International Journal of Computer Applications (0975 – 8887)
Volume 146 – No.6, July 2016

Piracy Detection App of Android Applications

Nandisha M. M.
Student, Department Computer Networking Engg.
Visveswaraya Technological University.India.

S. L. Deshpande
Prof. Department of Computer Networking Engg
Visveswaraya Technological University. India

ABSTRACT
Rapid increase of Smartphone users worldwide has moved developers attention towards Mobile platform to create applications for Smartphone. Android is one such major mobile platform and also an open source operating system. With the rapid increase in the android applications some undesirable apps begin to show up. Two kinds of such apps are pirated and malware. This focuses on piracy of application in android market, because one developer pirates the other developer's work. Two type of piracy present in android application, first copied java source code and other is graphical asset. This paper discusses on how to identify pirated application both graphical and java source code piracy. Androguard tool analyze similarity of bytecode in applications. With this tool, add extra feature to Combined Graphical asset comparison in Androguard. Finally, experimented result shows within 50sec can compare 20 Mb application graphical similarities and 120sec for source code comparison using Androsim (Snappy compressor).

General Terms
Security, Algorithms, Androguard, Smartphones.

Keywords
Android, Androguard, Piracy, Malware, Sourcecode, Graphical asset, Similarity, Backend, Frontend.

1. INTRODUCTION
Mobile platform is one of the world top five future trends in information technology [6]. Every year millions of Smartphone are sold in the market. Because of its usage and easiness to handle the data everywhere, the importance of Smartphone has increased in our daily life when compared to desktop and laptops. All operations available in Smartphone are same as desktop computer, so user's attention moved towards Smartphone. According to Statistics [1] number of Smartphone users in United States at the end of 2016 reaches 207.2 million, number of Smartphone users worldwide reaches more than 2 billion. As per statistics, in US, the number of smart phone users has increased from 62.6 million to 190.5 million from the year 2010 to 2015. Nearly 127.5 millions of device increased in United States only in 5 years. Assume overall world Smartphone devices.

World most Smartphone devices use two major mobile operating systems namely android and IOS. Android is developed by Google which is an open source platform it was launched to market in the year 2008. Most of Smartphone devices in the world use android operating system. Being the open source platform, its SDK is freely and easily available. It motivates to developer to develop an application in this platform. Google developed official application market called Google play store and is having 1.6million application in their dataset. Many third party applications are available in web, other than Google play store.

The format and Installation package of an application is called APK (Android Application package). APK includes AndroidManifest.xml, classes.dex, resources.arsc, META-INF and res files. It is easy to get the code from android application using Reverse Engineering tools. One developer pirates the other developer's work and makes an application similar to the original developer's application. In this paper describes about how to detect the pirated application, in terms of both frontend and backend similarity.

The following section as follows, section 2 describes literature survey about various android application similarity, section 3 for problem definition, section 4 architecture of application similarity, than graphical and code similarity technique and finally, experimental results.

2. LITEATURE SURVEY
Many similarity approaches have been proposed but final goal is to effective detection criteria. Two ways of application analysis are namely static and dynamic analysis. Dynamic analysis can examine the application at execution time only but Static analysis is without execution of application. Juxtapp is a static analysis and scalable infrastructure in android applications for detecting know malware and also identify piracy of applications based on code similarity and this architecture is ran on Amazon EC2 and is implemented in Hadoop[7]. Juxtapp more focus on scalability and working environment setup is difficult (Juxtapp requires nearly 100 minutes to completing 95k applications with on 100 8-core machines, 64GB RAM).

Androguard is static analysis tool for android application. Androguard is a powerful tool to decompile android applications and malware detection in applications. One of the features in Androguard is, application similarity identification using Androsim [2]. Androsim mainly detect the piracy of source code in applications and execution time depends on which compressor chooses. Androguard is open source tool, available freely to all users, but the issue is when downloading tool from [10] and run the tool in machine it only accepting Zlib and BZ2 compressor. This paper describes some feature which has been added to Androsim that work on snappy compressor. In further describes some compression library to increase the speed of application similarity identification.

Puppetdroid is a dynamic analysis of similarity measure of applications that more focuses on graphical similarity rather than coding similarity. Using Perceptual Hashing the system gets a hash of app screenshots so that two applications screen shot compare then find the similarity [9]. Because of dynamic analysis, each application screen short in devices and many different screen shot collections are difficult and risky.

