Received November 29, 2021, accepted December 23, 2021, date of publication December 24, 2021,
date of current version January 6, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3138628

Crystal Ball: From Innovative Attacks to Attack Effectiveness Classifier

HAREL BERGER[1], **CHEN HAJAJ**[2], **ENRICO MARICONTI**[3], **AND AMIT DVIR**[1]
[1]Ariel Cyber Innovation Center, Computer Science Department, Ariel University, Ariel 40700, Israel
[2]Ariel Cyber Innovation Center, Data Science and Artificial Intelligence Research Center, Industrial Engineering and Management Department, Ariel University, Ariel 40700, Israel
[3]UCL Department of Security and Crime Science, London WC1H 9EZ, U.K.

Corresponding author: Amit Dvir (amitdv@g.ariel.ac.il)

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

ABSTRACT Android OS is one of the most popular operating systems worldwide, making it a desirable target for malware attacks. Some of the latest and most important defensive systems are based on machine learning (ML) and cybercriminals continuously search for ways to overcome the barriers posed by these systems. Thus, the focus of this work is on *evasion attacks* in the attempt to show the weaknesses of state of the art research and how more resilient systems can be built. Evasion attacks consist of manipulating either the actual malicious application (problem-based) or its extracted feature vector (feature-based), to avoid being detected by ML systems. This study presents a set of innovative problem-based evasion attacks against well-known Android malware detection systems, which decrease their detection rate by up to 97%. Moreover, an analysis of the effectiveness of these attacks against VirusTotal (VT) scanners was conducted, empirically showing their efficiency against well-known scanners (e.g., McAfee and Comodo) as well. The VT system proved to be a great candidate for the attacks, as in 98% of the apps, less scanners detected the manipulated apps than the original malicious apps. As not all the attacks are effective in the same manner against the VT scanners, the *attack efficiency classifiers* are advised. Each classifier predicts the applicability of one of the attacks. The set of classifiers creates an ensemble, which shows high success rates, allowing the attacker to decide which attack is best to use for each malicious app and defense system.

INDEX TERMS Android malware, machine learning, malware detection.

I. INTRODUCTION

Malware can be formally defined as any damaging software that can harm a computer or mobile user [1]. Malware appears in various forms, such as ransomware, trojans, and viruses and targets all the Operative Systems (OSs) on the market. Mobile malware has been a rising threat in the past years (e.g. [2], [3]). As Android is the operative system on more than 80% of the smartphones sold on the market, Android malware are the most common and thus have the most impact. Android malware takes many forms, like piggybacking popular Android application PacKages (APKs), which can propagate to various Android markets. Lately, 10 million Android users were attacked by an Android malware named GriftHorse that lured users to assign themselves to a paid service incorporated in benign-like apps. The developers of this malware acquired over 1.5$ million [4]. During the last decade, this type of events motivated a long list of researchers in the security community. Researchers have been working towards different approaches throughout the years. Signature-based detection systems [5]–[7], as well as studies on permissions usage [8]–[10] are some of the works carried out. Recently, ML-based systems is one of the areas of highest interest [11]–[14].

A never-ending marathon is held between the defenders and attackers on each cyberspace subdomain, and the Android malware detection is no different. Against the malware detection systems, the attackers generate *adversarial examples* (AE) that target the vulnerabilities in the malware detection systems, as proven by Goodfellow *et al.* [15]. These samples are manipulated to hide some features or mimic the properties of a different malware family class so that the detection system does not classify them correctly. A special case of AE involves the use of *evasion*

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

attacks (EA), in which an adversary generates a perturbed version of an instance that is mistakenly classified as benign by a detection system [16]–[20]. This study follows problem-based EAs, which manipulate the actual code of the app. The contribution of this work is twofold. First, it leverages weak spots in popular Android malware detection systems (e.g. Drebin [11], [13], [21]–[25], Sec-SVM [13], [24]–[26], DNN [21], FM [22]), by means of innovative evasion attacks. In addition, the attacks were validated by using them against some of the famous malware scanners accessible via VirusTotal (VT) [27]–[31]. Popular scanners like Comodo, McAfee, Symantec, and Alibaba showed a significant decrease in detection rates, as a result of this study's evasion attacks. To extend the perspective on the attacks, a comparison of the suggested evasion attacks was conducted with Android-HIV (AHIV) [23], a well-known problem-based evasion attack study. This study's evasion attacks resulted in a greater increase in terms of evasion rates compared to AHIV attacks.

Second, each attack was analyzed comprehensively, including the evasion increase and VT app fail rates. Thanks to these analyses, a classifier that predicts if an evasion attack will be successful was implemented. An aggregation of multiple classifiers (which are correlated with this study's evasion attacks), resulted in a decision function that is able to predict and suggest which attack may be most effective depending on the defensive systems in place. Both the decision function and the classifiers are new approaches that can cause serious threats to conventional detection systems.

The remainder of this paper is organized as follows. First, related work is presented and discussed in Section II. Second, evaluation metrics and the dataset are discussed in Section III. The Android malware detection systems exploited in this paper are provided in Section IV, together with the attacker models. In Section V, the attacks are presented, including the comparison between the evasion attacks of this work and Android HIV. Next, the attack effectiveness classifiers are introduced and evaluated in Section VI. A discussion about the evasion attacks and the classifiers is provided in Section VII. Finally, the conclusions of the paper are presented in Section VIII.

II. RELATED WORK

This section describes previous work in the field of Android malware detection systems. The systems that are discussed are based on ML, categorized in four main approaches. This is the first part, in Section II-A. In addition, ML malware detection systems are vulnerable to EAs. Two forms of evasion attacks are known, problem-based attacks and feature-based attacks. Problem-based attacks [13], [32]–[37] (which is the type of attack implemented in this study) incorporate perturbations into the sample itself, while feature-based attacks [38]–[44] convert the sample into a feature vector and change the values of features. Feature-based attacks are implemented easily and automatically by an ML [45]–[49]. However, when implementing the changes into an application, the functionality of the app can be severely damaged [23], [30], [50], [51]. As this study implements problem-based EAs, the focus of Section II-B is problem-based EAs.

A. ANDROID MALWARE ML-BASED DETECTION SYSTEMS

This section reviews well-known ML-based detection systems for Android malware and discusses four main Android malware detection approaches. The first is static analysis, which includes gathering string values from the manifest file and the Smali code files. The second approach follows the control flow graph (CFG) of the application, which traces the connection between API calls used in the app. The third approach inspects the behavior of the app and gathers information on the OS's behavior while the app is running, along with network usage, etc. The fourth approach analyzes Bytecode sequences.

A general approach in Android malware detection systems is static analysis. A well-known static-analysis detection system in the Android malware domain is Drebin [11]. Drebin collects 8 types of features from the APKs. Drebin extracts permissions requests, software/hardware components and intents from the manifest file, and suspicious/restricted API calls, permissions that have been used in the app's run and URL addresses from the Smali code. Several updated versions of Drebin were published. For example, Sec-SVM [13] is an improved version, that uses an evenly-weighted approach toward the same feature-set. Another version is a DNN version [21], which uses the same features to originate a deep neural network. A factorization machine of Drebin was published in [22]. The DroidAPIMiner [52], [53] is a detection system similar to Drebin. It analyzes API calls and permissions. The authors analyzed dataflow to recover package names and frequently used parameter values. Mmda [54] analyzes static features as well, including permissions, hardware features and receiver actions. Chan and Song [55] inspected permissions and API calls. Sato *et al.* [56] listed permissions and intents as their feature set. They studied the frequency of the use of specific features in benign and malicious apps. Ma *et al.* [57] studied API calls in means of sequence, frequency and usage to classify benign and malicious APKs. The static analysis of Android malware detection systems inspects several features of the apps which are easy to understand and deploy. Most of the static features can be enumerated and manipulated [13], [36], [58], [59].

MaMaDroid [12] is an example of a different approach. This detection system extracts features from the CFG of an app. MaMaDroid [12] generates an API call tree based on packages and families. After an abstraction process of the API calls, it analyzes the sequence of the API calls performed by the app, to model its behavior. A similar approach to identifying malicious third-party libraries in apps was used by Backes *et al.* [60]. The authors also used function and app profiling, to analyze different versions of the same third-party library. The authors used Merkle Trees [61] to model their libraries' profiles. Monitoring the sequence of API calls seems to be a better idea than the syntax analysis approach and changing the app's flow is more complicated. But, EAs that change the flow of the application such as [23], [58], [62]–[64] succeed in evading this kind of malware detection system.

Several parameters such as network usage, CPU and battery levels are the focus of the third approach. An example of an Android malware detection using these features is Andromaly [14]. A similar system was presented by Shabtai *et al.* [65], who focused on network usage by measuring the network traffic patterns of apps. The authors learned the statistics of network packets a user sent and received, the time between packets, etc. Shabtai *et al.* [66] proposed an Android malware detection system using app resource consumption with emphasis on temporal abstraction. The authors aggregated the data during an app run, including both user interactions like clicks and OS behavior such as the CPU and network usage. Madam [67] correlates features at four levels, i.e., kernel, application, user and package, to detect malicious activity. Wang *et al.* [68] dissected kernel tasks such as number of pages on VMs, change of states of tasks. The third approach is based on the behavior of the Android device while running various apps. This behavior may depend on the app and is therefore hard to generalize.

Bytecode inspection is the fourth approach in Android malware detection. Dalvik Bytecode Frequency Analysis [69] looks for popular Dalvik Bytecode instructions found in malware apps. TinyDroid [70] studied sequences of Dalvik Bytecode instructions. The authors gathered families of Bytecode instructions under a symbol and used n-grams [71] to create the features for the learning algorithm.

As each approach individually did not suffice, researchers began to combine various methods. Martín *et al.* [72] fused dynamic and static analysis of Android apps. They combined the analysis of API calls (static) and transitions and probabilities of execution states (dynamic), creating AndroPyTool [73], an open source tool for Android malware research. A combination of static analysis of permission requests and dynamic testing of actual usage of these permissions was introduced in [74]. MARVIN [75] analyzes the behavior of Android apps through static analysis of their structure, certificates, etc. with the addition of dynamic analysis of phone activity, dynamic code loading and more. Androtomist [76] analyzes static features such as permissions, intents, API calls and Java classes, along with behavioral features like network traffic, and inter-process communication. The combination of several methods seems more accurate. However, analyzing multiple feature sets, for example static and behavioral feature sets, requires huge amount of resources, in means of memory and CPU. A recent extended survey on Android malware detection systems can be found in [77].

