Clone Analysis and Detection in Android Applications

Haofei Niu, Tianchang Yang, Shaozhang Niu
Beijing Key Lab of Intelligent Telecommunication Software and Multimedia,
Beijing University of Posts and Telecommunications, Beijing 100876, China
1052824556@qq.com, gunzai-00@163.com, szniu@bupt.edu.cn

***Abstract*—With the rapid development of mobile application markets, smartphones and mobile devices are more and more popular and widely used. However, at the same time that brings more disadvantages to users such as the probability of user information leakage is higher. Among any other threats, Android application cloning has brought tremendous risks to the market environment. It is not always feasible to distinguish the applications by comparing application release dates or file sizes. In this paper, we perform a systematic analysis in Android applications. We test the difference of obfuscated and not obfuscated applications by Androguard to verify its resistance against Proguard at first. Then explain that modified applications may bring some disadvantages to unsuspecting users by smali instrumentation. We compute the similarity value of modified and original apps with Androguard. Finally, we use a testing tool based Robotium to detect the details such as the UI of applications. The experimental results proved the effectiveness of the system in similarity detection. And we present the application of image processing technology in the similarity detection.**

Keywords-Android; Smali instrumentation; Clone analysis; Similarity

I. Introduction

Smartphones have got much popularity and demand over the last few years. The recent report shows that the amount of Android smartphone hits reached 550,000 every day in July of 2011 and 850,000 in February of 2012 [1,2]. More and more applications can be available from the download service provided by Android application stores or other third-party providers. Google Play, the largest and most popular app market, currently consists of over 2,284,573 applications submitted by various developers [3]. Most Android markets depend on users' feedback instead of reviewing applications while Apple's App Store tends to vet applications when they are submitted. Therefore, besides some advantages, it is also likely to bring some security risks to users such as personal privacy-sensitive data may be stolen. More applications can be searched through the search engine because Google allows the Android markets to self-regulate. Due to potential vulnerability to malicious applications, the policy has been criticized. A large quantity of user sensitive data is stored in smartphones, for example, the call logs, contacts, personal location information and application authentication-related user names, passwords, and so on.

Generally, Android protects the user through the sandbox which attempts to make sure that one application cannot tamper with another application or the system. The developer has to statically ask for permission to use that resource by declaring it in the manifest file of the application if he wants to meet the demand of an application to access a restricted resource. When one user wants to install an application, Android will remind the user that it needs some restrained resource such as location data and that he is authorizing the application to use the specified resources. The application will not be installed if the user do not allow to grant the permissions. However, immature developers could stealthily collect information of users by adding malicious program [4].

Incidents of cloning applications occur frequently due to the openness of Android markets. Developers can submit applications on official or any other third-party markets. Most of them tend to offer free applications which contain in-game billing for extra content or are ad-supported though they could have got money directly from their applications. The wild proliferation of applications has brought great risks to the ecosystem, including end users, app market operators and app developers, so it is significant to maintain a friendly market environment to inspire developers to keep on creating applications. That providing financial compensation for the work of developers is one of important solutions. App repackaging could pose a serious threat to end users as well as app developers. Malicious users can do great damage to the revenue stream and violate the intellectual property of original app developers, and add malicious program to infect users who do not doubt the security of applications.

Wu Zhou et al. [5] proposed and implemented a system named DroidMOSS which applies a fuzzy hashing technique to accurately analyze the differences between repackaged and original applications to measure the similarity of apps, finding that 5% to 13% of apps hosted in six third-party Android markets are repackaged. Jonathan Crussell et al. [6] found that more than 141 applications had been the victims of cloning by using a tool called DNADroid which robustly computes the similarity of two applications by contrasting program dependency graphs between methods in tested applications. Wu Zhou et al. [7] designed and developed a tool called AppInk for Android apps, which embeds and extracts watermark automatically and proved the validity of the tool in preventing app repackaging through analyzing its robustness in resisting additive, subtractive and distortive attacks.

