Volume 6, Issue 3, March 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Detect Malware Patten in Android Application Using Genetic Algorithm and Identify Malicious Behavior Using Neural Network

[1] Navdeep Singh, [2]Dr. Vinay Gautam
[1]M.Tech Student, [2]Professor
[1, 2] Department of Computer Science and Engineering, Chandigarh Engineering College Landran (PTU) Punjab, India

Abstract: *MALWARE is a kind of bad code which capture the sensitive information from the computer without knowing to the user and perform unwanted operation behind the scene. In Android malware is an code which effect on other running application in smartphone. Generally in android malware are running in the form of background service it overall effect on user interface of application. In our research we are introducing an new technique by using this we are identify the malware pattern in the application also we are checking the malware pattern background service of application*

Keywords: Malware, Code Clone, , Malware Analysis, Android operating system

I. INTRODUCTION

Duplicate code is a computer programming term for a sequence of source code that occurs more than once, either within a program or across different programs owned or maintained by the same entity. Duplicate code is generally considered undesirable for a number of reasons.[1] A minimum requirement is usually applied to the quantity of code that must appear in a sequence for it to be considered duplicate rather than coincidentally similar. Sequences of duplicate code are sometimes known as code clones or just clones, the automated process of finding duplications in source code is called clone detection.

The following are some of the ways in which two code sequences can be duplicates of each other:

- character-for-character identical
- character-for-character identical with white space characters and comments being ignored

II. MALWARE SOURCE CODE

Malware**,** short for malicious software, is any software used to disrupt computer operations, gather sensitive information, gain access to private computer systems, or display unwanted advertising.[1] Malicious software was called computer virus before the term malware was coined in 1990 by Yisrael Radai.[2] The first category of malware propagation concerns parasitic software fragments that attach themselves to some existing executable content. The fragment may be machine code that infects some existing application, utility, or system program, or even the code used to boot a computer system.[3] Malware is defined by its malicious intent, acting against the requirements of the computer user, and does not include software that causes unintentional harm due to some deficiency.

Malware may be stealthy, intended to steal information or spy on computer users for an extended period without their knowledge, as for example Regin, or it may be designed to cause harm, often as sabotage (e.g., Stuxnet), or to extort payment (CryptoLocker). '

III. ANDROID SMART PHONE

Android is a modern mobile platform that is designed to be truly open source. Android applications can use advanced level of hardware and software, as well as local and server data, exposed through the platform to bring innovation and value to consumers. Android platform must have security mechanism to ensure security of user data, information, application and network. Open source platform needs strong and rigorous security architecture to provide security. Android is designed with multilayered security that provides flexibleness needed for an open platform, whereas providing protection for all users of the platform designed to a software stack, android includes an operating system, middleware and core application as a complete. Android powers hundreds of millions of mobile devices in more than 190 countries around the world. Android architecture is designed with keep ease of development ability for developers. Security controls have designed to minimize the load on developers. Developers have to simply work on versatile security controls. Developers are not familiar with securities that apply by defaults on application. Android is also designed with focused on user's perspective. Users can view how applications work, and manage those applications.

IV. HOW TO GET RID OF CODE DUPLICATION?

The main advantage of removing duplicated code instead of "dead" and "gold-plated code" is that it can be done cheaply and quickly. Recent advances in static code analysis and hardware performance made possible tool-based localization of code duplication in industrial contexts. However, while many of the existing tools offer reliable and accurate results, the usefulness of the detection results can be worlds apart. The effectiveness of a given approach or tool depends very much on the context in which it is used. There are a number of aspects that have to be always considered when choosing a code duplication detection tool:

- Can the tool specify the minimal and maximal size of a code fragment that constitutes a clone? Size is typically measured in contiguous lines of code that form a duplicate.
- Can the tool cope with code formatting and decoration issues, like white spaces, annotation, and comments, when searching for duplicates?
- Does the tool recognize identifiers renaming, e.g., changing the name of variables?
- Can the tool cope with insertion/deletion/modification of code? That is, does it detect only exact duplicates, or duplicates in which small amounts of code have been altered?
- Does the tool follow statements on multiple lines?
- Are the results presented in a usable way?
- Is the application scalable enough?