B. PROBLEM-BASED EVASION ATTACKS ON ML-BASED DETECTION SYSTEMS

In the field of problem-based evasion attacks, three forms of attacks should be mentioned. The first engages in the art of camouflage, where the attack tries to conceal suspicious strings and values of the app using obfuscation and encryption [13], [36], [62]. The second adds noises to the app [23], [50], [78], [79]; e.g., non-invoked functions. The last attempt is to change the nature of the app [23], [62]–[64]. It analyzes the previous flow of the app, and changes the code of several function calls.

The first type of problem-based EA is to camouflage specific parts of the app. One well-known example of camouflage is the use of obfuscation or encryption. Demontis *et al.* [13] used obfuscation of suspicious strings, API calls, and packages. Another example of camouflage is packing an app inside another app. DaDidroid [62] used a similar obfuscation method as the one found in Demontis *et al.* [13], with the addition of packing. The authors wrapped the original malware inside a benign app. Reflection, which allows an app to masquerade its nature at runtime, is also a classic obfuscation method. In reflection, an attacker runs a code that is loaded on runtime. Rastogi *et al.* [36] introduced an attack, which uses Demontis *et al.*'s [13] approach along with reflection.

Another typical approach to problem-based EAs on ML-based detection systems includes adding noise to the application, thus misleading the detection system's labeling of benign and malicious apps. A stub function/code injection is a well-known example of noise added to an app that does not add any real functionality to it, but may change its flow. In Android HIV [23], the authors implemented non-invoked dangerous functions against Drebin and a stub function injection against MaMaDroid. A recent example of this type of attack can be found in [50], where the authors implanted benign codes in malicious apps to evade Drebin and Sec-SVM [13]. Pierazzi *et al.* [50] implanted non-used benign parts from benign APKs to malicious APKs for misclassification by Drebin and Sec-SVM as well. GADGET [78] generates API call based EAs against Android malware detection systems. Three types of attacks were incorporated in this system, which included appending no-op functions to the app, encoding short API calls and obfuscating strings. Cara *et al.* [79] injected non-operational classes to the end of function calls to evade classification of malicious apps.

The third approach involves in manipulation of the app's flow. A detection system which is based on the app flow analysis (i.e. MaMaDroid), may misclassify apps whose app flow changed [23], [62]. One of the ways to achieve this goal is function outlining/inlining. A function outlining breaks a function into smaller functions. Function inlining replaces a function call with the entire function body. Stub function can also be used to break the app flow [23], [63], [64]. A result of such a change may be an ML misclassification of a malicious application.

III. EVALUATION METRICS AND DATASET

In this section, the application dataset and a number of evaluation metrics are described. First, the dataset is presented, followed by evaluation metrics for each attack: evasion robustness, evasion increase rate, VT app fail rate and evasion trend.

A. DATA COLLECTION

The APK files evaluated throughout the paper were gathered from the AndroZoo dataset [80], downloaded from the Google Play market [81] and Drebin's dataset [11], where the first includes benign APKs and the latter includes the malicious ones.[1]

B. EVASION ROBUSTNESS (ER)

To evaluate the robustness of detection systems against evasion attacks, the proportion of malicious instances was computed for which the detection system was evaded; this is the metric of *ER*, with respect to the robustness of the detection system (similar to the analysis provided in [85]). The ER is computed as the true positive rate (TPR) of the malicious apps. The TPR metric is used for the original detection rate ($Orig_{DR}$) as well, measuring the TPR of the original malicious apps (prior to the evasion attacks).

C. EVASION INCREASE RATE (EIR)

The evasion robustness depicts the detection rate in the presence of an evasion attack. However, a more precise analysis of the effectiveness of an evasion attack can be suggested. The EIR compares the original detection rate ($Orig_{DR}$) and the *ER*, as in [86], [87]. Eq. 1 formulates the *EIR* metric:

$$EIR = 1 - \frac{ER}{Orig_{DR}} \tag{1}$$

D. VT APP FAIL RATE

The VT app fail rate measures the similarity between the former Android malware and their manipulated counterparts. This metric is specifically for a given set of detection systems. It views these systems as a one big detection system. Therefore, it is tailored for the VT system. For each app x, and for each scanner $sc \in SC$ where SC is the set of VT scanners, the function $H_{SC}(x)$ is defined: $H_{SC}(x) = \sum_{sc \in SC} \mathbb{1}_{sc}(x)$ (where $\mathbb{1}_{sc}(x)$ equals 0 if a specific scanner sc labels a sample app x as benign, and 1 if it labels the sample as malicious.). In an evasion attack, a set of perturbations is added to x, therefore resulting in x'. An app fail rate is defined as follows: $A\_FR_{SC}(x) = \frac{H_{SC}(x) - H_{SC}(x')}{|SC|}$.

As an evasion attack runs on a set of app samples X, the final formulation of the *VT app fail rate* is as follows in Eq. 2.

$$FR_{SC}(X) = \sum_{x \in X} A\_FR_{SC}(x) \tag{2}$$

E. EVASION TREND

This metric extends the VT app fail rate, adding an overall view of each evasion attack by the VT scanners according to the attack's trends. For example, assume that a former malware had 5 scanners detecting it as malicious. If the manipulated app is detected in the second scan by at least 6 scanners, it will be added to the higher trend. On the other hand, if the manipulated app is detected in the second scan by at most 4 scanners, it will be assigned to the lower trend. Alternatively, it will be related to the equal trend. The term f_{t_type} defines a specific trend.

Each evasion attack was evaluated according to the three trends (higher, lower, equal). The evasion trend (ET_{t_type}) function was defined as the sum of running the relevant f_{t_type} function on the former and manipulated apps divided by the amount of former/manipulated apps. The formulation of the ET function is presented in Eq. 3:

$$ET_{t\_type} = \frac{\sum_{x \in X} f_{t\_type}(H_{SC}(x) - H_{SC}(x'))}{|X|} \tag{3}$$

IV. TARGETED SYSTEMS AND ATTACKER MODELS

Four Android malware detection systems are described in this section: Drebin [11], Sec-SVM [13], a Factorization Machine (FM) [22] and DNN [21]. Drebin has several features, one of which includes requested permissions. Sec-SVM, FM and DNN are advanced models based on Drebin's feature set. In addition, three attacker models are described: Drebin Model Access, Data Access and Zero-knowledge. These models are the basis for the evasion attacks against the malware detection systems.

A. TARGETED SYSTEMS

Drebin [11] is a lightweight Android malware detection system. During its run, Drebin creates the following observations on each APK sample: (1) Component lists: The observations include *IntentActionList*, *ServiceList*, *ActivityList*, and *BroadcastReceiverList*. These observations are gained from the manifest file by following the component tags: intent, service, activity, receiver. (2) Requested permissions: These observations describe the permissions that were requested by the app. Drebin maps them according to the uses-permission tags from the manifest file. (3) Suspicious APIs: These observations relate to the APIs defined in the detection system. An example of a suspicious API is *setWifiEnabled()*. (4) Restricted APIs and used permissions: Drebin maps API calls to permissions. Drebin looks for these calls to understand which permissions are actually used by the application. Next, Drebin lists the permissions that correspond to permissions requested in the manifest as used permissions. Finally, it adds the APIs mapped to permissions that were not mentioned in the manifest file as restricted API calls. (5) URLDomains: The classifier lists the URLs mentioned in the Smali code files. The list of these observations and their weights are defined in this paper as ***Drebin reports***.

Overall, the classification process of Drebin is as follows: (1) Drebin saves the original labels of each app. A label of 0 is used for benign, and 1 for malicious. (2) During the training process, Drebin inspects each training APK file.

[1] Recent works [21], [30], [82]–[84] used a similar approach, choosing the Drebin dataset as their malicious dataset and apps from Androzoo as their benign dataset.

It assigns each observation a corresponding weight W_s and sums the weights. (3) Drebin computes the maximum weight that corresponds to a benign app as t using SVM. (4) In the test phase, if $\Sigma W_s > t$, Drebin predicts a label of 1. Otherwise, it predicts 0. It saves the weights and labels. (5) Drebin compares its predicted labels to the original labels, and produces an accuracy rate.

Sec-SVM [13] is a more secure version of Drebin. It uses the same features. However, it is a more evenly-weighted feature set. Such an approach was proven to improve the detection system of the classifier under an evasion attack.

A **factorization machine** (FM) [22] was implemented based on Drebin's feature set (setting aside the web domains). The FM model is designed to capture interactions between features within high dimensional sparse datasets efficiently.

A basic **DNN** [21] was also tested on the Drebin feature set. This DNN incorporates two fully-connected hidden layers. Each layer consists of 160 neurons. The activation function is ReLU. For optimization, the Adam optimizer is used with 150 epochs, a 128 sized mini-batch, and a learning rate of 0.001.

B. ATTACKER MODELS

This section describes three attacker models. The first two models depict a gray-box attacker, which has access to the Drebin reports or the training data. The last model is a zero-knowledge attacker. Each model describes an attacker that can manipulate specific parts on an APK, but not the training data or the model itself. First, the additional permission family statistics used in the attacker models are described. Then, the attacker models: Drebin Model Access (DMA), Data Access (DA), and Zero-Knowledge (ZK) are discussed. The models are summarized in Table 1.

TABLE 1. Attacker models.

Model	Access to training data	Access to the model
Drebin Model Access	X	V
Data Access	V	X
Zero-Knowledge	X	X

C. PERMISSION FAMILY STATISTICS

The occurrences of each set of permissions were extracted from both the benign and malicious apps. This process resulted in a list of each group, which maps the set of permissions and their probability. Each of these sets is denoted as a *permission family*. An adversary who has access to these statistics understands how to mimic a benign app. For example, if a permission family from the benign app's group is [X,Y,Z] and a malicious app requests permissions [A,X,B,Y,Z,C] the attacker will conceal permission requests [A,B,C] to mimic the benign app's group behavior.

D. DMA ATTACKER MODEL

DMA is the first attacker model. This model has access to Drebin as a gray-box during the manipulation process. It can send apps to Drebin, and gain the Drebin report of each app in return. Drebin enumerates the observations on the app in the report and sends the report back to the attacker. The Drebin report leads the attacker to the significant features the adversary needs to obfuscate.

E. DA ATTACKER MODEL

The second attacker, DA, has access to the dataset used to train the classifier. This model has no access to the model during the manipulation process. The data allows the adversary to create the permission families.

F. ZK ATTACKER MODEL

ZK, the third attacker model, has no knowledge of the trained classifier or the training data during the manipulation process.

