On Benign Features in Malware Detection

Michael Cao
Univ. of British Columbia, Canada
michaelcao@ece.ubc.ca

Sahar Badihi
Univ. of British Columbia, Canada
shrbadihi@ece.ubc.ca

Khaled Ahmed
Univ. of British Columbia, Canada
khaledea@ece.ubc.ca

Peiyu Xiong
Univ. of British Columbia, Canada
gbxpeiyu@ece.ubc.ca

Julia Rubin
Univ. of British Columbia, Canada
mjulia@ece.ubc.ca

ABSTRACT

This paper investigates the problem of classifying Android applications into malicious and benign. We analyze the performance of a popular malware detection tool, Drebin, and show that its correct classification decisions often stem from using benign rather than malicious features for making predictions. That, effectively, turns the classifier into a benign app detector rather than a malware detector. While such behavior allows the classifier to achieve a high detection accuracy, it also makes it vulnerable to attacks, e.g., by a malicious app pretending to be benign by using features similar to those of benign apps. In this paper, we propose an approach for deprioritizing benign features in malware detection, focusing the detection on truly malicious portions of the apps. We show that our proposed approach makes a classifier more resilient to attacks while still allowing it to maintain a high detection accuracy.

ACM Reference Format:
Michael Cao, Sahar Badihi, Khaled Ahmed, Peiyu Xiong, and Julia Rubin. 2020. On Benign Features in Malware Detection. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE'20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3324884.3418926

1 INTRODUCTION

The popularity of mobile phones has increased rapidly in the past decade. Together with the increased popularity, their wide adoption greatly stimulated the growth of mobile malware. A number of approaches have recently emerged to support Android malware detection [1, 2, 4, 5, 13, 14, 16, 17, 21, 22, 26, 27, 29]. Most approaches rely on extracting application (app) features and then training a machine learning classifier to distinguish between benign and malicious apps based on these features. They are typically evaluated on malware from the Malware Genome Project [28], Drebin [4], AMD [25], and VirusTotal Academic [23] datasets and on contemporary benign apps from the Android Google Play app store. The results of the evaluation show high malware detection abilities, typically with an accuracy of 90% and above.

To reproduce and study the reasons behind these high detection ratios, we experimented with a popular malware detection tool Drebin [4], which captures Android app characteristics by extracting features from the app's code and Android manifest file and further uses SVM [8] as the underlying classifier. We choose Drebin as it is known for its high accuracy and runtime efficiency [14, 18]. We used malicious apps from six recent snapshots of the VirusTotal Academic dataset [23] that were available to us (2016–2019). For benign apps, we sampled the AndroZoo repository [3], selecting apps that appeared in Google Play, were deemed benign by all antiviruses on VirusTotal, and corresponded in years to our malicious samples. We refer to the produced dataset as VT.

We performed time-aware experiments [19], training Drebin on malware and benign samples from earlier years and testing it on later years, while controlling for the size of the training and testing sets (see Section 2 for details). Our experiments showed a very high detection accuracy (F-measure of 96%), consistent with reports in earlier work. However, when interpreting the model learned by the tool and studying apps used in training and testing, we observed that the classification decisions are often based on the presence/absence of features deemed benign by the classifier rather than those deemed malicious.

Problem Description. Despite the high accuracy, this learned pattern is ineffective and even harmful. We demonstrate that by defining and executing two types of attacks: (1) a lightweight mimicry attack [24], where the attacker is assumed to be able to collect a surrogate dataset of representative benign samples (e.g., from Google Play) and to know the feature space used for classification and (2) a black-box invasion attack, where the attacker has access to benign samples but not the classifier feature space. For both attacks, we show that malicious apps can easily bypass detection just by adding features commonly used by benign apps in the surrogate dataset.

To investigate the reasons behind the success of the attacks, we inspected the most frequent features used by benign and malicious apps of our dataset, collectively. Figure 1 shows the difference between the fraction of malicious and benign apps using each of the 200 most frequent features in the VT dataset. Intuitively, popular features more common in malicious apps appear above the x-axis, popular features more common in benign apps appear below it, and the length of the bar represents the magnitude of the difference in feature appearances in malicious vs. benign apps.

