Detecting Sensor-Based Repackaged Malware

Boyu Liu, Duanyue Yun, Xin Guo, Xiao Ji, Huiyu Song
Data Science Institute
Columbia University, USA
{bl2791, dy2400, xg2331, xj2247, hs3160}@columbia.edu

Shirish Singh, Gail Kaiser
Dept. of Computer Science
Columbia University, USA
{shirish, kaiser}@cs.columbia.edu

***Abstract*—Android is the most targeted mobile OS. Studies have found that repackaging is one of the most common techniques that adversaries use to distribute malware, and detecting such malware can be difficult because they share large parts of the code with benign apps. Other studies have highlighted the privacy implications of zero-permission sensors. In this work, we investigate if repackaged malicious apps utilize more sensors than the benign counterpart for malicious purposes. We analyzed 15,297 app pairs for sensor usage. We provide evidence that zero-permission sensors are indeed used by malicious apps to perform various activities. We use this information to train a robust classifier to detect repackaged malware in the wild.**

***Index Terms*—Repackaged Malware, Malware Detection, Mobile Sensors, Malicious Payload, Third-party libraries**

I. Introduction

Repackaged malware is built by decompiling a benign app and slightly modifying the code by adding a malicious payload [1], either through third-party libraries or external API calls. Detection of repackaged malware is challenging for two primary reasons: 1) high code similarity with the benign app, and 2) malicious code embedded in libraries.

The privacy implications of zero-permission sensors have previously been studied [2], [3]. Malware often uses additional sensors to steal information. In contrast, benign apps use sensors to provide functionality to the user. The sensor data can be utilized either in the app code or via third party libraries. In this study, we found evidence that repackaged malware utilize additional sensors to perform malicious activities. We developed a tool to extract sensor types actually used by the apps (Figure 1). We incorporated the sensor-related features into various classification models for detecting malware. The contributions of our work are three-fold:

- We developed a tool to extract the sensors actually used by an app as opposed to the sensors mentioned in the Android manifest file but never used.
- We found evidence that the proportion of malicious apps using sensors is higher than that of benign apps.
- We built and compared five classifiers with sensor and library usage features to detect repackaged malware.

II. Dataset

To perform the study, we used 15,297 original-repackaged Android app pairs from the Repack repository [1]. The 15,297 repackaged apps correspond to 2,776 original apps, making a total of 18,073 apps. According to past studies [4], not all repackaged apps are malware (some are simply plagiarism). We relied on VirusTotal [5] reports to vet the apps in the dataset. VirusTotal aggregates detection results from over 60 antivirus engines, making its malware detection robust enough to be treated as the ground truth. In our study, we consider an app to be benign if the app was not flagged by any antivirus engine and an app to be malware if more than two antivirus tools flag the sample. We disregard all other samples (those flagged by only one antivirus tool). After incorporating the ground truth using the VirusTotal reports, we have 2,004 benign apps remaining from the original app samples. Among the repackaged apps, there were 9,142 malware and 2,858 benign apps. We disregard the benign repackaged apps. Consequently, the final labeled dataset contains 2,004 benign apps and 9,142 malicious repackaged apps. Section III-B discusses how we evaluate the models to accommodate the imbalanced dataset.