V. EVASION ATTACKS

An evasion attack is engineered for each attacker type (i.e., attacker model). Each attack transfers the embedded knowledge of the model or the training data to a manipulated app that is designed to be classified wrongly as benign.

In this section, an attack starting point template is described in Section V-A. This is a common starting point for each one of the evasion attacks. Then, the attacks targeting the detection systems are described in Section V-B. The experimental design is described in Section V-C and the results are analyzed and discussed in Section V-D.

The evasion attacks were tested not only against academic detection systems, but also on the set of industrial AVs from VT. The attacker has no information on the VT scanners, their detection methods or training data. Therefore, each one of the evasion attacks should be considered a zero-knowledge attack on the VT scanners. The VT scanner evaluation proves that these attacks are effective not only on the targeted systems but also on well-known VT scanners. Lastly, a comparison between the evasion attacks to a recent problem-based evasion attack named Android HIV is presented in Section V-E.

Algorithm 1 Attack Starting Point Template

```
1: procedure At. starting point($APK$, [$Add\_data$])
2:     $Manifest, Smali \ldots \leftarrow depackage(APK)$
3:     $Manifest, Smali \ldots \leftarrow$
   $Attack\_Vector(Manifest, Smali \ldots, [Add\_data])$
4:     $APK \leftarrow Repackage(Manifest, Smali \ldots)$
5:     return $APK$
```

A. ATTACK STARTING POINT TEMPLATE

This section describes a starting point template to the evasion attacks, which is a common starting point for each attack implemented in this study. The template consists of the following steps: (1) The algorithm's input is an APK and additional data if available (i.e., Drebin report/permission families' list); (2) Depackage the APK to the Smali code files, manifest file and other subordinate files (line 2); (3) Run the attack vector using the files obtained from the depackage process and the additional data (line 3); (4) Repackage the APK (line 4), and return it as an output of the algorithm (line 5)

B. MANIFEST BASED EVASION ATTACKS

This section introduces three novel evasion attacks. The following attack vectors manipulate parts of the manifest file, and specifically the permissions app requests. It should be noted that all targeted systems use Androguard [88] to extract and list native permission requests. The attacks are named MB1, MB2 and MB3. Each attack is defined by its formal attacker model, which describes its capabilities. For example, attack **Manifest based attack type 1 (DMA)** refers to the first attack which is termed MB1. The attacker model which defines the capabilities of this attack - the DMA attacker model - is discussed in Section IV-B.

Manifest based attack type 1 (DMA), also known as **MB1**. The adversary scans the report from Drebin in parallel to the manifest file. It finds the requested permission section in the report that is correlated to the specific tags in the manifest file. Next, it switches the *uses-permission* tags which describe a permission request to a newer version of the permission request, a *uses-permission-sdk-23* tag. The complete attack procedure follows these steps: (1) The procedure's input are the files depackaged from the APK and a list of observations produced by the Drebin report on this APK; (2) For each observation *s*, run steps 3-4; (3) Find the observation in the manifest file; (4) SDK_23: Replace the uses-permission tag in *s* to a uses-permission-sdk-23 tag; (5) Return the new manifest file and all the other depackaged files as output.

Manifest based attack type 2 (DA), also known as **MB2**. The attacker observes the permission family statistics in parallel to the manifest file. It finds the top *n* families from the benign apps dataset. In this study, WLOG, $n=3$ was used. Next, it changes the manifest file accordingly. For each app, it randomly picks a family to which to change. The attacker switches the uses-permission tags of all the permission tags that do not match a member of the family to a new uses-permission-sdk-23 tag. The complete attack function takes the following steps: (1) The procedure's input are the files depackaged from the APK and the permission family statistics; (2) Random_family: pick a random family from the top 3 families in the benign app statistics; (3) For each permission *p* in the manifest file, run steps 4-7; (4) If *p* is part of the family, continue without any change; (5) SDK_23: Replace the uses-permission tag of *p* to a new uses-permission-sdk-23 tag; (6) Return the new manifest file and all the other depackaged files as output.

Manifest based attack type 3 (ZK), also known as **MB3**. The attacker blindly changes all of the uses-permission tags to uses-permission-sdk-23 tags. The complete attack procedure implements the following steps: (1) The procedure's inputs are the files depackaged from the APK; (2) For each permission *p* in the manifest file, run step 3; (3) SDK_23: Change the uses-permission tag in *p* to a uses-permission-sdk-23 tag; (4) Return the new manifest file and all the other depackaged files as output.

The manifest based attacks, which are presented in this section, follow changes in the Android SDK versions. Specifically, the uses-permission-sdk-23 tag was introduced on devices running Android version 6.0. with sdk lower than 23 cannot run the manipulated apps. However, as of May 2021 [89], 88% of the Android devices around the globe run Android version 6.0 or a newer version. Therefore, the evasion attacks can be considered a threat for 88% of the Android users. Also, the use of the new uses-permission-sdk-23 tag is a part of the runtime permissions model. As a result, users will be notified in run-time to grant permissions. This may affect certain malware that wants to stay under the radar without actively interacting with users. However, not every user pays attention to the permission they grant [90]. For example, kids who download free apps which incorporate malicious activity may grant permissions as an automatic process to achieve full functionality of the app without any thought about the consequences. Granting only the Internet permission (which is common for benign and malicious apps, as they tend to send and receive information from the Internet) and the SMS permission produce a platform for malicious surveillance [90]. The number of requests from the user is small, and therefore may be considered a small interruption for the user.

The changes to the apps due to the MB attacks do not alter any functionality of the manipulated apps. Nevertheless, a random subset of 10% of the applications of every evasion attack was tested using DroidBot [91]. Each app was installed and run on an emulator. Using the DFS-policy of DroidBot, a subset of 5 actions were tested on the former and manipulated apps. One-hundred percent of the apps retain basic functionality of the former malicious apps.

C. EXPERIMENT SETTINGS

The experiments in this research assess the effectiveness of the evasion attacks against Drebin, Sec-SVM, FM and DNN. The experiments were run on an Intel(R) Core(TM) i7-4510U CPU with 8 GB RAM with GeForce 840M GPU. The implementation of Drebin for this study was taken from GitHub [92]. A version of Sec-SVM was obtained from the authors of [50]. The implementation of DNN was taken from GitHub [21]. The implementation of the FM model was based on polylearn [93]. The VT scan was run during 2020-2021.

The dataset for this experiment consisted of ∼60K benign apps from Androzoo (acquired from the Googleplay app store) and ∼6K malicious apps from the Drebin dataset (for more details on the source of the data, see Section III). The Drebin dataset was used for the experiments; as the basic detection machine, Drebin [11], was built with the use of this dataset. The Sec-SVM, DNN and FM also use the Drebin dataset as their source for malicious apps. The benign apps were replaced as the original apps were not available. However, the original detection rates of each one of the detection systems (Table 2) are similar to their detection rates in the original papers.[2] This different balance between benign and malicious apps was referred to in a recent paper [94] in the Android malware detection field, which noted that the real-life ratio of benign/malicious Android applications is 90/10.

In addition, the results were validated by using 5-fold cross validation. The benign data were not split to test/train since the goal was to assess the attack's evasion rate, as in [23], [30], [50]. Overall, $\sim 60k$ benign apps and $\sim 4.8k$ malicious apps were used as training data, and $\sim 1.2k$ malicious apps as test data.

An **initialization phase** preceded each attack, including the classifier detection rate evaluation which included the following steps: (1) A trained classifier was trained on the benign and malicious dataset. Malicious test apps were sent to the trained classifier. The classifier labeled each test app. (2) The labels were accumulated to reveal the detection rate, which is named the **initial detection rate**.

D. RESULTS

The effectiveness of the MB evasion attacks was evaluated by the ER, VT app fail rate (FR_{App}), EIR, and the ET (mentioned in Section III).

1) **Evasion Robustness:** The results are presented in Table 2. The original apps and three evasion attacks were evaluated against the targeted systems and VT scanners. The detection systems that were tested were Drebin, Sec-SVM, FM, and DNN. Six popular VT scanners were chosen (Comodo, Alibaba, McAfee-GW-Edition (McAfee), AegisLab, SymantecMobileInsight (Symantec) and Microsoft), in light of the fact that they all have a high original detection rate. They are representatives of the behavior of the VT system.[3]
The results show that the MB evasion attacks decreased the detection rate of all of the detection systems. Drebin sustained a fair evasion robustness of at least 75% with the MB attacks. The balance between the different feature types allowed Sec-SVM to detect more malicious samples, resulting in a minor decrease in its ER. On the other hand, the FM version suffered from a significant loss in the detection rate, namely ~70%. The DNN version of Drebin suffered a total loss in the detection rate, resulting in no manipulated app that was detected by the machine. Drebin and its updated versions extracted features outside the manifest file and thus their detection rate should not have zeroed when confronting the MB attacks. However, the DNN model also had an ER of 0%. As in its former study [21], this model is susceptible to evasion attacks.

[2]In addition, a subset of random samples from a newer dataset, CIC-ANDMAL2017, were examined for the feasibility of the MB evasion attacks. There were no errors in the manipulation process of the newer apps, and the functionality of each app was retained (based on DroidBot as well).

[3]The results of the 64 scanners for the clean data and evasion attacks, along with the classifiers, dataset and attacks can be found in: https://github.com/harelber/Android-crystal-ball.

TABLE 2. ER of Drebin, Sec-SVM, FM, DNN, along with the VT scanners Comodo, Alibaba, McAfee-GW-Edition, AegisLab, Symantec and Microsoft. The minimal ER of each detection rate/VT detection system/scanner appears in bold.

	$Orig_{DR}$	MB1	MB2	MB3
Drebin	0.96	**0.75**	0.78	0.76
SecSVM	0.96	0.88	**0.86**	0.89
FM	0.97	**0.27**	0.34	0.39
DNN	0.97	**0**	**0**	**0**
Comodo	0.9	**0.2**	0.34	0.35
Alibaba	0.73	0.76	**0.01**	0.09
McAfee	0.75	0.95	**0.04**	0.15
AegisLab	0.99	0.97	**0.39**	0.43
Symantec	1.00	0.99	0.8	**0.53**
Microsoft	0.88	0.88	**0.8**	0.81

A subset of the most popular VT scanners' results is presented as an extension to the explored detection systems. As can be seen, the scanners' ER rates vary, but some significant ER rates can be pinpointed. The ERs of Alibaba and McAfee in the case of MB2 were less than 5%. Comodo's ER with MB1 was 20%. AegisLab's ER was less than 40% with MB2, where its former detection rate was 99%. Symantec's original detection rate was 100%. Its ER was almost half, namely 53%. Microsoft's ER sustained a fair rate of 80%.
In conclusion, the MB attacks were effective against Drebin. Its secured version Sec-SVM was more robust against them. The FM version was less robust than Sec-SVM. Also, a DNN version of Drebin was assessed. However, it failed to detect any manipulated app. Moreover, some of the popular VT scanners suffered a great loss against these attacks.

