In other words, this case shows that the evasion attack that manipulated the MaMaDroid features dramatically affected the DRR of the Base classifier.

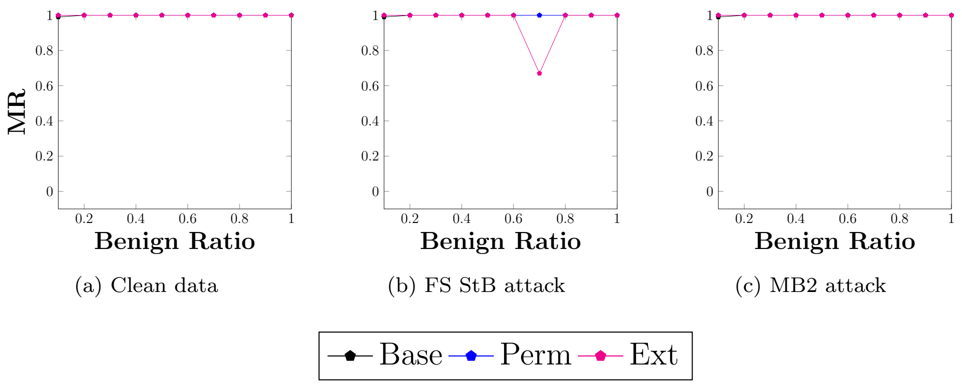


Figure 7: MR results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with clean data (a), and two types of manipulated data (b-c) - Full Statistical (FS) StB and MB2 attacks. The MR represents the Model Reliability of each model.

Fig. 6c presents the DRR of the MB2 attack. In this case, the Perm and Ext classifiers were more confident in their predictions. The anomalies were less dramatic, from a DRR rate of 1 to more than 0.6. The DRR of the Base classifier was equal to the DRR of clean data, as the MB2 attack did not change the feature set of the original MaMaDroid, which the Base classifier learns. Fig. 7 presents the results of the MR evaluation of the Base, Ext, and Perm models. This metric reflects the disorder in each classifier when confronting clean data (Fig. 7 a), and evasion attacks (Fig. 7b, Fig. 7c). The figures show that for each classifier and in most points, the MR was 1. In other words, there was a neglect rate of disorder in each of these classifiers. This is straightforward, as the inner model of each of these classifiers was DT, where in Section 5 the MR rate was 1 due to the binary probability pairs in most cases.

In conclusion, the enhanced MaMaDroid2.0 feature set (Ext classifier) was shown to create a fair alternative model for both Base and Perm classifiers. This model was shown to identify attacks against MaMaDroid and Drebin alike. The ER rate of the Ext classifier was found to be higher than that of the Base classifier on the attack that targets the MaMadroid feature set, with more than 40% on each point of the interval. A similar result was found for the Perm classifier and the attack that targets the permission set, where the ER of the extended version is higher than the ER of the Perm classifier in each point of the interval. The DRR and MR metrics showed a similar behavior between the Perm and Ext classifiers. The DRR of the Base classifier was slightly lower, by at least 10% on each point of the interval. The models of MaMaDroid1.0 (Section 5) failed to detect the StB attacks, decreasing to an ER of 0. In comparison, MaMaDroid2.0 achieved an ER of more than 0.6 at each point in the same case. Also, MaMaDroid2.0 achieved an improvement in the ER rate in comparison to the Perm classifier when facing the MB attacks (for example, 0.1 in the case of MB2). Thus, the results show that MaMaDroid2.0 took a meaningful step toward reducing StB attacks and MB attacks.

7. Discussion and Conclusions

While the field of Android malware detection is of great interest and many different aspects have been researched, there is one aspect that has not received sufficient attention. In this work, the neglected aspect, namely the ratio of benign instances in the model's training set, is specifically analyzed. A previous work (Pendlebury et al., 2019) analyzed the effect of changing the benign ratio in the training data. However, Pendlebury et al. (Pendlebury et al., 2019) did not analyze the effects of different benign ratios in the presence of evasion attacks, as done in this work. The findings show that the effect of this parameter on the model's performance is significant in terms of its ability to detect unseen malicious instances (Fig. 2(a)), its confidence (Fig. 3(a)), and its reliability (Fig. 4(a)). Note that these effects occur even when there is no adversary manipulation on the malicious instances (i.e., no attack).