The figure shows that 59 of the most frequently-used features (29.5%) originate primarily from benign apps. Moreover, almost half of them appear in 30%–50% more benign than malicious apps.

ASE'20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3418926

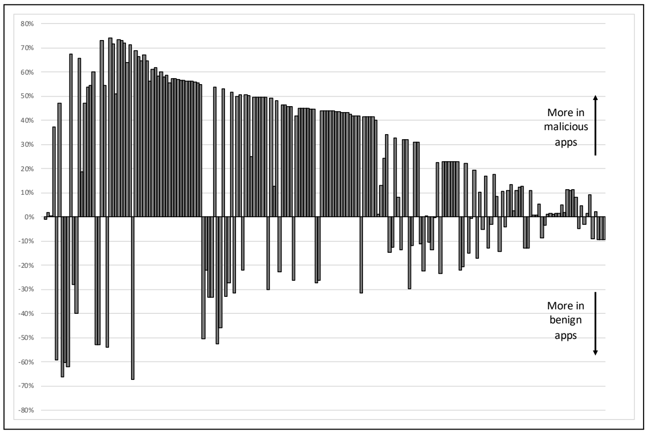

Figure 1: 200 most popular features in the training dataset

Some examples of such features include user account management to avoid re-authentication, notifications and pop-up messages, and logging used for debugging purposes. These features reflect functionalities that are beneficial for users and app developers. Malware developers do not have incentives to include such functionalities: their goals are rather to minimize interactions with the user, stimulate re-authentication for stealing user credentials, etc. They also do not intend to troubleshoot and improve their app, as long as the malicious behavior can occur. As a result, features corresponding to these behaviors are missing in malware datasets.

A classifier learning from such data is likely to assign high benign weights to features predominately used in benign. While that helps to achieve a high detection accuracy, it opens the door to attacks, i.e., simply adding benign functionality to malicious apps, such as pop-up messages and logging, can flip the detection from malicious to benign. In fact, the API responsible for successful re-authentication is the 12th highest-weighted benign feature learned by Drebin. Adding this feature by itself to the previously correctly classified malware from the VT dataset flips the detection result in 127 out of 4404 cases.

Main Idea and Novelty. To address this problem, we propose to *deprioritize features originating predominantly from benign apps*, ensuring that the classification decisions are made based on malicious rather than benign indicators. The main insight behind our work is that malicious apps are capable of including behaviors similar to those of benign apps (in addition to the malicious payload). Learning benign indicators thus increases the attack surface and should be avoided.

Specifically, we propose an approach that decreases the number and the weights of benign features learned by an SVM classifier, improving its reliability while maintaining high detection accuracy. Our work targets benign features only and thus differs from prior approaches that address the problem by manipulating the space of both malicious and benign features, e.g., Demontis et al. [9] who improve the robustness of Drebin by evenly distributing all feature weights and Chen et al. [7] who transform the binary feature space into continuous probabilities encoding the distribution in each class (either benign or malicious).

Table 1: Samples –VT dataset

Year	#M/#B Samples	Malware Size (MB)	Benign Size (MB)
2016	1,544/1,544	3.0	9.1
2017	5,019/5,019	5.3	11.7
2018	4,513/4,513	3.0	12.4
2019	200/200	3.2	13.0

Additional approaches proposed in earlier work suggest to train a one-class classifier (an outlier detector) on either benign samples only [5] or malicious samples only [1]. The former approach is not effective as it will mostly learn benign features and will deem malware having such features benign. The latter approach will have a similar problem and can deem benign apps malicious. Our work differs as we train a classifier on features extracted from both malicious and benign samples while using features extracted from benign samples as counter-examples, i.e., examples of "non-maliciousness", rather than major factors in classification decisions. Our preliminary evaluation of the proposed approach shows that it is effective for identifying malware in presence of benign behaviors.

