International Journal of Early Childhood Special Education (INT-JECSE)
DOI: 10.9756/INT-JECSE/V13I2.211190 ISSN: 1308-5581 Vol 13, Issue 02 2021

ROBUST APP CLONE DETECTION BASED ON SIMILARITY OF UI STRUCTURE

[1]Dr.N.Sudha. 2Dr.G.Chitra Ganapathi, [2]Dr. R.Kumar
[1]Professor, [2] Professor, [3]Asscociate Professor
Department of Computer Science and Engineering
CMS College of Engineering and Technology,
Coimbatore, Tamil Nadu, India

App cloning is a serious danger to the mobile app market, since it not only depletes the income of original developers but also facilitates in the spread of infections. App clone detection has received a lot of interest in our academic world, and various algorithms have been presented, the majority of them rely on code or visual similarities between apps. In the field, however, sly plagiarists may alter the code or content of the User Interface (UI), rendering present methods useless.
It proposes a reliable UI structure similarity-based app clone detection solution. Plagiarists are more likely to alter content features (such as background colour) than structure features (such as general hierarchical structure, widget hierarchy).

INTRODUCTION

With over 2.8 million apps in Google Play and billions of downloads in the last year, mobile apps have become increasingly popular. By 2020, the mobile app business is anticipated to be worth $189 billion, attracting millions of developers, including malevolent coders and hackers. Because of the openness and popularity of the Android operating system, software clones have been around since the PC era. The plagiarists' purpose in app clone is to obtain subscribers and recognition by imitating the main function, UI, and even product names and trademarks of legal apps. Furthermore, third-party Android markets have grown and expanded in great numbers, all of which are available on the Android platform. legitimate developers and plagiarists to upload their apps. Most app stores' review mechanisms are ineffective, and management based on user input data is too late to prevent the spread of software clones. According to recent research, app clones and phone apps are frequently detected in Google Play, and Google Play takes a long time to delete them.Because app clones can hurt developers and the mobile app ecosystem, it's vital to identify them as soon as possible. There have been several approaches described for detecting Android app clones. We propose a new method for detecting android app clones based on UI structural similarities that is obfuscation-resistant and effective even when the UI content has been changed. On several fronts, we evaluated our strategy's Experiments revealed that we could reach a 99.6% accuracy rate, and we discovered a number of real-world app clones.

LITERATURE SURVEY

APP CLONE AND FAKE APP DETECTION

There have been numerous studies into android app clone detection. The majority of prior methods rely on static features derived from code. To create the fingerprint, Zhou used the fuzzy hashes of each approach. Zhou et al.(2012)split an app's code into primary and non-primary components. modules, then a semantic To solve the difficulty of detecting "piggybacked" apps, a feature fingerprint for each principal module can be extracted. To detect code reuse in Android apps, Just App used feature hashing. Wang et al. (2019) presented a two-phase strategy that combines coarse-grained detection using light-weight static characteristics and fine-grained detection using more detailed features. A number of solutions for efficiently filtering TPLs have been proposed to eliminate the effects of third-party libraries.Androguard was developed to compare the similarities between apps based on control flow graph to capture the high-level semantic information of the code. DNADroid was created by Desnos and Geoffroy to check app repackaging using a programme dependency graph. Finally, they use a filter to exclude improbable clones before employing a subgraph isomorphism to compare the remaining PDG pairs that passed the filter. AdDarwin divides PDGS into connected components and extracts each component's semantics vector. The semantic vectors can also be used to identify external libraries. To find the centroid, Chen et al. extracted the methods and built a 3Dcontrol flow graph termed 3D-CFG. The centroid was then used to group comparable apps together.Differing from code similarity based approaches, several studies extract fingerprints from other files within apps to detect clones. Quadra was implemented based on the comparison of the resource files, this approach is resilient to code obfuscation, but will be affected by some changes in the resources. A furtherstudy was proposed to

demonstrate that a very low proportion of identical resource files in two apps is a reliable evidence for repackaging. View droid and Mass Vet statically analyzed the UI code to extract a graph that expresses the user interface state and transitions.