2) **Evasion Increase Rates:** The robustness evaluation can be extended in light of the original detection rate. A higher EIR indicates that more manipulated samples can evade the detection system/scanner. As depicted in Fig. 1, Drebin's EIR is at most ~ 0.2 as it uses multiple components and API calls as its feature set. In a similar manner, Sec-SVM's EIR low rate of 0.1 is explained by the use of the same features as Drebin. The DNN's EIR increases to 1.00.
Interesting insights can be drawn from the EIRs of the VT scanners. As can be seen, the EIRs of Alibaba and McAfee in MB1 drop by several percent and 20%, respectively. However, in MB2 their EIRs rise to more than 90%. In MB3 their EIR is approximately 80%. Comodo's EIR is at least 60% in each one of the attacks. AegisLab's EIR is similar to zero in MB1, and 60% in the MB2 and MB3 attacks. Finally, Symantec's EIR is negligible in the MB1 attack, increases to 20% in the MB2 attack, and increases even more to ~40% in the MB3 attack.

3) **VT App Fail Rates:** An extended analysis of the innovative attacks of this study was conducted, in light of the apps. Each evasion attack changed a slightly

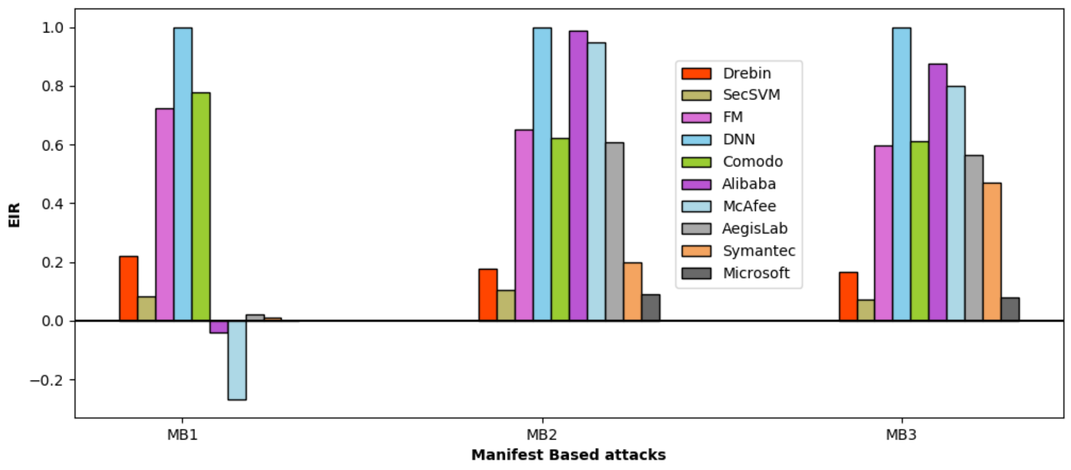

FIGURE 1. Evasion Increase Rates of the three detection systems and six famous VT scanners.

different set of lines in the manifest file of every app. Therefore, an observation of the VT scanner in light of each app might add information on the effect of the evasion attack. The *VT App Fail Rate* defines the difference in the amount of scanners that detect the clean malicious app sample and its manipulated counterpart. For MB1, an average decrease of 7.7 VT scanners was identified. For MB2, the average decrease of VT scanners was 6.2. The average decrease of VT scanners for MB3 was 6.05. In conclusion, each evasion attack decreased the effectiveness of the VT system. As the VT system consists of ∼64 scanners, the *VT App Fail Rate* of ∼ 10% of each evasion attack presents clear maliciousness activity, yet is still effective in the evasion of multiple scanners.

4) **Evasion Trend:** A close inspection was conducted into the trends of the VT scanners. The trends of the scanners in relation to each app sample were investigated in light of the fact that it was established that a constant ∼ 10% of the VT scanners failed to detect the evasion attacks. Table 3 describes the trends of detection rate after the manipulation in comparison to the former app scans. If in the former malware x scanners detected it as malicious and at least $x + 1$ scanners detected the manipulated app as malicious, the app was assigned to the higher trend. Otherwise, if the evasion attack app sample was detected in the second scan by at most $x-1$ scanners, it was assigned to the lower trend. Elsewhere, it was assigned to the equal trend.

In all of the MB attacks, the number of scanners detecting the app decreased after the evasion attack in at least 98.4% of the apps. Less than 1% of the apps in each attack had an identical number of scanners before and after the manipulation. Only MB1 had a 0.7% of the apps with a higher number of scanners after the manipulation in comparison to the former apps. As depicted in Fig. 1, the MB1 evasion attack created an increasing

TABLE 3. Evasion Trends of the MB attacks. The leading trend in each one of the attacks is the lower trend.

	Lower	Equal	Higher
MB1	98.4%	0.8%	0.7%
MB2	99.5%	0.5%	0%
MB3	99.9%	0.01%	0%

ER in some of the scanners (e.g. McAfee). Therefore, the attack created an opposite effect on the system as a whole, of a positive higher trend, as more scanners detected the attack in comparison to the clean data counterpart. In conclusion, the higher trend was found in more than 98% of the apps in each attack. Therefore, the higher trend in Table 3 depicted the proportion of VT scanners that detected the clean data but failed to detect the manipulated data. The effect of the attacks was shown with the subset of six scanners but had an overall effect of the system as a whole. Most of the scanners detected the clean data but failed to detect the manipulated data. Therefore, the higher trend had the lead over the other two trends. These insights should be considered in future research.

To conclude this section, the Drebin and its advanced versions suffered from a decrease in detection rate when faced with the novel MB evasion attacks. In addition, the results show that VT scanners suffer from a great loss in detection rate as a consequence of these evasion attacks. Also, for each malicious app, the evasion attacks caused a decrease of ∼ 6 scanners in average that detected the malicious app as malicious. At last, the evasion trends assured that as a whole, the VT system had a decrease in means of amount of scanners that identify malicious APK samples after this study's evasion attacks.

E. ANDROID HIV

Android HIV [23] (AHIV) is a recent work on Android malware detection systems and evasion attacks. The work evaluated Drebin to find evasion attacks that decrease their detection rates. Their evasion attack, JSMA [95], crafts adversarial examples using the forward derivatives of the classifier. As the MB attacks and AHIV attack are both Problem-Based evasion attacks that target the Drebin classifier, a comparison between the two works was suitable. The authors of AHIV stated that evasion attacks that target the manifest file are risky in terms of app functionality. As stated in Section V-B, the MB attacks target the manifest file. However, as stated in the Section V-B, the manipulation process does not change the malicious activity of the app. Therefore, a comparison between the AHIV and MB attacks is viable.

AHIV includes several attacker models. The first attacker model is F, in which the attacker has access only to the feature set. The next attacker model is FT, where the attacker has access to the feature set and the training dataset. The third attacker model is FB, in which an attacker has access to the defense model as a black-box and the feature set. The FTB attacker model is the final attacker model, which merges the FB and FT attacker models. As the attacker models of AHIV are slightly different from the attacker models of this study, each attack is compared to the closest attacker model.

The MB attacker models were the Drebin Model Access, Data Access and Zero-knowledge. The Drebin Model Access and its MB1 correlated attack incorporate access to the *Drebin reports*. The reports hold the important features of any app that the attacker sends to Drebin. Therefore, it is compared to the FB/FTB attacker models. The MB2 attack and the Data access model include access to the statistics on the training dataset. As such, this model can be referred to as a T model, because the only data it knows is the training dataset. However, the AHIV attacker models include the F and FT attacker models, not a T model. Consequently, the MB2 attack and Data Access model are compared to both F and FT with a reservation. The MB3 attack is a zero-knowledge attack. As a result it uses a much simpler model than the F attacker model and thus can be compared to the F attacker model but the MB3 attacker's knowledge is more limited than the F attacker model. Hence, it is compared to the Baseline detection rate, and the F attacker model-based attack.

The authors of AHIV kindly shared the trained models of their attacks with this research. As their test data was from the Drebin dataset, the MB attacks were run on the Drebin dataset apps and tested against the trained models obtained from the authors. The results of the evasion rates of the MB attacks and the AHIV attacks are summarized in Table 4. As stated above, the MB2 attack was compared to the attacks that are based on the F and the FT attacker models. As such, it attained a better evasion rate than the F attacker model. Compared to the attack that is based on the FT attacker model, the MB2 attack achieved a lower evasion rate. The MB3 attack reached a better evasion rate than the Baseline and the F attacker model's attack. While better than the F attacker model, the attacker in the MB3 attack has no prior knowledge and thus is a more evasive and destructive attack. The MB1 attack trailed behind its correlative attacks by 16%. It involves a more extensive process than MB2 and MB3 and its results are worse.

In conclusion, the MB2 and MB3 evasion attacks performed better than the AHIV evasion attacks. The MB3 was better in each standard, including evasion rate and knowledge of the model/data. The MB2 attack outperformed the F attacker model's attack. It had different knowledge than the F and FT models.

TABLE 4. A comparison of the evasion rates of the AHIV attacker model based attacks and the MB attacks. The JSMA evasion attacks (based on the F, FT, FB, FTB attacker models) of AHIV are compared to the MB attacks, and each MB attack to its relative attack of AHIV.

Baseline	MB3	F	MB2	FT	FB	MB1	FTB
1%	80%	40%	81%	99%	99%	83%	100%

VI. ATTACK EFFICIENCY CLASSIFIER

The analysis of the VT scanners' robustness against evasion attacks in Section V established that the MB attacks decrease the detection rate of the scanners. This decrease was successful although no information on the training dataset of the scanners or on their feature set, nor any information on the scanners' model were provided. Nonetheless, some MB attacks did not achieve high EIRs against specific scanners. For example, the EIR of MB1 attack against the McAfee-GW-Edition was negative. In other words, the attack assisted the scanner in detecting the malicious samples. As the purpose of an evasion attack is to evade detection, the VT analysis marked a dramatic failure of the MB attacks. This case as well as the case of the MB1 attack and the Comodo scanner motivate the need to predict successful attacks.