In this paper, we conduct a systematic analysis in Android applications. We make use of Androguard to compare the

National Natural Science Foundation of China (No. 61370195,U1536121).

features of obfuscated applications and not obfuscated. Secondly, we modify the application to gain some important information such as user password by smali and baksmali [8], compute the similarity value of modified and original apps. Finally, we further detect the similarity of two applications by a tool based Robotium.

The remaining part of this paper is arranged as follows. We present some relevant background knowledge in Section 2, followed by the principle of Proguard, Smali, Androguard, and Robotium in Section 3 while Section 4 gives the implementation of our system. Finally, we evaluate and discuss the experimental result and limitation in Section 5 and conclude this paper in Section 6.

II. Background and Threat Analysis

This section gives a brief introduction to the related knowledge that can help to understand the analysis.

A. *Background*

- Application signing: An app must be digitally signed by a certificate which contains the private key of the developer before being distributed to the market [9]. An application that has not been signed cannot be installed on mobile phones successfully and used. Android decides the owner of applications by the certificate, two applications have same owner if there is the same key in the signatures.
- Components of Android Application: In application layer, Android applications consist of four standard components [10,11] that realize different functions of applications. These components include: Activities, the first interface of human-computer interaction, which present information and operation results to users, the information can be obtained through resources available; Services, by which the background operation not directly related to the UI is achieved; ContentProviders, which can provide relevant processed information to Activities; and BroadcastReceivers, by which information of the application framework is passively received. Android offers a message routing system based on URIs for interactive communication between the components. Messages can be passed between the components through Intent, the carrier of information. Each of the four components is independent and closely related to each other.
- Android Application Structure: Android applications are released in compressed packages named Android Packages (APKs) [12]. The packages mainly include these directories or files: assets, lib, META-INF, res, AndroidManifest.xml, classes.dex, resources.arsc. Assets and res contain some resource files such as image files. Lib contains the dynamic library file. Signing messages are stored in META-INF, AndroidManifest.xml is the essential XML file in every Android application, it contains many parameters which Android requires to run the app, including names of Activities, the permissions the app needs and the API version and so on. It is possible that developers specify some extra information the application may use with the XML file such as advertising parameters. ".dex" is a file format that can be run directly on a virtual machine in Android system. Resources.arsc is the main resource file which contains most of the resources reference to strings.

B. *Threat analysis*

Cloning is a common phenomenon which exists in various of application software available [13,14]. It occurs when there is similar code and different ownership in two applications, so it is different from code reuse [15,16,17]. A cloned application is defined as a copied and modified app of another app, and thus has a quantity of same code as the original [18]. A plagiarized application is a cloned app that was frequently copied from a user to another. It is demonstrated that indeed there are a lot of cloned applications on mobile markets.

A plagiarist may modify the cloned code to avoid detection. Evasion techniques include: High level modifications, Method Restructurings, Control Flow Alterations, Addition and Deletion, Reordering and so on.

Malicious users may add malicious program into the original application instead of modifying the user interface of application, unsuspecting application users' sensitive information may be stolen when using the software. It is not always practicable to identify cloned applications by comparing application release dates or file sizes. In this paper, we focus on the threat brought by cloned applications and the similarity detection of the apps.

III. Methodology

In this section, we introduce the method of analyzing applications, including the principle of Proguard, smali instrumentation, and how to use Androguard and Robotium.

We present the principle of Proguard at first. Secondly, we simply analyze the format of smali files. Then we introduce the principle of Androguard. Finally, we perform the analysis through the detector based Robotium.

A. *Proguard*

Proguard is an obfuscator provided by Android, it can transform the names of Classes, Methods, Variables to some letters that are not easily recognized by users, for example, "a,b,c,d" [19]. Some relevant files are generated after running Proguard such as dump.txt, mapping.txt, seeds.txt, usage.txt. Dump.txt is an internal structure description file of class files in APK, mapping.txt is a code mapping table of obfuscation, seeds.txt presents the classes and members that are not obfuscated, usage.txt specifies some code that cannot be seen because of deletion in program source code.

