A Non-Intrusive Machine Learning Solution for Malware Detection and Data Theft Classification in Smartphones

Sai Vishwanath Venkatesh[1][0000−0001−6568−6259], Prasannakumaran D[2], Joish J Bosco[1], Pravin Kumaar R[1], and Vineeth Vijayaraghavan[1]

[1] Solarillion Foundation, Chennai, India
{saivishwanathv,pravin.kumaar99,vineethv}@ieee.org
joishbosco99@gmail.com
[2] SSN College of Engineering, Chennai, India
prasannakumaran18110@cse.ssn.edu.in

Abstract. As smartphones strive to provide more versatility and functionality to satiate their growing demand, more user data becomes vulnerable and exposed to attackers. Successful mobile malware attacks could steal a user's location, photos, or even banking information. Due to the lack of post-attack strategies, firms also risk going out of business due to data theft. Thus, there is a need to not only detect malware intrusion in smartphones but to also identify the data that has been stolen in order to assess, aid in recovery and prevent future attacks. In this paper, we propose such a machine learning solution* which is accessible, non-intrusive and can perform intrusion detection and stolen data classification for any app under supervision. We do this with Android usage data obtained from publicly available data collection framework– *SherLock*. We test the performance of our architecture for multiple users on real-world data collected using the same framework. Our architecture exhibits less than 9% inaccuracy in detecting malware and can classify the type of data that is being stolen with 83% certainty.

Keywords: Data Classification · Malware Detection · Cybersecurity · Smartphone

1 Introduction

Currently, Android has more than 1.6 billion active users, which accounts for more than 70% of the global market share of mobile operating systems. As a result, the application market for Android is flooded with apps. We define *malicious app* or *malware* as Android applications that present themselves to the user as benign, but secretly steal user information in the background. Although the Android application store (Google Play Store) verifies apps for malicious intent upon release, it does not aggressively track updates from these verified

*https://github.com/PrasannaKumaran/AndroidDataTheft

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

apps and cannot account for third-party apps downloaded independently by the user. A report released in 2020 by McAfee Advanced Threat Research and Mobile Malware Research [18] suggests that malware developers roll out malware through verified apps in Google Play as updates to shield themselves from preliminary verification. Undetected malware attacks can steal sensitive information from users such as photos, documents and browsing data. Data breaches are extremely disastrous for small and midsize firms and businesses. A report by the U.S. Securities and Exchange Commission [21] states that 60% of small firms can not recuperate from data breaches and go out of business within 6 months. The IBM "Cost of a Data Breach Report 2020" [12] suggests that companies establish an incident response (IR) plan to determine the damage done by the breach and contain it as soon as possible. It goes on to state that companies with an IR plan save an average of $2 million in the event of a data breach. Furthermore, the report projects an increase in the costs of data breaches due to the COVID-19 pandemic and the increase in digital reliability. This calls for a need to not only detect malicious attacks but also identify the stolen data to assess the damage, strategically recover and prevent future attacks. Performing this can help in understanding malware trends and aid in malware prevention research.

We propose a novel two-stage machine learning approach to detect malicious attacks for any app under supervision and identify the data stolen by the attack to aid in assessment and recovery.

The course of this paper is as follows: Section 2 discusses relevant research in the field of malware detection. In Section 3, we describe the dataset used in our study extensively. We elucidate the steps taken to make the data computationally feasible in Sections 4 and 5. Later, in Section 6 we outline our model architecture and describe the parameters of its evaluation. In section 7 we report and discuss our findings. Finally we conclude our work and discuss future scope to this research in Section 8.

2 Related Work

Mobile malware detection has been an active and broad area of research for the past several years. Static analysis was one of the first major mobile malware detection approaches proposed [10, 19]. Here, the source code of the target malware is analyzed to identify semantic signatures. Although static analysis can detect malware even before running the app, static analysis systems fail when the malware uses obfuscation techniques such as code encryption and repackaging. Dynamic analysis techniques [6, 9] address code obfuscation and encryption in malware detection by executing the source code of the application in an isolated environment to analyze runtime characteristics based on frequency. However, this proves to be a bottleneck in systems that use dynamic analysis as clean and noiseless data is hard to achieve and implement in real-world scenarios. Static and dynamic methods additionally require super-user (root) access since they require source code to be executed. Furthermore, Moser et al. [16] suggest that

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

the rate of developing rule-based solutions can not match the fast rate of new malware released to the world. Thus, these solutions will fail to perform for new malware since they are rule-based and discrete solutions.

