Detecting Camouflaged Applications on Mobile Application Markets

Su Mon Kywe[1(✉)], Yingjiu Li[1], Robert H. Deng[1], and Jason Hong[2]

[1] School of Information Systems, Singapore Management University, Singapore, Singapore
{monkywe.su.2011,yjli,robertdeng}@smu.edu.sg
[2] Human Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, USA
jasonh@cs.cmu.edu

Abstract. Application plagiarism or application cloning is an emerging threat in mobile application markets. It reduces profits of original developers and sometimes even harms the security and privacy of users. In this paper, we introduce a new concept, called camouflaged applications, where external features of mobile applications, such as icons, screenshots, application names or descriptions, are copied. We then propose a scalable detection framework, which can find these suspiciously similar camouflaged applications. To accomplish this, we apply text-based retrieval methods and content-based image retrieval methods in our framework. Our framework is implemented and tested with 30,625 Android applications from the official Google Play market. The experiment results show that even the official market is comprised of 477 potential camouflaged victims, which cover 1.56 % of tested samples. Our paper highlights that these camouflaged applications not only expose potential security threats but also degrade qualities of mobile application markets. Our paper also analyze the behaviors of detected camouflaged applications and calculate the false alarm rates of the proposed framework.

Keywords: Camouflaged applications · Application plagiarism · Cloning

1 Introduction

With the growing number of third-party applications on mobile market places, it becomes increasingly hard to manage these applications and ensure that they are authentic, secure and of high quality. One of the emerging problems that the market owners encounter is plagiarism or cloning of mobile applications. During cloning, malicious parties copy all or parts of original applications and create similar applications or the clones. Such application plagiarism causes two main problems in mobile application markets. Firstly, it allows malicious parties to siphon revenues from original developers by replacing the advertisement libraries of plagiarised applications or by selling the clones with different prices to users. It has been shown that original developers, who are the victims of plagiarism, lost

J. Lee and J. Kim (Eds.): ICISC 2014, LNCS 8949, pp. 241–254, 2015.
DOI: 10.1007/978-3-319-15943-0_15

14 % of their advertising revenues and 10 % of their user base to the attackers [6]. Secondly, there are cases, where attackers add malicious payloads to the clones of popular applications and threaten the security and privacy of mobile application users. In a recent study by Zheng M. et al. [30], cloning is even regarded as one of the main distribution channels of mobile malwares.

Thus, to hinder application plagiarism, a number of clone detection methods have been proposed in [3,7,13,29]. However, these methods only focuses on repackaged applications, which are the clones created from the reverse-engineered codes of original applications. As such, these methods only search for code similarities among applications, consequently missing out a different set of clones, called camouflaged applications. Hence, in this paper, we introduce the concept of camouflaged applications. Camouflaged applications are applications whose external information, such as application names, icons, user interfaces or application descriptions, are cloned. These clones may or may not have similar codes as original applications but like other clones, they plagiarise and take advantage of other applications without consensus from original developers. They are not only confusing and harmful to the users but also discourage application development by affecting developers' reputation and monetary profits.

Therefore, in this paper, we propose a detection framework for finding camouflaged applications. Our method is based on external features of applications and applies text similarity and image similarity measurements, calculated by information retrieval systems. Although information retrieval systems have been applied to detect phishing web pages, we are the first to apply these technologies to efficiently detect camouflaged applications in mobile platforms. Our detection framework is tested with 30,625 Android applications from Google Play market. The experiment shows that 477 applications (1.56 %) are potential camouflaged applications. We further analyze the behaviors of detected camouflaged applications and inspect the false alarms rate of our detection method. A total of 44 false positives, which is 9.22 % of tested application samples, are identified.

Our paper is organized as follows. Problem definition of camouflaged applications and threat model are provided in Sect. 2. Background information about information retrieval systems and repackaged applications are given in Sect. 3. Our detection framework is proposed in Sect. 4 and our experiment results are shown in Sect. 5. Discussion about our findings, limitations of our method and future direction are provided in Sect. 6. After that, related work on repackaged applications are summarized in Sect. 7 and we conclude the paper in Sect. 8.