Appearance similarity evaluations in android applications [3] is static analysis approach, more focuses on graphical assist of application. Decompile android applications and scan for text and images elements, five type of feature extracted for text

similarity, namely content of text, colour of text, background colour of text, size of text, style of text and finally, apply greedy algorithm for similarity score. For image, extract features are size of image, colour histogram of image element, 2D haar wallet transformations for image element, finally similarity comparison. Resultant Average of both text and image element get the final similarity score.

3. PROBLEM DEFINATION

From the above, we observed that Androguard tool can use small developer to big vendors can compare their application with other applications but only backend source code. Problem is every developer need to know their application similarity to another application with same features, before it is published to market, but the environment setup to check similarity of application is difficult, tools for similarity check are not available easily to developer, both source code and GUI similarity needs to be checked. In market places depends on two approaches to eliminate applications, one is review based and reactive approach. Former one requires security examination and mostly expert manual review, and next reactive approach, which requires user reporting, user ratings and user policing as indicators that an application is pirated. Developer or vendor can check their application with other many applications and use any one of approach take further actions if application pirated. About graphical comparison, when check backend code at the same time can compare GUI element based on comparison of text and image as a feature, used in applications.

4. ARCHITEATURE

In this architecture separated two sections, one for backend source code comparison and another, frontend graphical similarity comparison shows in Figure 1. Classes.dex includes all java bytecode which compiled from java source code and res folder include graphical related elements. After decompile of apk can get all text visible in display screen and images are available in drawable folders. In the next section how to detect graphical similarity between applications and further some discuss on Androsim with snappy compressor to increase speed of Androguard.

5. GRAPHICAL SIMILARITY OF APPLICATION

The growth of applications in Android marketplaces like Google play store and other third-party marketplaces is due to lack in security. GUI between the legitimate or popular app and malicious app is somewhat similar, so that it becomes difficult for Android users to differentiate them. Backend java source code piracy identification is not solving the piracy problem because world dealing with non technical people called as users. Before user fakery because of frontend, need to avoid this problem. Hence the motive is to detect both backend (java source code) and frontend (GUI) application similarity.

There are three ventures to compare likeliness of changed Android applications. This includes application preprocessing, Feature extraction and similarity comparison. As already discussed in case of apk, it is necessary to extract the GUI related features inside resource directory. When an application is complied, xml files get converted to binary format. AAPT (Android Asset Packaging Tool) can be used to get original xml code in resource directory. This set-up compares the two android applications based on GUI Similarity, from two significant features namely image and text.

5.1 Image Similarity

Because of large set of images inside an apk, makes it difficult to compare each and every image. Cryptographic hash function can be used to check the integrity of data. Example: SHA-1, MD5, SHA-256[4]. Small partial data modification changes the lot of variation in hash values, is called avalanched effect. This method of identifying the similarity of image is too difficult because if the slight modification of image like background color is changed, the image is cropped or rotated or if just one pixel is modified out of the original image, it is not possible to match the hash of the image to an already existing one. This is not practical solution to check the similarity of images in applications. Need to detect similar images, even if they have been modified a little. For this scenario perceptual hashing algorithm works on fingerprint of data by deriving it various features from content (images).

Mainly three perceptual algorithm namely AHash, PHash, DHash.

- **AHash (Average Hash or Mean Hash):** This methodology change over the pictures into grayscale 8x8 pictures and sets the 64 bits in the hash in view of whether the pixels quality is more noteworthy than the normal shading for the picture.
- **PHash (Perceptive Hash):** This calculation is like past, however utilize a discrete cosine changes (DCT) and looks at in view of frequencies instead of shading qualities.
- **DHash (Difference hashing):** Like AHash and PHash, DHash is truly easy to actualize and is significantly more precise than it has any privilege to be. As a usage, DHash is almost same as AHash however it performs much better. While AHash concentrates by and large values and PHash assesses recurrence designs, DHash tracks gradients.