However, all these approaches that were focused on static fingerprints may be vulnerable to advanced obfuscation techniques and clone attacks. Recent years, researches mainly focused on detection techniques. Charlie et al. Proposed that they are the first to use runtime UI birthmarks to Android app clone detection. Yury et al.(2019) Extracted UIs from apps and analyzed the extracted screenshots to detect impersonation. Luka et al. (2018) Extracted features from the attribute of text components and image components to calculate similarity of UI.Malisa et al. (2019)(detected mobile app spoofing attacks which can be regarded as an advanced partial UI clone attack by leveraging user visual similarity perception, they conducted an extensive online user study and the result told how likely the user is to fall for a potentialclone attack. Our approach differs from them in that our birthmark information is extracted from interface structureinformation.

SOFTWARE PLAGIARISM DETECTION

Hyun-il*et al(2009).* Stack pattern-based birthmarks were used, which required the source code. Myles and Christian performed a static analysis of executables and proposed k-gram based static birthmarks at the op-code level. To measure the code clones in the smart contract ecosystem, he and his colleagues utilised a fuzzy hashing method.. Haruaki *et al(2015).* To detect Java programme theft, four forms of static code-level birthmarks were proposed. System call based birthmarks, core value based birthmarks, and dynamic API based birthmarks are all examples of dynamic software birthmarks.

UI- INFORMATION MINING

Information from Android app GUIs has also been analysed for software engineering, in addition to mobile security. To mine human generated app traces, add natural language interfaces, identify the inconsistency between intention and app behaviours, detect aggressive mobile advertisements, and build conversational bots, recent researchers extracted screen semantics from Android app GUIs and task flow from Android app GUI layouts. Thomas et al. (2019) proposed an automated creating semantic content technique annotations for mobile app UIs, which might be used to create new data-driven design apps and allow for fast flow search over massive datasets of interaction mining data. Siva and Christoph created P2A, a platform that allows developers to automate human perception and data entry operations. Biplabet.al (2015)and his colleagues Presented ERICA, a system that uses a scalable, human-computer method to interaction mining existing Android apps without requiring them to be modified in any way, and may be used for usability testing as well as online app indexing and searching.

EXISTING SYSTEM

Static programming although birthmark-based procedures are faster than dynamic alternatives, code repetition and the rising use of advanced obfuscation techniques may have a significant impact on detection outcomes.

Furthermore, an attacker might easily elude detection without changing the app's look, such as through UI cloning assaults. For these new attack vectors, static techniques are ineffective.

Recent research has primarily focused on detection approaches that rely on visual cues, as plagiarists would like to retain the look and feel of the UI identical to the original apps.App compatibility these approaches can handle complex UI-based assaults such as text alteration in widgets, background picture substitution, and widget size manipulation, among others. However, some plagiarists may alter the UI's widget characteristics in order to avoid discovery. Plagiarists change the background colour of the UI title widget and add or remove widgets from the bottom section of the UI.

DRAWBACKS

Genuine reviews are impossible to discern. cannot distinguish between fraud users and malware indicators. Process of executing programme and analysing code permission methods takes time.

PROPOSED SYSTEM

We have suggested a novel approach for detecting android app clones based on UI structural similarities that is obfuscation-resistant and efficient even when the UI content has been modified.

We compared our approach against a state-of-the-art code-based app clone detection approach on multiple sets of real-world apps. The results of the experiments indicated that our method performed better, with a low false positive and false negative rate.We have implemented a prototype system and applied it to more than 404,650 app pairs. Experiments revealed that we could reach a 99.6% accuracy rate, and we discovered a number of real-world app clones.