In what follows, we describe our experimental setup, our approach, and preliminary results supporting the ideas proposed in this work.

2 EXPERIMENTAL SETUP

Subject Datasets. To construct a set of recent malware samples, we collected samples from the VirusTotal dataset [23]. The complete dataset contains around 1.5 billion samples. For academic purposes, VirusTotal provides snapshots of randomly selected samples, known as VirusTotal Academic. We acquired six academic snapshots between the years 2016 and 2019 and removed samples duplicated in these snapshots.

We collected benign samples from AndroZoo [3] – a popular repository with over 9.5 million Android apps from various markets. We randomly sampled benign apps from AndroZoo, ensuring the apps are from the same time period as our malicious samples. If there were multiple versions of an app, we collected the latest sample. To ensure all our apps were truly benign, we additionally filtered for apps that were not detected by any of the anti-viruses on VirusTotal. The overall distributions of samples and the average size of a sample (apk file), per year, is given in Table 1.

We performed time-aware experiments, training on samples from earlier years and testing on samples from later years [17, 19]. Specifically, we trained on apps from 2016–2017 (13,126 apps in total) and tested on apps from 2018–2019 (9,426 apps in total). We refer to these datasets as training and testing, respectively. To perform a uniform experiment and to avoid biasing the classifiers toward either malicious or benign apps, we chose to perform our experiments on a balanced dataset using a 50-50 ratio [2].

Malware Classification. Drebin [4] is one of the first approaches for Android malware detection. Yet, it is still one of the most accurate and performant. It uses a lightweight static analysis to extract eight different types of features from the app bytecode and manifest file: requested permissions, such as permission to access the Internet or device info; permissions actually used by the app; main app components [11], such as activities and services; used hardware components [10], such as Bluetooth and camera; used intent filters [12], which are responsible for the inter- and intra-component communication; API calls guarded by permissions; suspicious API calls, such as `getDeviceId()`; and network addresses encoded in the app. Drebin further embeds all features into a multi-dimensional vector space, where each dimension is either 0 or 1, and uses that as input to a linear Support Vector Machine (SVM) [8]. The authors of the tool shared the implementation of the feature extraction component with us and we implemented the training and testing part in consultation with the authors.

3 PROPOSED APPROACH

We now present our approach for deprioritizing benign features in malware detection, called BClean. Similar to existing feature normalization techniques, e.g., [15], BClean operates on the vector space V of feature values extracted from malicious and benign apps in the training dataset. It transforms V into an augmented vector space V', while modifying (some of) the values. However, unlike existing normalization techniques, BClean focuses only on features that appear in more benign than malicious apps in the training dataset (i.e., features below the x-axis in Figure 1), aiming to prevent the classier from deeming these features as strong indicators of "benignness".

As discussed earlier, Drebin extracts binary features – either 0 or 1 – which indicate the presence or absence of a particular feature in an app. It then uses linear SVM to find the hyperplane that maximizes the margin between the malware and benign classes. SVM assumes that the data it works with is in a standard range (0 to 1 in case of Drebin); otherwise the values have to be normalized because features in different ranges skew classification accuracy: for example, when the values of feature f_1 range from 0 to 1000 and of feature f_2 – from 0 to 1, f_1 becomes more important and dominates f_2 when looking for the hyperplane.

We leverage that property to reduce the importance of the features that appear predominantly in benign samples. Our approach is inspired by the Inverse Document Frequency (IDF) metric used in information retrieval for reflecting the importance of a word given the collection of documents by assigning lower weights to words that appear in many documents [20]. Similarly, BClean captures the importance of a feature in the training dataset by assigning lower weights to features that appear in more benign than malicious apps. It does so proportionally to the difference in appearance of that feature between the classes. Specifically, given a feature f with a value f_v (either 0 or 1) that appears in $f_{\#B}$ benign and $f_{\#M}$ malicious training samples, BClean transforms f_v as follows:

$$\textsc{BClean}(f) = \begin{cases} \dfrac{f_v}{f_{\#B} - f_{\#M}} & \text{if } f_{\#B} > f_{\#M} \\ f_v & \text{otherwise} \end{cases}$$

As the values of $f_{\#B}$ and $f_{\#M}$ are constant for a given training dataset, the value of f is modified consistently in all feature vectors extracted from the training samples. Moreover, features used more frequently in benign vs. malicious apps get proportionally smaller values, deprioritizing these features for SVM.