2 Problem Definition

Informally, camouflaged applications are defined as "copycat" applications or "confusingly similar" applications. There have been a lot of such applications on both official Google Play store and Apple's iTunes store. Generally, the features being cloned in camouflaged applications are icons, names, screenshots and descriptions. For instance, there are camouflaged applications with very similar

names, such as "Irate Birds" for the official "Angry Birds" and "Snip the Rope" for the official "Cute the Rope"[1]. Moreover, some camouflaged applications focus on screenshots to deceive users. For example, fake Pokemon Yellow application used Nintendo's popular Game Boy RPG as its application screenshots. It even managed to rise to top 3 position on iTunes store before being removed [19].

Camouflaged applications may exist on different application markets of the same platforms or across different platforms. According to Zhou et al. [29], 5–13% of the applications from unofficial Android market places are cloned from the official Google Play market. In addition, some clones may also spans across different platforms, such as Android or iOS. For instance, fake versions of popular iOS applications, such as Infinity Blade II[2] and Temple Run[3], appeared on Google Play, even before their official releases in Android version.

Market owners have imposed various developer policies for trademarks, copyrights, and patents of applications. For instance, Google Play has a policy for impersonation, stating (1) not to pretend as another company, (2) not to link to another website to represent itself as another application and (3) not to use another application's branding in title and description [20]. Moreover, Google Play's Trademark Infringement policy suggests to use distinct name, icon and logo and not to use those that are "confusingly similar" to another company's trademark. However, according to Liebergeld et al. [14], there is insufficient market control in Google Play market, because uploaded applications are not checked upfront on whether they indeed follow the policies. The policy enforcement relies heavily on feedbacks from users and developers.

Threat Model: The main goal of attackers is to trick users into installing their camouflaged applications. There are two ways by which users can install applications on their mobile devices. One way is to use default installer applications, such as Play Store or iTune Store, on mobile devices. Another way is to use desktop browsers, download applications from the providers' websites and later synchronize the applications to their mobile devices. In both cases, there are two situations in which user can be tricked to install the camouflaged applications. One is during the search and another is after the user goes to the detailed information page.

– When browsing applications or searching for an application, users can only observe application icons, application names and publishing company names. Some users download applications directly from the search results, instead of going to the detailed pages. Therefore, these three pieces of information play an important role in tricking the users. Although the ranking algorithm used by the Google also plays a role, it is out of scope of our paper.

[1] http://arstechnica.com/gaming/2012/08/google-play-cracks-down-on-confusingly-similar-apps/.
[2] http://www.pocketgamer.co.uk/r/Android/Infinity+Blade+II/news.asp?c=43572.
[3] http://m.androidcentral.com/temple-run-android-still-isnt-out-anything-else-just-malware.

– In the detailed information pages, application descriptions and screenshots are the main visual elements for users. Thus, they also play a critical role in tricking the users by attackers.

There are several ways in which attackers can gain profit for creating camouflaged applications. Table 1 summarizes different attackers' motivations as well as various possible attacks from camouflaged applications. Attack type may vary from mild copy-right violation and information theft to severe phishing and malware attacks. From the table, we can see that in addition to users and developers, other third-parties, such as banks and telecom providers, can be adversely affected by camouflaged applications.

Table 1. Categorizing attacks of camouflaged applications

Attacker motivation	Attack type	Mainly affected parties
Replacing advertise libraries	Copy-right violation	Developers
Creating paid version of free applications	Copy-right violation	Developers
Selling users' information to third parties	Information theft	Users
Stealing users' bank credentials	Phishing	Users and banks
Sending premium SMSes	Malware	Users and telecom providers

3 Background

3.1 Information Retrieval Systems

Information retrieval systems are used for retrieving relevant information from a collection of information resources. Most information retrieval systems includes two processes: indexing and retrieving. During indexing, the systems process documents that are either text documents or image, and extract useful information from them. During retrieving steps, query objects that are also processed, cleaned and their useful information are extracted. Then, similarity distance are measured between the query document and a collection of documents by using their representations. Ranked or sorted results are then returned to the users, together with the similarity scores.

Information retrieval techniques have also been used to detect phishing websites [24,25]. However, the traditional phishing detection methods cannot be applied directly on platform providers' websites, such as Google Play Store. This is because camouflaged applications and original applications can be featured on the same official website. Thus, meta-data analysis of web contents, such as hyper-links, web titles, web links, etc., cannot be applied in detecting camouflaged applications.

