DecisionDroid: A Supervised Learning-Based System to Identify Cloned Android Applications

Ayush Kohli
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL, USA
akohli@siu.edu

ABSTRACT

This study presents DecisionDroid, a supervised learning based system to identify cloned Android app pairs. DecisionDroid is trained using a manually verified diverse dataset of 12,000 Android app pairs. On a hundred ten-fold cross validations, DecisionDroid achieved 97.9% precision, 98.3% recall, and 98.4% accuracy.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; **Software reverse engineering**; Mobile and wireless security;

KEYWORDS

Android, cloned apps, cloned apps, security, supervised learning

ACM Reference format:
Ayush Kohli. 2017. DecisionDroid: A Supervised Learning-Based System to Identify Cloned Android Applications. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 3 pages.
https://doi.org/10.1145/3106237.3121277

1 PROBLEM AND MOTIVATION

More than 85% smartphone users now use the Android OS [10]. As of June 2017, the Google play store (official Android market place) has more than 2.9 million Android applications [2]. However, users can customize their Android systems relatively easily and install apps from unofficial third-party marketplaces (e.g., GetJar, SlideMe, and AppBrain). Sometimes plagiarists modify popular apps obtained from the official marketplace by replacing some of the classes (e.g., ad library) or by injecting malicious code, and upload those to an alternative marketplace [5] or even to the Google play store [18]. This practice not only violates the intellectual property of the original developer but also makes the users of the cloned apps vulnerable to malicious code.

ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3121277

However, identifying cloned app pairs is difficult as plagiarists can alter different features of an app to evade detection [18]. Moreover, most of the existing techniques [5, 9, 18, 21, 22] require pairwise comparisons, which are not scalable (i.e., complexity $O(n^2)$) for market-scale app analysis. Therefore, this research aims to build a scalable and resilient system to identify cloned app pairs.

2 RELATED WORK

Researchers have proposed several techniques to identify cloned Android pairs. Popular techniques include pairwise comparisons based on similarities in opcode sequences (i.e., DroidMOSS [22] and JuxtApp [9]), similarity searches based on functionalities (i.e., PiggyApp [21]), pairwise comparisons of the program dependency graphs (i.e. DNADroid [5] and AnDarwin [6]), visual similarities (i.e DroidEagle [16] and ViewDroid [18]), and similarities based on statistical features like number of activities, permissions, and resource files (i.e., ResDroid [15] and FSquaDRA [19]).

Techniques based on a particular set of features perform very well on a particular type of clone but fail on others. For example, a plagiarist can beat opcode based techniques (e.g., DroidMOSS and JuxtApp) using code obfuscations. Statistical features based techniques are immune to obfuscations but they often report false positives on auto-generated apps (e.g., apps created by Appinventor). Moreover, a plagiarist can easily change one byte of a resource file (i.e., image files) to alter its digest. While view graph based techniques (e.g., ViewDroid) are immune to obfuscations or resource modifications, these techniques fail on apps with small view graphs (e.g., single UI apps).

3 APPROACH AND UNIQUENESS

Since techniques based on a particular type of feature fail on certain clones, a robust and resilient solution must use combinations of the existing methods. To accomplish this goal, we implement DecisionDroid, a supervised learning based system that combines multiple detection approaches. The following subsections describe our training dataset generation steps and the DecisionDroid architecture.

3.1 Training Dataset Generation

A resilient supervised learning based technique requires a diverse training dataset. We obtained Android clone datasets from five prior studies [1, 4, 11, 16, 20] and failed to obtain datasets from other four studies (e.g., [6, 15, 17, 18]) due to no response from the authors.

Table 1: Manually verified app pairs

Dataset	Lazy	Amateur	Grey	Non-clone
Androzoo (clone) [1]	1,156	1,199	295	1,324
Centroid [4]	163	0	0	0
DroidAnalytics [20]	387	703	1	0
DroidEagle [16]	25	0	0	0
Piggyapp [11]	1,186	11	1	15

To ensure accuracy of our training dataset, we manually verified each of the clone pairs after executing the apps on the Memu Android emulator[1]. We categorized each of the pairs into one of the following four categories:

(1) **Lazy clones** are pairs of apps with the exact same layouts and features, where only the icons or splash screens may differ.
(2) **Amateur clones** are pairs of apps providing the same features, where the UI components remain the same but the color schemes, language, images, and positioning may differ.
(3) **Grey clones** are pairs of apps providing similar features, where one app provides additional functionalities not offered by the other.
(4) **Non-clones** are pairs of apps with significantly different features.

Table 1 shows the number of manually verified pairs from the five datasets. Our training dataset is robust with not only different types of clone pairs but also non-clones pairs that are highly similar in features (e.g., resource, class names). In addition to these 6,830 manually verified pairs, our training dataset of 12,000 app pairs also includes 625 auto generated non-clone pairs (i.e., Appinventor created apps) and 4,545 randomly selected non-clone pairs.

3.2 DecisionDroid Architecture

DecisionDroid is composed of the two modules: 1) feature extractor, 2) similarity calculator. We implement the feature extractor module of DecisionDroid in Java using the libraries provided by IC3-DialDroid [3]. Table 2 shows the set of features used by DecisionDroid. While DecisionDroid uses several features employed by prior studies, its use of Intent attributes and exit points make it unique. Intents characterize the communication among app components and are very hard to change without altering app behavior. Since ViewDroid[18] uses only explicit intents with inaccurate resolution, it creates incomplete view graphs. On the contrary, view graphs constructed by DecisionDroid are based on state-of-the art resolution of both explicit and implicit intents. The feature extractor module stores the extracted features in a MySQL database using following four steps:

(1) Parses the digests of resource files from `MANIFEST.MF`.
(2) Parses the signer certificate (*.rsa) from `META-INF` directory.
(3) Parses the `AndroidManfiest.xml` file to extract package name, permissions, components, entry points, and intent filters.