In order to evaluate these effects on manipulated data, three novel evasion attacks were suggested. While the attacks share a common ground, which is their general flow, they differ in terms of the attacker model (i.e., the knowledge the attacker holds).

The empirical evaluation of this work (in Fig. 2) shows that all three attacks decreased the ER of the tree-based models. The distance-based models proved to be strong against the attacks, but their ER was already low on the clean data (and remained low on the manipulated data using the attacks). The DT model was found to be the best model in terms of ER. Although its decrease in ER was the most disastrous among the models, its ER with original applications was the best, and also with the Random StB attack. It was also found to be very reliable (i.e., its MR value was 1.0), even in the most challenging ratio of more than 90% of the dataset being benign apps. The DRR metric (in Fig. 3) shows that most of the models (aside from the Adaboost model), lose their confidence in the presence of evasion attacks. However, the MR metric (in Fig. 4) completes the view so that although the confidence may decrease, the DT model is still the best and the most reliable model of the six. Consequently, improvement of the original MaMaDroid feature set and creation of MaMaDroid2.0 was suggested. This improved version proved to be robust against evasion attacks against the original MaMaDroid and against other evasion attacks, as shown in Fig. 5. In comparison to the original MaMaDroid model and a permission-based model, it was found to be the best model in means of ER (Fig. 5), DRR (Fig. 6) and MR (Fig. 7).

The new version of MaMaDroid is an excellent example of using the CFG to identify behaviors of benign and malicious APKs. In general, a CFG of software tries to organize the order of commands the software runs. A malware detection system that is based on a CFG attempts to find the differences between the order of commands in malicious and benign apps to address the task of the classification of new samples. As the experiments in this study showed, a shift in the order of commands can cause miss-classification. For example, the Black Hole StB variant demonstrates the effect of moving some of the commands from the malicious order of commands to an unknown order of commands. In other words, the order turns to chaos. The different variants modify the order of commands in an app, which results in the alteration of its CFG. When the modified CFG is inspected by the detection machine, the detection machine may miss-classify it as benign.

This course of action can be generalized to other domains as well. Therefore, the CFG-based detection systems now confront the following challenge: Are they secure enough against the ominous chaos that will occur? It seems that an extension of the feature set or a hybrid approach may aid in dealing with this issue. The combination of different features seems a promising step towards more accurate solutions against multiple types of malware that exploit different vulnerabilities. These inquiries should be addressed in future work.

8. Acknowledgment

This work was supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber Directorate of the Prime Minister's

Office. The authors would like to thank Dr. Lucky Onwuzurike for his technical aid in regard to MaMaDroid.

References

Aafer, Y., Du, W., Yin, H., 2013. Droidapiminer: Mining api-level features for robust malware detection in android, in: International Conference on Security and Privacy in Communication Systems, Springer. pp. 86–103.

Ali-Gombe, A., Ahmed, I., Richard III, G.G., Roussev, V., 2016. Aspectdroid: Android app analysis system, in: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp. 145–147.

Allen, F.E., 1970. Control flow analysis. SIGPLAN Not. 5, 1–19. URL: `https://doi.org/10.1145/390013.808479`, doi:`10.1145/390013.808479`.

Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. Androzoo: Collecting millions of android apps for the research community, in: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories, IEEE. pp. 468–471.

Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.J., Srivastava, M., Chang, K.W., 2018. Generating natural language adversarial examples. arXiv preprint arXiv:1804.07998 .

Apruzzese, G., Colajanni, M., 2018. Evading botnet detectors based on flows and random forest with adversarial samples, in: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), IEEE. pp. 1–8.

Arora, A., Peddoju, S.K., Conti, M., 2019. Permpair: Android malware detection using permission pairs. IEEE Transactions on Information Forensics and Security .

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket., in: Ndss, pp. 23–26.

Aswini, A., Vinod, P., 2014. Droid permission miner: Mining prominent permissions for android malware analysis, in: The Fifth International Conference on the Applications of Digital Information and Web Technologies, IEEE. pp. 81–86.