Additionally, as observed by [96], malicious executable files were found on online sandboxes long before their respective attacks were published publicly. The authors did not provide an absolute statement about the reasons such files were found at the time. One possible explanation, raised by the authors, was that advanced attackers tested their malicious files before sending them to the wild. As a consequence, from the perspective of an attacker, creating a malicious machine that predicts the feasibility of an attack is of great value to malicious entities.

As a result of both the VT analysis and the observation of [96], described above, the following interesting questions emerged:

1) Could an ML-based system predict whether an evasion attack on a given sample is successful or not? The results of the evasion attacks show that a few attacks are detectable quite easily while others demonstrate a high evasion rate. The same question of course is relevant for the targeted systems (e.g. Drebin, Sec-SVM). However, these systems are based on an open-source code implementation. The feature sets are known and the data for the classification is given. Therefore, it is not a great challenge for an attacker to understand how to make a successful attack against these systems.

In contrast, such an information is unknown when dealing with the VT scanners. Therefore, it is more interesting to explore this question with regard to VT scanners. To address this question, a classifier can be implemented. This classifier will be termed an **attack efficiency classifier (AEC)**.

2) A brute force (BF) attacker runs an evasion attack against the VT scanner on each malicious sample it obtains. It can be considered as an AEC that predicts each sample as class 1, where class 1 means a successful attack. The motivation of such a comparison is the cost of an attack, where running an attack costs one unit, and not running it saves one unit. The BF attacker costs n units where n symbolizes the number of malicious samples. An AEC costs $n-t$ units, where t resembles the apps in which the AEC predicts the attack would not succeed. Therefore, the attack should not be run. Can the AEC be more precise than the BF attacker and therefore cost efficient?
3) Does a combination of evasion attacks upgrade the total evasion rate? Given a specific VT scanner and a specific malicious APK, a group of A evasion attacks, and a set of AECs, one for each evasion attack, the following scenario is an option: First, the AEC of the first evasion attack produces a prediction of failure. Then, the AEC of the second evasion attack produces a prediction of success. The other $A-2$ evasion attacks produce a prediction of failure. When each attack is viewed separately, the AEC's success in predicting a successful attack is limited by the success of the specific attack. A combination of attacks, and in correlation, a type of combination between the AECs, may produce a stronger evasion attack and a more complex AEC. In the above case, the choice is clear - the second evasion attack. However, there might be a case where more than one evasion attack on a specific sample is predicted as successful. How should one choose between the attacks? A decision function should be advised, i.e., by randomly picking one of the attacks, or by a more complex approach.

The remainder of the section is organized as follows: First, the formal definitions are listed in Section VI-A. Next, the experiments are described in Section VI-B. Finally, the experimental results are analyzed and discussed in Section VI-C.

A. FORMAL DEFINITIONS

An AEC was implemented to assess the forecast of a successful attack. The data that such an AEC learns is the classification of the original malicious samples. It is more intuitive to produce an AEC with the manipulated features of the malicious apps, as it tries to learn an attack. Such an AEC would try to assess the structure of the attack. However, in such cases, the attacker needs to run the attack for each app it sends to the AEC. When the AEC predicts that an attack will fail, the attacker wastes the running time and resources on this app. Saving the time and resources is the motivation of the AEC. Since the AEC's goal is to predict if an attack on a specific app will be successful, its actual purpose is to forecast the success/failure, not to observe the actual attack procedure. Therefore, the chosen features are the features of the original malicious apps. For each evasion attack, the AEC learns the original features of an app, and a version of the label obtained from the targeted malware detection system it tries to evade for the manipulated version of the app. The label for the AEC is the opposite label gained from the targeted system. In other words, if the targeted system detected the manipulated version, the label for the AEC is 0, and vice versa. Based on the original features, the AEC tries to predict whether the theoretic evasion attack will succeed or not. The AEC requires formal definitions. The following definitions are presented in a general manner to be applied on other targeted systems than the VT scanners as well. Given a malicious sample x, its counterpart evasion attack x', a set of targeted systems S and a set of evasion attacks A, the following definitions apply:

1) **AEC:** For each evasion attack $a \in A$, an AEC C is termed C_a, where a resembles an attack. For example, C_{MB1} is the AEC of MB1. The AEC analyzes the features of the original app x in the training or test processes.
2) **Prediction of a Sample:** The term $C_a(x)$ will be used to describe the prediction of C_a on the original malicious sample x.
3) **AEC Labels:** The term $s(x'_a)$ will be used to describe the label of a system $s \in S$ produces for the malicious counterpart x'_a of x. Similarly, the label of the AEC is denoted $s_a(x)$. Since the AEC works in an opposite manner compared to the scanner, $s_a(x) = \neg s(x'_a)$.
4) **Brute-Force Attacker:** The Brute-Force attacker uses the evasion attack on any app. The AEC will be evaluated against the BF attacker. The AEC and BF attacker will demonstrate the difference between using the attack any time vs using it in a more complex way.
5) **Attack Feasibility:** An ensemble of AECs is defined as a combination of a group of AECs trying to predict the success of a set of evasion attacks A.
An *attack feasibility (AF)* checks if one of the AECs predicted an evasion attack as a success (i.e., $\exists C_a(x) = 1$ s.t. $a \in A$). The AF is tested against a BF attacker, which outputs 1 for every app (in a similar manner to the AEC vs BF).
6) **LR Attacks' Decision Function:** For each case where more than one AEC predicts a successful attack attempt, a decision function is implemented to choose between the AECs. A decision function can take many courses. In this study, a logistic regression ADF_{LR} approach was chosen. First, an LR model, LR, is generated based on the evasion attack data in a similar manner to an AEC. For each app, the three LR models output a probability. The attacks' decision function

outputs the argmax over the multiplication of the *LR* model and the AECs:

$$ADF_{LR}(x) = argmax(LR_a(x) * C_a(x) \mid x \in X, a \in A) \tag{4}$$

7) **Random attacks' decision function:** To test the effectiveness of the decision function, a random choice between the AECs which predict a successful attack is chosen as a baseline, and termed ADF_R. The argmax function of the LR decision function only chooses between successful attack predictions if their amount is more than 1. To match this issue, the ADF_R only chooses between the indexes of the AECs which output 1. First, the indexes of the AECs which result in 1 are defined with an indicator function $\mathbb{1}_A$:

$$\mathbb{1}_A(x) = \{a \mid C_a(x) = 1, a \in A\}$$

Then, the random choose function is defined as follows:

$$ADF_R(x) = random(\mathbb{1}_A(x)) \tag{5}$$

B. ATTACK EFFICIENCY CLASSIFIER EXPERIMENTS

The data obtained from VT in the test apps presented in Section V-C were used in the AEC experiments. In addition, apps from the AMD [97] and StormDroid [98] datasets were used for the AEC's experiments. In total, ~6K malicious apps were used as a dataset for the AEC experiments (a similar number to those used in the former experiments). The results were validated by using 5-fold cross validation. The data of each scanner included one set of malicious app features and three sets of labels - one for each evasion attack variant of the app. PyCaret [99] was used for the evaluation. The VT scanners Comodo, McAfee-GW-Edition, SymantecMobileInsight and Microsoft were selected for the test cases.

The assessment comprised three parts:

1) The F_β score, which serves as a weighted harmonic mean of precision and recall metrics was used to compare several algorithms at once and assess the effectiveness of the AECs. As both recall and precision are important for the assessment, each AEC was optimized once in the precision direction and the other with recall. A range of β values between 0.2-5 with increments of 0.2 were tested. For each β value, the model with the highest F_β score was chosen.[4]
2) After choosing the best model for each β value, a more general overview was observed. For values lower than or equal to 1, the optimization direction was precision. The most frequent model that was chosen for the β values served as the precision model. For β values higher than 1, the recall was optimized, and a model was chosen in a similar manner to the precision model. Therefore, for each scanner and evasion attack, two models were chosen - one that was found to be the most effective in terms of precision for values lower than/equal to 1, and one for the values higher than 1, with recall optimization.
3) Two evaluation metrics were used to evaluate the AECs, Attack Feasibility, and Decision Function:
   a) **Success rate:** The main goal of the AECs is to predict the success rate of an evasion attack. This rate is the true positive rate (TPR) of the labels and the predictions. The TPR in this context measures the amount of predictions of a successful attack, which correlates to an actual successful attack divided by the amount of actual successful attack labels. Therefore, it is a suitable metric to evaluate the accuracy of the AECs predicting a successful attack.
   b) **Save rate:** The second goal of the AECs is to save resources in cases where the AECs predict a failed attack. Therefore, an evaluation metric was created to assess the resources saved by the AECs. The metric for save rate is the true negative rate (TNR). In other words, the part of the apps that was not manipulated due to a prediction of a failed attack, which correlates with actual failed attacks is divided by the amount of actual failed attack labels (which represents apps that should not be manipulated and therefore will not require resources). The amount of resources saved in the attack process is identified in this manner.

[4]An overview of the chosen models can be found in the Appendix.

C. RESULTS

The purpose of the AECs was to predict the success of an evasion attack. Tables 5-7 show the metrics with respect to the AECs, AF and DF of the VT scanners: Comodo, McAfee-GW-Edition (McAfee), SymantecMobileInsight (Symantec), and Microsoft. Each AEC is compared to its correlative Brute-Force attacker, based on their success rates. Moreover, the save rates of the AECs are presented in comparison to no save rates on the part of BF.

The results of the AECs, presented in Table 5, show that each AEC produces an equal or higher success rate than its Brute-Force opponent. The gap between the BF and the AEC lies between few percents, in cases like the MB1 against the Comodo VT scanner with the Precision optimization, which results in a gap of 1%, to cases where the gap can reach 40%, as in the case of the MB3 attack compared to Symantec, with the Precision optimization. For each attack, scanner and optimization factor, the save rate of the AEC is different, ranging from no savings at all in the case of the MB1 attack against the Comodo VT scanner with the Recall optimization, to a save rate of 96% in the case of MB3 compared to the Microsoft VT scanner with the Precision optimization. Also, in most of the cases, the optimization factor did not change the success rate dramatically with regards to the same attack and scanner. The BF attacker did not show any intention of saving resources. This is apparent from its definition, i.e., an attack that attempts to run itself on each sample without any consideration of the waste of resources.

The attack feasibility results in Table 6 show that the ensemble of multiple AECs is still a better choice than a BF attacker. In all of the cases, the attack feasibility success rate is better than the BF attacker. In most of the cases (excluding Microsoft), the difference between the optimization factors is not huge. Also, in most of the cases, the AF shows that the ensemble of attacks leads to a better success rate than its subordinate AECs. This is intuitive, as the group of attacks compensate for the cases where one/two of the attacks fail. However, it is vital for the assessment of the AECs. As in the results of the AECs, the BF attacker does not suggest any savings in resources.

