RESEARCH ARTICLE OPEN ACCESS

Software Program Plagiarism Detection Using Longest Common Subsequence Method

Mahalakshmi S.[1], Kavitha S.[2]
[1](M.Phil Research Scholar, Department of Computer Science, Auxilium College, Vellore-6)
[2](HOD & Asst. Prof ,Department of Computer Science, Auxilium College, Vellore-6)

----------**********----------

Abstract:

Software plagiarism is an unlawfully stealing other person source code or program code this become serious issue for common open source program company and other software companies. It violates the logical possessions of software developers and has been a stern problem, diversity from open source code use again, software product theft to smart phone application repackaging. This research is presents new technique for software plagiarism that achieves to compare two programs similarity to find execution path of the program. The proposed system used the symbolic execution and weakest requirement reasoning to capture the semantics of execution paths and to find path distinction. Path Deviation Method is more resilient to current automatic obfuscation techniques, compared to the existing detection mechanisms. In addition, since LCS method is a formal program semantics-based method, it can provide a guarantee of resilience against many known obfuscation attacks. The results indicate that LCS method is both effective and efficient in detecting software plagiarism.

Keywords— **SCDG, VaPD, DKISB, MOSS,SIM,LCS,PDG,LHMM.**

----------**********----------

I. INTRODUCTION

Software plagiarism is an act of theft other's software by illegitimately copying their code, applying code obfuscation techniques to make the code look different and then claiming that it is one's own program in a way violating the terms of original license. In recent years, software plagiarism has become a serious anxiety for honest software companies and open source communities. It violates the intellectual property of software developers and has been a severe problem, ranging from open source code reuse, software product stealing to smartphone application repackaging. The stolen code can be used by plagiarists to reduce the cost of their software development.

The popular smartphone applications may be repackaged and injected with malicious payload to accelerate the propagation of malware. According to a recent study, it was found that 1083 (or 86.0%) of 1260 malicious app samples were repackaged versions of legitimate apps with malicious payloads. Moreover, the booming of software industry gives plagiarists more opportunities to steal other's code. The burst of open source projects (e.g., SourceForge.net has more than 430, 000 registered open source projects with 3.7 million developers and more than 4.8 million downloads a day provides plenty of easy targets for software thieves, since source code is easier to understand and modify than executable binaries.

The existing automatic code obfuscation tools (e.g., Loco, Sand Mark) can change the syntax of a program while preserving its semantics and therefore will help plagiarists to evade detection. Therefore, automated software plagiarism detection is greatly desired. However, automated software plagiarism detection is very challenging. For one reason, source code of suspicious programs is usually not available to plaintiff. The analysis of executables is much harder than source code analysis. Besides, code obfuscation is also an enormous obstacle to automatic software plagiarism detection. Code obfuscation is a technique to convert a sequence of code into a different sequence that conserve the semantics but is much more difficult to understand or analyze.

Rapid development of internet technologies simplified sharing any kinds of data. Extremely notable is also sharing the source codes. Consequently, today's "copy paste" generation is a subject of a notable problem of plagiarism. It is present in many areas, from educational and research areas to software development.

There are two types of plagiarism are more occurs:

1. **Textual plagiarisms**: this type of plagiarism usually done by students or researchers in academic enterprises, where documents are identical or typical to the original documents, reports, essays scientific papers and art design.

2. **A Source Code Plagiarism**: also done by students in universities, where the students trying or copying the whole or the parts of source code written by someone else as one's own, this types of plagiarism it is difficult to detect.

Based on the above two facts, there are two necessary requirements for a good software plagiarism detection scheme

- Capability to work on suspicious executables without the source code.
- Resiliency to code obfuscation techniques.

An important differential between source code plagiarism and free text plagiarism is that the methods used to detect both of these differ. Source code detection is a well-understood area that has not recently been the focus of much research. It is thought to be easier to detect source code plagiarism than free text plagiarism since the language that can be used is constrained to a set of defined key words and since any plagiarism is most likely intra-corpal in nature.

Free text plagiarism contains an effectively unlimited number of possible words that can be used and plagiarism may be intra or extra-corpal. Research on detecting plagiarism in free text is more recent and ongoing and has become possible due to the increasing availability of cheap computer processing power.