3.2 Repackaging and Code-Based Detectors

Cloned applications are often the result of repackaging, which includes recovering source codes of original applications and illegally re-compiling them with

different developers' certificates. Repackaging is common in Android application platform. In Android applications, Java source code are compiled into the Dalvik executable (DEX) format and run in Dalvik virtual machines. Dalvik byte codes can be easily reverse engineered by publicly available online tools, such as dex2jar and jd-gui.

As the repackaged clones are created from source codes of original applications, their source codes are similar to certain extent. Thus, code-based detectors can be used to detect repackaged applications. Generally, there are three types of code similarity detectors: feature-based, structure-based and PDG (Program Dependence Graph)-based. Feature-based detectors extracts features, such as number or size of classes, methods, loops, variables, from the applications and detects their similarities. Structure-based detectors convert applications into a stream of tokens and compare their streams. On the other hand, PDG-based detectors construct PDGs from the applications and compare them to derive the similarity scores. Many other code-based detectors, that have been proposed for repackaged applications, will be discussed more in Sect. 7.

4 A Framework for Detecting Camouflaged Applications

Accuracy and scalability are the key factors, considering the number of third-party applications in mobile markets. Thus, the goal of our paper is to have a lightweight simple detection system, which can efficiently detects the camouflaged applications. The implementation of our framework should allow developers to check their applications before submitting to the application stores. It can also used by Google Play for vetting before or after the application submission.

Our system leverages on the light-weight information retrieval systems, such as text retrieval and content-based image retrieval systems. There are four features with which we try to find camouflaged applications: application name, description, icon and screenshot. Application name and descriptions are handled by text retrieval systems, while application icon and screenshots are handled by image retrieval system. Figure 1 shows the architecture of our detection system. Our detection system includes four main steps: crawling, indexing, querying and detecting.

4.1 Crawling

First, we need a collection of existing applications, with which the potential camouflaged applications are compared. This application collection can be from different markets of different mobile platforms, depending on where we want to detect camouflaged applications. For instance, if we want to detect camouflaged applications, which are uploaded on unofficial Android markets, existing application collection should be crawled from official Google Play market and tested applications should be crawled from unofficial Android markets. However, if we want to detect camouflaged applications on Google Play's Android market, which are copied from iTunes market, the existing application collection

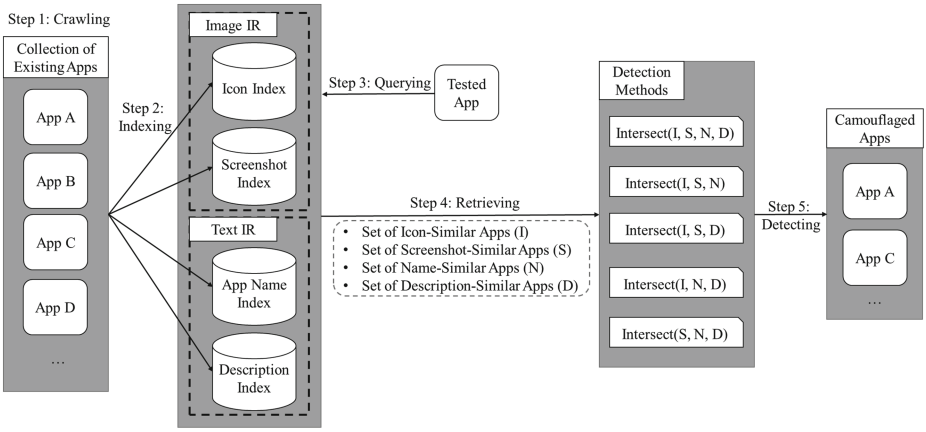

Fig. 1. Framework for detecting camouflaged applications

should be from official Google Play market and tested applications should be from Apple's iTunes market.

Our framework is independent of mobile platform and application market. It can be used on any platforms or markets as long as the market displays application names, icons, screenshots and descriptions. In our experiment, we crawled applications from official Android market and detect camouflaged applications within the same market. We use unofficial Google API to crawl App info, such as id, name, developer, rating as well as application description, icon and screenshots. Total of 30,625 applications are crawled for the experiment.