ADVANTAGES

- Becausesemantics-preserving obfuscation techniques do not influence runtime behaviour, dynamic software birthmarks are accurate and obfuscation-resistant.
- To capitalize on the popularity of the original apps, app clones are likely to have UIs that are similar to them.

- Plagiarists are more likely to change the content aspects of UIs, with little effect on the runtime presentation of the apps towards the user.

PROBLEM DEFINITION

Our academic community has given app clone detection a lot of attention, and several algorithms have been presented, most of which rely on code or visual similarities of the apps. However, in the field, sneaky plagiarists may specifically modify the code or content of the User Interface (UI), rendering current approaches worthless. We offer a powerful app clone detection method based on UI structural similarities in this study. Our approach is based on the discovery that content features (such as background colour) are more likely to be modified by plagiarists, whereas structure features (such as overall hierarchy structure, widget hierarchy structure) are relatively stable.

APP CLONE'S

An Android app clone occurs when one programmeddeliberately copies the function, UI, and even product names and brands of another lawful app in order to obtain subscribers and recognition. To gain a more comprehensive and systematic understanding of the features of app clones.

APP PREPROCESSING

We add attributes to all of the activities listed in the AndroidManifest.xml file to explicitly launch each activity using explicit intents. To decompress and repackage programmes, we use the Apk tool. We add the "intent-filter" tag to each activity node, along with the "category" and "action" sub tags to the "intent filter" tag, to the AndroidManifest.xml file. It's worth noting that the "category" tag must include the "android: name" property with the value 'android.intent.category.launcher". In the "new" APK, I repackage and sign the updated files..

UI DATA EXTRACTION

Developers often configure the user interface in the Android platform by specifying it in XML files or Java code. Developers can utilise the official SDK's widgets directly in the interface implementation process, but they can also create their own invocation widgets. As a result, extracting structure features directly from static analysis is not possible. TakingIt's also difficult to collect UI information dynamically, given the size of apps in different markets. In general, when dynamic testing tools traverse all of the components, they capture UI information. It normally constructs an app's widget tree, examines each widget's trigger methods, and then traverses the app via creating inputs. However, analysing every entry point and traversing all components takes time during the dynamic app automation process. Furthermore, previous research suggests that the code coverage of current dynamic testing techniques is inadequate.

The "AM" tool is used to implement the UI traversaThe ‘‘bounds’’ attribute represents the position and the area of the rectangular region that widget controls. On account of that the arrangement of the widgets in the dumped file is outside-to-inside, We convert it into the topto- bottom arrangement in order to facilitate the extraction of structures

Fig: Extraction of structures

STRUCTURE FEATURE XTRACTION

Each activity in an Android app can be adjusted to portrait (default) or landscape orientation by app developers. The characteristic android: screen Orientation allowed us to determine the mode. The UI is given via a widget tree in either mode, and each widget controls a specific rectangular region in the activity window and can respond to user actions. "bounds=[x1,y1][x2,y2]" denotes the area of the rectangular region, where x1 and x2 are the lower and upper bounds of the transverse axis, respectively, and y1 and y2 are the lower and upper bounds of the longitudinal axis, respectively.At last, we define the layer that is not belong to an overlap layer and each overlap layer as an ‘‘IndependentLayer’’.

We extract characteristics from three dimensions for each activity's widget data, which may be utilised to describe the visual effects of the screen from various angles.

CONCLUSION AND FUTURE ENHANCEMENTS

It presented a novel approach to detect Android app clones based on the similarity of UI structure. We extract UI structure features from view hierarchy information to generate the dynamic software birthmark. The result shows that our approach can effectively detect different types of clone attacks, including repackaging, functional cloning and UI cloning, with low false positive and false negative rate. We believe that our implemented prototype system can effectively detect and defend app cloning on the app market level. Finally, we validate the proposed system with extensive experiments on real-world App data collected from the Apple's App Store. A unique perspective of this approach is that all the evidences can be modeled by statistical hypothesis tests, thus it is easy to be extended with other evidences from domain knowledge to robust app clone detection. Experimental results showed the effectiveness of the proposed approach. In the future, we plan to study more effective fraud evidence and analyze the latent relationship among rating, review, and rankings. Moreover, we will extend our clone detection approach with other mobile App related services, such as mobile Apps recommendation, for enhancing userexperience.