Athalye, A., Carlini, N., Wagner, D., 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, in: International Conference on Machine Learning, pp. 274–283.

Aydogan, E., Sen, S., 2015. Automatic generation of mobile malwares using genetic programming, in: European Conference on the Applications of Evolutionary Computation, Springer. pp. 745–756.

Backes, M., Bugiel, S., Derr, E., 2016. Reliable third-party library detection in android and its security applications, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 356–367.

Bekkers, L., van't Hoff-de Goede, S., Misana-ter Huurne, E., van Houten, Y., Spithoven, R., Leukfeldt, E.R., 2023. Protecting your business against ransomware attacks? explaining the motivations of entrepreneurs to take future protective measures against cybercrimes using an extended protection motivation theory model. Computers & Security 127, 103099.

Berger, H., 2022. Mamadroid2.0 code repository. https://github.com/harelber/MaMaDroid2.0.

Berger, H., Hajaj, C., Dvir, A., 2020. Evasion is not enough: A case study of android malware, in: International Symposium on Cyber Security Cryptography and Machine Learning, Springer. pp. 167–174.

Berger, H., Hajaj, C., Mariconti, E., Dvir, A., 2021. Crystal ball: From innovative attacks to attack effectiveness classifier. IEEE Access .

Berger, H., Hajaj, D.C., Dvir, D.A., 2022. Problem-space evasion attacks in the android os: a survey. URL: `https://arxiv.org/abs/2205.14576`, doi:`10.48550/ARXIV.2205.14576`.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Srndic, N., Laskov, P., Giacinto, G., Roli, F., 2013. Evasion attacks against machine learning at test time, in: European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 387–402.

Brama, H., Dery, L., Grinshpoun, T., 2022. Evaluation of neural networks defenses and attacks using ndcg and reciprocal rank metrics. arXiv:2201.05071.

Brooks, S., Gelman, A., Jones, G., Meng, X.L., 2011. Handbook of markov chain monte carlo. CRC press.

Cai, H., Jenkins, J., 2018. Towards sustainable android malware detection, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ACM. pp. 350–351.

Cara, F., Scalas, M., Giacinto, G., Maiorca, D., 2020. On the feasibility of adversarial sample creation using the android system api. Information 11, 433.

Carlini, N., Wagner, D., 2017. Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy, IEEE. pp. 39–57.

Chen, L., Hou, S., Ye, Y., Chen, L., 2017. An adversarial machine learning model against android malware evasion attacks, in: Asia-Pacific Web and Web-Age Information Management Joint Conference on Web and Big Data, Springer. pp. 43–55.

Chen, L., Hou, S., Ye, Y., Xu, S., 2018a. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks, in: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, IEEE. pp. 782–789.

Chen, S., Xue, M., Tang, Z., Xu, L., Zhu, H., 2016. Stormdroid: A streaminglized machine learning-based system for detecting android malware, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ACM. pp. 377–388.

Chen, T., Mao, Q., Yang, Y., Lv, M., Zhu, J., 2018b. Tinydroid: a lightweight and efficient model for android malware detection and classification. Mobile Information Systems 2018.

Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K., 2019. Android hiv: A study of repackaging malware for evading machine-learning detection. IEEE Transactions on Information Forensics and Security .

ChenJunHero, 2018. droidapiminer code. github. `https://github.com/ChenJunHero/DroidAPIMiner`.

Choo, K.K.R., 2011. The cyber threat landscape: Challenges and future research directions. Computers & security 30, 719–731.

Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E., 2005. Semantics-aware malware detection, in: 2005 IEEE symposium on security and privacy (S&P'05), IEEE. pp. 32–46.

Damashek, M., 1995. Gauging similarity with n-grams: Language-independent categorization of text. Science 267, 843–848.

Dang, H., Huang, Y., Chang, E.C., 2017. Evading classifiers by morphing in the dark, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 119–133.

Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F., 2017. Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing .

Ding, C., Luktarhan, N., Lu, B., Zhang, W., 2021. A hybrid analysis-based approach to android malware family classification. Entropy 23, 1009.