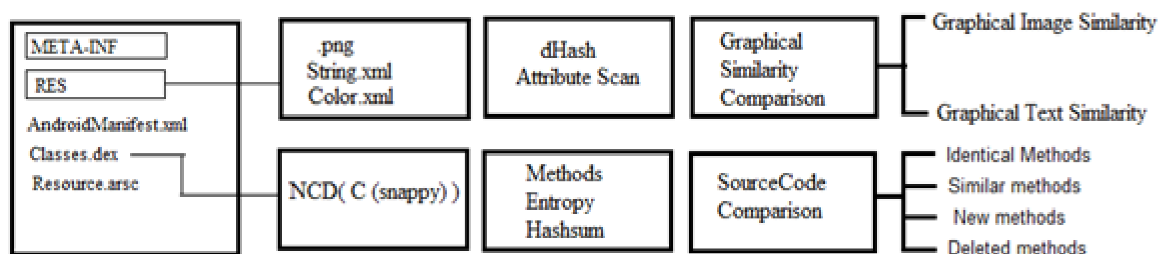

Figure 1: Architecture diagram of application similarity in both cases graphical and source code

For better performance and speed, choose DHash algorithm which compute the difference in Radiance between neighbor pixels, detecting the relative gradient direction. In detail the DHash algorithm has four steps; first, convert the image into

gray scale image because it reduces each pixel value to a luminous intension value. For example, a white pixel (255, 255,255) will be reduced to an intensity value of 255 and black for 0.Second Soften the image to a common size or shrinking the image to a common base size, here important thing is resizing or stretching an image which won't affect its hash value and all images in application are normalized to a common size. After the previous two steps compare adjacent pixels which are left with a list containing intensity values, then compare each intensity value to its adjacent value for each row resulting in an array of binary values. Finally, resultant difference converts it into bits to make it easy to store as a hexadecimal string.

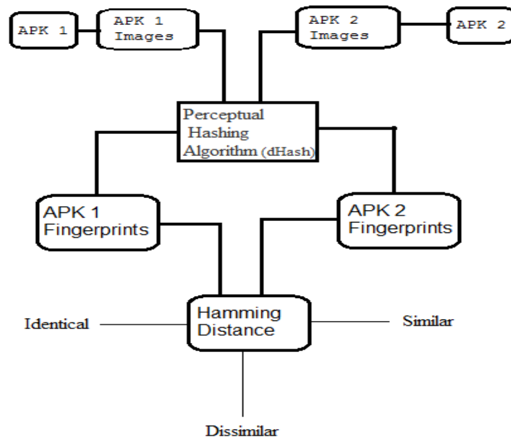

Figure 2: Image comparison using perceptual algorithm

After completion of above four steps, resulting 64 bit hexadecimal image fingerprint that is called image hash. Algorithm 1 shows the concept of image comparison. Resultant hash can be stored in database once calculated shown in Figure 2. To check similarity between two images hamming distance is used to compare two image fingerprints. Now fix the acceptable range A. If the both fingerprint match exactly same that is called identical image or Low distance values will represents the images are similar, high distance values represent that the images are different, this shown in Figure 2. This is one way to detect similarity between images.

Algorithm 1: Two Image comparisons using Perceptual hash

I1 = 16bit hexadecimal fingerprint

I2 =16bit hexadecimal fingerprint

Where Image1=I1 Image2=I2

Hamming Distance= I1 (XOR) I2

Assume A=10

Where A=acceptable value

IF hamming distance= 0

Images are identical

Else if hamming distance >0 && <A

Images are Similar

Else

Images are Dissimilar

5.2 Text Similarity

In application preprocessing remove the package of an application and convert to binary xml to original xml using aapt(Android Asset Packaging Tool) and In order to compare GUI similarity of applications in feature extraction, GUI related symptom of an app are extracted and later for similarity comparison it is expressed in feature vector. In subtle element, it incorporates an arrangement of elements identified with the visual connection of a showcase, which incorporates every text appearing in the screen with a few distinctive properties like context of text, size of text, Background of text, color of text, textual style of content.

To extricate the setting estimation of the context component, have to scan the attribute "android:text" to get its value otherwise have to index strings.xml file to get the text value. For this component, indicate it as f1. To get a context color, need to scan attributes "android: textColor" in color.xml this indicate as f2. Same like above f1 and f2 staying three "android: background", android: textSize", "android: textStyle" denoted as f3, f4, f5 respectively.