All these issues are important for the effectiveness of a duplication detection tool, as follows. First, the tool should allow easy setting of the size of a code fragment considered to be a duplicate. Fixed (preset) sizes are not an option: Too small sizes will discover too many duplicates, which are useless. Too large sizes will yield too few duplicates. Second, code is in practice rarely duplicated 'verbatim'. Small-scale changes such as formatting, indentation, comments, variable renaming, and insertion/removal of small code fragments, occur when duplicating code. Hence, a duplication detection tool should be able to detect duplicates in presence of such changes, or its usability will be limited. On the other hand, not too many changes should be allowed, otherwise the whole notion of a duplicate is lost. Third, the detected duplicates must be presented in a way that makes their analysis, searching, and understanding fast and effective, otherwise this information cannot be used for software improvement. Finally, duplication tools must handle large code bases of millions of lines of code fast enough so the detection cost does not offset its advantages. A careful balance must be struck between all these aspects to have a truly useful duplication tool.

V. LITERATURE SURVEY

Shahid Ahmad Wani1,Shilpa Dang2 [1] has present in their Research Code reuse is a common activity in software development and is one of the main reasons for code clones. A code clone is a part of the source code that is identical, or highly similar, to another part (clone) in terms of structure and semantics. Various clone detection techniques and tools have been proposed over last few years. Code cloning is found to be a more somber and serious problem in industrial software systems. A large number of clone detection tools are available and in order to make use of the right tool for detection of clones very important. The aim of this study is to analyze various clone detection tools. This study would help to decide which tool is best suitable for detection of code clones. We present the background concepts of cloning, a generic clone detection process and a comparison of four clone detection tools. Keywords: Clone detection, Code clone, Code Fragment, Dynamic pattern matching (DPM).

Jian Chen 1, Manar H. Alalfi 2, Member, ACM, IEEE, Thomas R. Dean 1, and Ying Zou[2] has present in their Research Android is currently one of the most popular smartphone operating systems. However, Android has the largest share of global mobile malware and significant public attention has been brought to the security issues of Android. In this paper, we investigate the use of a clone detector to identify known Android malware. We collect a set of Android applications known to contain malware and a set of benign applications. We extract the Java source code from the binary code of the applications and use NiCad, a near-miss clone detector, to find the classes of clones in a small subset of the malicious applications. We then use these clone classes as a signature to find similar source files in the rest of the malicious applications. The benign collection is used as a control group. In our evaluation, successfully decompile more than 1 000 malicious apps in 19 malware families. Our results show that using a small portion of malicious applications as a training set can detect 95% of previously known malware with very low false positives and high accuracy at 96.88%. Our method can effectively and reliably pinpoint malicious applications that belong to certain malware families.

Xin Sun, Yibing Zhongyang, Zhi Xin, Bing Mao, Li Xie [3] has present in their Research Recently smartphones and mobile devices have gained incredible popularity for their vibrant feature-rich applications (or apps). Because it is easy to repackage Android apps, software plagiarism has become a serious problem. In this paper, we present an accurate and robust system DroidSim to detect code reuse. DroidSim calculates similarity score only with component-based control flow graph (CB-CFG). CB-CFG is a graph of which nodes are Android APIs and edges represent control flow precedence order in each Android component. Our system can be applied to detect repackaged apps and malware variants. We evaluate Droid Sim on 121 apps and 706 malware variants. The results show t**Jian Chen 1, Manar H. Alalfi 2, Member, ACM, IEEE, Thomas R. Dean 1, and Ying Zou[2]** has present in their Research Android is currently one of the most popular smartphone operating systems. However, Android has the largest share of global mobile malware and significant public attention has been brought to the security issues of Android. In this paper, we investigate the use of a clone detector to identify known Android malware. We collect a set of Android applications known to contain malware and a set of benign applications. We extract the Java source code from the binary code of the applications and use NiCad, a near-miss clone detector, to find the classes of clones in a small subset of the malicious applications. We then use these clone classes as a signature to find similar source files in the rest of the malicious applications. The benign collection is used as a control group. In our evaluation, successfully decompile more than

1000 malicious apps in 19 malware families. Our results show that using a small portion of malicious applications as a training set can detect 95% of previously known malware with very low false positives and high accuracy at 96.88%. Our method can effectively and reliably pinpoint malicious applications that belong to certain malware families.

That our system has no false negative and a false positive of 0.83% for repackaged apps, and a detection ratio of 96.60% for malware variants. Besides, ADAM is used to obfuscate apps and the result reveals that ADAM has no influence on our system.