Machine learning approaches were introduced to swiftly aid in detecting new malware as they are released. Notable works using these approaches include [4, 5, 8, 22] that outperform static and dynamic methods by modelling network usage for detection. Bläsing et. al [6] used various anomaly detection methods to detect malware using system and network data collected. Ronen et. al [17] goes on to detect and classify the family of detected malware by analysing Dalvik bytecode from Android devices. However, these works fail to address the security risk for any end user trying to obtain bytecode. This exposes the phone to further vulnerabilities due to the need for root access. There is a need for non-intrusive malware detection systems based on low privilege information such as usage statistics. This would allow easier user applicability and ensure better security over super-user vulnerabilities.

We propose modeling malware on usage statistics data and we consider one of the largest and most granular datasets for mobile sensor and software sampling - Sherlock dataset. As a result of the dataset's versatility, it is suitable for a multitude of use-cases. Since it does not require root access to probe its data, it is safe and reproducible for malware detection. Zheng et al. [24] explored usage patterns, relationship between mobile usage and the state (benign/malicious) of the application for this data. Wassermann et al. [23] used low-level system features from this dataset coupled with sampling techniques to deal with the inherent class imbalance and detect malicious actions performed on a smartphone.

Although current research tackles malware detection extensively it fails to address data theft classification to aid damage assessment and recovery from data breaches.

We use the SherLock dataset to develop a machine-learning based malware detection pipeline that is capable of identifying the type of data stolen.

3 Dataset

The SherLock Dataset [15], spanning over 10 billion records and involving over 50 volunteers is the result of a real-world data collection experiment to obtain low-level Android usage data alongside emulated malware. Such statistics do not require root access, therefore making any solution developed on the dataset more secure under real-world circumstances since rooting exposes a mobile phone to further vulnerabilities.

The experiment introduces two data collection agents to the mobile phones provided to the volunteers – *Sherlock* and *Moriarty*. *Moriarty* emulates malicious actions on the volunteer's mobile phones randomly through the course of the experiment, generating distinct labels between malicious and benign actions. Meanwhile, *Sherlock* logs usage attributes and statistics in the background.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

Table 1. Categories of Data Theft

Malware Service Type	Target information
Contacts	Phonebook data
GPS	User coordinates (latitude and longitude)
URL	Web address of every page visited by the user recently
Audio Records	Audio records collected during the session
Contacts	Names and Phone numbers
BrowserInfo	Account details, bookmarks and browser history
Photos	Images from gallery

3.1 Sherlock Data Collection Agent

One of the ways Sherlock logs phone attributes is through *Pull Probes* which extract data periodically at a constant sampling rate. For our experiments we consider the most frequently sampled pull probe in Sherlock named *T4*, which has a sampling rate of 5 seconds. T4 probes Global System Features as well as Local Application Features.

Global System Features (GSF): These features pertain to attributes with a global scope in the Android system such as network traffic, CPU and memory utilization, I/O interrupts and Wi-Fi related data. There are a total of 128 Global System Features.

Local Application Features (LAF): Alongside Global System Features, Linux-level data [1] for every running application is sampled. This includes process-specific features such as the scheduling priority, number of bytes transferred, number of threads and kernel-level features used by an application at the time instant. There are a total of 56 Local Application Features.

Local Application Features used in context with Global System Features together provide a rich feature set to determine if a given app exhibits malicious behaviour.

3.2 Moriarty Malicious Agent

Moriarty presents itself to the user as a benign application, such as a game or a browser depending on the version of the app but covertly performs malicious actions. The malware emulated by each version is dissimilar to its precursor and targets different vulnerabilities in each version as illustrated in Table 1. The malware used by Moriarty are behavioural copies of malware found in the real-world.