B. *Smali*

Smali files can be obtained by decompiling APK with static analysis tools such as apktool or baksmali. Each class of application will generate a corresponding separate smali file after decompiled. The smali files describe some basic information related to class, for example, ".class" specifies a class name, ".super" declares a super class, ".source" presents the source file name. Every class is composed of fields and methods. There are two types of fields, static field and instance field, and there are also two types of methods, direct method and virtual method. The method starts with ".method" and ends with ".end method". The smali code describes a function by using the method name, parameter and returned value. We can modify the generated smali code after decompiling the application by smali and baksmali tools. The application can be used again after repackaged and re-signed. Figure 1 presents part of code reference to a smali file called MainActivity.smali. From the perspective of the structure of Dalvik byte code, Android applications are a collection of many classes. In fact, it records the relationship between the classes, which can be represented by directed acyclic graph.

```
.class public Lcn/itcast/files/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"


# annotations
.annotation system Ldalvik/annotation/MemberClasses;
    value = {
        Lcn/itcast/files/MainActivity$ButtonClickListener;
    }
.end annotation


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 13
    invoke-direct {p0}, Landroid/app/Activity;-><init>()V

    return-void
.end method
```

Figure 1. Part of MainActivity.smali.

C. Androguard

Androguard [20] is an open source static analysis tool. A fast analysis of APK can be performed by toolkits provided by Androguard. File permissions, major components, MainActivity information and so forth will be printed by entering command "androapkinfo.py" on the console. The screen will output parsed AndroidManifest.xml file of APK by using "androaxml.py". We can evaluate the potential risk of APK by using "androrisk.py". The different character can be gained by "androdiff.py". We can also compute the similarity of two applications by "androsim.py".

D. Robotium

There are two methods called setUP and tearDown in Robotium [21,22], of which the former is used to initialize settings, for example, starting an activity, initializing the resource, and the latter for garbage disposal and resource recovery.

We analyze the applications by a tool based Robotium. To run the test successfully, we need to configure AndroidManifest.xml file accordingly.

IV. System Design and Implementation

To confirm the similarity of Android applications, we conduct the detection through methods based on above methodology. In this section, we present the concrete implementation in detail. It mainly includes the following parts.

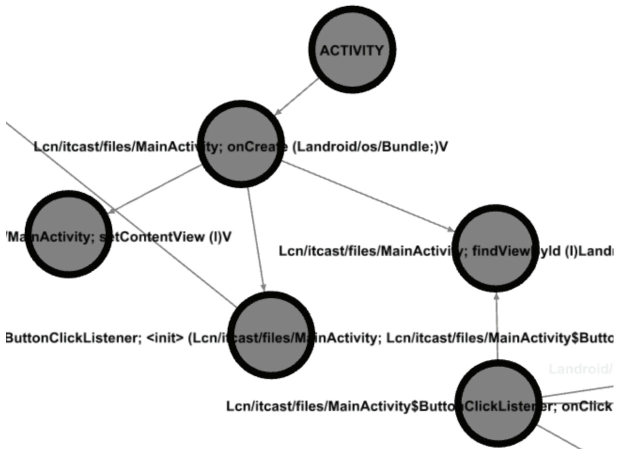
ACTIVITY
Lcn/itcast/files/MainActivity; onCreate (Landroid/os/Bundle;)V
MainActivity; setContentView (I)V
Lcn/itcast/files/MainActivity; findViewById (I)Land
ButtonClickListener; <init> (Lcn/itcast/files/MainActivity; Lcn/itcast/files/MainActivity$Butto
Landroid/
Lcn/itcast/files/MainActivity$ButtonClickListener; onClick

Figure 2. Node dependency graph.

