ESEC/FSE: U: DecisionDroid: A Supervised Learning-Based System to Identiy Cloned Android Applications

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

ABSTRACT

This paper presents the design and evaluation of DecisionDroid, a supervised learning based system to identify cloned Android app pairs. DecisionDroid combines a set of 13 different features and is highly resilient. We trained DecisionDroid using a manually verified diverse dataset of 18,562 app pairs. On a hundred ten-fold cross validations, it achieved 96.8% precision, 97.2% recall, and 98.3% accuracy.

CCS CONCEPTS

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

KEYWORDS

android, resilient, repackaged, clone, supervised learning, marketplace, scalable

ACM Reference Format:
Ayush Kohli. 2018. ESEC/FSE: U: DecisionDroid: A Supervised Learning-Based System to Identiy Cloned Android Applications. In *Proceedings of Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE'2018).* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

1 PROBLEM AND MOTIVATION

More than 85% smartphone users now use the Android OS [18]. One of the reasons for the large percentage of market share is due to the multitude of Android apps available on the Google play store. As of June 2017, the Google play store (official Android market place) has more than 2.9 million Android applications [5]. Moreover, users can customize their Android systems and install applications unofficial third-party marketplaces (e.g., GetJar,SlideMe, and AppBrain). However, pirates can 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 [8] or even to the Google play store [30]. 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.

Identifying cloned android app pairs remains a challenging problem as pirates can alter different features of the applications to evade detection [30]. Moreover, most of the existing techniques [8, 17, 30, 33, 34] require pairwise comparisons, which are not scalable for market-scale app analysis, i.e., complexity $O\left(n^2\right)$. Therefore, this research aims to build a scalable and resilient system to identify cloned app pairs.

On this goal we build *DecisionDroid*, a supervised learning based system based 21 different similarity scores between a pair of apps. The results of our 10-fold cross validation using a training dataset of 18,562 app pairs achieved very high precision (96.8%), recall(97.2%), and accuracy (98.3%).

The primary contributions of this research are:

- DecisionDroid, a supervised learning based system to identify cloned app pairs.
- An empirical investigation of features that are able to predict cloned app pairs.
- A large-scale real-world and manually verified dataset of cloned app pairs.

2 APPROACH AND UNIQUENESS

Both the popularity and the open-source nature of the Android platform has facilitated a large number of reverse-engineering tools such as Apktool [29], Proguard [19], Apk Editor Pro [2], Androguard [10], DashO [3], and Dex2Jar [21] for Android apps. Using theses tools a pirate can modify app layouts, rename / split /merge classes and packages, alter images and graphics, replace ad libraries, and insert malicious codes. The lack of a single-entry point also makes it easier for a pirate to insert malicious code inside an apk without changing existing classes [11].

A pirate can make various changes while creating a cloned app from the original app. However, most of the existing solutions to identify cloned apps rely on between one to three categories of similarity measures. As a result, these solutions may work well on a particular type of clones but would fail if a pirate adopts a different set of modifications. We encounter this challenge by developing *DecisionDroid*, that combines 13 different features using supervised learning algorithms. The following subsections describe our research method to generate a training dataset, select features, extract features from apps, compute similarity scores between app pairs, and generation of supervised learning based models.

FSE'2018, November 2018, Florida, USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

Table 1: Overview of our training dataset

Dataset	Lazy	Amateur	Grey	Non-clone
Appinventor*	0	0	0	625
Androzoo (clone)* [4]	1,374	1,277	307	6,505
Centroid* [7]	163	0	0	0
Codematch* [14]	349	0	0	649
DroidAnalytics* [32]	365	703	1	0
DroidEagle* [27]	25	0	0	0
Piggyapp* [20]	1,187	11	6	15
Random (non-clones)	0	0	0	5,000
Total: 18,562	**3463**	**1991**	**314**	**12,794**

*Manually verified using Android emulator

2.1 Training Dataset Generation

A resilient supervised learning based classifier requires a diverse dataset in order to obtain multiple categories / types of clones. Hence, we obtained Android clone datasets from six prior studies [4, 7, 14, 20, 27, 32] and were unable to obtain datasets from other four studies (e.g., [9, 25, 28, 30]) due to no response from the authors.

To ensure the accuracy of our dataset, we manually validated each of the pairs using the Memu Android emulator and categorized each pair based on the following clone categorization scheme adopted from a prior study[30].

(1) **Lazy clone** : App pairs which appear exactly and have the same features where only features like icons or splash screens differ.
(2) **Amateur clone** : App pairs which have same features and UI components remain the same but color schemes, language, images, and position may differ.
(3) **Grey clone** : App pairs which have similar features however one application provides additional functionality not available in others.