The app contains labels indicating whether an action executed is benign or malicious. Furthermore, the details of malicious actions such as the type of data stolen, number of bytes transmitted and time taken to transfer the stolen information are logged along with the labels. To collect sufficient information for the experiment, the volunteers were reminded to use the Moriarty app at least once every couple of days.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

For our experiments we have considered a computationally feasible subset of the SherLock dataset. It consists of data collected during the first quarter of 2016, with over 300 million records, spanning across 5 users.

4 Data Pre-processing

We aim to enable efficient data merging between Local Application Features (LAF) and Global System Features (GSF). Let g and n denote the number of GSF and LAF. Assuming there are m apps running at the same time, each Global System Feature would correspond to multiple LAF at that instant of time. The vector space of application data (LAF at time t), denoted by Ω_t for any time instant t is represented in Equation (1).

$$\Omega_t = \begin{Bmatrix} \omega_{11} & \omega_{12} & \dots & \omega_{1n} \\ \omega_{21} & \omega_{22} & \dots & \omega_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{m1} & \omega_{m2} & \dots & \omega_{mn} \end{Bmatrix} \tag{1}$$

Consequently, if a relational join operation between GSF and LAF was performed it would lead to the generation of GSF duplicates for every running application with a shape of $(m,\ g+n)$. The size of this data denoted by S_{np} is $m*(g+n)$ memory units. With the dataset spanning over 300 million records, it becomes essential to reduce memory consumption to expedite the data handling and modeling process. Therefore to overcome duplicates, Ω_t is transformed into a row vector of shape $(1,\ m*n)$ by performing $PIVOT$ operation represented in Equation (2), thus obtaining a functional dependency with time.

$$PIVOT(\Omega_t) := \{\omega_{ij} \mid i \in M \ and \ j \in N\} \tag{2}$$

M = Set of all applications on the device
N = Set of local application features

As a result of using $PIVOT(\Omega_t)$ to merge with GSF as opposed to using Ω_t, we obtain a shape of $(1,\ g+m*n)$ and size of this data denoted by S_p is $1*(g+m*n)$. The size comparison of the data obtained from merging GSF with and without pivot operation is illustrated in Equation (3). Therefore, with an increase in the number of applications the overall throughput decreases.

$$\begin{aligned} & g+m*n << g*m+m*n \\ \implies \quad & S_p << S_{np} \end{aligned} \tag{3}$$

For the first quarter of 2016 in SherLock, $g = 128$ features and $n = 56$ features with an average of $m = 55$ apps running at any given time.

We observed that S_{np} / S_p was 3.2 indicating that the pivot operation was effective in reducing the size of the merged data. We obtain a dataset with 14,234 features and 5.81 million records on merging this data with Moriarty labels.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

5 Feature Selection

We strive to reduce the dataset to its most informative features for smooth and utilitarian processing. On closer inspection of the 14,234 features, we discovered that 12,726 features had more than 70% null values in them, and we obtain 1508 features as a result of their removal. However, this remains significantly large for us to process, considering that we have 5.8 million records.

To further reduce the feature set, we pursue a feature selection method that ensures relevance towards our objective – malware detection and target classification. We considered LightGBM [13] as it has proven to be fast and scalable especially when implemented on high dimensional datasets [7]. Using this technique we reduce our feature space to 150 and 100 important features for malicious detection and target classification respectively. With the features reduced to less than 15% of 1508 features, we now implement stepwise forward selection [11] – an iterative method to determine the least number of features required to obtain any given model's best performance. Using stepwise forward selection we reduce the features required to detect malware to 10 features and the features required to determine the data targeted by malware to 16 features.

As a result of our feature selection approach, the feature set is reduced to approximately 0.1% of the original feature set. Table 2 lists the most important features that were considered for modeling.