Dini, G., Martinelli, F., Saracino, A., Sgandurra, D., 2012. Madam: a multi-level anomaly detector for android malware, in: International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, Springer. pp. 240–253.

Elish, K., Elish, M., Almohri, H., 2022. Lightweight, effective detection and characterization of mobile malware families. IEEE Transactions on Computers .

Enck, W., Ongtang, M., McDaniel, P., 2009. On lightweight mobile phone application certification, in: Proceedings of the 16th ACM conference on Computer and Communications Security, ACM. pp. 235–245.

Frenklach, T., Cohen, D., Shabtai, A., Puzis, R., 2021. Android malware detection via an app similarity graph. Computers & Security 109, 102386.

Geyer, C.J., 1992. Practical markov chain monte carlo. Statistical science , 473–483.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 .

Google, 2008. GooglePlay app market. GooglePlay website. https://play.google.com/store/apps/.

Google, 2017. Android security 2017 year in review. https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection, in: European Symposium on Research in Computer Security, Springer. pp. 62–79.

Hu, W., Tan, Y., 2017. Generating adversarial malware examples for black-box attacks based on gan. arXiv preprint arXiv:1702.05983 .

Huynh, N.A., Ng, W.K., Ariyapala, K., 2017. A new adaptive learning algorithm and its application to online malware detection, in: International Conference on Discovery Science, Springer. pp. 18–32.

Ikram, M., Beaume, P., Kaafar, M.A., 2019. Dadidroid: An obfuscation resilient tool for detecting android malware via weighted directed call graph modelling. arXiv preprint arXiv:1905.09136 .

Kabakus, A.T., 2022. Droidmalwaredetector: A novel android malware detection framework based on convolutional neural network. Expert Systems with Applications 206, 117833.

Kang, B., Kang, B., Kim, J., Im, E.G., 2013. Android malware classification method: Dalvik bytecode frequency analysis, in: Proceedings of the 2013 Research in Adaptive and Convergent Systems, pp. 349–350.

Kuppa, A., Grzonkowski, S., Asghar, M.R., Le-Khac, N.A., 2019. Black box attacks on deep anomaly detectors, in: Proceedings of the 14th International Conference on Availability, Reliability and Security, ACM. p. 21.

Li, B., Vorobeychik, Y., Chen, X., 2016. A general retraining framework for scalable adversarial classification. arXiv preprint arXiv:1604.02606 .

Li, C., Mills, K., Niu, D., Zhu, R., Zhang, H., Kinawi, H., 2019. Android malware detection based on factorization machine. IEEE Access 7, 184008–184019.

Li, D., Li, Q., 2020. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. IEEE Transactions on Information Forensics and Security 15, 3886–3900.

Li, D., Li, Q., 2020. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. IEEE Transactions on Information Forensics and Security 15, 3886–3900. doi:`10.1109/TIFS.2020.3003571`.

Li, J., Ji, S., Du, T., Li, B., Wang, T., 2018a. Textbugger: Generating adversarial text against real-world applications. arXiv preprint arXiv:1812.05271 .

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., Ye, H., 2018b. Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics 14, 3216–3225.

Li, Y., Yang, Z., Guo, Y., Chen, X., 2017. Droidbot: a lightweight ui-guided test input generator for android, in: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), IEEE. pp. 23–26.

Lindorfer, M., Neugschwandtner, M., Platzer, C., 2015. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis, in: 2015 IEEE 39th annual computer software and applications conference, IEEE. pp. 422–433.

Lindorfer, M., Volanis, S., Sisto, A., Neugschwandtner, M., Athanasopoulos, E., Maggi, F., Platzer, C., Zanero, S., Ioannidis, S., 2014. Andradar: fast discovery of android applications in alternative markets, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer. pp. 51–71.

Maggi, F., Frossi, A., Zanero, S., Stringhini, G., Stone-Gross, B., Kruegel, C., Vigna, G., 2013. Two years of short urls internet measurement: security threats and countermeasures, in: proceedings of the 22nd international conference on World Wide Web, pp. 861–872.

Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G., 2015. Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security 51, 16–31.