A. Motivation and Scope of this Research Work

Plagiarism has become very common in educational institutions. Students copy without any hesitation other student's assignments, both text and source code, to complete their work in time or to complete their work in a better way. Many students seldom care to put their time and effort into doing the assignments on their own when it is far simpler and effortless to copy from someone else. However, it is necessary to differentiate the original work from plagiarized work.

Software plagiarism has become a serious threat to maintaining a healthy and trustworthy environment in the software industry. In 2005 there was an intellectual property lawsuit filed by Compuware against IBM. As a result, IBM paid $140 million in fines to license Compuware's software and an additional $260 million to purchase Compuware's services. Examples such as this point to a critical need for computer aided, automated software plagiarism detection techniques that are capable of measuring code similarity.

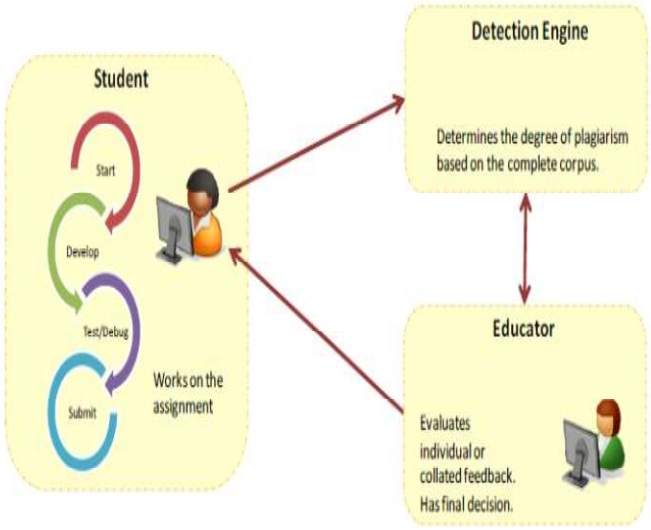


Fig.1 Workflow followed in traditional detection engines.

There is an alarming rise in plagiarism due to the widespread use of internet. Internet is an enormously huge repository of information which can be accessed easily from almost anywhere.

This has made it very difficult to control plagiarism. Since the task of manually detecting plagiarism in a large document database is very tedious and time-consuming, efforts are continuously being made to automate the process. There exist many different plagiarism detection techniques and numerous tools based on these techniques.

There are two main categories of techniques for source code plagiarism detection: **attribute-counting-based** and **structure-based** comparison. Attribute-counting-based techniques consider the number of occurrences of different attributes in a file following certain criteria and different similarity measures are used to obtain the similarity between files. Structure-based techniques derive information on program structure and obtain similarity scores based on this information. Attribute-counting algorithms are simple to implement and execute faster. Structure-based methods, on the other hand, are more reliable since they gather details of program structure for comparison of programs. However, structure-based methods are computationally expensive. Hence, the aim of

this research is to develop a new strategy which combines the advantages of both the categories.

B. Objectives of this Research

- To provide new review of the existing technologies for source code plagiarism detection.
- To combine the advantages of attribute-counting- and structure-based code plagiarism detection techniques and design a new strategy which can effectively figure out plagiarized source code files.
- To derive a fast and efficient method to detect plagiarisms in source code files written in all type of programming languages.