Table 2. Features Selected for Proposed Architecture

Model Stage	GSF	LAF
Malware Detector	totalmemory_used_size, totalmemory_freesize, traffic_totalrxpackets	dalvikprivatedirty_Moriarty, dalvikprivatedirty_WhatsApp, dalvikpss_Samsung Push Service, otherpss_SherLock, rss_SherLock, uidrxbytes_Moriarty, num_threads_SherLock
Target Classifier	–	utime_SherLock, rss_SherLock, utime_Moriarty, stime_Moriarty, importance_SherLock, lru_SherLock, dalvikprivatedirty_SherLock, vsize_Hangouts, num_threads_Moriarty, rss_Hangouts, otherpss_Hangouts, dalvikpss_Hangouts, num_threads_SherLock, utime_Unified Daemon, otherprivatedirty_Context Service, vsize_Chrome

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

6 Experimental Framework

Knowing the kind of data the malware steals could be of more use during data breach assessment compared to just detecting the presence of a malicious action. We propose a two-stage architecture illustrated in Fig. 1 to classify data targeted by a positively detected malware. Our approach detects if a malicious action occurs in the first stage and if positively detected, classifies the data targeted during the malicious action in the second stage.

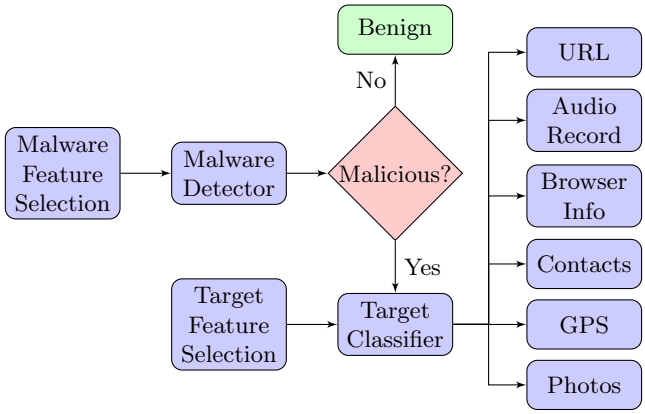

Fig. 1. Two-stage Architecture

Malware Detection: We primarily consider supervised tree-based models (Extra Trees, Random Forest, Decision Tree and XGBoost) [2, 3] for malicious detection since they have proven to be effective for the data in use [14, 23, 24]. Anomaly detection methods are suggested by Mirsky et al. [15] due to the sparse frequency of malicious records observed in the data as compared to benign (1:90). We aim to identify if anomaly detection methods are effective as per prior assumption, therefore we consider a tree-based anomaly and outlier detection method–Isolation Forest.

Target classification: We pass the values detected as malicious in the first stage to further classify the data targeted in this stage. This is a multi–class classification problem to determine the type of data targeted by the malware as seen in Table 1. We consider Extra Trees, XGBoost and K-Nearest Neighbours for this task.

6.1 Evaluation Metrics

Malware Detection: We propose using *False Omission Rate* (FOR) and *False Positive Rate* (FPR) to evaluate the performance of a malware detector. Accu-

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

racy and True Positive Rates as considered by [4,20,22,24] are not ideal metrics as they evaluate the model's performance using the true positive values of the majority class. These values are generally high for highly imbalanced data such as *SherLock* and therefore compensate for the impreciseness in classifying the minority class.

We aim to reduce the number of instances where a malware is misclassifed as benign. Therefore we consider *False Omission Rate* (FOR) and *False Positive Rate* (FPR) to evaluate the performance of the malware detector.

FOR Illustrated in Equation (4) this metric indicates the fraction of benign actions that are misclassified

$$\begin{aligned} FOR &= \frac{Number\,of\,benign\,records\,predicted\,as\,malicious}{Total\,number\,of\,malicious\,predictions} \\ &= \frac{False\,Negatives}{False\,Negatives + True\,Negatives} \end{aligned} \tag{4}$$

FPR Illustrated in Equation (5) this metric indicates the fraction of malicious records that go undetected by the malware classifier

$$\begin{aligned} FPR &= \frac{Number\,of\,malicious\,records\,falsely\,predicted\,as\,benign}{Total\,number\,of\,malicious\,records} \\ &= \frac{False\,Positives}{False\,Positives + True\,Negatives} \end{aligned} \tag{5}$$

Although each metric can be used individually, we propose using both FOR and FPR in conjunction to discover a detector with an overall good-fit for detecting presence of malware. A lower FOR signifies the success of the first stage of our architecture (malware detection). Meanwhile, a lower FPR signifies a smaller error that will cascade to the next stage. Ideally, both FOR and FPR need to be minimised to improve performance in data classification stage of our proposed two-stage architecture.