1. For the substance estimation of the content component, which has a place with string sort, characterize by:

$$SIM(f_{1i}, f_{1j}) = 1 - \frac{d(f_{1i}, f_{2j})}{\max(length(f_{1i}), length(f_{1j}))}$$

Where: function d is the edit distance of two strings.

2. The color highlight h is a three-tuple array (R, G, B), which contains the red, green and blue part estimation of the shading. The SIM capacity is characterized as:

$$SIM(f_{2i}, f_{2j}) = 1 - \frac{(|ri-rj|+|gi-gj|+|bi-bj|)}{3\times255}$$

3. Also, the background color of the content component is taken care of with the same capacity SIM as above.

4. For the text size f4 the function is:

$$SIM(f_{4i}, f_{4j}) = 1 - \frac{|f_{4i}-f_{4j}|}{\max(f_{4i}, f_{4j})}$$

5. See the yield of SIM capacity as 1 if the content style is the same, 0 generally.

After all above five result namely f1, f2, f3, f4, f5 combine together to get the score of the similarity degree of the two text element. For understanding purpose can take 5 results as 1.25, 2.5, 2.25, 1.75, 1.6 for f1, f2, f3, f4, f5 respectively. To find Sct (the likeness score of two applications texts component), proposed a calculation to figure the closeness among all texts appearing in two applications and break down the most conceivable correspondence among content components from two unique applications [3]. Algorithm 2 describes two text element comparisons in applications. Example takes two applications android1.apk and android2.apk. Decompile the both applications using reverse engineering tool. Resultant two folders created namely android1 and android2. In res folder inside android1 application there is a values folder that contains string.xml file. This file includes all texts visible on android device screen. Scan all the text and store in database. Do same for android2 application. Now two text files ready for comparison using sequence matcher algorithm or any. Resultant value converts into percentage of text similarity.

Algorithm 2: Two apps' text element similarity score calculation

Example takes two applications: android1.apk and android2.apk

1) Convert binary xml to original xml using aapt (Reverse Engineering tool).

2) After decompiled two applications it's create two folder namely android1 and android2.
3) Scan both applications, String.xml file in values folder and get all text visible on android screen.
4) Store separately both application texts.
5) Compare two text file using sequence matcher or any text compare.
6) Finally, put result into percentage of graphical text similarity.

6. ANDROGUARD FOR APPLICATION SIMILARITY

In what manner would we be able to say two numerical article are indistinguishable, it is simple simply need to analyze all characters in that sentence, however in what capacity would we be able to say two numerical item similar not identical?. Shannon entropy, Sequence matcher, Descriptional entropy, is not practical solution for our purpose [11]. Many real world compressors are presented for similarity distance identification, in those one of the real world compressors is Normalized Compression Distance (NCD) [2]. NCD's working speed depends on compression library. In detail, identify similarity between two elements *X* and *Y* is defined as *dNCD*(*X, Y*). To understanding algorithm of NCD, let us take two elements X and Y.

- $C(X)$ and $Lx = L(C(X))$;
- $C(Y)$ and $Ly = L(C(Y))$;
- $C(X/Y)$ and $Lx/y = L(C(X/Y))$;

Where X|Y is the succession of X and Y, C is the compressor, and L is the length of a string.

Then *dNCD*(*X, Y*) is defined by:

$$d_{NCD}(X,Y) = \frac{L_{X|Y} - \min(L_X, L_Y)}{\min(L_X, L_Y)}$$

The NCD depends on the comparability of components. A compressor C is ordinary if the accompanying four sayings are fulfilled up to an added substance O (log n), where n is the maximal binary length of the components required in the inequalities:

1) Idempotency: C (xx) = C(x), and C (ε) = 0, where ε is the empty string.
2) Monotonicity: C (xy) _ C(x).
3) Symmetry: C (xy) = C (yx).
4) Distributivity: C (xy) + C (z) _ C (xz) + C (yz).

For this situation of pick a best compressor, essential thing is compressor must fulfill above four inequalities. In a genuine occasion there is a timing limitation to execute the calculation. Here pressure rate is not an essential element to pick the compressor in the event that it fulfills with the accompanying principles:

1) Compressor regards the four inequalities,
2) Compressor C(x) can compute a satisfactory measure of time.