II. Problem Statement

Zhenzhou Tian, Qinghua Zheng, Ting Liu and Ming Fan[7] proposed that DKISB (Dynamic Key Instruction Sequence Birthmark) which generates birthmarks for both the plaintiff and defendant program, and then make the plagiarism decision according to the similarity of their birthmarks but comments should safely discard that take overhead.Wang Chunhui, Liu Zhiguo and Liu Dongsheng[8]. Proposed JPLAG Tool which is developed by java and this paper analyzes and expatiates the reasons and the methods about the code's plagiarism, and thinks there are two phases in preventing this plagiarism: one is preventing plagiarism from occur, the other is to detect cases of plagiarism when the preventative measures fail. Preventing plagiarism methods mainly include the valid course assignment design and to forbid the electronic copy. This paper describes a code's editor software which has been implemented use Java. When the preventative measures fail, this method describes an automatic tool to help instructor find the suspicious targets. These phase's aim is to cut down the plagiarism and improve the ability of the student's programming but this method cannot find plagiarism behaviors such as replacing procedure calls by the procedure body or replacing some codes which the function is same. Snehal N. Nayakoji, S. P. Sonavane.[9] proposed Subgraph Isomorphism Technique and Software Birthmark.The software birthmark results from the intrinsic characteristics of the program which could be used to determine the similarity between two programs. This Technique demonstrates the way to extract code signature and to design software birthmark along with the idea of using sub graph isomorphism to detect the source code theft of JavaScript programs but sometimes it creates more complexity in Birthmark Generator and Birthmark Comparison. It consumes high time to generate Subgraph Isomorphism. Yoon-Chan Jhi, Xinran Wang, Xiaothat[11] proposed that the technique Value Based Plagiarism Detection (VaPD) characterization is a method based on runtime values. By exploiting runtime values that can hardly be changed or replaced, the code characterization technique is resilient to various control and data obfuscation techniques. This approach directly examines executable files and does not need to access the source code of suspicious programs. It analyzed a number of real world programs and the results effective in identifying software plagiarism. According to Chanchal Kumar Roy and James R. Cordy[4] Some of the reasons for source code plagiarism may include.

1. ***Simple Reuse***: codes can be reused simply because of their logical structure
2. ***Limited Knowledge:*** when students have limited knowledge of a programming language or proper understanding of the programming task at hand.
3. ***Time Constraints****:* when projects cannot be completed within the required time frame, plagiarism might be a way out.
4. ***Coincidence:*** when two students come up with very similar or same solutions for a task, using the same programming structure but by sheer coincidence.

III. Proposed Solution

Longest Common Sub Sequence (LCS) Method to compare the semantic similarity of two codes, one from the plaintiff and the other from the suspicious code, constructed based on the LCS dynamic programming algorithm, with basic blocks as the sequence elements by trying more than one path, the code similarity scores from LCS collectively to model program semantics similarity. Note that LCS is different from the longest common substring. Because LCS allows skipping non-matching nodes, it naturally tolerates noises inserted by obfuscation techniques.

IV. Proposed LCS Algorithm

The two ways of line-by-line comparison between the source codes which is text based as well as string token based. To compare lines from File A with all lines in File B and then lines from File B with all lines from File A, this creating a two way line comparison of two source codes.

Step1: *Read file A*

Step2: *Convert the file into Text Format.*

Step 3: *Check the Data repository*

Step 4: *It's Empty*

Then

Stored file into Data repository.

Step 5: *It's not empty*

Stored file into Data repository.

For each line in file A

For each file I to N in Repository

For each line in file I in Repository

Compare line from file A with line from file I

If line from I is contained in line from A

Count string as word score

Increment number of similar lines

End if

If line from A is contained in line from I

Count string as word score

Increment number of similar lines

End if

Increment I=I+1;

End for

End for

End for

Step 6*: Display the Word Score (plagiarism of upload file).*

End.

The proposed algorithm based on recursive LCS algorithm. In this recursive based LCS algorithm at each step, compare two strings from the original uploaded file and already upload file which is in repository. The running time of the algorithm is very easy to compute. LCS only has a single pair of nested loops, which require O (m) time. This algorithm computes the length of the longest common subsequence, not the subsequence itself.

However, this algorithm can easily pull through the succession by tracing it through the files. Start at uploaded file1 string1 (0,0). Here that the value of LCS [0][0] was the maximum of all string values of the neighboring file string. So simply recomputed LCS [0][0] and note which string gave the maximum value. Then move to that string (it will be one of (1,1), (0,1) or (1,0)) and repeat this until hit the boundary of the all file in the repository. Every time to pass through a string (i,j) where S1[i]=S2[j], than the a matching pair and print S[i]. At the end, printed the longest common subsequence in O (m+n) time.

V. WORKING AND SIMULATION SCENARIO

The LCS based plagiarism detection approach leverages selected source code or documents to characterize a code fragment, it can be evaluated by the LCS algorithm that can search each of strings present in the source code and find similarity between the source codes. Here discuss about the impact of presenting plagiarism detection method and potential results compare with previous methods. It is string based similarity finding algorithm this support all types of programming languages because it's considered only the string and identifiers and find the optimal matches of the source codes. In the implantation the algorithm in the use of the .NET frames work.