Target Classification: Target Classification is a multi-class classification task that involves predicting what kind of data has been stolen by the malware. The different types of malware stolen were given equal importance and hence equal weights were considered for all the classes. Therefore, the average F1-score is the metric of choice used to evaluate the model in this stage.

7 Results and Discussions

To evaluate the performance of our proposed architecture, we consider training and testing on all the users combined. Each user has been proportionally sampled

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

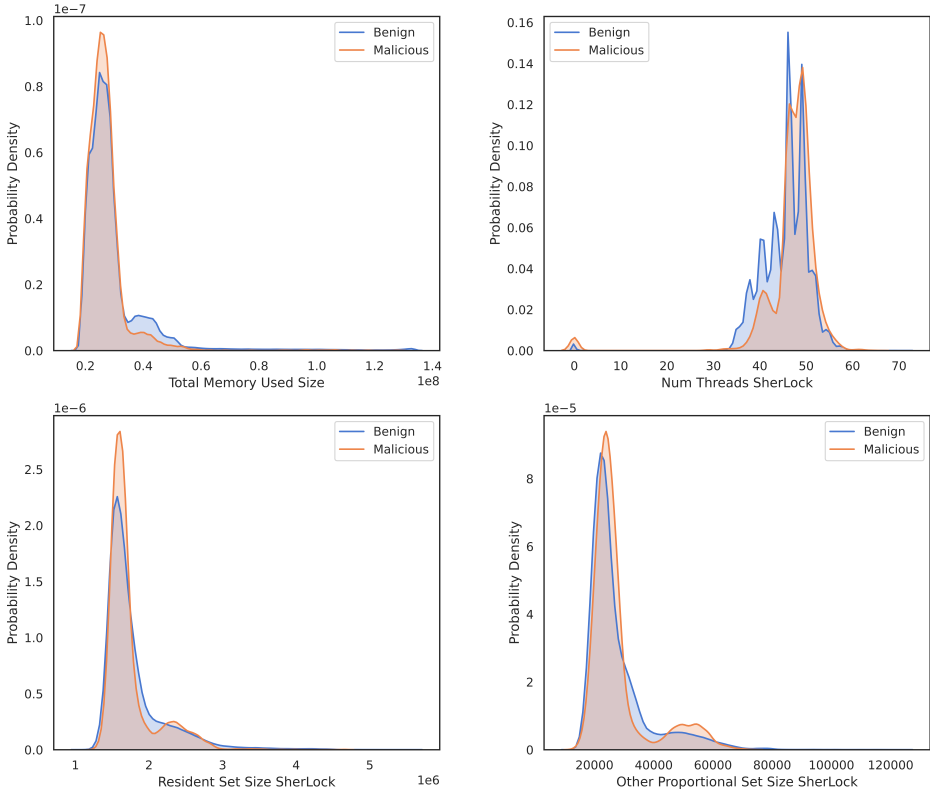

Fig. 2. Density distributions of some of the most important features

(stratified) while splitting the data into 75% for training and 25% for testing. Since the proposed architecture consists of two stages, it cascades performance at each level. We report the results at each stage for a deeper understanding of our model's performance.

7.1 Malware Detection

Tree-based classifiers display superior performance for this task as illustrated in Table 3. This is due to their ability to capture discrete and categorical information more accurately.