4.2 Indexing

The second step of most information retrieval systems is indexing. During indexing, a collection of documents are cleaned and processed to get ready for queries. We call both texts and images as "documents". Indexing can be done offline and just one time. Therefore, it is suitable for a large collection of documents. For each application in our 30,625 crawled applications, we create a name index, description index, icon index and screenshot index. Name and description indexes are created by text retrieval engines, while icon and screenshot indexes are handled by image retrieval engine.

Text Indexing. There are many types of text retrieval systems, such as boolean model, vector space model, probabilistic models. Most of them can be plugged and played in our detection framework. However, in our experiment, vector space model is used as it applies similar-word matching instead of exact-word matching algorithm. In the vector space model, each document is represented by a weighted vector in high-dimensional space. The weights from vectors are measured by TF-IDF scheme, which stands for Term Frequency (TF) and Inverse Document

Frequency (IDF). Open-source software, such as Lucene [18], can be used to implement TF-IDF scheme. Tokenizing, stemming and removal of stop words are all handled by Lucene.

Image Indexing. Similar to text retrieval methods, there are also many types of image retrieval methods. They extract visual features from the images and index those features with a pointer to the parent image. The extracted features include colors, color distributions, textures or joint histograms, which involve both color and texture information. Different algorithms have their own advantages and disadvantages on performance and robustness depending on the applied scenarios and types of images. We choose auto color correlogram algorithm [9], which uses the spatial correlation of colors. The algorithm is tested using SIMPLIcity data set [23] and is shown to be both effective and inexpensive in general purpose situations [17]. Note that our framework can also be easily modified to use other visual information retrieval algorithms. We use an open-source software, LIRE [16], to perform the visual information retrieval.

4.3 Querying and Retrieving

The third step is to query the index databases with potential camouflaged applications. In our case, the same 30,625 crawled applications are used as potential camouflaged applications. For each queried application, we retrieved applications, which have similar user interfaces but are from different developers. Information retrieval systems are used to calculate the similarity scores, and developer ID information, obtained from Google Play website, is used to ensure that similar applications are not from the same developer.

For each query, information retrieval systems calculate the cosine similarity score between query document and a set of indexed documents. The cosine similarity score measures the similarity distance between two vector representations of documents. The score ranges from 0 to 1, where similarity score of 0 represents two totally different documents and similarity score of 1 represents two totally similar documents. The retrieved similarity score are then used to rank the documents. In our case, retrieved set of applications is sorted based on the decreasing similarity scores, meaning the most similar ones are on the top of the list. We only use top-ten similar applications in each retrieved set to reduce false positives.

The output of each queried application is four sets of similar applications, namely I, S, N and D, where

- I is a set of applications that have similar icons as queried application,
- S is a set of applications that have similar screenshots as queried application,
- N is a set of applications that have similar names as queried application and
- D is a set of applications that have similar description as queried application.

Each set contains at most ten similar applications and many sets have fewer than ten applications. Note that although we use the same application set

for indexing and querying, different application set can also be applied in our architecture if we want to differentiate camouflaged applications across different markets.

4.4 Detecting

The fourth step of our framework is detection. Our detection method is different intersection sets of the four retrieved set I, S, N and D. This step generates the following five different result sets for each potential camouflaged application.

- Intersect(I,S,N,D) is a set of applications that have similar icons, screenshots, names and descriptions as queried application,
- Intersect(I,S,N) is a set of applications that have similar icons, screenshots and names as queried application but are not included in Intersect(I,S,N,D),
- Intersect(I,S,D) is a set of applications that have similar icons, screenshots and descriptions as queried application but are not included in Intersect(I,S,N,D),
- Intersect(I,N,D) is a set of applications that have similar icons, names and descriptions as queried application but are not included in Intersect(I,S,N,D),
- Intersect(S,N,D) is a set of applications that have similar screenshots, names and descriptions as queried application but are not included in Intersect (I,S,N,D).

Since these sets contain very similar applications from different developers, they are considered as camouflaged applications. Nonetheless, there can also be false alarms, where the result set contains non-camouflaged applications. False alarms are created because information retrieval methods cannot differentiate them, although they are obvious to normal users that they are not camouflaged applications.