TABLE 5. AECs' results The success (Suc) and save rates for each AEC of the 4 VT scanners: Comodo, McAfee, Microsoft and Symantec. Each AEC is optimized by precision (P) or recall (R). The Brute-Force Success (BF-Suc) rates are presented as well for comparison. No save rate is presented for the BF, in light of the fact that its definition is to attack by means of each app sample.

Scanner	Opt	Attack	Save	Suc	BF-Suc
Comodo	P	MB1	0.27	0.81	0.8
Comodo	P	MB2	0.83	0.86	0.63
Comodo	P	MB3	0.66	0.81	0.63
Comodo	R	MB1	0	0.8	0.8
Comodo	R	MB2	0.54	0.77	0.63
Comodo	R	MB3	0.28	0.7	0.63
McAfee	P	MB1	0.52	0.05	0.05
McAfee	P	MB2	0.46	0.97	0.94
McAfee	P	MB3	0.22	0.88	0.85
McAfee	R	MB1	0.52	0.05	0.05
McAfee	R	MB2	0.39	0.96	0.94
McAfee	R	MB3	0.17	0.87	0.85
Microsoft	P	MB1	0.94	0.18	0.13
Microsoft	P	MB2	0.75	0.26	0.18
Microsoft	P	MB3	0.96	0.4	0.18
Microsoft	R	MB1	0.01	0.13	0.13
Microsoft	R	MB2	0.37	0.22	0.18
Microsoft	R	MB3	0.39	0.24	0.18
Symantec	P	MB1	0.65	0.01	0.01
Symantec	P	MB2	0.57	0.34	0.2
Symantec	P	MB3	0.94	0.93	0.48
Symantec	R	MB1	0.65	0.01	0.01
Symantec	R	MB2	0.57	0.34	0.2
Symantec	R	MB3	0.86	0.87	0.48

The results of the decision functions are presented in Table 7. The Comodo and McAfee's LR decision function produced better success rates than the Random decision function. In the Symantec scanner, the Random function slightly outperformed the LR function. In the case of the Microsoft scanner, the Random function was more successful in forecasting the correct attack. The LR function was believed to be a suitable solution for the decision between attacks, but the random function demonstrated an impressive fight. The change in the success rate between the optimization factors that are incorporated in the AECs for the most part is moderate.

In conclusion, the AECs showed that the predictability of a successful attack is achievable. A BF attacker is a great challenge for the AEC predicting an attack. Nonetheless, the success rate of the AECs is better than that of the BF. Moreover, their save rates demonstrated that wise use of

TABLE 6. Attack feasibility results The success (Suc) and save rates for the attack feasibility (AF) of the 4 VT scanners: Comodo, McAfee, Microsoft and Symantec. Each AF is based on the AECs optimized by precision (P) or recall (R). The Brute-Force Success (BF-Suc) rates are presented as well for comparison. No save rate is presented for the BF, due to the fact that its definition is to attack by means of each app sample.

Scanner	Opt	Attack	Save	Suc	BF-Suc
Comodo	P	AF	0.31	0.93	0.91
Comodo	R	AF	0	0.91	0.91
McAfee	P	AF	0.2	0.97	0.96
McAfee	R	AF	0.29	0.97	0.96
Microsoft	P	AF	0.73	0.52	0.4
Microsoft	R	AF	0.01	0.4	0.4
Symantec	P	AF	0.59	0.69	0.48
Symantec	R	AF	0.58	0.68	0.48

TABLE 7. Decision function results The success (Suc) and save rates for the decision functions (DF-function), both random (DF-Random) and logistic regression (DF-LR) of the 4 VT scanners: Comodo, McAfee, Microsoft and Symantec. Each DF is based on the AECs optimized by precision (P) or recall (R).

Scanner	Opt	Random-Suc	LR-Suc
Comodo	P	0.83	0.85
Comodo	R	0.76	0.81
McAfee	P	0.77	0.87
McAfee	R	0.76	0.86
Microsoft	P	0.48	0.41
Microsoft	R	0.18	0.13
Symantec	P	0.58	0.55
Symantec	R	0.53	0.52

resources does not harm the evasion rate, as their success rate sustained a fair rate against the BF attacker and outperformed the success rate of BF. The attack feasibility showed better results for the BF. In most cases, this advantage is accompanied by a noticeable save rate. The decision functions showed that while one AEC is good, a combination of multiple AECs can create an advanced approach against detection systems. The choice between several attacks is challenging. A logical solution like logistic regression was theorized to be more promising than a random choice. The results show that the LR solution is not the best option for each of the VT scanners. The best decision function is yet to be found.

VII. DISCUSSION

This section discusses the main insights from previous sections. First, the evasion attacks and their mitigation options are discussed, and then the contributions of the AECs, attack feasibility and attack decision functions are presented.

A. EVASION ATTACKS

The successful attacks on the targeted systems were the result of the focus of these detection systems on the permission tags of the Android manifest. As the Android APK has several components, the malicious activity can be encapsulated in other parts of the app, for example, in the Smali code files. As a result, Drebin and its evenly weighted version, Sec-SVM, were less affected by the MB attacks. The DNN and FM detection rates decreased by at least $\sim$70%. The surprising result was that the VT scanners also failed to detect some of the apps, and in some cases like Alibaba in the MB2 attack or McAfee in the MB3 attack, miserably failed to detect the malicious apps. The conclusion of this phenomenon is that the VT scanners also suffer from the problem of excessively relying on the permission tags.

The MB evasion attacks were compared to the AHIV attacks. Fine preprocessing of the Smali code files will identify the manipulations and eliminate them from the code. For example, the following process can identify the non-invoked JSMA functions: First, run an enumeration of signatures of the functions in the Smali code files to get the names of the functions. Store it as *f_names_list*. Second, for each file in the Smali code files, scan the list. Erase a name of a function from the *f_names_list* if it is called in the file (neglecting the signature of the function). Following this process, the *f_names_list* is shortened. Finally, erase each function that remains in the list.

Although the mitigation techniques of both MB and JSMA evasion attacks are easy to implement, the MB attacks have two clear advantages over JSMA attacks. First, their simplicity. The actual manipulation of the code done by the MB attacks is the change of the name of a tag in the manifest file. The amount of the code is constant and small. The code insertion of HIV is also constant but includes a new function addition and an API call for this function. Second, the knowledge the adversary needs to orchestrate the attack. Each attacker model in the JSMA attack is defined with additional data to run the attack, such as: the training set, an AEC's black-box, or the feature set. The MB3 attack does not require any prior knowledge in order to run.

Despite the advantages of the MB attacks, they have several limitations. First, the MB1 attack was not efficient against the McAfee and Alibaba VT scanner. Actually, the attack improved their detection rate. Furthermore, its effectiveness against the other VT scanners was limited (excluding Comodo). The 3 MB attacks caused a small decrease in the detection by the Microsoft VT scanner. Additionally, a preprocess of the manifest file to identify the use of the different name tags will eliminate the effect of the attacks, in the same way it is done in the preprocess of Android HIV.

B. AECs' CONTRIBUTION

The AECs in Section VI demonstrate the importance of good analysis of the feature set to predict the success of an attack. A complex adversary who knows when to run its attack is more challenging than a Brute Force attacker that has a bag of attacks and runs them all. A Brute Force attacker raises more alarms when identified by the detection system. Also, as mentioned in Section VI, it is costly. In future work this cost may be analyzed by means of running time, memory, or resources needed for the attack. In each of the cases, the AECs' success rate was better than the Brute Force attack. The result was a more accurate and efficient model, which means a more engaging and challenging adversary to the targeted malware detection systems.

The attack feasibility and attack decision function showed an improvement in the faults of the MB attacks. The AF showed better performance than the BF in all cases. The save rate of the AF was specific for each scanner and optimization factor. Nonetheless, in most of the cases, the AF saved resources. The results of the attack decision function raised the following question: Is there a perfect model for choosing between the AECs? A subtle function like logistic regression was believed to be a promising choice. However, the random function and the decision function resulted in a tie. The best function to choose between the AECs is still lacking. Overall, it was proven that an ensemble of more than one AEC, with the addition of a simple decision function, fixes the flaws of one evasion attack. Merging the AF and DF creates an attack system that fuses resource maintenance and a smart decision engine between attacks that may threaten various detection systems or anti-viruses.

As the power of these AECs is great for an adversary, a mitigation technique may be suggested. The detection machine may be updated by the use of a random distortion to the labeling mechanism. A flipping of the label of the answer from malicious to benign can be added if the detection machine senses that an attacker is trying to build an AEC against it. This slight change can make it hard for the attacker to create a precise AEC against the detection system. Some parameters such as the distortion extent or the memory needed for this kind of defense mechanism should be taken into account.

VIII. CONCLUSION AND FUTURE WORK

Malware detection systems based on machine learning techniques are widely used in the information security community. These systems are the targets of various evasion attacks. Recent studies show that adaptive and intelligent attackers can easily find the weak spots of such systems. However, such attackers lack the intel of whether the attack will succeed in finding this weak spots or not.

The goal of this work was to shrink this gap. First, this study introduced several innovative evasion attacks against popular Android malware detection systems, like Drebin, Sec-SVM, FM, and DNN. The evasion attacks included the masking of permission requests. It was found that the targeted systems' detection rate decreased, ranging from 13% in the case of Sec-SVM to almost 97% in the case of DNN. Drebin and its successor versions are only one type of Android malware detection systems. Consequently, the exploration of different kinds of Android malware detection systems is still a possible course of action in the future. For example, hitherto it has not been established whether or not these attacks affect hybrid malware detection systems. It may be interesting to check whether or not a detection machine that gathers its feature set from the code files and from the actual run of the app will be affected by the MB attacks. In other words, will changing the manifest file of an APK dramatically change the classification of a hybrid detection system?

The impact of the evasion attacks on VT scanners was presented. It was proved that although the evasion attacks target the open-source detection systems, they decrease the detection rate of popular VT scanners, up to 70%. However, questions such as how these attacks affect other AV frameworks and whether or not they can evade other detection machines without any knowledge on their model remain open. Future work will include other testing frameworks to test the effectiveness of the MB attacks.

Moreover, the attack efficiency AECs were defined. Forecasting a successful attack is a complex process. An absolute strategy was not established and may not exist, but the fundamentals were presented to answer an interesting aspect of evasion attacks: Does a good predictor of a successful attack exist? Is a predictor that uses a tree algorithm better than a neural network with this type of classification task, or perhaps a simple SVM is the best method? If there is a specific predictor that is superior to the others, does a combination of various predictors improve the total forecast? What is the best ensemble method to use? All of these questions are left for future work.