However, contrary to our prior assumption, the tree-based outlier detection method – Isolation Forest fails to detect malware with an FOR of 0.79. On observing the density distributions of some of the most important features (Fig. 2) we discover an overlap between malicious and benign distributions. Anomaly detection methods are effective to identify outliers from distributions [20]. Since unsupervised and anomaly detection methods rely on the malware to exist outside benign distribution, these methods may fail to detect malicious activity for this data.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

Table 3. Malware Detection Results

Classifier	False Omission Rate	False Positive Rate
Decision Tree	0.063	0.222
Extra Trees	**0.087**	**0.019**
Random Forest	0.088	0.058
XGBoost	0.646	0.296
Isolation Forest	0.793	0.976

With the least FOR of all the models considered (illustrated in Table 3), Decision Tree and Extra Trees are the best malware detectors with 6.3% and 8.7% FOR respectively. However, on closer inspection of the Decision Tree detector we observe that it can only achieve this accuracy at the cost of 22.2% FPR. Since this is not desirable for a performance cascading architecture as discussed in Section 6.1, we use Extra Trees to determine if an action is malicious before we classify its target in the next stage of our two-stage model.

7.2 Progressive learning

To achieve good detection, it is necessary for any user to be trained using the SherLock framework before the user can successfully monitor a newly installed app from the market. The time taken by each user to train the detector with SherLock would desirably need to be reduced, which can be done by minimising the required train data for the detection task. Our detector tackles this problem by combining all the users we have and performs stratified training and testing. Our detector exhibits the same accuracy with a decrease in train size as the number of users it has learned from increases. This is visualized in Fig. 3 where we consider a threshold of 0.15 FOR to analyse the change in required train data for an Extra Trees detector trained on 1-5 users. To achieve the threshold FOR when our detector had trained only on a single user, the detector required atleast 76% train data. However, our detector reduces the percentage of train data required from each user as it learns from more users. When the model was trained on 5 users, it required only 52.5% of the train data to achieve the threshold FOR.

Table 4. Target Classification Results

Models	Average F1 Score	Class-wise F1 Score (Support)					
		Audio Record (5)	BrowserInfo (13)	Contacts (2,343)	GPS (522)	Photos (662)	URL (91)
Extra Trees	0.82	0.44	0.58	0.99	0.99	0.99	0.90
XGBoost	**0.83**	**0.44**	**0.64**	**0.99**	**0.98**	**0.99**	**0.89**
K-Nearest Neighbors	0.79	0.40	0.50	0.99	0.98	0.99	0.86

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

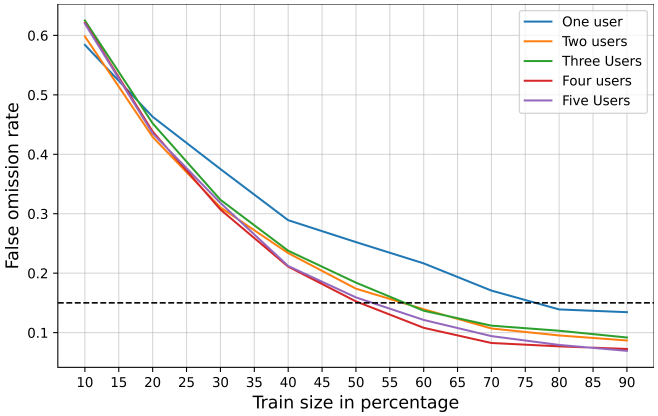

Fig. 3. Progressive learning with increase in users

7.3 Target Classification

Table 4 illustrates the results for classifying the target of the malicious actions predicted by the first stage. Due to the non-linearity posed by the data stream, we considered tree-based algorithms such as Extra Trees and XGBoost. Although XGBoost and Extra Trees display comparable performances, we prefer XGBoost to be integrated with our final pipeline since it has proven to be more scalable than the latter and displays the highest average performance of the models considered for the second stage.

With less than 9% inaccuracy in detecting malware from the first stage, we can predict with 83% certainty on what kind of data is being stolen when we use an Extra Trees detector (Table 3) coupled with an XGBoost classifier (Table 4).

Furthermore, by using our feature selection approach we maintain the aforementioned model performance with the feature set reduced to approximately 0.1% of the original set.