Table 1 shows an overview of our training dataset. In total, we manually verified 12,937 app pairs from the six dataset obtained from prior studies. We also manually verified 625 auto-generated app pairs created using Appinventor. The training data contains a variety of types of clones. In addition to the 13,562 manually verified app pairs, our training dataset also included 5,000 randomly selected non-clone pairs. Our manual verification also indicates that many of the app pairs marked as clones in prior studies [4, 20] are false positives.

2.2 Feature Selection

DecisionDroid relies on features to detect cloned app pairs. We define a *feature* as a characteristic of an app that can be used to distinguish itself from other apps. Unfortunately, Android clone detection techniques based on a particular feature may work well on certain types of clones but would fail on others.

While most of the prior studies used at most three categories of features, DecisionDroid leverages 13 different categories of features (Table 2). For each of the feature, we provide a brief rationale on why that feature may be useful to identify cloned app pairs. The Limitations column in Table 2 also describes potential limitations of each feature to identify cloned pairs. These limitations also points out potential drawbacks of detection approaches based on only two or three features. Eight out of the 13 categories of features used in DecisionDroid were also used in prior studies. However, the usage of four intent based features (IT, IF, XP, and NP) are unique to DecisionDroid.

2.3 Feature Extraction

The feature extractor of DecisionDroid is built using libraries provided by IC3-Dialdroid [6] and extracts the thirteen categories of features described in Table 2 according to following seven steps.

(1) Parses the digests of resource files [RF] from MANIFEST.MF.
(2) Parses the signer certificate (*.rsa) from META-INF directory to bet certificate[CRT].
(3) Parses the layout XML files inside the res/layout directory to determine the layout components [LT].
(4) Parses the AndroidManfiest.xml file to extract package name [PK], Launcher [L], permissions [PR], entry points [NP], and intent filters [IF].
(5) Performs static program analysis to extract class name [CL], intents [IT] and exit points [XP].
(6) Computes the size of an apk file in terms of the number of bytes [AS].
(7) Uses intent matching algorithm provided by DIALDroid [6] to compute View graphs [VG].

Upon completion, the attribute extractor stores the attributes of each app in MySQL database comprising of 40 tables. Our database is relational, normalized to facilitate efficient storage. We have implemented indexes on several columns to facilitate faster joins and queries.

2.3.1 Similarity calculator. We wrote a python script to compute the similarity scores (SS) for a pair of apps. We use the *overlap similarity coefficient* as our similarity measure over other similarity measures such as the *Jaccard similarity coefficient*, since a recent study [13] has shown the former performing better in identifying cloned pairs. For a feature(`F`) present in both app `A` and app `B`, the overlap similarity score is computed using the following formula:

$$SS_F\left(A,B\right) = \frac{\left(|A_F \cap B_F|\right)}{\min\left(|A_F|,|B_F|\right)}$$

The similarity scores for eight out of the thirteen categories of features were computed using the overlap similarity coefficient. To improve the efficacy of the CL, we exclude ad libraries, common libraries as identified in a prior work [14]. We also exclude the resource files [RF] belonging to more than 5% of the apps. For the remaining five features, we compute similarity scores as following.

- We use the graph-tool [23] to compute the similarity between view graphs [VG] of two apps.
- We use the Levenshtein Distance ratio to compute the similarity in package names [PK] between two apps.