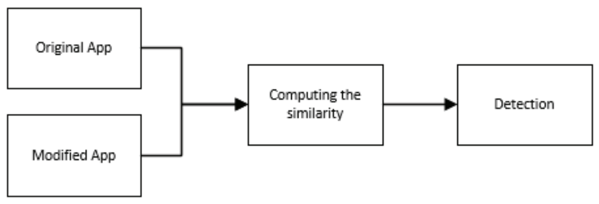

Figure 3. Clone analysis.

A. Build node dependency graph

Graphic files can be obtained from Android applications with Androguard, we can view and edit the files by a tool named Gephi. In the system, firstly, we use Androguard to detect applications of obfuscated and not obfuscated to explain its resistance against the obfuscation technology. It will generate a node dependency graph, which is clearly illustrated in figure 2.

B. Clone analysis and detection

Plagiarism between applications is analyzed by the comparison of extracted features from the applications [23,24]. The evasion techniques can hardly change the relationship of program dependence graph (PDG) [25,26]. There should be the coincident dependencies between the input and output variables if the function of copied program is the same as original. The calculation of similarity value is based on Androguard. We conduct the detection based Robotium, and the analysis process is shown in figure 3.

In the system, we demonstrate that the application pairs are similar by examining some information of them, for instance, the UI.

V. EXPERIMENT AND EVALUATION

To verify the influence of obfuscation on Android applications, we confirm the difference [27] of obfuscated and not obfuscated applications at first, then evaluate the similarity of them and conduct the cloning detection at last.

A. Experimental data

We use Proguard provided by Android to obfuscate the App at first before detecting the difference and similarity. The test data is separately shown in figure 4 and figure 5. From the figures, we can know that there are no new methods and deleted methods between obfuscated and not obfuscated applications and the similarity value is 100.000000%, so we conclude that Androguard can defend against the transformation based Proguard.

```
Elements:
          IDENTICAL:    4031
          SIMILAR:      0
          NEW:          0
          DELETED:      0
          SKIPPED:      0
NEW METHODS
DELETED METHODS
```

Figure 4. Defference of applications.

```
Elements:
          IDENTICAL:    4031
          SIMILAR:      0
          NEW:          0
          DELETED:      0
          SKIPPED:      0
         --> methods: 100.000000% of similarities
```

Figure 5. Similarity of applications.

There may be malicious programs in modified applications, which bring threat to users such as infecting unsuspecting users to gain the user password. The verification can be conducted by smali instrumentation. We analyze the smali files and get the control flow and data flow of programs after decompiling the application, then we can get the login password. The verification system is shown in figure 6. We know the password is “changshengjian” from information printed by logcat.

The UI of application is analyzed through the detection program, as is shown in figure 7.

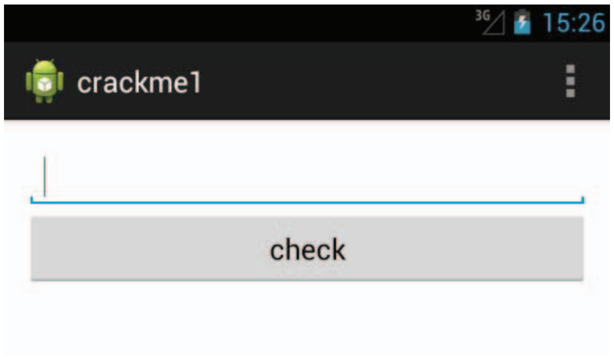

Figure 6. Verification system.

```
public class UITextTest extends
ActivityInstrumentationTestCase2<MainActivity>{
   private Solo solo;
   public UITextTest() {
      super("cn.itcast.app", MainActivity.class);
      }
   public void setUp() throws Exception {
      solo=new Solo(getInstrumentation(), getActivity());
      }
   public void testUI() throws Exception {
      boolean expected = true;
      boolean actual =solo.searchText("Hello")
&&solo.searchText("world");
      assertEquals("This and/or is are not found",
expected, actual);
   }
}
```

Figure 7. Detection program.