Stepwise forward selection for malware detection (illustrated in Fig. 4) reveals that we only require 10 features to determine if an action is malicious to achieve a minimum FOR and FPR of 0.087 and 0.019 respectively. Fig. 5 illustrates stepwise forward selection for target classification and suggests that we require only 16 features to categorize the type of data stolen. As a result of using such a small feature set, we minimize our throughput and processing time drastically.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

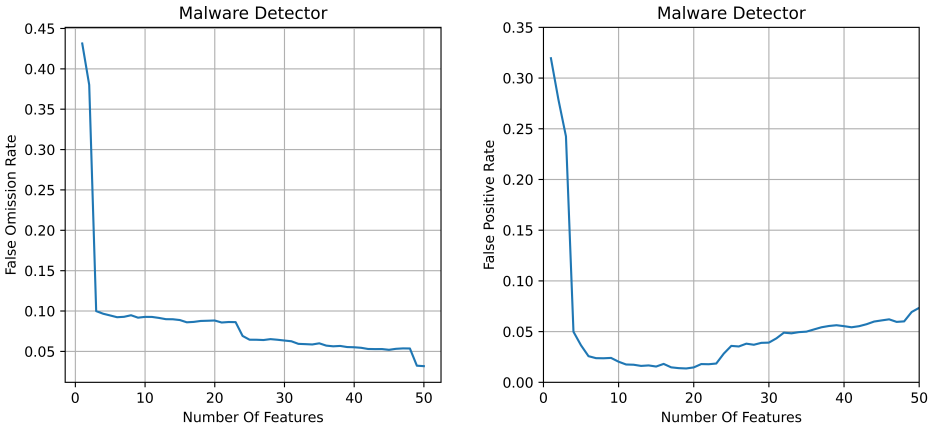

Fig. 4. Stepwise Forward Selection Convergence – Malware Detection

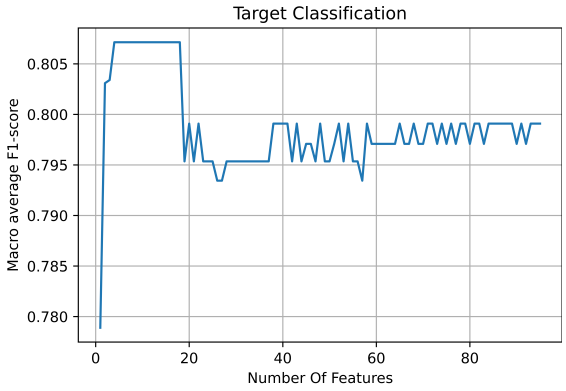

Fig. 5. Stepwise Forward Selection Convergence – Target Classification

8 Conclusion

In this paper, we propose and successfully test a two-stage machine learning model on the SherLock dataset to detect malicious actions in a smartphone and identify the type of data it steals. We successfully reduce one of the largest datasets for malware classification (SherLock) to 0.1% salient features of its initial feature set using our data preprocessing techniques. Furthermore, we go on to propose using False Omission Rate and False Positive Rate in conjunction to evaluate malware detectors. With just 8.7% inaccuracy in detecting malware from the first stage, our model can predict the kind of data stolen with 83%

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

certainty when we use an Extra Trees detector coupled with an XGBoost classifier. We exhibit our detector's robustness with the gradual decrease in the required train data from one user to achieve the aforementioned performance by training on more users and data. Anomaly detection techniques for malware fail, since malicious actions do not lie outside benign distributions as conventionally expected.

Although the proposed model reduces the percentage of train data required by a user to the minimum, malware detection is still dependent on user behaviour to work. There exists the need for a truly user-independent machine learning solution for malware detection to enhance user experience and ergonomics.