REFERENCES

1. (2017). Number of Android Apps. [Online]. Available: http://www. appbrain.com/stats/number-of-android-apps
2. H. Wang, H. Li, and Y. Guo, ''Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of Google play,'' in Proc. World Wide Web Conf. (WWW), 2019, pp.1988–1999.
3. (2019). Mobile App Market to Grow 270 to 189 Billion By 2020. [Online]. Available: https://venturebeat.com/2016/11/02/mobile-app-market-togrow- 270-to-189-billion-by-2020-with-games-accounting-for-55
4. H. Wang, X. Wang, and Y. Guo, ''Characterizing the global mobile app developers: A large-scale empirical study,'' in Proc. IEEE/ACM 6th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft), May 2019, pp.150–161.
5. C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi, ''AdRob: Examining the landscape and impact of Android application plagiarism,'' in Proc. 11th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys), 2013, pp.431–444.
6. H. Wang, Y. Guo, Z. Ma, and X. Chen, ''WuKong: A scalable and accurate two-phase approach to Android app clone detection,'' in Proc. Int. Symp. Softw. Test. Anal. (ISSTA), 2015, pp.71–82.
7. L. Malisa, K. Kostiainen, M. Och, and S. Capkun, ''Mobile application impersonation detection using dynamic user interface extraction,'' in Proc. Eur. Symp. Res. Comput. Secur. Cham, Switzerland: Springer, 2016, pp. 217– 237.
8. J. Zhu, Z. Wu, Z. Guan, and Z. Chen, ''Appearance similarity evaluation for Android applications,'' in Proc. 7th Int. Conf. Adv. Comput. Intell. (ICACI), Mar. 2015, pp.323–328.
9. L. Malisa, K. Kostiainen, and S. Capkun, ''Detecting mobile application spoofing attacks by leveraging user visual similarity perception,'' in Proc. 7th ACM Conf. Data Appl. Secur. Privacy, Mar. 2017, pp. 289–300.
10. H. Wang, Z. Liu, J.Liang, N. Vallina-Rodriguez, Y. Guo, L.Li, J. Tapiador, J. Cao, and G. Xu, ''Beyond Google play: A large-scale comparative study of Chinese Android app markets,'' in Proc. Internet Meas. Conf., Oct. 2018, pp. 293–307.
11. Y. Hu, H. Wang, R. He, L. Li, G. Tyson, I. Castro, Y. Guo, L. Wu, and G. Xu, ''Mobile app squatting,'' in Proc. Web Conf. WWW, 2020, pp.1–12.
12. Y. Zhou and X. Jiang, ''Dissecting Android malware: Characterization and evolution,'' in Proc. IEEE Symp. Secur. Privacy, May 2012, pp.95–109.
13. H. Wang, H. Li, L. Li, Y. Guo, and G. Xu, ''why are Android apps removed from Google play? A large-scale empirical study,'' in Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories (MSR), 2018, pp.231–242.
14. H. Niu, T. Yang, and S. Niu, ''Clone analysis and detection in Android applications,'' in Proc. 3rd Int. Conf. Syst. Informat. (ICSAI), Nov. 2016, pp. 520–525.
15. J. Zheng, K. Gong, S. Wang, Y. Wang, and M. Lei, ''Repackaged apps detection based on similarity evaluation,'' in Proc. 8th Int. Conf. Wireless Commun. Signal Process. (WCSP), Oct. 2016, pp.1–5.
16. A. Desnos and G. Geoffroy, ''New 'open source' step in Android application analysis,'' in Proc. 10th Annu. PacSec Conf., 2012, pp.13.