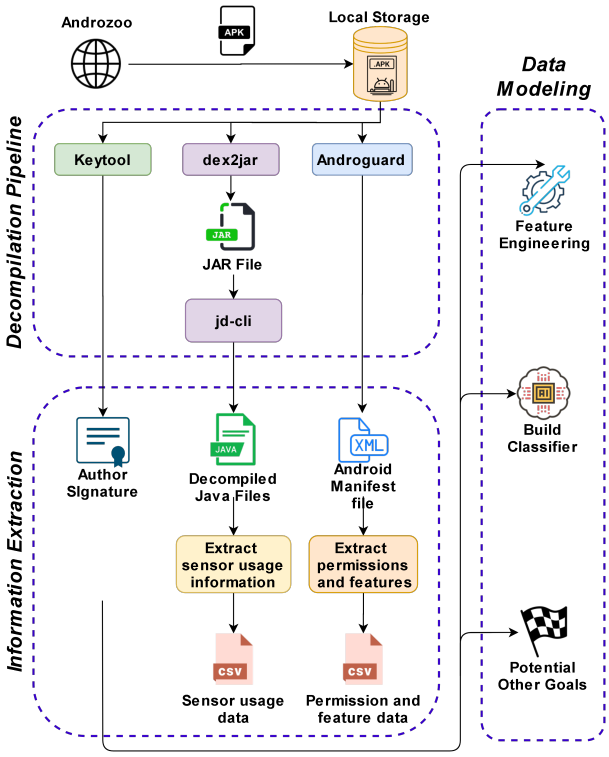

Fig. 1. Overview of our framework

A. Pipeline of Sensor Feature Extraction

We built a pipeline to extract sensor features. To improve the validity of our static code analysis and reduce the noise

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

introduced in the code base, we searched for sensor usage starting from the activities and services files. We aimed to keep the sensors that are actually used in the app. Algorithm 1 shows how we extracted the sensor usage features from Android package (APK) files.

Algorithm 1 Sensor Feature Extraction

Result: Return a list L of sensors used by an app
Collects a list l of $activities$ and $services$ of the app
for *file in* l **do**
  for $line$ *in file* **do**
    if $line$ *contains package/library* **then**
      Explore files inside the package/library;
      Generate function list F using sensors;
    else if $line$ *USES sensor* **then**
      Add sensor type into L;
    else if $line$ *contains function in* F **then**
      Add sensor type into L;
  end
end

B. Basic Feature Generation

Besides sensor usage features, we also extracted other machine learning interpretable features from the APK files. We used Androwarn [6] to generate features from six categories: Permissions from Android Manifest file, Packages, Hardware, Intents, Classes, and Leaks.

III. Experiment Design and Results

A. Statistical Testing

To obtain an understanding of whether sensor usage features could be helpful in malware detection, we conducted a hypothesis test comparing the proportions of apps using at least one sensor between malicious and benign apps. Let p_1 denote the proportion of malicious apps using at least one sensor, and p_2 denotes that of benign apps. Our null hypothesis is that p_1 is the same as p_2, and the alternative hypothesis is that p_1 is greater than p_2, which implies that the proportion of malicious apps using sensors is higher. These can be summarized in the mathematical notation below:

$$H_0 : p_1 = p_2 \tag{1}$$

$$H_a : p_1 > p_2 \tag{2}$$

The test statistic is calculated as follows:

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} = 7.405$$

where $\hat{p}_1$, $\hat{p}_2$ are the group specific proportions calculated using the data sample, $\hat{p}$ is the overall proportion, n_1, n_2 are the corresponding group sizes.

When rounding to 3 decimal places, the p-value is 0. Therefore, at 1% significance level, we reject the null hypothesis that the proportions of sensor usage in malware and benign apps are the same. We find evidence that the proportion of malicious apps (0.187) using sensors is higher than that of benign apps (0.137).

B. Modeling

1) Training and Testing Data: We trained our models on 80% of the original benign and repackaged malicious apps (8,916 apps) and tested on the remaining 20% (2,230 apps). The dataset was split randomly. We excluded repackaged benign apps from the training dataset. As mentioned in the Repack paper [1], these repackaged benign apps are plagiarized or copied versions of the original apps. They might not have any additional features and just be simple replicas of the original app. However, these 2,858 repackaged benign apps were used for testing.