Application comparison in Androguard project has the following steps:

- •*Create* signatures for each method.
- Detect all identical methods
- Detect which all methods are similar by using NCD (with Snappy compressor).

Androsim after execute it produces and detects the following elements as an output in android applications:

- *Identical methods.*
- *Similar methods.*
- *New methods.*
- *Deleted methods.*

For more information refer [2, 11] because this paper gives importance to graphical similarity identification. Androguard (Androsim) detect the java bytecode similarity. Extend this Androguard feature into detect graphical similarity application in efficient manner.

7. EXPERIMENTEAL RESULTS

In code similarity main concept is choose best compressor to increase the speed of similarity check. From the above section in case of choosing the compressor, compressor must satisfy four inequalities. For Androguard, Compression libraries such as Zlib, LZMP, B2Z, and Snappy satisfy these inequalities [2]. In this paper, Experimented to identify the speed of compressor with text data (.pdf/.docx) as an input. Testing environment includes Oracle Virtual box with dual core processor, Ubuntu 14.04LTS operating system 64 bits, 2GB RAM. Fig 3 and 5 shows the variation of compressed file size with respect to various compression libraries. Fig 4 and 6 shows the time difference between various compressors, text data is used as an input file (.pdf) with file size 10762187 bytes and (.docx) file with size 28535845 bytes respectively.

Observe the Fig 3,4,5,6 LZMP (Lempel–Ziv–Markov chain algorithm) which gives a good compression rate 10.7Mb pdf file converted to 7.5Mb but taking more time, compares to other compressors, but snappy gives high speed compression. Example Fig 3 LZMP takes 4.5 sec for pdf file, for the same file snappy compress 0.0172 sec. As already discussed compression rate is not an important factor but Snappy satisfy the speed in both cases but gives the worst compression rate shown in Fig 3 and 5. For time constraints Snappy is a good compressor and NCD can check similarity between elements quickly using Snappy.

Experiment between application similarity, downloaded applications from official market and other third party application markets. Conduct experiment on data set which includes official market apk's and another dataset third party application market apk's, finally experimented result shows that compare to Zlib and xz2, snappy completes its work 40% quicker than other two compressors. Experimental setup: five core processors, 4 GB RAM and Ubuntu 14.0LTS operating system. 20Mb applications hardly take 3 min to check similarity using Zlib compressor same applications using snappy take 120sec. Similarity result variation between two compressors hardly 3% more than snappy. For example if

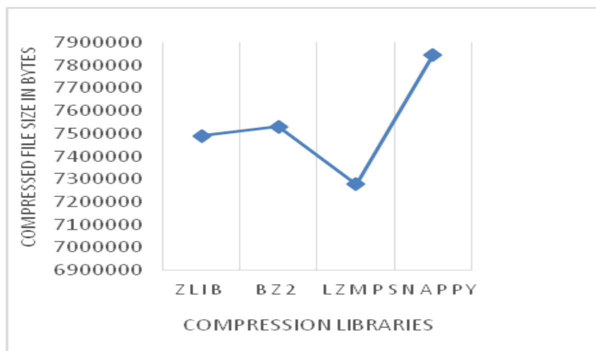

Figure 3: Variations in Compressed file size (.pdf)

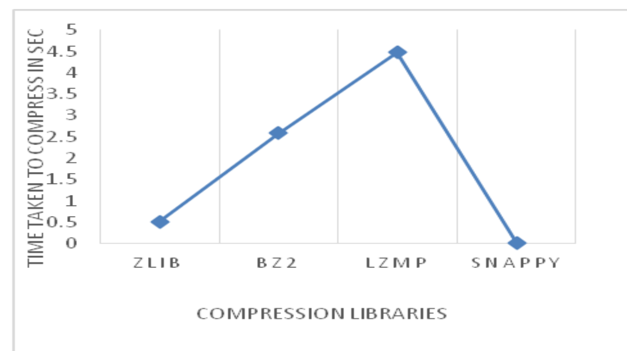

Figure 4: Variations in Time with various compressors (.pdf)