[1]http://www.memuplay.com/

Table 2: Features selected to train classifier

Feature	Definition
Components	The essential building blocks of an Android application declared in the `AndroidManifest.xml` handling UI, background service, data store, or broadcast message.
Content providers	Represents data stores for an app.
Entry points	Classes responsible for handling incoming intents.
Exit points	Classes issuing outgoing intents.
Intents	Messaging objects facilitating inter-component communications between two components of the same app or different apps.
Intent filters	Define the types of intents that components of an app can handle.
Launcher	The entry point when a user launches an Android application.
Package name	Serves as an unique identifier of the application.
Permissions	The type of system data and features that an app wants to access.
Resource files	The XML or image files packaged inside an application.
View graph	A directed graph representing how Android display can switch from one view to another view during the execution of an app.

Table 3: Cross validation results

Algorithm	Precision	Recall	Accuracy	F-score
Adaboosting	96.7%	96.4%	97.3%	0.97
Decision Tree	96.1%	96.7%	97.1%	0.96
Gradient tree boosting	96.6%	98.1%	97.8%	0.97
Naive Bayes	82.9%	98.4%	91.2%	0.90
Random Forest	**97.9%**	98.3%	**98.4%**	**0.98**
SVM	92.4%	93.0%	94.1%	0.93

(4) Performs static analysis to extract intents and exit points.

We implement the DecisionDroid similarity calculator module in Python. We use overlap similarity (i.e., $similarity(a, b) = (a \cap b)/min(|a|, |b|)$) as our similarity measure for statistical features, since a recent study [7] shows that overlap similarity performs better than the Jaccard similarity (i.e., $similarity(a, b) = (a \cap b)/(a \cup b)$) in identifying cloned app pairs. We use Levenshtein ratio to compute the similarities in package names and the Python graph tool library [13] to compute view graph similarities.

4 RESULTS AND CONTRIBUTIONS

We computed 21 different similarity scores for each of the app pairs. To reduce potential overfitting, we eliminated 12 features based on a recursive feature elimination [8] technique. We used the Scikit-learn [12] implementations of six commonly used supervised algorithms on the remaining nine features. We validated each of the algorithms using 10-fold cross-validations [14], where the dataset was randomly divided into 10 groups and each of the ten groups

was used as test dataset once, while the remaining nine groups were used to train the classifier. We repeated this process over a hundred times and computed the mean performances of the classifiers. The results suggest that DecisionDroid is highly accurate in identifying clone app pairs with a Random Forest based model performing the best. We have extracted features from 200000 Android apps obtained from five different marketplaces. We are currently working on scaling the classifier model on this large dataset.

ACKNOWLEDGEMENTS

I would like to thank Amiangshu Bosu for guiding me through this research and Alec Waichunas for assisting in dataset verification.

REFERENCES

[1] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. 468–471.

[2] AppBrain. 2017. Number of Android apps. https://www.appbrain.com/stats/number-of-android-apps. (2017).

[3] Amiangshu Bosu, Fang Liu, Danfeng (Daphne) Yao, and Gang Wang. 2017. Collusive Data Leak and More: Large-scale Threat Analysis of Inter-app Communications. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. 71–85.

[4] Kai Chen, Peng Liu, and Yingjun Zhang. 2014. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 175–186.

[5] Jonathan Crussell, Clint Gibler, and Hao Chen. 2012. Attack of the clones: Detecting cloned applications on android markets. In *European Symposium on Research in Computer Security*. Springer, 37–54.

[6] Jonathan Crussell, Clint Gibler, and Hao Chen. 2013. Andarwin: Scalable detection of semantically similar android applications. In *European Symposium on Research in Computer Security*. Springer, 182–199.

[7] Olga Gadyatskaya, Andra-Lidia Lezza, and Yury Zhauniarovich. 2016. Evaluation of Resource-Based App Repackaging Detection in Android. In *Nordic Conference on Secure IT Systems*. Springer, 135–151.

[8] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene selection for cancer classification using support vector machines. *Machine learning* 46, 1 (2002), 389–422.

[9] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. 2012. Juxtapp: A scalable system for detecting code reuse among android applications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 62–81.

[10] International Data Corporation (IDC). 2016. Smartphone OS Market Share, 2016 Q3. http://www.idc.com/promo/smartphone-market-share/os. (2016).

[11] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting. *IEEE Transactions on Information Forensics & Security (TIFS)* (2017).

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Tiago P Peixoto. 2014. The graph-tool python library. *figshare* (2014).

[14] Payam Refaeilzadeh, Lei Tang, and Huan Liu. 2009. Cross-validation. In *Encyclopedia of database systems*. Springer, 532–538.

[15] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. 2014. Towards a scalable resource-driven approach for detecting repackaged android applications. In *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 56–65.

[16] Mingshen Sun, Mengmeng Li, and John Lui. 2015. Droideagle: seamless detection of visually similar android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 9.

[17] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. 2015. Wukong: a scalable and accurate two-phase approach to android app clone detection. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 71–82.

[18] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Peng Liu. 2014. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 25–36.

[19] Yury Zhauniarovich, Olga Gadyatskaya, Bruno Crispo, Francesco La Spina, and Ermanno Moser. 2014. FSquaDRA: fast detection of repackaged applications. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 130–145.

[20] Min Zheng, Mingshen Sun, and John CS Lui. 2013. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 163–171.

[21] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. 2013. Fast, scalable detection of piggybacked mobile applications. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 185–196.

[22] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. 2012. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 317–326.