4 PRELIMINARY EVALUATION

We evaluate the accuracy and reliability of our proposed solution under the two attacks described below. We compare it with the baseline Drebin implementation (denoted by Orig) and with an implementation where we applied the standard IDF normalization instead of BClean (denoted by IDF).

4.1 Attack Models

We define two types of attacks. The first is a *lightweight mimicry attack* [6, 24], where the attacker is assumed to know the feature space and be able to collect a surrogate dataset of representative benign samples (e.g., from Google Play). We apply the attack on all malware samples from our VT training dataset. For the benign surrogate dataset, we use samples from VT testing as these samples are representative benign applications from Google Play within the appropriate time range. We assume the attacker extracts features used by apps from this dataset, orders them by the frequency of usage, and then injects features to malicious samples based on that order, aiming to make malicious samples look more benign. Our attack is lightweight as the attacker is assumed to have no access to a dataset of malicious samples.

In the *black-box invasion attack*, the attacker is only able to collect the surrogate dataset of benign apps (as described above) but has no access to the classifier feature space. The attacker then combines a malicious sample with a randomly picked benign app, aiming to hide the malicious behavior. This is a simpler, more brute-force attack, which can be easily executed in practice. To evaluate the attack, we pair each malware sample in the VT testing dataset with a benign sample in our surrogate dataset; we refer to the produced dataset as VTA (for VT-augmented).

4.2 Results

To assess detection accuracy, we calculated precision (P), recall (R), and F-Measure (F) achieved by each solution when detecting malware. Precision represents the ratio of malicious samples that are correctly predicted as malicious out of all samples predicted as malicious. Recall represents the ratio of malicious samples that are correctly predicted as malicious out of all truly malicious samples. F-measure is a harmonic mean aiming to balance between precision and recall, calculated as $2 \times \frac{P \times R}{P + R}$.

The first three rows in Table 2 show the accuracy measures for the Orig, IDF, and BClean solutions. Consistently with experiments reported by prior work, we obtained a very high detection accuracy for Orig (99% precision, 94% recall, and 96% F-measure) and the accuracy of both IDF and BClean solutions is equivalently high (97% and 96% F-measure, 95% and 93% recall, respectively), attesting to the high ability of all these solutions to separate malicious and benign samples in the training data.

Performance under mimicry attack. To focus on malware detection capabilities and fairly compare different solutions under this attack, we fix the benign samples misclassification rate at 1% for each solution, and calculate the recall while varying the number of injected features. Figure 2 shows the results of this experiment, where the x-axis shows the number of injected features (from 0 to 200) and the y-axis shows the detection rate (recall).

Table 2: Accuracy of Orig, IDF, and BClean

Solution	Training and Testing of VT		
	P	R	F
Orig	99	94	96
IDF	99	95	97
BClean	99	93	96
Solution	Black-Box Evasion Attack		
	P	R	F
Orig	98	67	80
IDF	99	63	77
BClean	99	88	93

While the detection rate of both Orig (solid line) and IDF (dotted line) rapidly decreases, dropping to 50% after just 24 infected features, BClean maintains a high detection rate of above 80% until 108 injected features and drops to below 50% at around 140 injected features, attesting to the improved resilience of BClean against the mimicry attack.

When inspecting the reasons for that improvement of BClean when compared with the baseline approaches, we observed that the 20 most frequent features in the benign surrogate dataset (which are used for the attack first) are also frequently used in the training dataset: they rank from #1 to #68 in the list of most frequent features. Almost 85% of them (17 out of 20) appear in more benign than malicious training apps (below the x-axis in Figure 1). As such, Orig and IDF are more likely to deem these features strongly benign for classification purposes.