Images of resource folders in applications may be copied from other applications and modified as well. We can detect the images by image processing technology, for example, sift algorithm [28]. Figure 8 and Figure 11 are two original pictures downloaded from internet. Modified files are shown in Figure 9 and Figure 12. The detection result is shown in Figure 10 and Figure 13. It shows that seven matching points are found from Figure 9, and the similarity value of Figure 11 and Figure 12 is 0.929145.

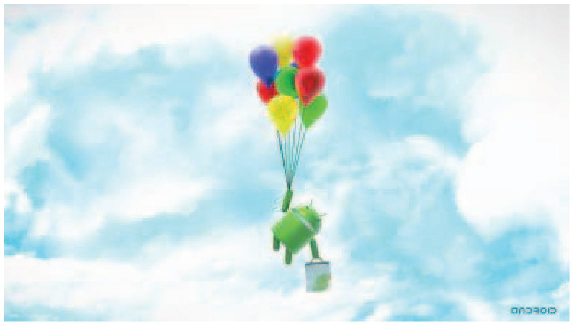

Figure 8. Original balloon.

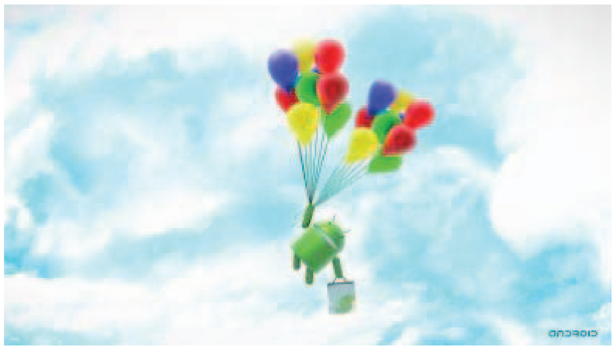

Figure 9. Modified balloon.

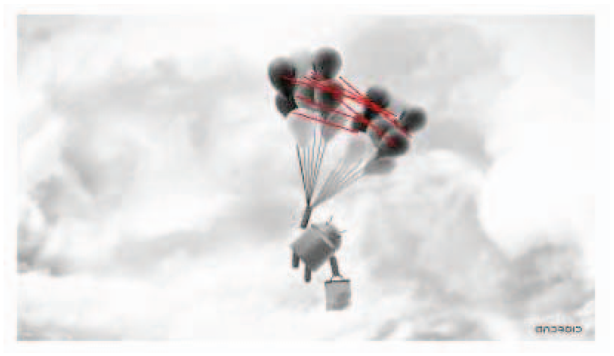

Figure 10. Matching result of balloon.

B. Discussion

There may be varieties of potential threats in modified applications, we just give the example of password to explain that cloned applications may bring some disadvantages to unsuspecting users. Password is the important information of users, the leakage of user password will lead to significant damage to the user. Firstly, we use Androguard to generate the node dependency graph, then measure the difference and similarity value of two applications , and detect the details such as the UI of applications by the detector based Robotium at last.

Jonathan Crussell et al. contrast program dependency graphs between methods to compute the similarity of tested applications by DNADroid. They confirmed that the application pairs identified by DNADroid are in fact similar through manual verification, such as examining their GUI and user interactions. Wu Zhou et al. analyze the differences between repackaged and original apps to measure the similarity of apps by DroidMOSS which applies a fuzzy hashing technique. In measuring software similarity, DroidMOSS needs to use a sliding window to produce the fingerprint, and compute the hash value to compare against a reset point to ulteriorly localize repackaged changes. The system obtains the similarity of two apps by extracting their instruction sequences into relevant fingerprints.

There are still some weaknesses in our detection method. We can increase the reliability of detection by the GUI ripping based technique [29]. Some plagiarists may adopt powerful anti-reverse analysis methods, for example, advanced program transformations, to evade similarity detection. Full automatic analysis remains challenging for Android applications though some researchers have developed some models for reversing engineer by fully or partially automated analysis techniques.