References

1. Linux manual, https://man7.org/linux/man-pages/man5/proc.5.html
2. scikit-learn, https://scikit-learn.org/stable
3. Xgboost, https://xgboost.readthedocs.io/en/latest/
4. Arora, A., Garg, S., Peddoju, S.K.: Malware detection using network traffic analysis in android based mobile devices. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies. pp. 66–71 (2014). https://doi.org/10.1109/NGMAST.2014.57
5. Bekerman, D., Shapira, B., Rokach, L., Bar, A.: Unknown malware detection using network traffic classification. In: 2015 IEEE Conference on Communications and Network Security (CNS). pp. 134–142 (2015). https://doi.org/10.1109/CNS.2015.7346821
6. Bläsing, T., Batyuk, L., Schmidt, A., Camtepe, S.A., Albayrak, S.: An android application sandbox system for suspicious software detection. In: 2010 5th International Conference on Malicious and Unwanted Software. pp. 55–62 (2010). https://doi.org/10.1109/MALWARE.2010.5665792
7. Chen, C., Zhang, Q., Ma, Q., Yu, B.: Lightgbm-ppi: Predicting protein-protein interactions through lightgbm with multi-information fusion. Chemometrics and Intelligent Laboratory Systems **191**, 54–64 (06 2019). https://doi.org/10.1016/j.chemolab.2019.06.003
8. Chen, Z., Yan, Q., Han, H., Wang, S., Peng, L., Wang, L., Yang, B.: Machine learning based mobile malware detection using highly imbalanced network traffic. Information Sciences **433-434** (04 2017). https://doi.org/10.1016/j.ins.2017.04.044
9. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. **32**(2) (Jun 2014), https://doi.org/10.1145/2619091
10. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. p. 627–638. CCS '11, Association for Computing Machinery, New York, NY, USA (2011), https://doi.org/10.1145/2046707.2046779
11. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques (01 2012). https://doi.org/10.1016/C2009-0-61819-5
12. IBM: Cost of a Data Breach Report 2020 (2020), https://www.ibm.com/security/digital-assets/cost-data-breach-report/

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17

13. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 3149–3157. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
14. Memon, L., Bawany, N., Shamsi, J.: A comparison of machine learning techniques for android malware detection using apache spark. Journal of Engineering Science and Technology **14**, 1572–1586 (06 2019)
15. Mirsky, Y., Shabtai, A., Rokach, L., Shapira, B., Elovici, Y.: Sherlock vs moriarty: A smartphone dataset for cybersecurity research. pp. 1–12 (10 2016). https://doi.org/10.1145/2996758.2996764
16. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. pp. 421–430 (01 2008). https://doi.org/10.1109/ACSAC.2007.21
17. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. CoRR **abs/1802.10135** (2018), http://arxiv.org/abs/1802.10135
18. Samani, R.: McAfee mobile threat report q1 (2020), https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf
19. Schmidt, A.., Bye, R., Schmidt, H.., Clausen, J., Kiraz, O., Yuksel, K.A., Camtepe, S.A., Albayrak, S.: Static analysis of executables for collaborative malware detection on android. In: 2009 IEEE International Conference on Communications. pp. 1–5 (2009). https://doi.org/10.1109/ICC.2009.5199486
20. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: "andromaly": A behavioral malware detection framework for android devices. J. Intell. Inf. Syst. **38**(1), 161–190 (Feb 2012), https://doi.org/10.1007/s10844-010-0148-x
21. U.S. Securities and Exchange Commission: The need for greater focus on the cybersecurity challenges facing small and midsize businesses (2015), https://www.sec.gov/news/statement/cybersecurity-challenges-for-small-midsize-businesses.html
22. Wang, S., Chen, Z., Zhang, L., Yan, Q., Yang, B., Peng, L., Jia, Z.: Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. pp. 1–6 (06 2016). https://doi.org/10.1109/IWQoS.2016.7590446
23. Wassermann, S., Casas, P.: Bigmomal: Big data analytics for mobile malware detection. In: Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity. p. 33–39. WTMC '18, Association for Computing Machinery, New York, NY, USA (2018), https://doi.org/10.1145/3229598.3229600
24. Zheng, Y., Srinivasan, S.: Mobile app and malware classifications by mobile usage with time dynamics. In: Barolli, L., Takizawa, M., Xhafa, F., Enokido, T. (eds.) Advanced Information Networking and Applications. pp. 595–606. Springer International Publishing, Cham (2020)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_17