In fact, we calculated the fraction of these 20 features that are placed in the upper 10th percentile of all benign features learned by a solution. We use this metric because one cannot directly compare feature weights between different solutions as the weights are not normalized. For Orig and IDF, 11 (55%) and 14 (70%) of the 20 most frequent features in the benign surrogate dataset, respectively, are in the upper 10th percentile of all benign features learned by a model in training. However, BClean successfully deprioritized these features, placing only 2 of them (1%) in the upper 10th percentile.

As one example, the feature corresponding to the re-authentication scenario described in the introduction appears in 2% malicious and 50% benign application in the training dataset. In the trained model, it is ranked as the 12th top benign feature by Orig, 24th by IDF, and only 7716th by BClean, which is able to successfully reduce the importance of this feature.

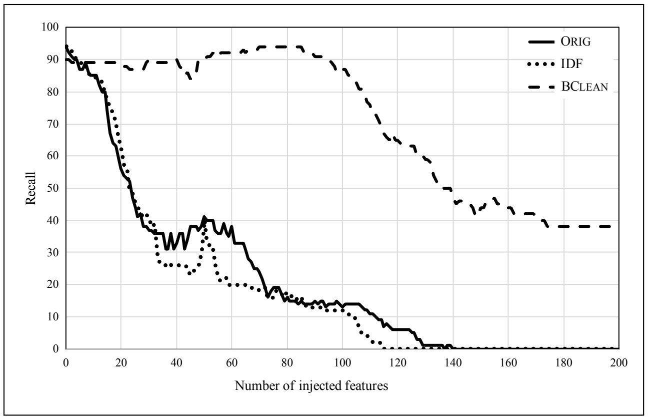

Figure 2: Mimicry attack on VT

Furthermore, we calculated the fraction of the 200 most frequent features used in benign apps (and injected in this attack) that are in the upper 10th percentile of all benign features learned by a solution. Our calculations show that only 25 of these features (13%) are in the upper 10th percentile of benign features learned by BClean, while this number is 85 (43%) and 56 (28%) for the Orig and IDF solutions, respectively. This demonstrates the effect of applying BClean and its ability to reduce the importance of frequent benign features used in classification decisions.

Performance under black-box evasion attack. Rows 4–6 in Table 2 show the accuracy of all three models when classifying the VTA samples produced in the black-box attack. The overall F-measure achieved by Orig under the attack is 80% (vs. 96% for the original testing experiment in row 1 of the table), with a recall of 67% (vs. 94% in row 1). The F-measure and recall of IDF also drop to 77% and 63%, respectively (vs. 97% and 95% in row 2 of the table). However, the F-measure and recall of the BClean solution remains high under this attack as well, at 93% and 88%, respectively (vs. 96% and 93% in row 3 of the table).

Pairing malicious and benign apps for the VTA dataset adds 39 features on average to each malicious sample; on average, only 2 of these features are in the upper 10th percentile of benign features learned by BClean, while this number is 14 and 10 for the Orig and IDF solutions, respectively. As features added under this attack are not as effective in flipping the detection accuracy as features selected in mimicry attack, the detection accuracy for Orig and IDF does not drop as low as in the experiment in Figure 2.

Yet, the drop for these solutions is still substantial while BClean is able to maintain a high accuracy also under these brute-force attack model that can easily be carried out by the most naïve attacker. This experiment, again, demonstrates the effectiveness of our solution to sustain attacks by deprioritizing benign features used in detection.

5 CONCLUSION

In this paper, we investigated the problem of using benign features for malware detection. Our work shows that the high detection accuracy of machine-learning-based approaches trained on datasets commonly used in academic literature does not necessarily stem from a real ability to detect malware. Instead, these approaches effectively become benign detectors, which makes them prone to evasion attacks. We proposed an approach for de-prioritizing benign features used in malware detection and compared it with two baseline techniques, w.r.t. to both accuracy and resilience to attacks. The results of our experiments show that our proposed approach is efficient for achieving accurate and reliable detection outcomes, outperforming the baseline techniques in two different types of attacks. As part of our future work, we intend to apply our technique to additional malware classifiers and datasets of malicious samples. We also intend to further investigate and interpret the reasons for correct classification decisions, i.e., by separating between cases where malware is correctly classified based on its similarity to other malware vs. cases where malware is correctly classified based on the lack of benign features.