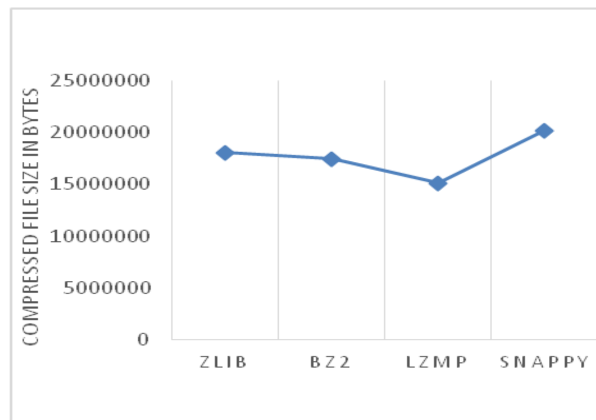

Figure 5: Variations in Compressed file size (.docx)

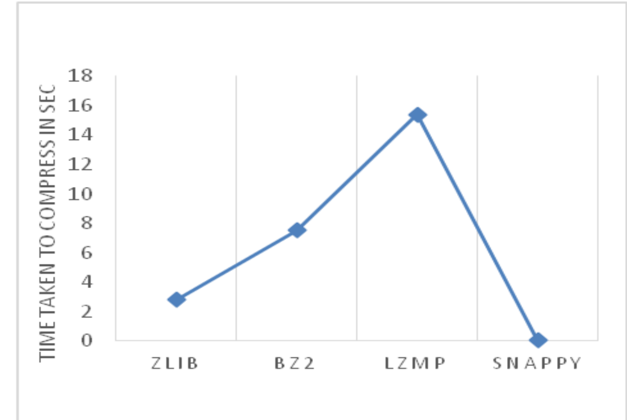

Figure 6: Variations in Time with various compressors (.docx)

Zlib give 75% of similarity between two applications, Snappy gives below 72% similarity for same applications. Same way for graphical comparison of two applications with 20Mb size, all text and image comparison gives a result within 50sec with acceptable result.

8. CONCLUSION AND FUTURE SCOPE

The proposed methodology implemented graphical similarity in terms of text and image similarity and methodology also focused on the code similarity of android application. The method displayed the comparison result, as well as in terms of percentage. The proposed method increase the application comparison speed and also added graphical similarity feature to Androguard. Existing methods are expensive in terms of cost and complexity. Proposed method reduces the burden on developer, part of simplifying environmental setup, available as an open source. Opted experiment result as shown the improved the efficiency over existing system. In future, proposed methodology extends to compare all layouts inside layout folder, after decompile of apk's, which gives the more accuracy of graphical comparison of android applications.

9. ACKNLOWGEMENT

Authors acknowledge the kind support and encouragement of their organization, Visveswaraya Technological University Belagavi, India.

10. REFERENCES

[1] http://www.statista.com/statistics/201182/forecast-of-smartphone-users-in-the-us/

[2] Anthony Desnos. 2012 Honeynet project. Android: Static Analysis Using Similarity Distance ESIEA: Operational Cryptology and Virology Laboratory (CVO).

[3] Jiawei Zhu, Zhengang Wu, Zhi Guan, and Zhong Chen. 2015 Appearance Similarity Evaluation for Android Applications. 7th International Conference on Advanced Computational Intelligence, Mount Wuyi, Fujian, China; March 27-29, 2015.

[4] http://blog.iconfinder.com/detecting-duplicate-images-using-python/

[5] http://www.hackerfactor.com/blog/?/archives/529-Kind-of-Like-That.html

[6] http://www.cio.com/article/2923453/careers-staffing/5-hot-trends-in-software-development-hiring.html#slide2

[7] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. Juxtapp: A Scalable System for Detecting Code Reuse among Android Applications. Intel labs and UC Berkely.

[8] *Saung Li*. 2012 Juxtapp and DStruct: Detection of Similarity among Android Applications. Electrical Engineering and Computer Sciences University of California at Berkeley.

[9] A. Gianazza, F. Maggi, A. Fattori, L. Cavallaro, and S. Zanero, "Puppetdroid: A user-centric ui exerciser for automatic dynamic analysis of similar android applications," ArXiv e-prints, Feb. 2014. Computer Science - Cryptography and Security.

[10] https://code.google.com/p/androguard/

[11] http://phrack.org/issues/68/15.htm