Table 2: Description of features to identify cloned app pairs

Feature name	Description	Rationale	Limitations
1. Class name [CL]	An android executable is comprised of a collection of Java classes. Each class has a name and and contains a blueprint for object creation. Classes contain codes that are responsible for the behavior of an app.	A cloned app may contain classes that have same names as the original app.	- Obfuscation tools such as Proguard can rename, split or merge classes. - Many of the class files (e.g., ad library, common library) are shared by multiple apps.
2. Certificate [CRT]	Android apks are signed using a RSA public key that verifies the publisher of an app.	A cloned app should have a different signer than the original app.	- Certificates are useful to identify non-clones but are not useful to determine clones.
3. Resource files [RF]	Resource files are the images and XML files packed inside an APK file. During the application signing process, the `Jarsigner` tool computes the SHA1 digest of each file inside an unsigned apk and adds it to the `MANIFEST.MF` file.	A cloned application may share resource files with the original app.	- The digest of a resource file can be changed very easily by just changing a bit. - Many of the resource files (e.g., facebook share icon) are shared by multiple apps.
4. Layout [LT]	Layouts are a subset of resource files that describe the visual structures of the UIs of an application. Layouts are represented as XML files and are located under the /res/layouts directory.	Since the layouts are responsible for rendering the UI of an application, a cloned application may share layouts with the original app.	- Services do not have any layout. - Layouts can be created programmatically, therefore making layout based detection approaches unsuccessful.
5. Intents [IT]	Intents are messaging objects to establish communication between two components of the same app or two different apps. There are two type of intents, i) explicit– where the receiver is explicitly specified, ii) implicit– where the action to be performed is specified.	As Intents define how components communicate with each other, an cloned app is highly likely to issue similar intents as the original app.	-Explicit intents are identified by [CL] and therefore are prone to class renaming. - Implicit intents with frequent standard actions (e.g.,`Intent.ACTION.SEND` and `Intent.ACTION.VIEW`) are issued by a large number of apps.
6. View graph [VG]	The view graph of an app is a directed graph G(V,E), where each node in V represents a UI and E is a set of edges $< a, b >$ such that $a \in V$, $b \in V$ and the smartphone display can switch from view a to view b by user interaction or other triggers [30].	The view graph of an app is based on its core behavior, and therefore a cloned app may have similar view graph as the original.	-Attackers can add additional views to display ads. - Applications with very small number of views (i.e., less than four) may have very similar view graphs. - Background services don't have any views.
7. Launcher class[L]	The launcher class, which is defined inside the `AndroidManifest.xml` file is the main entry point of an Android application. When a user starts an application, code starts executing from the `onCreate` method inside the Launcher class.	The name of the launcher class of an app an its clones might be the same.	-Auto-generated applications and applications generated using frameworks could share the same launcher class name.
8. - Package name [PK]	Android package name, also known as the Google Play ID, is the unique identifier of an application. It can be found in the `AndroidManifest.xml` file.	A cloned app may have the exact same package name, especially if it is uploaded to an alternative marketplace other than the original market.	Package name can be easily changed by a repackaging tool such as Apk editor pro.
9. Permissions[PR]	Android applications run in a sandboxed environment. Therefore, the application has to explicitly request permission to access any resources or data. These permissions are declared in an `AndroidManifest.xml` file.[1]	Cloned applications may share application permissions since majority of the features would be the same.	- Changing the list of permissions is straightforward. -To avoid permission-based detections, a cloned app can remain underprivileged or over-privileged[12].
10. APK size [AS]	The Android APK size is the total sum of the zip archive which consists of all the resource and compiled code files.	A lazily cloned application may have similar APK size.	- Dead code [30] and useless resources [25] can be added to increase APK size.
11. Intent filter [IF]	An intent filter specifies the type of intents an application component can receive. Intent filters are declared in the `AndroidManifest.xml` and are characterized by three attributes i) action, ii) category and iii) data.	A cloned app providing same functionalities as the original app is more likely to also declare similar intent filters.	A cloned application could add / remove IFs based on the additional functionalities that it includes. For example, a cloned app may add additional intent filters to eavesdrop on system wide broadcasts.
12. Exit points [XP]	Exit points of an app are the classes that may issue outgoing intents.	A cloned app may have exit points with same names as the original app.	- An attacker can add additional exit points to leak data.
13. Entry points [NP]	Entry points of an app are the classes that may be activated to handle an incoming intent.	A cloned app may have entry points with same names as the original app.	- Entry points can be easily modified and an attacker can add new entry points to collect data from android devices.

Figure 1: Importance of features to identify cloned app pairs

Table 3: Similarity score (SS) conversion table

Category	Range
Low	$SS < 0.33$
Medium	$0.33 \leq SS < 0.66$
High	$0.66 \leq SS < 0.95$
Very High	$0.95 \leq SS$

- We divide the size of the smaller apk by the size of the larger apk to compute APK size [AS] similarity.
- For certificate [CRT] and Launcher [L], an exact match indicates a score of 1 otherwise the score would be 0.

The SS values are on a continuous scale with values between 0 to 1, where 1 indicates an exact match but 0 indicates no match. However, if feature values on a continuous scale are used to train a supervised learning based classifier, there are high potentials for overfitting [15]. Therefore, we converted SS values to a ordinal scale based on Table 3. We empirically determined the ranges for the categories in our conversion table based on the frequency distributions of the similarity scores in our training dataset.

3 EVALUATION

The evaluation of DecisionDroid answers two questions as described in the following subsections.

3.1 Which features are the most important to identify cloned app pairs?

Figure 1 shows the distribution of similarity scores for the nine most important features to distinguish cloned app pairs from non-clones. Based on those distributions, we conclude:

- Each of the individual features are prone to both false positives and false negatives, which further validates the need for a combined approach like DecisionDroid.
- Approaches based on RF, CL and PK have lower false positives compared to the other features but their rate of false negatives are not negligible.