Keivanloo[4] has present in their research a new application of Semantic Web and Artificial Intelligence in software analysis research. We show on a concrete example - clone detection for object-oriented source codethat transitivity closure computation can provide added value to the clone detection community. Our novel approach models the domain of discourse knowledge as a mixture of source code patterns and inheritance trees represented as Directed Acyclic Graphs. Our approach promotes the use of Semantic Web and inference engines in source code analysis. More specifically we take advantage of the Semantic Web and its support for knowledge modeling and transitive closure computation to detect semantic source code clones not detected by traditional detection tools.

Yang Yuan [5]has introduces CMCD, a Count Matrix based technique to detect clones in program code. The key concept behind CMCD is Count Matrix, which is created while counting the occurrence frequencies of every variable in situations specified by pre-determined counting conditions. Because the characteristics of the count matrix do not change due to variable name replacements or even switching of statements, CMCD works well on many hard-to-detect code clones, such as swapping statements or deleting a few lines, which are difficult for other state-of-the-art detection techniques. We have obtained the following interesting results using CMCD: (1) we successfully detected all 16 clone scenarios proposed by C. Roy et al., (2) we discovered two clone clusters with three copies each from 29 student-submitted compiler lab projects, (3) we identified 174 code clone clusters and a potential bug from JDK 1.6 source files.

Chanchal K. Roy∗,a, James R. Cordya , Rainer Koschkeb[6] has present many techniques and tools for software clone detection have been proposed. In this paper, we provide a qualitative comparison and evaluation of the current state-of-the-art in clone detection techniques and tools, and organize the large amount of information into a coherent conceptual framework. We begin with background concepts, a generic clone detection process and an overall taxonomy of current techniques and tools. We then classify, compare and evaluate the techniques and tools in two different dimensions. First, we classify and compare approaches based on a number of facets, each of which has a set of (possibly overlapping) attributes. Second, we qualitatively evaluate the classified techniques and tools with respect to a taxonomy of editing scenarios designed to model the creation of Type-1, Type-2, Type-3 and Type-4 clones. Finally, we provide examples of how one might use the results of this study to choose the most appropriate clone detection tool or technique in the context of a particular set of goals and constraints. The primary contributions of this paper are: (1) a schema for classifying clone detection techniques and tools and a classification of current clone detectors based on this schema, and (2) a taxonomy of editing scenarios that produce different clone types and a qualitative evaluation of current clone detectors based on this taxonomy.

Randy Smith and Susan Horwitz[7] has present most previous work on code-clone detection has focused on finding identical clones, or clones that are identical up to identifiers and literal values. However, it is often important to find similar clones, too. One challenge is that the definition of similarity depends on the context in which clones are being found. Therefore, we propose new techniques for finding similar code blocks and for quantifying their similarity. Our techniques can be used to find clone clusters, sets of code blocks all within a user-supplied similarity threshold of each other. Also, given one code block, we can find all similar blocks and present them rank-ordered by similarity. Our techniques have been used in a clonedetection tool for C programs. The ideas could also be incorporated in many existing clone-detection tools to provide more flexibility in their def initions of similar clones.