The conclusion of the assessment of the Brute-Force attack and the AECs is simple - running an attack on each malicious APK is not cost effective and may raise too many alarms. The issue of how to measure the costs of an evasion attack is still at large, and is a topic for future research. In addition, the full assessment of the AFs and DFs is incomplete. The correct decision function is an interesting option for improvement. The fundamentals are presented, but the full capacity of using both as a full system is still an open issue. Moreover, the mitigation techniques of the efficiency AECs will be explored in the future.

APPENDIX A COMPARISON OF THE AECs OF THE SELECTED VT SCANNERS

The subset of VT scanners, analyzed in Section VI, were the target of the attack efficiency assessments, to predict the success rate of an attack. This appendix is an overview of the analysis of these scanners and the creation of the attack efficiency AECs. Eighteen classification algorithms were tested for the AEC using the PyCaret Framework [99]. A model was implemented for each algorithm,. The performance of each model was evaluated. For each β value in the range of 0.2-5, a F_β score was calculated for each model. The best model was defined as the model which achieved the highest F_β score for most of the β values. This evaluation was run twice with two different hyper-parameter optimizations: precision and recall. The results are presented in Table 8. Each model is described by the VT scanner (Comodo, McAfee-GW-Edition (McAfee), SymantecMobileInsight, and Microsoft), the evasion attack, the name of the ML algorithm with optimization of precision(p) or recall(r), and three metrics: recall, precision and accuracy. Each pair of models was correlated with a specific VT scanner and the evasion attack was compared to a Brute-Force attacker, which runs the evasion attack on every APK sample.

TABLE 8. Results of the models described by scanner (Scanner), evasion attack (Attack), and specific algorithm (Algorithm). The evaluated scanners included Comodo, McAfee-GW-Edition (McAfee), Symantec, and Microsoft. The optimization factor appears in brackets. Each model was evaluated by means of recall (Rec), precision (Prec), and accuracy (Acc).

Scanner	Attack	Algorithm	Rec	Prec	Acc
Comodo	MB1	LDA(p)	0.8	0.8	0.8
Comodo	MB1	ADA(r)	1	0.79	0.79
Comodo	MB1 - BF	-	1	0.79	0.79
Comodo	MB2	KNN(p)	0.75	0.8	0.72
Comodo	MB2	XGB(r)	0.92	0.78	0.78
Comodo	MB2 - BF	-	1	0.64	0.64
Comodo	MB3	GB(p)	0.89	0.79	0.79
Comodo	MB3	XGB(r)	0.98	0.72	0.75
Comodo	MB3 - BF	-	1	0.64	0.64
McAfee	MB1	RF(p)	0.67	0.05	0.48
McAfee	MB1	RF(r)	0.6	0.06	0.52
McAfee	MB1 - BF	-	1	0.04	0.04
McAfee	MB2	GB(p)	0.99	0.97	0.97
McAfee	MB2	LGBM(r)	1	0.97	0.97
McAfee	MB2 - BF	-	1	0.95	0.95
McAfee	MB3	KNN(p)	0.97	0.87	0.86
McAfee	MB3	XGB(r)	0.98	0.88	0.88
McAfee	MB3 - BF	-	1	0.85	0.85
Symantec	MB1	LR(p)	0.33	0.02	0.71
Symantec	MB1	LR(r)	0.47	0.02	0.67
Symantec	MB1 - BF	-	1	0.01	0.01
Symantec	MB2	RF(p)	0.92	0.36	0.64
Symantec	MB2	RF(r)	0.89	0.35	0.66
Symantec	MB2 - BF	-	1	0.21	0.21
Symantec	MB3	KNN(p)	0.91	0.93	0.92
Symantec	MB3	XGB(r)	0.96	0.86	0.91
Symantec	MB3 - BF	-	1	0.48	0.48
Microsoft	MB1	QDA(p)	0.11	0.14	0.83
Microsoft	MB1	XGB(r)	0.97	0.12	0.15
Microsoft	MB1 - BF	-	1	0.11	0.11
Microsoft	MB2	NB(p)	0.32	0.27	0.69
Microsoft	MB2	XGB(r)	0.88	0.26	0.46
Microsoft	MB2 - BF	-	1	0.2	0.2
Microsoft	MB3	LDA(p)	0.13	0.47	0.82
Microsoft	MB3	XGB(r)	0.87	0.26	0.48
Microsoft	MB3 - BF	-	1	0.2	0.2

It can be seen that in the Comodo VT scanner case, the MB1 models are identical to the BF attacker. For both MB2 and MB3 models, the precision and accuracy were higher than the BF attacker. The McAfee case is more questionable - the recall of the MB1 model was worse $\sim$ 33%. However, the total accuracy was better. The MB2 and MB3 models present a tradeoff between precision and recall. With reference to Symantec, the MB1 model presented similar results to the McAfee MB1 model. The MB2 presented a good tradeoff between recall and precision. The MB3 model presented an effective decrease in recall, which resulted in an impressive increase in precision of more than 40%. Microsoft presented poor performance, with slightly better precision in most cases, accompanied by a significant decrease in recall.

In conclusion, the Comodo scanner models showed promising results in each metric that was inspected. The same is true for MB2 and MB3 models of the McAfee scanner. All in all, in most of the cases of the four VT scanners evaluated, the price of a small decrease in recall was worthwhile.

ACKNOWLEDGMENT
The authors would like to thank Xiao Chen for the Android HIV models. They would also like to thank Prof. Lorenzo Cavallaro, Prof. Pierazzi Fabio, Pendlebury Feargus, and Cortellazzi Jacopo for their help regarding problem-based evasion attacks and their Sec-SVM implementation. Last but not least, a great thank is sent to Noa Hadad, Dean Carmi, and Elon Ezra for their implementation of the algorithms and evaluation system for VirusTotal.

REFERENCES

[1] J. Aycock, *Computer Viruses and Malware* (Advances in Information Security) vol. 22. Springer, 2006. [Online]. Available: https://dblp.org/rec/books/sp/Aycock06.bib, doi: 10.1007/0-387-34188-9.
[2] N. Leavitt, "Malicious code moves to mobile devices," *Computer*, vol. 33, no. 12, pp. 16–19, 2000.
[3] M. Hypponen, "Malware Goes mobile," *Sci. Amer.*, vol. 295, no. 5, pp. 70–77, Nov. 2006.
[4] M. Humphries. (2021). *Over 10m Android Phones Infected With Grifthorse Malware*. [Online]. Available: https://www.pcmag.com/news/over-10m-android-phones-infected-with-grifthorse-malware
[5] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," *Mobile Inf. Syst.*, vol. 4, no. 1, pp. 33–49, 2008.
[6] A. Ojugo and A. Eboka, "Signature-based malware detection using approximate boyer Moore string matching algorithm," *Int. J. Math. Sci. Comput.*, vol. 5, no. 3, pp. 49–62, Jul. 2019.
[7] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in *Proc. 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Jul. 2013, pp. 163–171.
[8] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, 2009, pp. 235–245.
[9] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2013, pp. 400–410.
[10] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.
[11] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.
[12] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, J. G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Trans. Priv. Secur.*, vol. 22, no. 2, p. 14, 2019.
[13] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 711–724, Jul. 2019.
[14] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.
[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.
[16] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proc. USENIX Secur. Symp.*, 2006, pp. 241–256.
[17] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," 2016, *arXiv:1606.04435*.
[18] N. Rndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 197–211.
[19] D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, 2013, pp. 119–130.
[20] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Proc. Netw. Distrib. Syst. Symp.*, 2016, pp. 21–24.
[21] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3886–3900, 2020.
[22] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi, "Android malware detection based on factorization machine," *IEEE Access*, vol. 7, pp. 184008–184019, 2019.
[23] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 987–1001, 2020.
[24] H. Chen, J. Su, L. Qiao, and Q. Xin, "Malware collusion attack against SVM: Issues and countermeasures," *Appl. Sci.*, vol. 8, no. 10, p. 1718, Sep. 2018.
[25] H. Bostani and V. Moonsamy, "EvadeDroid: A practical evasion attack on machine learning for black-box Android malware detection," 2021, *arXiv:2110.03301*.
[26] M. Melis, M. Scalas, A. Demontis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Do gradient-based explanations tell anything about adversarial robustness to Android malware?" 2020, *arXiv:2005.01452*.
[27] Virus Total. (2012). *Virustotal-Free Online Virus, Malware and URL Scanner*. [Online]. Available: https://www.virustotal.com/en
[28] P. Peng, L. Yang, L. Song, and G. Wang, "Opening the blackbox of VirusTotal: Analyzing online phishing scan engines," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 478–485.
[29] A. Salem, S. Banescu, and A. Pretschner, "Maat: Automatically analyzing VirusTotal for accurate labeling and effective malware detection," *ACM Trans. Privacy Secur.*, vol. 24, no. 4, pp. 1–35, Nov. 2021.
[30] H. Berger, C. Hajaj, and A. Dvir, "Evasion is not enough: A case study of Android malware," in *Proc. Int. Symp. Cyber Secur. Cryptogr. Mach. Learn.* Cham, Switzerland: Springer, 2020, pp. 167–174.
[31] H. D. Menéndez, D. Clark, and E. T. Barr, "Getting ahead of the arms race: Hothousing the coevolution of VirusTotal with a packer," *Entropy*, vol. 23, no. 4, p. 395, Mar. 2021.
[32] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," 2018, *arXiv:1804.07998*.
[33] G. Apruzzese and M. Colajanni, "Evading botnet detectors based on flows and random forest with adversarial samples," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–8.
[34] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 119–133.
[35] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "TextBugger: Generating adversarial text against real-world applications," 2018, *arXiv:1812.05271*.
[36] V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android anti-malware against transformation attacks," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, 2013, pp. 329–334.
[37] M. Zheng, P. P. Lee, and J. C. Lui, "Adam: An automatic and extensible platform to stress test Android anti-virus systems," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Berlin, Germany: Springer, 2012, pp. 82–101.
[38] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2013, pp. 387–402.
[39] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.
[40] L. Chen, S. Hou, Y. Ye, and S. Xu, "DroidEye: Fortifying security of learning-based classifier against adversarial Android malware attacks," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Aug. 2018, pp. 782–789.
[41] M. Shahpasand, L. Hamey, D. Vatsalan, and M. Xue, "Adversarial attacks on mobile malware detection," in *Proc. IEEE 1st Int. Workshop Artif. Intell. for Mobile (AIMobile)*, Feb. 2019, pp. 17–20.
[42] R. Shao, X. Lan, J. Li, and P. C. Yuen, "Multi-adversarial discriminative deep domain generalization for face presentation attack detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10023–10031.
[43] Q. Xu, G. Tao, S. Cheng, and X. Zhang, "Towards feature space adversarial attack," 2020, *arXiv:2004.12385*.
[44] I. A. Zikratov, V. Korzhuk, I. Shilov, and A. Gvozdev, "Formalization of the feature space for detection of attacks on wireless sensor networks," in *Proc. 20th Conf. Open Innov. Assoc. (FRUCT)*, Apr. 2017, pp. 526–533.