2) Evaluation Metrics: In this study, we use *malware precision*, *malware recall*, and *balanced accuracy* as the evaluation metrics. Malware precision and malware recall are the basic metrics that can indicate the ability of our models to detect repackaged malware. However, as described in Section II, the dataset is imbalanced, so when evaluating the models, we choose balanced accuracy as another metric, which can be used to assess the overall performance of our models on both repackaged malware and benign apps. The formula of balanced accuracy is as follows:

$$\text{Balanced accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{TN}+\text{FP}}\right) \tag{3}$$

where TP is True Positive, TN is True Negative, FN is False Negative, FP is False Positive.

3) Models: We trained five classifiers to detect repackaged malware on the training set containing sensor features, including Support Vector Machine (SVM), K-nearest neighbor (KNN), XGBoost, Logistic Regression (LR), and Deep Neural Network (DNN). Table I summarizes the methods we used and the optimal hyperparameter settings for each classifier.

- *SVM*: For hyperparameter tuning, we tried two different kernels: 'poly' and 'rbf'. We also tuned the parameter C, which controls the strength of regularization. After tuning, the model gives precision, recall and balanced accuracy of 0.959, 0.847 and 84.0%, respectively.
- *KNN*: Since the dataset is imbalanced, we used the Synthetic Minority Oversampling Technique (SMOTE) to add synthetic interpolated data to the benign class. After doing 5-fold cross validation, the optimal number of neighbors k is 5 and the optimal weight function is 'distance'[1]. KNN provides a test malware recall of 0.933 with sensor features and 0.928 without sensor features. The model achieves precision and balanced accuracy of 0.922 and 78.7% respectively.
- *XGBoost*: We tuned hyperparameters using Bayesian Optimizer to maximize the AUC score.The parameters 'max depth' and 'n_estimator' control the complexity of the

[1]'distance' : weight points by the inverse of their distance. Closer neighbors of a query point will have a higher weight than farther neighbors.

trees, while 'learning rate' and 'reg_alpha' prevent model over-fitting. We achieved malware precision, malware recall and balanced accuracy of 0.938, 0.894, and 81.2% respectively, with the highest AUC score of 0.928.

- *LR*: The model hyperparameter 'C' usually needs to be tuned, which is the inverse of the regularization parameter. When we use 'l2' norm as the penalty of LR, the optimal 'C' is 0.7. Under this optimal setting, the malware precision, malware recall and balanced accuracy of LR is 0.940, 0.806 and 78.6%. respectively.
- *DNN*: In order to tune parameters for DNN, we first tuned the number of layers to be 3 to prevent over-fitting. Keras-Tunner was used to find the best combinations of 'n_unites', 'learning rate' and 'activation', which maximize the precision of the validation set. We achieved malware precision, malware recall and balanced accuracy of 0.947, 0.879, and 83.1% respectively.

Our best model achieves a detection rate of 95% on repackaged malware. Table II shows the malware precision, malware recall, and balanced accuracy of the models. Among the five classifiers, SVM has the highest malware precision, and 95.9% of its predictions are true positive. When considering malware recall, KNN performs best, identifying 93.3% malware. Other models also have good performance. But the balanced accuracy is not optimal (SVM's balanced accuracy is 84.0%).

Model	Tuning Method	Optimal Hyperparameters
SVM	5-fold cross validation	kernel: 'rbf', C: 1
KNN	5-fold cross validation	n_neighbors: 5, weights: 'distance'
XGBOOST	Bayesian Optimizer	lr: 0.3579, max_depth: 5 n_estimators: 38 reg_alpha: 0.4868
LR	5-fold cross validation	C: 0.7
DNN	Keras Tuner Bayesian Optimizer	layer: 3, n_unit1: 32 n_unit2: 48, lr=0.0001 active_function: 'relu' optimizer: 'adam'