5 Experiment and Results

Out of 30,625 applications, we find that 477 applications (1.56 %) have 1 to 6 camouflaged applications. Figure 2 shows the exact number of camouflaged

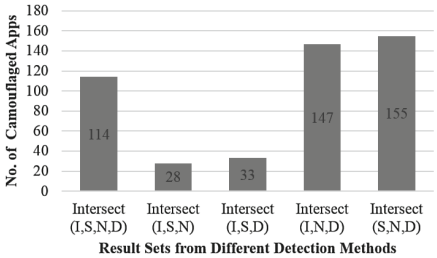

Fig. 2. Number of camouflaged applications for each detection method

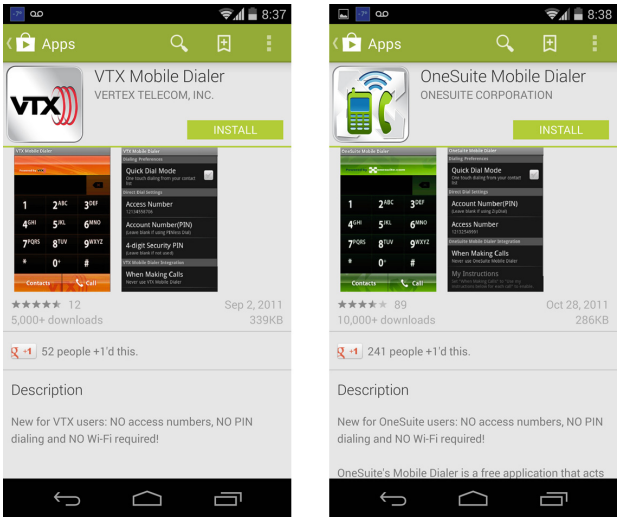

Fig. 3. Example of detected camouflaged application

application from each result set. According to the figure, we can see that Intersect(I,S,N,D), Intersect(I,N,D) and Intersect(S,N,D) reports more camouflaged applications than Intersect(I,S,N) and Intersect(I,S,D) methods.

An example of detected camouflaged applications, namely "VTX Mobile Dialer" and "OneSuite Mobile Dialer", is shown in Fig. 3. The two applications have the very similar screenshots, application name and description. Thus, they are reported in Intersect(S,N,D) set. However, they use different developer IDes as well as different contact information. The developer website and email address of "VTX Mobile Dialer" are https://www.vtxtelecom.com/ and *mobileapp@vtxtelecom.com*. On the other hand, the developer website and email address of "VTX Mobile Dialer" are http://www.onesuite.com/ and *mobileapp@onesuite.com*. Although they claim to be from different companies, their user interfaces are suspiciously similar. Therefore, they are regarded as camouflaged applications.

Determining the False Alarms: Determining the false positives and false negatives for camouflaged applications is a challenge, as we do not have any ground truth samples. Thus, we decide to do manual inspection on the result sets to determine the false positives. Though tedious, expert manual inspection has been a common way to test the efficiencies of information retrieval systems. To our surprise, a lot of the reported camouflaged applications have almost identical user interfaces. This makes our manual inspection easier.

Our manual inspection shows that the result sets contain a total of 44 false positives, which is 9.22 % of reported camouflaged applications. However, false positives exist only in the Intersect(I,N,D) and Intersect(S,N,D). Intersect(I,N,D) contains 21 false positive samples and Intersect(S,N,D) contains