REFERENCES

[1] Shahid Alam, Zhengyang Qu, Ryan Riley, Yan Chen, and Vaibhav Rastogi. 2017. DroidNative: Automating and Optimizing Detection of Android Native Code Malware Variants. *Computers & Security* 65 (2017), 230–246.

[2] Kevin Allix, Tegawendé F. Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 2016. Empirical Assessment of Machine Learning-Based Malware Detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proc. of Working Conference on Mining Software Repositories (MSR)*. 14–15.

[4] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. of Network and Distributed System Security Symposium (NDSS)*. 23–26.

[5] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining Apps for Abnormal Usage of Sensitive Data. In *Proc. of International Conference on Software Engineering (ICSE)*. 426–436.

[6] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks Against Machine Learning at Test Time. In *Proc. of uropean Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*. 387–402.

[7] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. 2018. Droideye: Fortifying Security of Learning-Based Classifier Against Adversarial Android Malware Attacks. In *Proc. of International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 782–789.

[8] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[9] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2017. Yes, Machine Learning Can Be More Secure! a Case Study on Android Malware Detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)* 16, 4 (2017), 711–724.

[10] Google Developers. 2020. Android Hardware. https://developer.android.com/reference/android/hardware/package-summary. last accessed August 2020.

[11] Google Developers. 2020. Application Fundamentals. https://developer.android.com/guide/components/fundamentals. last accessed August 2020.

[12] Google Developers. 2020. Intents and Intent Filters. https://developer.android.com/guide/components/intents-filters. last accessed August 2020.

[13] Yanick Fratantonio, Antonio Bianchi, William Robertson, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2016. TriggerScope: Towards Detecting Logic Bombs in Android Applications. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 377–396.

[14] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 3, Article 11 (2018).

[15] Anil K. Jain and Richard C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice-Hall.

[16] Jie Liu, Diyu Wu, and Jingling Xue. 2018. TDroid: Exposing App Switching Attacks in Android with Control flow Specialization. In *Proc. of International Conference on Automated Software Engineering (ASE)*. 236–247.

[17] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of Network and Distributed System Security Symposium (NDSS)*. 1–12.

[18] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. 2017. Context-Aware, Adaptive, and Scalable Android Malware Detection Through Online Learning. *IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI)* 1, 3 (2017), 157–175.

[19] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proc. of USENIX Security Symposium (USENIX)*. 729–746.

[20] Juan Ramos. 2003. Using TF-IDF to Determine Word Relevance in Document Queries. In *Proc. of Instructional Conference on Machine Learning (ICML)*. 133–142.

[21] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. 2016. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing (TDSC)* 15, 1 (2016), 83–97.

[22] Mingshen Sun, Xiaolei Li, John C.S. Lui, Richard T.B. Ma, and Zhenkai Liang. 2017. Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android. *IEEE Transactions on Information Forensics and Security (TIFS)* 12, 5 (2017), 1103–1112.

[23] VirusTotal. 2020. VirusTotal. https://www.virustotal.com/home. last accessed August 2020.

[24] David Wagner and Drew Dean. 2001. Intrusion Detection Via Static Analysis. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 156–168.

[25] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *Proc. of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 252–276.

[26] Wei Yang, Mukul R. Prasad, and Tao Xie. 2018. EnMobile: Entity-based Characterization and Analysis of Mobile Malware. In *Proc. of International Conference on Software Engineering (ICSE)*. 384–394.

[27] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. 2015. AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context. In *Proc. of International Conference on Software Engineering (ICSE)*. 303–313.

[28] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android Malware: Characterization and Evolution. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 95–109.

[29] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *Proc. of Network and Distributed System Security Symposium (NDSS)*. 50–52.