Table 4: Cross validation results

Algorithm	Precision	Recall	Accuracy	F-score
Adaboosting	95.9%	94.7%	95.3%	0.95
Decision Tree	96.1%	96.7%	97.1%	0.96
Gradient tree boosting	95.9%	97.1%	97.9%	0.97
Naive Bayes	79.0%	98.0%	91.9%	0.87
Random Forest	**96.8%**	**97.2%**	**98.3%**	**0.97**
SVM	95.2%	95.2%	97.2%	0.95

- VG and LT based approaches[26, 30] should have very high false positives.
- Naive features like AS do not perform worse than a sophisticated feature such as VG.

3.2 How does our supervised learning based approach perform in identifying cloned app pairs?

Using the 13 categories of features (Table 2), we computed 21 different similarity scores for each of the app pairs in the training dataset. To reduce potential overfitting, we eliminated five of those features based on a recursive feature elimination [16] technique. We used the Scikit-learn [22] implementations of six commonly used supervised algorithms on the remaining sixteen features. We validated each of the algorithms using 10-fold cross-validations [24], 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 (Table 4).

4 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 [34] and JuxtApp [17]), similarity searches based on functionalities (i.e., PiggyApp [33]), pairwise comparisons of the program dependency graphs (i.e. DNADroid [8] and AnDarwin [9]), visual similarities (i.e DroidEagle [27] and ViewDroid [30]), and similarities based on statistical features like number of activities, permissions, and resource files (i.e., ResDroid [25] and FSquaDRA [31]).

Techniques based on a particular set of features perform very well on a particular type of clone but fail on others. For example, a pirate 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 pirate 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).

REFERENCES

[1] "Request app permissions," 2011, [Online; Accessed: 2017.02.26]. [Online]. Available: https://developer.android.com/training/permissions/requesting.html

[2] "Apk editor pro," *URL: https://apkeditorpro.com/*, 2017.

[3] "Java obfuscator & android obfuscator," 2018, [Online; Accessed: 2018.04.10]. [Online]. Available: https://developer.android.com/training/permissions/requesting.html

[4] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16, 2016, pp. 468–471.

[5] AppBrain, "Number of Android apps," https://www.appbrain.com/stats/number-of-android-apps, 2017.

[6] A. Bosu, F. Liu, D. D. Yao, and G. Wang, "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*, ser. ASIA CCS '17, 2017, pp. 71–85.

[7] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 175–186.

[8] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 37–54.

[9] ——, "Andarwin: Scalable detection of semantically similar android applications," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 182–199.

[10] A. Desnos *et al.*, "Androguard," *URL: https://github.com/androguard/androguard*, 2011.

[11] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *IEEE security & privacy*, vol. 7, no. 1, pp. 50–57, 2009.

[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[13] O. Gadyatskaya, A.-L. Lezza, and Y. Zhauniarovich, "Evaluation of resource-based app repackaging detection in android," in *Nordic Conference on Secure IT Systems*. Springer, 2016, pp. 135–151.

[14] L. Glanz, S. Amann, M. Eichberg, M. Reif, B. Hermann, J. Lerch, and M. Mezini, "Codematch: Obfuscation won?t conceal your repackaged app," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 638–648.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[16] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1, pp. 389–422, 2002.

[17] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A scalable system for detecting code reuse among android applications," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2012, pp. 62–81.

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

[19] E. Lafortune *et al.*, "Proguard," *h ttp://proguard. sourceforge. net*, 2004.

[20] L. Li, D. Li, T. F. Bissyandé, J. Klein, Y. Le Traon, D. Lo, and L. Cavallaro, "Understanding android app piggybacking: A systematic study of malicious code grafting," *IEEE Transactions on Information Forensics & Security (TIFS)*, 2017.

[21] B. Pan, "dex2jar," 2015.

[22] 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, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014.

[24] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," in *Encyclopedia of database systems*. Springer, 2009, pp. 532–538.

[25] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang, "Towards a scalable resource-driven approach for detecting repackaged android applications," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 56–65.

[26] C. Soh, H. B. K. Tan, Y. L. Arnatovich, and L. Wang, "Detecting clones in android applications through analyzing user interfaces," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. IEEE Press, 2015, pp. 163–173.

[27] M. Sun, M. Li, and J. Lui, "Droideagle: Seamless detection of visually similar android apps," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2015, p. 9.

[28] H. Wang, Y. Guo, Z. Ma, and X. Chen, "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, 2015, pp. 71–82.

[29] R. Winsniewski, "Android–apktool: A tool for reverse engineering android apk files," 2012.

[30] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "Viewdroid: Towards obfuscation-resilient mobile application repackaging detection," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 25–36.

[31] Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, and E. Moser, "Fsquadra: fast detection of repackaged applications," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2014, pp. 130–145.

[32] M. Zheng, M. Sun, and J. C. Lui, "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, 2013, pp. 163–171.

[33] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of piggybacked mobile applications," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, pp. 185–196.

[34] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 317–326.