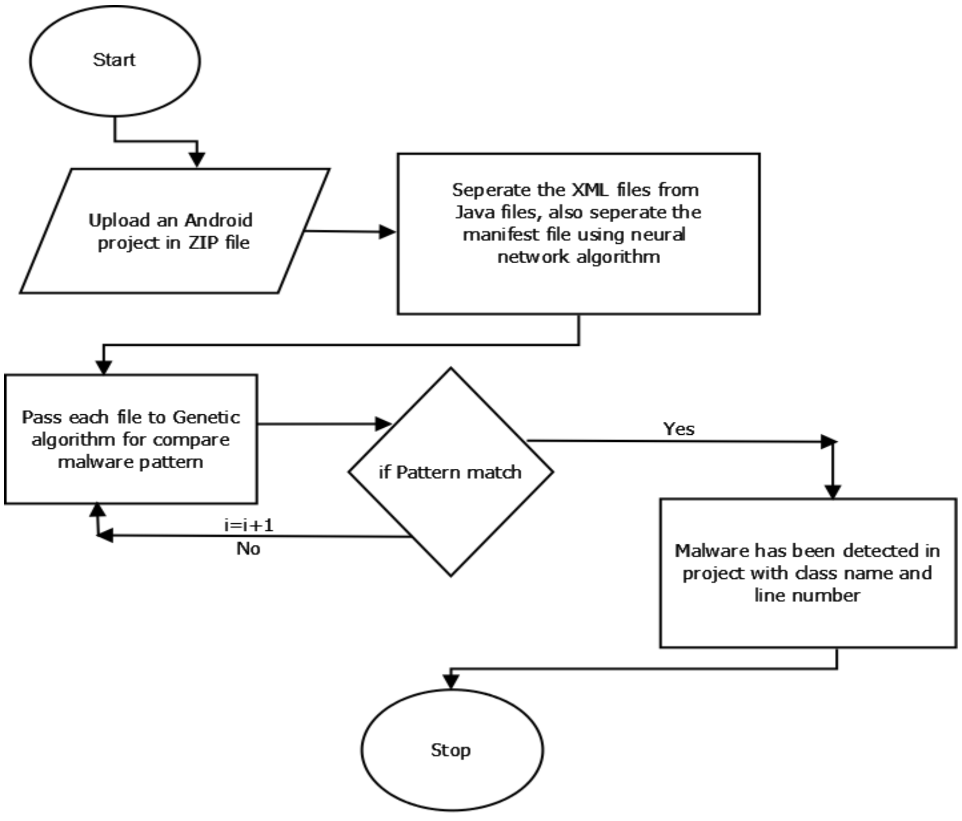
Chanchal K. Roy and James R. Cordy 2008, "Scenario-Based Comparison of Clone Detection Techniques. Reusing code fragments by copying and pasting with or without minor adaptation is a common activity in software development. As a result software systems often contain sections of code that are similar, called code clones. Code clones are often maintained separately and in time may di-verge significantly. A difficulty with such duplicated frag-ments is that if a bug is detected in a code fragment, all fragments similar to it should be checked for the same bug[32]. Duplicated fragments can also significantly increase the work to be done when enhancing or adapting code [35]. Fortunately, several (semi-)automated techniques for de-tecting code clones have been proposed, and there have been a number of comparison and evaluation studies to reate them. The most recent study, by Bellon et al. [7], provides a comprehensive quantitative evaluation of six clone detectors in detecting known observed clones in a number of open source software systems written in C and Java. Other studies have evaluated clone detection tools in other contexts [25, 8, 40, 41]. These studies have not only provided significant contributions to the clone detection research, but have also exposed how challenging it is to compare different tools, due to the diverse nature of the detection techniques, the lack of standard similarity definitions, the absence of benchmarks, the diversity of target languages, and the sensitivity to tuning parameters [1].To date no comparative evaluation has considered all of the different techniques available. Each study has chosen a number of state-of-the art tools and compared them using precision, recall, computational complexity and memory use. There is also as yet no third party evaluation of themost recent tools, such asCP-Miner[32],Deckard[20],cpdetector [25],RTF[4] and Asta[17].In this paper, we provide an overall comparison and evaluation of all of the currently available clone detection techniques, using both general criteria and a set of edit-based hypothetical scenarios for different clone types. In contrast to previous studies which concentrate on empirically evaluating tools, we aim to identify the essential strengths and weaknesses of both individual techniques in particular andgeneral approaches overall, with a view to providing a complete catalogue of available technology and its potential to recognize "real" clones, that is, those that could be create by the editing of intentionally reused code. Because this paper can fit only a basic summary, not all individual tools can be considered in detail here and our complete study can be found in a technical report [38].

Elmar Juergens, Florian Deissenboeck, Benjamin Hummel[8] has present The area of clone detection has considerably evolved over the last decade, leading to approaches with better results, but at the same time using more elaborate algorithms and tool chains. In our opinion a level has been reached, where the initial investment required to setup a clone detection tool chain and the code infrastructure required for experimenting with new heuristics and algorithms seriously hampers the exploration of novel solutions or specific case studies. As a solution, this paper presents CloneDetective, an open source framework and tool chain for clone detection, which is especially geared towards configurability and extendability and thus supports the preparation and conduction of clone detection research.

Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, Dawn Song[9] has present in their Research Mobile application markets such as the Android Marketplace provide a centralized showcase of applications that end users can purchase or download for free onto their mobile phones. Despite the influx of applications to the markets, applications are cursorily reviewed by marketplace maintainers due to the vast number of submissions. User policing and reporting is the primary method to detect misbehaving applications. This reactive approach to application security, especially when programs can contain bugs, malware, or pirated (inauthentic) code, puts too much responsibility on the end users. In light of this, we propose Juxtapp, a scalable infrastructure for code similarity analysis among Android applications. Juxtapp provides a key solution to a number of problems in Android security, including determining if apps contain copies of buggy code, have significant code reuse that indicates piracy, or are instances of known malware. We evaluate our system using more than 58,000 Android applications and demonstrate that our system scales well and is effective. Our results show that Juxtapp is able to detect: 1) *463*applications with confirmed buggy code reuse that can lead to serious vulnerabilities in real-world apps, 2) *34*instances of known malware and variants (*13*distinct variants of the GoldDream malware), and 3) pirated variants of a popular paid game.