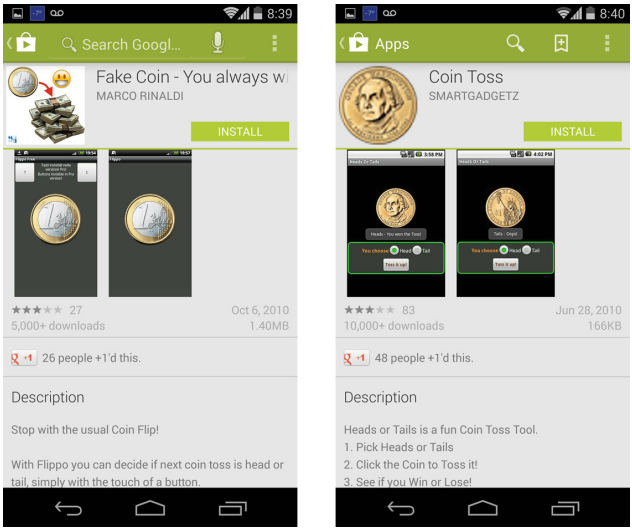

Fig. 4. Example of false-positive camouflaged applications

23 false positive samples. No false positive applications have been identified in Intersect(I,S,N,D), Intersect(I,S,N) and Intersect(I,S,D), which consider the similarity of both icons and screeenshots. This indicates that icons and screenshot similarity measures are great indicators of camouflaged applications.

Figure 4 shows an example of false alarm applications, called "Fake Coin - You always win!" and "Coin Toss". Although their user interfaces are similar, it is quite obvious to the real users that they provide different functions: the former application is for tricking friends and the latter application is for randomly tossing the coin. Therefore, these two applications should not be regarded as camouflaged applications.

6 Discussion

In this section, we will discuss about our findings on camouflaged applications as well as limitations of our method and future work.

Feature Selection in Detection Method: Our detection method is limited to camouflaged applications with at least three similar features. Nonetheless, there can still be camouflaged applications with only one or two similar features. For instance, there are camouflaged applications with only similar icons. Although our method can be easily extended to find applications with one or two similar features, many applications use very simple and easily searchable icons, such as a light bulb. Consequently, there are a lot of false alarms when we use only two features. Thus, it is still a challenge on how to ensure quality control on icons and names of applications in the market.

Applications from Open-Source Projects: Our result shows that there are applications, which are modified from open-source projects, such as e-book readers, music players and map applications. Although they use different contents, such different books or songs, and change the themes, the applications are still highly similar as they use source codes from the same projects. Thus, although they do not copy from each other, they are still considered as camouflaged applications in our framework.

Applications with Different Versions: We find out that some camouflaged applications claim to be different versions from one another. They use version differentiating words, such as "HD" (High Definition), "full", "II" (two), "plus" and "pro". However, many of them do not provide additional functionalities, although they claim to be upgraded versions. It is possible that a malicious attacker tries to attract more customers by claiming to provide upgraded version of the victim application. To solve this problem, application markets should enforce that developers use the same account, when they claim to provide upgraded version of an existing application. Our detection framework for camouflaged applications can serve as an automatic policy enforcement mechanism for these kind of applications.

Internationalized Applications: Another finding of our experiment is that many international companies, such as banks, have different applications developed for different countries and languages. Unfortunately, they also use different developer ID in Google Play to update them. For instance, "Banco Weng Hang, S.A." application uploaded by "Banco Weng Hang, S.A." provides banking services in Chinese, while "Wing Hang Bank" application uploaded by "Wing Hang Bank Ltd" provides the same services in English. This is actually a vulnerability, which allows attackers to impersonate as legitimate applications and launch phishing attacks.

7 Related Work

Studies on the repackaged applications have become popular recently. Zhou et al. [30] studies 1260 Android malwares and finds out that 1083 malwares are repackaged applications. Balanza et al. [1] analyzes a repackaged malware, called DroidDreamLight and states that trojanizing or repackaging is common form of infection in Android market. Jung et al. [11] launches repackaging attack on bank applications. Moreover, Vidas et al. [22] shows that some malwares are even repackaged with the valid certificates from original developers. It also proposes an authentication protocol for market applications which makes it difficult for an attacker to perform repackaging.

Chen et al. [2] also studies the underground economy of Android application plagiarism. Similarly, Gibler et al. [6] studies the impact of repackaged applications and finds out that 14 % of original developers' revenues and 10 % of user based are redirected to the attacker. Zheng et al. [26] presents various obfuscation techniques which allow automatic repackaging of original malwares to different variants. Transformed malwares are then used to test the robustness

of Android anti-virus systems. Potharaju et al. [21] uses permission information and estimates that 29.4 % of applications are likely to be plagiarized. They also detect repackaged applications using Deckard [10], which is a tree-based detection algorithm of cloned codes.