TABLE I
HYPERPARAMETER TUNING

Sensor Features	Malware Classifier	Malware Precision	Malware Recall	Balanced Accuracy
Used	SVM	**0.959**	0.847	**84.0**%
	KNN	0.922	**0.933**	78.7%
	XGBOOST	0.938	0.894	81.2%
	LR	0.940	0.806	78.6%
	DNN	0.947	0.879	83.1%
Not used	SVM	**0.945**	0.876	**82.1**%
	KNN	0.921	**0.928**	78.2%
	XGBOOST	0.931	0.895	79.7%
	LR	0.934	0.824	78.0%
	DNN	0.932	0.875	79.2%

TABLE II
MODEL EVALUATION

In order to evaluate the effect of the sensor features that we extracted from APK files on the performance of classification, we trained another five classifiers on the dataset without sensor features. The results are also listed in Table II. The models using sensor features have higher malware precision and balanced accuracy than the models training on the dataset without sensor features, indicating that sensor features actually bring new information to the classifiers and enable them to make more accurate predictions.

Figure 2 shows the ROC curves of the five classifiers. The solid and dotted lines represent classifiers with sensor features and without sensor features, respectively. The dotted lines are slightly to the right of the solid lines, and the AUCs of classifiers with sensor features are also higher than the AUCs of classifiers without sensor features. This study demonstrates that adding sensor-related features improves the classifier.

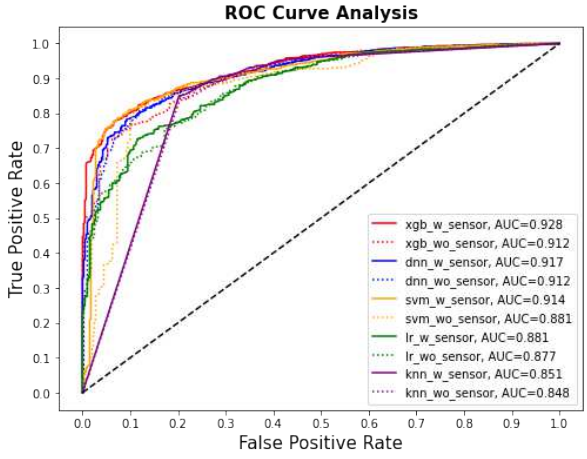

Fig. 2. ROC Curves of XGBoost, DNN, SVM, LR, and KNN

IV. CONCLUSION

We evaluated our work on a repackaged app dataset [1] containing 15,297 original-repackaged app pairs. To verify our hypothesis that a malicious app is more likely to use sensors, we conducted a two-proportion z-test. At 1% significance level, we find evidence that the proportion of malicious apps using at least one sensor is higher than that of benign apps. We used static analysis tools to extract features from the APK files and trained five classifiers to detect repackaged malware. Our models achieve 95% malware detection rate.

ACKNOWLEDGMENT

Gail Kaiser and Shirish Singh are supported in part by NSF CNS-1563555, CCF-1815494 and CNS-1842456.

REFERENCES

[1] L. Li, T. F. Bissyande, and J. Klein, "Rebooting research on detecting repackaged android apps: Literature review and benchmark," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[2] J. L. Kröger, P. Raschke, and T. R. Bhuiyan, "Privacy Implications of Accelerometer Data: A Review of Possible Inferences," in *3rd International Conference on Cryptography, Security and Privacy (ICCSP)*. ACM, January 2019, pp. 81–87.

[3] S. Singh, D. M. Shila, and G. Kaiser, "Side Channel Attack on Smartphone Sensors to Infer Gender of the User: Poster Abstract," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 436–437. [Online]. Available: https://doi.org/10.1145/3356250.3361939

[4] K. Khanmohammadi, N. Ebrahimi, A. Hamou-Lhadj, and R. Khoury, "Empirical study of android repackaged applications," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3587–3629, 2019.

[5] VirusTotal, "VirusTotal," https://www.virustotal.com/gui/, (Accessed on 11/15/2020).

[6] T. D., "maaaaz/androwarn: Yet another static code analyzer for malicious Android applications," https://github.com/maaaaz/androwarn, (Accessed on 11/15/2020).