Yujie Fan, Yanfang Ye, Lifei Chen[10] Due to its damage to Internet security, malware (e.g., virus, worm, trojan) and its detection has caught the attention of both anti-malware industry and researchers for decades. To protect legitimate users from the attacks, the most significant line of defense against malware is anti-malware software products, which mainly use signature-based method for detection. However, this method fails to recognize new, unseen malicious executables. To solve this problem, in this paper, based on the instruction sequences extracted from the file sample set, we propose an effective sequence mining algorithm to discover malicious sequential patterns, and then All-Nearest-Neighbor (ANN) classifier is constructed for malware detection based on the discovered patterns. The developed data mining framework composed of the proposed sequential pattern mining method and ANN classifier can well characterize the malicious patterns from the collected file sample set to effectively detect newly unseen malware samples. A comprehensive experimental study on a real data collection is performed to evaluate our detection framework. Promising experimental results show that our framework outperforms other alternate data mining based detection methods in identifying new malicious executables.

Flow Chart


Start
Upload an Android project in ZIP file
Seperate the XML files from Java files, also seperate the manifest file using neural network algorithm
Pass each file to Genetic algorithm for compare malware pattern
if Pattern match
Yes
i=i+1
No
Malware has been detected in project with class name and line number
Stop

VI. CONCLUSION

In this research study, we used a malware signature pattern to detect thee malicious codes in other Java source files of Android based mobiles andTablets. A single malicious code and its many variants belong to one family and are connected with an original code. Thus, identifying and detecting them by using a signature pattern fetched from a defected Java source file proved to be an accurate technique. We converted the Android APK files into Java source files, followed by a second step of passing them to Genetic Algorithm and examined them for malware clones. By following these step, we fetched 5 malware signature patterns to detect the

In this research study, we used a malware signature pattern to detect the malicious codes in other Java source files of Android based mobiles and Tablets. A single malicious code and its many variants belong to one family and are connected with an original code. Thus, identifying and detecting them by using a signature pattern fetched from a defected Java source file proved to be an accurate technique. We converted the Android APK files into Java source files, followed by a second step of passing them to Genetic Algorithm and examined them for malware clones.

REFERENCES

[1] Shahid Ahmad Wani1 , 2015"A Comparative Study of Clone Detection Tools" ⋆

[2] Jian Chen 1, Manar H. Alalfi 2, Member, ACM, IEEE, Thomas R. Dean 1, and Ying Zou "Detecting Android Malware Using Clone Detection "Chen J, Alalfi MH, Dean TR et al. Detecting Android malware using clone detection. JOURNAL OF COMPUTERSCIENCE AND TECHNOLOGY 30(5): 942–956 Sept. 2015. DOI 10.1007/s11390-015-1573-7

[3] Xin Sun, Yibing Zhongyang, Zhi Xin, Bing Mao, Li Xie " Detecting Code Reuse in Android ApplicationsUsing Component-Based Control Flow Graph"N. Cuppens-Boulahia et al. (Eds.): SEC 2014, IFIP AICT 428, pp. 142–155, 2014.cIFIP International Federation for Information Processing 2014

[4] Keivanloo 2012,"Clone detection meets Semantic Web-based transitive closure computation".

[5] Yang Yuan. 2011 , "CMCD: Count Matrix Based Code Clone Detection".

[6] Chanchal K. Roy∗,a, James R. Cordya, Rainer Koschkeb 2009 , "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach" .

[7] Randy Smith and Susan Horwitz 2009 , "Detecting and Measuring Similarity in Code Clones ".

[8] Chanchal K. Roy and James R. Cordy 2008, "Scenario-Based Comparison of Clone Detection Techniques."

[9] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel ,2008.CloneDetective – A Workbench for Clone Detection Research

[10] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, Dawn Song" Juxtapp: A Scalable System for Detecting Code Reuse among Android Applications"

[11] Yujie Fan[a], , Yanfang Ye[b], , Lifei Chen "Malicious sequential pattern mining for automatic malware detection"(2014)