The evaluated this algorithm on a set different type of programs to measure its obfuscation resiliency and scalability. These experiments conducted on small programs as well as large real-world production software. This evaluation results are compare with previous tools. In all of these experiments, the functions in the plaintiff program (or component) randomly, and test each of them to find similar code in the suspicious program. For each source codes are selected, to identify the starting blocks both in the plaintiff function and the suspicious program.

A. Getting Word Score of the Source Code

Once computed the path similarity scores (the lengths of the resulted LCS), then calculate the similarity score between the two functions. To assign a weight to each calculated LCS according to the plaintiff path length, and the function similarity score is the weighted average score. For each selected function in the plaintiff program, then compare it to a set of function in the suspicious program identified by the potential starting blocks and the similarity score of this function is the highest one among those. After calculate the similarity scores of the selected plaintiff functions, then output their weighted average score as the similarity score of the plaintiff and suspicious programs.

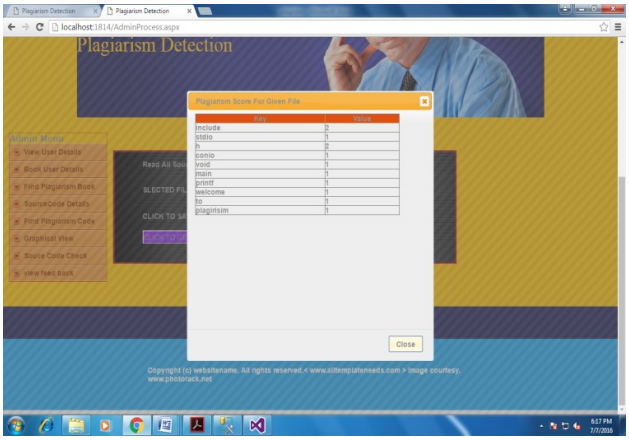
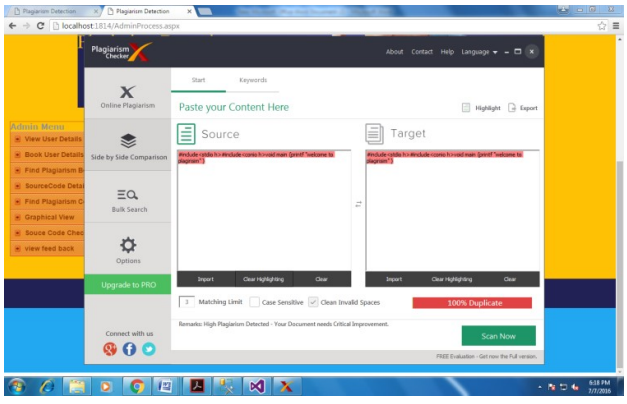


Fig.2 Getting Word Score

B. Find the Plagiarism Level

After getting word score of the source and finalize to the plagiarism level of the source code by using x checker.

Fig 3 Plagiarism Level



C. Usability between the Algorithms

By comparing these tools the most efficient is this LCS based approach.

- Many tools are sensitive to numerous small changes.
- All tools do not well for the majority of single refactoring, and many tools score rather badly.
- A striking result of the top-10 comparison is that the top-10's for GPlag, JPlag, Marble and MOSS are fairly similar, LCS is quite different to other algorithms.

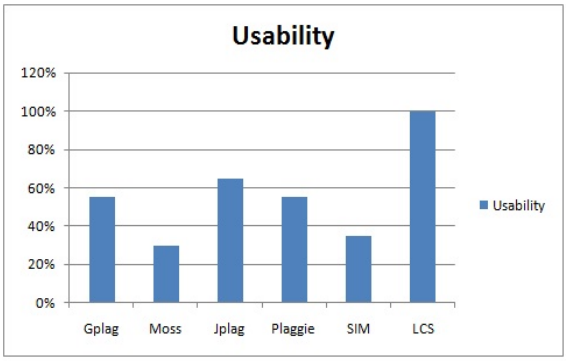


Fig. 4 Usability between the Algorithms.

The comparison of the approaches shown that still now their no tools that can detect or to prove that the source code has been plagiarize 100%, because each method and tool has advantages and limitation, according to the features and performance. This research is focused the limitations of approaches and summarize the performance of existing tools. Finally this proposed method gives good solutions to find the source code plagiarism.