Maiorca, D., Mercaldo, F., Giacinto, G., Visaggio, C.A., Martinelli, F., 2017. R-packdroid: Api package-based characterization and detection of mobile ransomware, in: Proceedings of the symposium on applied computing, pp. 1718–1723.

Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, E., Ross, G., Stringhini, G., 2018. MaMaDroid implementation. bitbucket. `https://bitbucket.org/gianluca_students/mamadroid_code/src/master/`.

Marjoram, P., Molitor, J., Plagnol, V., Tavaré, S., 2003. Markov chain monte carlo without likelihoods. Proceedings of the National Academy of Sciences 100, 15324–15328.

Martín, A., Lara-Cabrera, R., Camacho, D., 2019. Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. Information Fusion 52, 128–142.

Meng, G., Xue, Y., Mahinthan, C., Narayanan, A., Liu, Y., Zhang, J., Chen, T., 2016. Mystique: Evolving android malware for auditing anti-malware tools, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ACM. pp. 365–376.

Ming, J., Xin, Z., Lan, P., Wu, D., Liu, P., Mao, B., 2015. Replacement attacks: automatically impeding behavior-based malware specifications, in: International Conference on Applied Cryptography and Network Security, Springer. pp. 497–517.

Nguyen, T.T., Luong, A.V., Dang, M.T., Liew, A.W.C., McCall, J., 2020. Ensemble selection based on classifier prediction confidence. Pattern Recognition 100, 107104.

Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G., 2017. Mamadroid: Detecting android malware by building markov chains of behavioral models, in: Annual Symposium on Network and Distributed System Security.

Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G., 2019. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). ACM Transactions on Privacy and Security 22, 14.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L., 2019. \${\$tesseract\$}\$: Eliminating experimental bias in malware classification across space and time, in: 28th USENIX Security Symposium, pp. 729–746.

Piao, Y., Jung, J.H., Yi, J.H., 2016. Server-based code obfuscation scheme for apk tamper detection. Security and Communication Networks 9, 457–467.

Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L., 2019. Intriguing properties of adversarial ml attacks in the problem space. arXiv preprint arXiv:1911.02142 .

Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L., 2020. Intriguing properties of adversarial ml attacks in the problem space, in: 2020 IEEE Symposium on Security and Privacy, IEEE Computer Society. pp. 1308–1325. URL: `https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00073`, doi:`10.1109/SP40000.2020.00073`.

Project, A.O.S., 2006. Android open source project - github repository. https://github.com/aosp-mirror/platform_frameworks_base.

Rahbarinia, B., Balduzzi, M., Perdisci, R., 2017. Exploring the long tail of (malicious) software downloads, in: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE. pp. 391–402.

Rastogi, V., Chen, Y., Jiang, X., 2013. Droidchameleon: evaluating android anti-malware against transformation attacks, in: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ACM. pp. 329–334.

Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y., 2018. Generic black-box end-to-end attack against state of the art api call based malware classifiers, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer. pp. 490–510.

Salem, A., Paulus, F.F., Pretschner, A., 2018. Repackman: A tool for automatic repackaging of android apps, in: Proceedings of the 1st International Workshop on Advances in Mobile App Analysis, ACM. pp. 25–28.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G., 2013. Puma: Permission usage to detect malware in android, in: International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, Springer. pp. 289–298.

Saracino, A., Sgandurra, D., Dini, G., Martinelli, F., 2016. Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing 15, 83–97.

Shabtai, A., Kanonov, U., Elovici, Y., 2010. Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method. Journal of Systems and Software 83, 1524–1537.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y., 2012. "andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems 38, 161–190.

Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C., 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Information Security Technical Report 14, 16–29.

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

Shahpasand, M., Hamey, L., Vatsalan, D., Xue, M., 2019. Adversarial attacks on mobile malware detection, in: 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile, IEEE. pp. 17–20.

Shao, R., Lan, X., Li, J., Yuen, P.C., 2019. Multi-adversarial discriminative deep domain generalization for face presentation attack detection, in:

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 10023–10031.

Shin, S.Y., Kang, Y.W., Kim, Y.G., 2020. Android-gan: Defending against android pattern attacks using multi-modal generative network as anomaly detector. Expert Systems with Applications 141, 112964.