DroidMOSS [29] and Juxtapp [7,13] and apply fuzzy hashing on program instruction sequence and derive the similarity score by calculating the edit distance between two generated fingerprints. Crussell et al. [3] proposes DNADroid, which uses Program Dependence Graph(PDG) to determine code similarity. DNADroid is similar to our approach because it filters the applications based on application names, packages, markets, owners and descriptions. However, such filtering is performed only to make the PDG comparison more scalable for determining the similarity between two applications.

AnDarwin [4] applies Locality-Sensitive Hashing (LSH) to detect the repackaged applications. Zhou et al. [28] calls repackaged applications as "piggybacked" applications and proposes linearithmic search algorithm in a metric space to detect them. Desnos et al. [5] proposes an algorithm, which uses Normalized Compression Distance (NCD) to analyze the similarity and differences between two Android applications. Similarly, Lin et al. [15] apply thread-grained system call sequences to detect repackaged applications. Ko et al. [12] extract k-gram based software birthmarks from the dissembled codes and measure the similarity of DEX files.

Huang et al. [8] proposes an evaluation framework for detection algorithms of repackaged application by measuring their resilience to obfuscation methods. Different from other approaches, [27] proposes to use software watermarking to prevent repackaging. In summary, researchers have proposed different ways of detecting repackaged applications by measuring the source code similarity or software watermarking. However, none of them have yet considered camouflaged applications, which have very similar user interfaces, instead of similar source codes.

8 Conclusion

Our paper highlights the existence of camouflaged applications in mobile application markets as well as their exposed risk on application users and developers. Although there have been papers about repackaged applications and their copyright infringement, our paper is the first to introduce the concept of camouflaged applications and consider their user interface similarity. Our paper describes a proper threat model of camouflaged applications, including their attack scenarios and attackers' motivations. Moreover, we propose a simple, yet effective, detection framework, which applies text and image retrieval systems that are accurate and scalable in detecting camouflaged applications. The proposed framework is tested and the experiment result shows that 477 applications are camouflaged. We analyze these camouflaged applications, discuss their behaviors and calculate the false alarm rates. Our paper shows that detecting camouflaged applications is important, not only for maintaining a safe mobile application market but also for controlling the quality of mobile applications.

References

1. Balanza, M., Abendan, O., Alintanahin, K., Dizon, J., Caraig, B.: Droiddreamlight lurks behind legitimate android apps. In: Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software, MALWARE 2011, pp. 73–78. IEEE Computer Society, Washington, DC (2011)
2. Chen, H.: Underground economy of android application plagiarism. In: Proceedings of the First International Workshop on Security in Embedded Systems and Smartphones, SESP 2013, pp. 1–2. ACM, New York (2013)
3. Crussell, J., Gibler, C., Chen, H.: Attack of the clones: detecting cloned applications on android markets. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 37–54. Springer, Heidelberg (2012)
4. Crussell, J., Gibler, C., Chen, H.: Scalable semantics-based detection of similar android applications. In: 18th European Symposium on Research in Computer Security, ESORICS 2013, Egham, U.K. (2013)
5. Desnos, A.: Android: static analysis using similarity distance. In: Proceedings of the 2012 45th Hawaii International Conference on System Sciences, HICSS 2012, pp. 5394–5403. IEEE Computer Society, Washington, DC (2012)
6. Gibler, C., Stevens, R., Crussell, J., Chen, H., Zang, H., Choi, H.: Adrob: Examining the landscape and impact of android application plagiarism. In: Proceedings of 11th International Conference on Mobile Systems, Applications and Services (2013)
7. Hanna, S., Huang, L., Wu, E., Li, S., Chen, C., Song, D.: Juxtapp: a scalable system for detecting code reuse among android applications. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 62–81. Springer, Heidelberg (2013)
8. Huang, H., Zhu, S., Liu, P., Wu, D.: A framework for evaluating mobile app repackaging detection algorithms. In: Huth, M., Asokan, N., Čapkun, S., Flechais, I., Coles-Kemp, L. (eds.) TRUST 2013. LNCS, vol. 7904, pp. 169–186. Springer, Heidelberg (2013)
9. Huang, J., Kumar, S.R., Mitra, M., Zhu, W.-J., Zabih, R.: Image indexing using color correlograms. In: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR 1997), CVPR 1997, pp. 762–768. IEEE Computer Society, Washington, DC (1997)
10. Jiang, L., Misherghi, G., Su, Z., Glondu, S.: Deckard: scalable and accurate tree-based detection of code clones. In: Proceedings of the 29th International Conference on Software Engineering, ICSE 2007, pp. 96–105. IEEE Computer Society, Washington, DC (2007)
11. Jung, J.-H., Kim, J.Y., Lee, H.-C., Yi, J.H.: Repackaging attack on android banking applications and its countermeasures. Wirel. Pers. Commun. **73**(4), 1421–1437 (2013)
12. Ko, J., Shim, H., Kim, D., Jeong, Y.-S., Cho, S.-J., Park, M., Han, S., Kim, S.B.: Measuring similarity of android applications via reversing and k-gram birthmarking. In: Proceedings of the 2013 Research in Adaptive and Convergent Systems, RACS 2013, pp. 336–341. ACM, New York (2013)
13. Li, S.: Juxtapp and DStruct: detection of similarity among android applications. Master's thesis, EECS Department, University of California, Berkeley, May 2012
14. Liebergeld, S., Lange, M.: Android security, pitfalls and lessons learned. In: Gelenbe, E., Lent, R. (eds.) Information Sciences and Systems 2013. LNEE, vol. 264, pp. 409–417. Springer, Heidelberg (2013)