VI. CONCLUSION

Plagiarism is a complex topic, and the edge between useful activities that can help a student's educational growth (such as association with other students or "scrap writing" from trusted sources as a stepping stone to developing independent skills) and activities that will be punished as "dishonest" can cause misinterpretation. There is certainly room for more research into understanding motivations for plagiarism and how best to support students from different backgrounds. An identifying the source code is difficult task for software companies and the students assignments. The findings reported here identify a number of issues related to source code plagiarism that students find confusing and will therefore help teachers

provide effective support for students to understand (and avoid) this type of plagiarism. This approach provides LCS based algorithm give new solutions of the software source code plagiarism that produce good results compare to all other technique it support all type of languages and performance speedup also comparatively good.

VII. FUTURE WORK

In future work, in this research will focused about comment line of the source code and identified the keywords of source code will find the exact source code and flow of the source code and find the exact plagiarism of the source code.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their valuable comments, which greatly improved the readability of this paper. S.Mahalakshmi would also like to thank Ms. S.Kavitha ,HOD and Assistant Professor, Department of Computer Science, who guided for my work, and also express my whole hearted thanks to my parents and friends for their encouragements to bring this work to a successful completion.

REFERENCES

[1]. Chao Liu, Chen Chen, Jiawei Han, "*GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis"* U.S. National Science Foundation NSF ITR-03-25603 and IIS-03-08215/05-13678.2006.

[2]. Xinran Wang, Yoon-Chan Jhi, Peng Liu and Sencun Zhu.," *Behavior Based Software Theft Detection*", November 9–13, 2009, Chicago, Illinois, USA. Copyright 2009 ACM 978-1-60558-352-5/09/11.

[3]. Dr. Warren Toomey," *Code Similarity Detection in Multiple Large Source Trees using Token Hashes"*, PAN-09 3rd Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse and 1st International Competition on Plagiarism Detection, 2010.

[4]. Chanchal Kumar Roy and James R. Cordy *"A Survey on Software Clone Detection Research"* ,2007.

[5]. Asako Ohno and Hajime Murao," A Two –Step in Class Source Code Plagiarism Detection Method Utilizing Improved CM Algorithm and SIM" *International Journal of Innovative Computing, Information and Control* ICIC International©2011 ISSN 1349-4198 Volume 7, Number 8, August 2011.

[6]. Yoon – Chan jhi, Xinran Wang, Xiaoqi Jia and Peng liu," *Value-based program characterization and its application to software plagiarism detection***",** ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA.

[7]. Zhenzhou Tian, Qinghua Zheng, Ting Liu and Ming Fan," *DKISB: Dynamic Key Instruction Sequence Birthmark for Software Plagiarism Detection*", National Science Foundation of China the Ministry of Education Innovation Research Team (IRT13035), Key Projects in the National Science and Technology Pillar Program of China 2013.

[8]. Wang Chunhui, Liu Zhiguo and Liu Dongsheng," *Preventing and Detecting Plagiarism in Programming Course*", International Journal of Security and Its Applications Vol.7, No.5 (2013), pp.269-278 Http://dx.doi.org/10.14257/ijsia.2013.7.5.25.

[9]. Snehal N. Nayakoji, S. P. Sonavane.," *Code Birthmarks and Graph Isomorphism for Theft Detection*", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.1, January- 2014, pg. 470-476.

[10]. Lannan Luo Jiang Ming Dinghao Wu Peng Liu and Sencun Zhu," *Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software Plagiarism Detection*", FSE'14, November 16–22, 2014, Hong Kong, China Copyright 2014 ACM 978-1-4503-3056-5/14/11.

[11]. Yoon-Chan Jhi, Xinran Wang, Xiaoqi Jia, Sencun Zhu, Peng Liu and Dinghao Wu," Program Characterization Using Runtime Values and Its Application to Software Plagiarism Detection", 10.1109/TSE.2015.2418777, IEEE Transactions on Software Engineering.2015.

[12]. Y.-C. Jhi, X. Wang, X. Jia, S. Zhu, P. Liu, and D. Wu." *Value-based program characterization and its application to software plagiarism detection*". In *33rd International Conference on Software Engineering(ICSE 2011), the SEIP track*, 2011.

[13]. L. Luo, J. Ming, D. Wu, P. Liu, and S. Zhu, *"Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection,"* in *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, November 2014.