Spooren, J., Preuveneers, D., Desmet, L., Janssen, P., Joosen, W., 2019. On the use of dgas in malware: an everlasting competition of detection and evasion. ACM SIGAPP Applied Computing Review 19, 31–43.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., Pan, Y., 2016. Sigpid: significant permission identification for android malware detection, in: 2016 11th International Conference on Malicious and Unwanted Software, IEEE. pp. 1–8.

Sun, M., Tan, G., 2014. Nativeguard: Protecting android applications from third-party native libraries, in: Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks, ACM. pp. 165–176.

Talha, K.A., Alper, D.I., Aydin, C., 2015. Apk auditor: Permission-based android malware detection system. Digital Investigation 13, 1–14.

Tong, L., Li, B., Hajaj, C., Xiao, C., Zhang, N., Vorobeychik, Y., 2019. Improving robustness of ml classifiers against realizable evasion attacks using conserved features., in: USENIX Security Symposium, pp. 285–302.

Treadwell, S., Zhou, M., 2009. A heuristic approach for detection of obfuscated malware, in: 2009 IEEE International Conference on Intelligence and Security Informatics, IEEE. pp. 291–299.

Venugopal, D., Hu, G., 2008. Efficient signature based malware detection on mobile devices. Mobile Information Systems 4, 33–49.

Wang, H., Guo, Y., Tang, Z., Bai, G., Chen, X., 2015. Reevaluating android permission gaps with static and dynamic analysis, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE. pp. 1–6.

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X., 2014. Exploring permission-induced risk in android applications for malicious application detection. IEEE Transactions on Information Forensics and Security 9, 1869–1882.

Wang, X., Li, C., 2021. Android malware detection through machine learning on kernel task structures. Neurocomputing 435, 126–150.

WIRE, B., 2021. A year of lockdown sees a surge in mobile malware targeting banking, billing and covid-19 vaccines. https://www.businesswire.com/news/home/20210628005004/en/A-Year-of-Lockdown-Sees-a-Surge-in-Mobile-Malware-Targeting-Banking-Billing-and-COVID-19-Vaccines.

Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P., 2012. Droidmat: Android malware detection through manifest and api calls tracing, in: 2012 Seventh Asia Joint Conference on Information Security, IEEE. pp. 62–69.

Xu, Q., Tao, G., Cheng, S., Tan, L., Zhang, X., 2020. Towards feature space adversarial attack. arXiv preprint arXiv:2004.12385 .

Xu, W., Zhang, F., Zhu, S., 2013. Permlyzer: Analyzing permission usage in android applications, in: 2013 IEEE 24th International Symposium on Software Reliability Engineering, IEEE. pp. 400–410.

Xu, Z., Ren, K., Qin, S., Craciun, F., 2018. Cdgdroid: Android malware detection based on deep learning using cfg and dfg, in: International Conference on Formal Engineering Methods, Springer. pp. 177–193.

Yang, W., Zhang, Y., Li, J., Shu, J., Li, B., Hu, W., Gu, D., 2015. Appspear: Bytecode decrypting and dex reassembling for packed android malware, in: International Symposium on Recent Advances in Intrusion Detection, Springer. pp. 359–381.

Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X., 2020. Byte-level malware classification based on markov images and deep learning. Computers & Security 92, 101740.

Yuan, X., He, P., Zhu, Q., Li, X., 2019. Adversarial examples: Attacks and defenses for deep learning. IEEE Transactions on Neural Networks and Learning Systems .

Zhang, N., Yuan, K., Naveed, M., Zhou, X., Wang, X., 2015. Leave me alone: App-level protection against runtime information gathering on android, in: 2015 IEEE Symposium on Security and Privacy, IEEE. pp. 915–930.

Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M., Yang, M., 2020. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware, in: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security, pp. 757–770.

Zhao, G., Zhang, M., Liu, J., Wen, J.R., 2019. Unsupervised adversarial attacks on deep feature-based retrieval with gan. arXiv preprint arXiv:1907.05793 .

Zheng, M., Lee, P.P., Lui, J.C., 2012. Adam: an automatic and extensible platform to stress test android anti-virus systems, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer. pp. 82–101.