15. Lin, Y.-D., Lai, Y.-C., Chen, C.-H., Tsai, H.-C.: Identifying android malicious repackaged applications by thread-grained system call sequences. Comput. Secur. **39**, 340–350 (2013)
16. Lux, M., Chatzichristofis, S.A.: Lire: lucene image retrieval: an extensible java cbir library. In: Proceedings of the 16th ACM International Conference on Multimedia, MM 2008, pp. 1085–1088. ACM, New York (2008)
17. Marques, O., Lux, M.: Visual information retrieval using java and lire. In: Hersh, W.R., Callan, J., Maarek, Y., Sanderson, M. (eds.) SIGIR, p. 1193. ACM (2012)
18. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action: Covers Apache Lucene 3.0, 2nd edn. Manning Publications Co., Greenwich (2010)
19. Orland, K.: Fake pokemon yellow rises to no. 3 position on itunes app charts (2012)
20. Play, G.: Intellectual property
21. Potharaju, R., Newell, A., Nita-Rotaru, C., Zhang, X.: Plagiarizing smartphone applications: attack strategies and defense techniques. In: Barthe, G., Livshits, B., Scandariato, R. (eds.) ESSoS 2012. LNCS, vol. 7159, pp. 106–120. Springer, Heidelberg (2012)
22. Vidas, T., Christin, N.: Sweetening android lemon markets: measuring and combating malware in application marketplaces. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013, pp. 197–208. ACM, New York (2013)
23. Wang, J.Z., Li, J., Wiederhold, G.: Simplicity: semantics-sensitive integrated matching for picture libraries. IEEE Trans. Pattern Anal. Mach. Intell. **23**(9), 947–963 (2001)
24. Xiang, G., Hong, J.I.: A hybrid phish detection approach by identity discovery and keywords retrieval. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 571–580. ACM, New York (2009)
25. Zhang, Y., Hong, J.I., Cranor, L.F.: Cantina: a content-based approach to detecting phishing web sites. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 639–648. ACM, New York (2007)
26. Zheng, M., Lee, P.P.C., Lui, J.C.S.: ADAM: an automatic and extensible platform to stress test android anti-virus systems. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 82–101. Springer, Heidelberg (2013)
27. Zhou, W., Zhang, X., Jiang, X.: Appink: watermarking android apps for repackaging deterrence. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS 2013, pp. 1–12. ACM, New York (2013)
28. Zhou, W., Zhou, Y., Grace, M., Jiang, X., Zou, S.: Fast, scalable detection of "piggybacked" mobile applications. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013, pp. 185–196. ACM, New York (2013)
29. Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applications in third-party android marketplaces. In: Proceedings of the Second ACM Conference on Data and Application Security and Privacy, CODASPY 2012, pp. 317–326. ACM, New York (2012)
30. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: IEEE Symposium on Security and Privacy, pp. 95–109. IEEE Computer Society (2012)