[45] E. Aydogan and S. Sen, "Automatic generation of mobile malwares using genetic programming," in *Proc. Eur. Conf. Appl. Evol. Comput.* Cham, Switzerland: Springer, 2015, pp. 745–756.
[46] L. Chen, S. Hou, Y. Ye, and L. Chen, "An adversarial machine learning model against Android malware evasion attacks," in *Proc. Asia–Pacific Web Web-Age Inf. Manage. Joint Conf. Web Big Data*. Cham, Switzerland: Springer, 2017, pp. 43–55.
[47] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, *arXiv:1702.05983*.
[48] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao, "Replacement attacks: Automatically impeding behavior-based malware specifications," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Cham, Switzerland: Springer, 2015, pp. 497–517.
[49] G. Zhao, M. Zhang, J. Liu, and J.-R. Wen, "Unsupervised adversarial attacks on deep feature-based retrieval with GAN," 2019, *arXiv:1907.05793*.
[50] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ML attacks in the problem space," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1308–1325.
[51] A. Salem, F. F. Paulus, and A. Pretschner, "Repackman: A tool for automatic repackaging of Android apps," in *Proc. 1st Int. Workshop Adv. Mobile App Anal.*, Sep. 2018, pp. 25–28.
[52] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining API-level features for robust malware detection in Android," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, 2013, pp. 86–103.
[53] *Droidapiminer Code*, ChenJunHero, GitHub, San Francisco, CA, USA, 2018.
[54] K. Wang, T. Song, and A. Liang, "Mmda: Metadata based malware detection on Android," in *Proc. 12th Int. Conf. Comput. Intell. Secur. (CIS)*, Dec. 2016, pp. 598–602.
[55] P. P. K. Chan and W.-K. Song, "Static detection of Android malware by using permissions and API calls," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 1, Jul. 2014, pp. 82–87.
[56] R. Sato, D. Chiba, and S. Goto, "Detecting Android malware by analyzing manifest files," *Proc. Asia–Pacific Adv. Netw.*, vol. 36, nos. 23–31, p. 17, 2013.
[57] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for Android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.
[58] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on Android malware," *Comput. Secur.*, vol. 51, pp. 16–31, Jun. 2015.
[59] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, "Mystique: Evolving Android malware for auditing anti-malware tools," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 365–376.
[60] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in Android and its security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 356–367.
[61] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1987, pp. 369–378.
[62] M. Ikram, P. Beaume, and M. A. Kaafar, "DaDiDroid: An obfuscation resilient tool for detecting Android malware via weighted directed call graph modelling," 2019, *arXiv:1905.09136*.
[63] Y. Piao, J.-H. Jung, and J. H. Yi, "Server-based code obfuscation scheme for APK tamper detection," *Secur. Commun. Netw.*, vol. 9, no. 6, pp. 457–467, Apr. 2016.
[64] M. Sun and G. Tan, "NativeGuard: Protecting Android applications from third-party native libraries," in *Proc. ACM Conf. Secur. Privacy Wireless Mobile Netw. (WiSec)*, 2014, pp. 165–176.
[65] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Comput. Secur.*, vol. 43, no. 6, pp. 1–18, 2014.
[66] A. Shabtai, U. Kanonov, and Y. Elovici, "Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method," *J. Syst. Softw.*, vol. 83, no. 8, pp. 1524–1537, Aug. 2010.
[67] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Mar. 2016.
[68] X. Wang and C. Li, "Android malware detection through machine learning on kernel task structures," *Neurocomputing*, vol. 435, pp. 126–150, May 2021.
[69] B. Kang, B. Kang, J. Kim, and E. G. Im, "Android malware classification method: Dalvik bytecode frequency analysis," in *Proc. Res. Adapt. Convergent Syst. (RACS)*, 2013, pp. 349–350.
[70] T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "TinyDroid: A lightweight and efficient model for Android malware detection and classification," *Mobile Inf. Syst.*, vol. 2018, pp. 1–9, Oct. 2018.
[71] M. Damashek, "Gauging similarity with n-grams: Language-independent categorization of text," *Science*, vol. 267, no. 5199, pp. 843–848, 1995.
[72] A. Martín, R. Lara-Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Inf. Fusion*, vol. 52, pp. 128–142, Dec. 2019.
[73] alexMyG. (2019). *A Framework for Automated Extraction of Static and Dynamic Features From Android Applications*. [Online]. Available: https://github.com/alexMyG/AndroPyTool
[74] H. Wang, Y. Guo, Z. Tang, G. Bai, and X. Chen, "Reevaluating Android permission gaps with static and dynamic analysis," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1–6.
[75] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, Jul. 2015, pp. 422–433.
[76] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha, "Two anatomists are better than one-dual-level Android malware detection," *Symmetry*, vol. 12, no. 7, p. 1128, 2020.
[77] A. Alzubaidi, "Recent advances in Android mobile malware detection: A systematic literature review," *IEEE Access*, vol. 9, pp. 146318–146349, 2021.
[78] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art API call based malware classifiers," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 490–510.
[79] F. Cara, M. Scalas, G. Giacinto, and D. Maiorca, "On the feasibility of adversarial sample creation using the Android system API," *Information*, vol. 11, no. 9, p. 433, Sep. 2020.
[80] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 468–471.
[81] *GooglePlay App Market*, Google, Mountain View, CA, USA, 2008.
[82] C. Wang, L. Zhang, K. Zhao, X. Ding, and X. Wang, "AdvAndMal: Adversarial training for Android malware detection and family classification," *Symmetry*, vol. 13, no. 6, p. 1081, Jun. 2021.
[83] R. Labaca-Castro, L. Muñoz-González, F. Pendlebury, G. D. Rodosek, F. Pierazzi, and L. Cavallaro, "Universal adversarial perturbations for malware," 2021, *arXiv:2102.06747*.
[84] S. K. Sasidharan and C. Thomas, "ProDroid—An Android malware detection framework based on profile hidden Markov model," *Pervas. Mobile Comput.*, vol. 72, Apr. 2021, Art. no. 101336.
[85] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik, "Improving robustness of ML classifiers against realizable evasion attacks using conserved features," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 285–302.
[86] J. Chen, D. Wu, Y. Zhao, N. Sharma, M. Blumenstein, and S. Yu, "Fooling intrusion detection systems using adversarially autoencoder," *Digit. Commun. Netw.*, vol. 7, no. 3, pp. 453–460, Aug. 2021.
[87] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," 2018, *arXiv:1809.02077*.
[88] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," *Proc. Black Hat Abu Dhabi*, pp. 77–101, 2011.
[89] Statecounter GlobalStats. (2021). *Mobile & Tablet Android Version Market Share Worldwide*. [Online]. Available: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide
[90] PYMNTS. (2021). *NuData: Countering Account Hijacking With Behavioral Analytics*. [Online]. Available: https://www.pymnts.com/news/biometrics/2021/nudata-countering-account-hijacking-with-behavioral-analytics/
[91] Y. Li, Z. Yang, Y. Guo, and X. Chen, "DroidBot: A lightweight UI-guided test input generator for Android," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 23–26.
[92] *Drebin Implementation*, Daniel Arp, GitHub, San Francisco, CA, USA, 2014.

[93] V. Niculae. (2016). *Polylearn—A Library for Factorization Machines and Polynomial Networks for Classification and Regression in Python*. [Online]. Available: https://contrib.scikit-learn.org/polylearn/

[94] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 729–746.

[95] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 372–387.

[96] M. Graziano, D. Canali, L. Bilge, A. Lanzi, E. Shi, D. Balzarotti, M. van Dijk, M. Bailey, S. Devadas, and M. Liu, "Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence," in *Proc. 24th USENIX Secur. Symp. (USENIX Security)*, 2015, pp. 1057–1072.

[97] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Bonn, Germany: Springer, 2017, pp. 252–276.

[98] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A streaminglized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 377–388.

[99] M. Ali. (2020). *PyCaret: An Open Source Low-Code Machine Learning Library*. [Online]. Available: https://pycaret.org/

HAREL BERGER received the B.Sc. degree in computer science from Bar-Ilan University, Ramat Gan, Israel, in 2016, and the M.Sc. degree in computer science and mathematics from Ariel University, Ariel, Israel, in 2018, where he is currently pursuing the Ph.D. degree in mobile security and network security with the Department of Computer Science.

CHEN HAJAJ received the B.Sc. degree in computer engineering, the M.Sc. degree in electrical engineering, and the Ph.D. degree in computer science from Bar-Ilan University. From 2016 to 2018, he was a Postdoctoral Fellow at Vanderbilt University. He is currently a Faculty Member with the Department of Industrial Engineering and Management, the Head of the Data Science and Artificial Intelligence Research Center, and a member of the Ariel Cyber Innovation Center. His research interests include machine learning, game theory, and cybersecurity, specifically, the focuses of his work are on how to detect and robustify the weak-spots of AI methods (adversarial artificial intelligence) and personalized and preventative medicine using data-science-based methods.

ENRICO MARICONTI is currently a Lecturer at the UCL Department of Security and Crime Science. He has been part of the SECReT DTC and during his Ph.D. degree, he focused mainly on malware detection. His main focus has been the use of AI and statistical models for detecting automated malicious activities as well as advanced threats on the internet. Alongside this area of research, he is trying to expand the use of AI in detecting other cybercrime activities, such as hate crime on social media.

AMIT DVIR received the B.Sc., M.Sc., and Ph.D. degrees from Ben-Gurion University, Beer Sheva, Israel, all in communication systems engineering. From 2011 to 2012, he was a Postdoctoral Fellow at the Laboratory of Cryptography and System Security, Budapest, Hungary. He is currently a Faculty Member with the Department of Computer Science and the Head of the Ariel Cyber Innovation Center, Ariel University, Israel. His research interest includes enrichment data from encrypted traffic.