Zhiwu, X., Ren, K., Song, F., 2019. Android malware family classification and characterization using cfg and dfg, in: 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), IEEE. pp. 49–56.

Zikratov, I.A., Korzhuk, V., Shilov, I., Gvozdev, A., 2017. Formalization of the feature space for detection of attacks on wireless sensor networks, in: 2017 20th Conference of Open Innovations Association (FRUCT), IEEE. pp. 526–533.

Zulkifli, A., Hamid, I.R.A., Shah, W.M., Abdullah, Z., 2018. Android malware detection based on network traffic using decision tree algorithm, in: International Conference on Soft Computing and Data Mining, Springer. pp. 485–494.

Appendix A. Appendix - Additional Evasion attacks against MaMaDroid2.0

This appendix reviews four additional evasion attacks tested on MaMaDroid2.0, in a similar manner to the evaluation presented in Section 6. The attacks were: Random StB attack, Black Hole Statistical StB attack, MB1 attack, and MB3 attack. Each attack was evaluated using the ER, DRR, and MR metrics.

The results in Fig. A.8a and A.8b show that the ER of the model using the base features (Base), against the StB attacks (Random and Black Hole Statistical attacks) resulted in a maximum of 0.4 at each point, and decreased to 0 in the case of the Black Hole Statistical StB attack. The extended classifier using the base features and the *permission set* (Ext) and the classifier that was based solely on the *permission set* (Perm), sustained a better ER, with 0.6-0.8 on every point. The addition of the *permission set* again was proven to strengthen the basic classifier against the StB attacks.

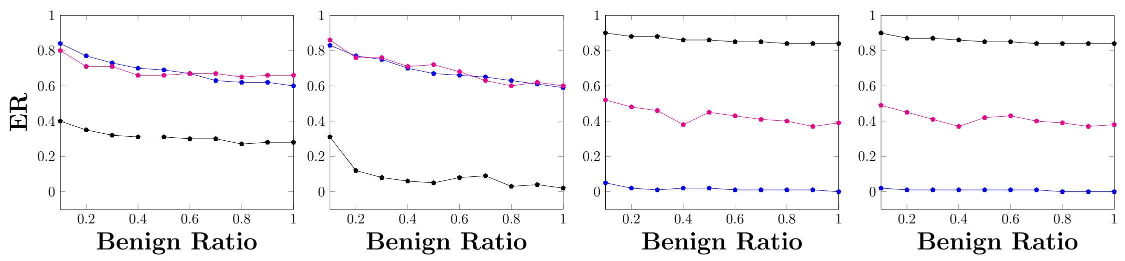
Fig. A.8c and A.8d show the results of the MB1 and MB3 attacks. The results of the ER of the Perm and Ext models are now distinguishable. The Ext classifier demonstrated an advantage of ~ 0.5 over the Perm classifier at every point. However, the Base classifier displayed an advantage over the Ext classifier as well. The Base classifier was better than the Ext classifier by ~ 0.4. The Base classifier was not affected as the MB attacks changed the *permission set*, which the classifier did not analyze. The merge of the features of MaMaDroid1.0 and the permission set resulted in a less effective classifier against the MB attacks. However, the robustness against the StB attacks was of great value.

As before, these results show that both the Base and Perm models were weak against the attacks that target their initiative - the StB attacks against MaMaDroid, and the MB attacks against the Perm classifier. They also show that the extended version, MaMaDroid2.0, is a classifier that presents a meaningful defense against both attacks.

Fig. A.9a- A.9d present the results of the DRR evaluation of the Base, Ext, and Perm models. This metric reflects the effect of evasion attacks on these classifiers. A lower DRR indicates an effect on the classifier. Even if the attack gains a high ER, it could be improved more easily in order to become a successful attack. The figures demonstrated that the three models have an identical DRR for every attack.

Fig. A.10a- A.10d present the results of the MR evaluation of the Base, Ext, and Perm models. This metric reflects the disorder in each classifier when confronting evasion attacks. The figures show that for each classifier and at most points, the MR was 1. In other words, there was a neglect rate of disorder in each of these classifiers. This is straightforward, as the inner model of each one of these classifiers is DT, where an MR rate of 1 due to the binary probability pairs in most cases was proven in Section 5.