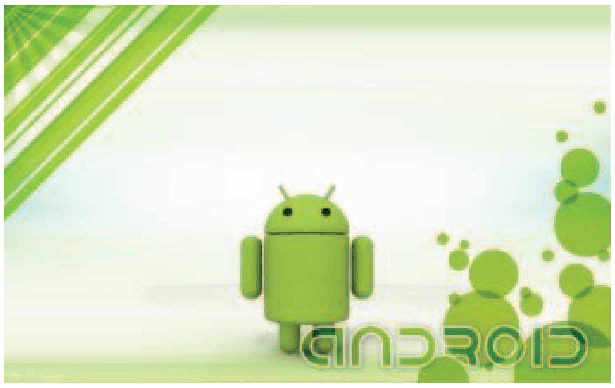

Figure 11. Original bubble.

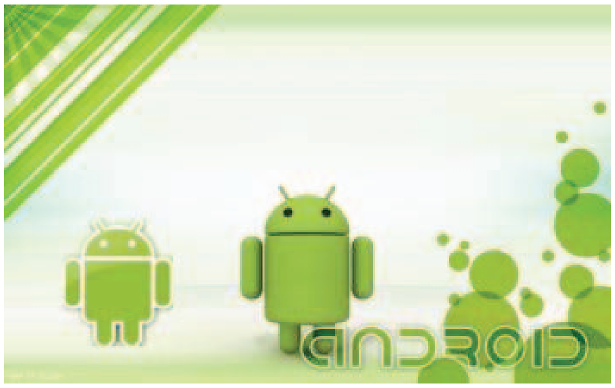

Figure 12. Modified bubble.

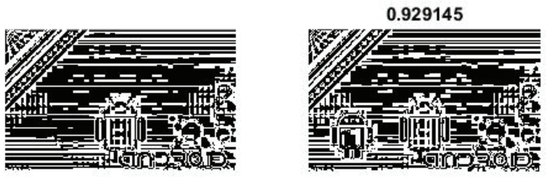

Figure 13. Matching result of bubble.

VI. Conclusions

In this paper, we perform a systematic analysis in Android applications. We test the difference of obfuscated and not obfuscated applications by Androguard to verify its resistance against Proguard. Then explain that modified applications may bring some disadvantages to unsuspecting users by smali instrumentation. We compute the similarity value of modified and original apps with Androguard. And we use a testing tool based Robotium to detect the details such as the UI of applications. It is not always feasible to distinguish the applications by comparing application release dates or file sizes. The experimental results proved the effectiveness of the system in similarity detection. We will focus on improving the efficiency of the system in future. Images of resource folders in applications may be copied from other applications and modified as well, so we also present the application of image processing technology in the similarity detection, it needs to be taken into account that there may be cloned images when we detect the whole application. The next step we will pay more attention to this aspect.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (No. 61370195，U1536121).

REFERENCES

[1] G. Kumparak. Android Now Seeing 550,000 Activations Per Day. Jul, 2011. http://techcruch.com/2011/07/14/android-now-seeing-550000-activations-per-day/.

[2] M. Burns. 850k daily android activations, 300m total devices, says andy rubin. Feb, 2012. http://techcrunch.com/2012/02/27/850k-android-activations-daily-300m-total-devices-says-andy-rubin/.

[3] AppBrain. Number of Android applications. July, 2016. http://www.appbrain.com/stats/number-of-android-apps.

[4] M Christodorescu, S Jha, C Kruegel. Mining specifications of malicious behavior. Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM, 2007:5-14.

[5] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. DroidMOSS: Detecting Repackaged Smartphone Applicatons in Third-party Android Marketplaces. In Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy, CODASPY '12, February 2012.

[6] Jonathan Crussell, Clint Gibler, and Hao Chen. Attack of the Clones: Detecting Cloned Applications on Android Markets. In 17th European Symposium on Research in Computer Security, ESORICS 2012, September 2012.

[7] Wu Zhou, Xinwen Zhang, and Xuxian Jiang. AppInk: Watermarking Android Apps for Repackaging Deterrence. In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, ASIACCS '13, pages 1-12, New York, NY, USA, 2013.