In conclusion, the Ext classifier proved to be a promising option, as its robustness against both types of attacks - the StB and MB attack - was



(a) Random StB attack (b) BHS StB attack (c) MB1 attack (d) MB3 attack

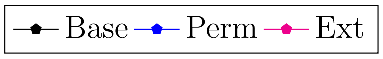
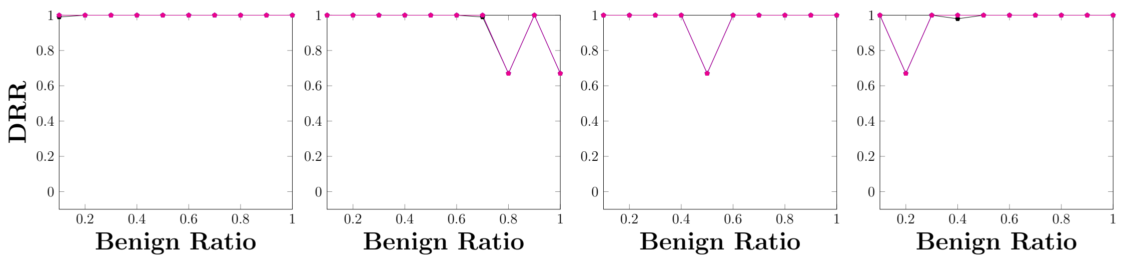


Figure A.8: ER results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with four types of manipulated data by evasion attacks (a-d) - Random StB attack, Black Hole Statistical StB attack, MB1 attack, and MB3 attack. The ER represents the evasion robustness of each model.



(a) Manipulated data - Random StB attack (b) Manipulated data - Black Hole Statistical StB attack (c) Manipulated data - MB1 attack (d) Manipulated data - MB3 attack

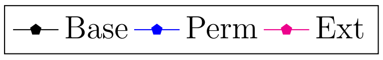
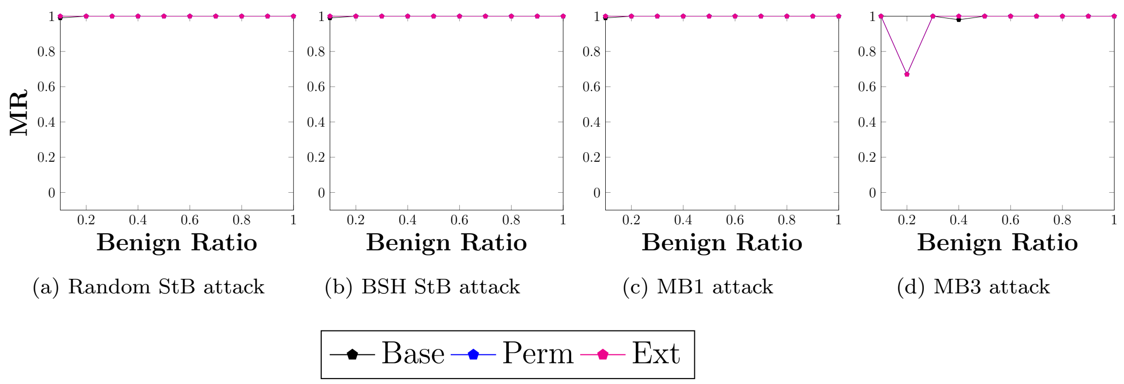


Figure A.9: DRR results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with four types of manipulated data by evasion attacks (a-d) - Random StB attack, Black Hole Statistical StB attack, MB1 attack, and MB3 attack. The DRR represents the Defense Reciprocal Rank of each model.

shown to be better than the Base and Perm classifiers.



(a) Random StB attack (b) BSH StB attack (c) MB1 attack (d) MB3 attack

Figure A.10: MR results: The Base model, the Permission-based (Perm) model, and the Extended (Ext) model. The models were tested with four types of manipulated data by evasion attacks (a-d) - Random StB attack, Black Hole Statistical StB attack, MB1 attack, and MB3 attack. The MR represents the Model Reliability of each model.