[8] Smali/Baksmal-An Assembler/Diassembler for the dex format. http://bitbucket.org/JesuFreke/smali. July, 2016.

[9] Sign Your App. http://developer.android.com/studio/publish/app-signing.html. July,2016.

[10] Han Jideng, Zhang Wen, et al. Research of component hijacking vulnerability on Android platform. Journal of Network New Media. 2014, 3(06):15-19. (in Chinese)

[11] Long Lu, Zhichun Li, Zhenyu Wu, et al. ChEX: statically vetting Android apps for component hijacking vulnerabilities. Proceedings of the 2012 ACM conference on Computer and communications security, 2012: 229–240.

[12] Clint Gibler, Jonathan Crussell, Jeremy Erickson, et al. AndroidLeaks:automatically detecting potential privacy leaks in android applications on a large scale. Proceedings of the 5th international conference on Trust and Trustworthy Computing, 2012: 291-307.

[13] Hamid Abdul Basit, Stan Jarzabek. Detecting higher-level similarity patterns in programs. Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005:156-165.

[14] Md. Sharif Uddin, Chanchai K. Roy, et al. On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems. Proceedings of the 2011 18th Working Conference on Reverse Engineering. WCRE '11, 2011:13-22.

[15] T. Kamiya, S. Kusumoto, and K.Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 28(7):654-670, 2002.

[16] Ira D. Baxter, Andrew Yahin, Leonardo Moura, et al. Clone Detection Using Abstract Syntax Trees. Proceedings of the International Conference on Software Maintenance. ICSM '98, 1998:368-377.

[17] Huiqing Li, Simon Thompson. Incremental clone detection and elimination for erlang programs. Proceedings of the 14th international conference on Fundamental approaches to software engineering: part of the joint European conferences on theory and practice of software. FASE '11/ETAPS '11, 2011:356-370.

[18] Clint Gibler, Ryan Stevens, Jonathan Crussell, et al. AdRob:examining the landscape and impact of android application plagiarism. Proceeding of the 11th annual international conference on Mobile systems, applications, and services, 2013: 431-444.

[19] ProGuard. http://directory.fsf.org/wiki/ProGuard. July, 2016.

[20] Androguard-Reverse engineering, Malware and goodware analysis of Android applications...and more. http://code.google.com/archive/p/androguard/. July,2016.

[21] Hrushikesh Zadgaonkar. Robotium Automated Testing for Android. Birmingham: Packt Publishing, 2013.

[22] Robotium-User scenario testing for Android. http://code.google.com/p/robotium/. July, 2016.

[23] H Sajnani, V Saini, J Svajlenko, et al. SourcererCC: scaling code clone detection to big-code. Proceedings of the 38th International Conference on Software Engineering, 2016:1157-1168.

[24] V Sajnani, H Sajnani, J Kim, C Lopes. SourcererCC and SourcererCC-I: tools to detect clones in batch mode and during software development. Proceedings of the 38th International Conference on Software Engineering Companion, 2016:597-600.

[25] Raghavan Komondoor, Susan Horwitz. Using Slicing to Identify Duplication in Source Code. Proceedings of the 8th International Symposium on Static Analysis. SAS '01, 2001:40-56.

[26] Jeanne Ferrante, Karl J. Ottenstein, Joe D. Warren. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 9 Issue 3, July 1987, pages 319-349. ACM New York, NY, USA.

[27] Taweesup Apiwattanapong, Alessandro Orso, Mary Jean Harrold. A Differencing Algorithm for Object-Oriented Programs. Proceedings of the 19th IEEE international conference on Automated software engineering. ASE '04, 2004:2-13.

[28] SHI Wenchang, ZHAO Fei, et al. Improving Image Copy-Move Forgery Detection with Particle Swarm Optimization Techniques. China Communications, 2016,13(1):139-149.

[29] D Amalfitano, AR Fasolino, P Tramontana, et al. Using GUI ripping for automated testing of Android applications. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ASE, 2012, 43(9):258-261.