Abstract— The reused code in software field will degrade the efficiency in development and in execution phases. To identifying similar code fragments is more challenging in case of code theft. In case of code theft many automated obfuscation code transformations are used to hide the stolen code. To handle such obfuscated code with a strong evidence of a suspicious program some methods are used based on the runtime values of the program during its execution. The runtime values are core values dynamic characterization of program executable, this technique is resilient in data and control obfuscation techniques. The dynamic taint analyzer will extract and refine the runtime executables of the core values. The value based plagiarism detection method (VaPD) is used to measure the two code fragments by using longest common subsequence method. The similarity measuring algorithms are used to check codes which belong to same ancestry.

Index Terms— Obfuscation, Software plagiarism detection, Value dependency graph, Dynamic code identification.

II INTRODUCTION

To identify similar code fragments among two dissimilar programs or in the identical program is very significant in some applications. The duplicate codes found in the same program may corrupt effectiveness in both development phase and in execution phase they can confuse programmers and lead to potential errors degrade cache performance. The code detection techniques such as clone detection are capable of discovering and refactoring the similar code fragments to improve the program efficiency [1]. The clone detects and removes the clone’s promises decrease software maintenance cost of probably the same magnitude. It shows simple and realistic methods for detecting accurate and closes to clone over random code fragments in program source code with the help of abstract syntax trees [2].

If those programs are independently developed by two different programmers, and duplicated code can be suggested to be code theft or software plagiarism. In code theft cases, software plagiarizers can use different code transformation techniques to hide stolen code from detection. To design a code obfuscator uses some tools to convert the program into equivalent which is critical to analyze the code and in reverse engineering [3]. The code classification and identification techniques should help to detect or find semantically equivalent code in which two code fragments belong to the same lineage.

Previous works are classified in two desired requirements: at first, Resiliency in automated semantics-preserving obfuscation converting tools is easy to transform most of the syntactic features like strings [3], second the Ability to do work on binary executable of the suspected programs, in some applications like code theft cases, the source code of suspect software products cannot be argued unless until a strong evidences have been collected.

The existing works are widely classified into four categories they are static and dynamic based plagiarism detection, obfuscation techniques and smart phone app repackaging detection. It introduce a approach to dynamic characterization of executable programs. After the examining different runtime values of some executable programs, the runtime values of a program are tough to be eliminated by semantics-preserving transformation like optimization techniques, obfuscation techniques, etc. The runtime values are called core values. Here the core values are the values during runtime execution of a program, so it is dynamic program execution. It shows how to extract the values helpful for the core-values and how to apply this runtime core values to find the solutions in software plagiarism detection.

It implements a value extractor with a dynamic taint analyzer and value refinement with the generic processor emulator, it implements a value-based software plagiarism detection method (VaPD) that uses the similarity measuring algorithm based on sequences of the core values and value dependence graphs constructed from the extracted core values.

The sandmark [4] is obfuscated tool used to obfuscate the code and it can also helpful to find obfuscated source code. The KlassMaster [5], Thicket C obfuscator [6], and Loco/Diablo [7].

1. The new approach called code characterization method based on runtime values.
2. The use of runtime values that can hard to replaced, this code characterization technique is strong to all data and control obfuscation techniques.
3. The VaPD method shows strong evidence against suspect program and plaintiff programs by a set of real world programs.

2 Related Works

The code obfuscators use obfuscation techniques to hide the stolen codes, the code obfuscations are semantic preserving transformations. This technique is not simple like adding and removing a variable to the code. The obfuscation code transformations are based on potency, resilience and cost. The obfuscated techniques are not easy to read. Christian collberg provided taxonomy of some automated obfuscating techniques which are based on reverse engineering. Most obfuscated code transformations are lexical structure, names of variables, code formatting. Collberg mainly concentrates on data structures and flow of control of application. The code obfuscator transforms the code by manipulating the control or data flow of the program by adding new abstractions which is not effective on main output of the program [3].

Collberg also provided the Sandmark is one of the code obfuscation tools which are used for software protection. The
sandmark is used to evaluate the effectiveness and performance of the software based methods. In sandmark code obfuscated algorithms are used to measure the performance and effectiveness by combing and adding algorithms. It provides protection from the malicious attackers. Watermarking is a technique used to discourage illegally redistributing the code by embedding or changing the message into program. The fingerprint technique is also used in sandmark [4]. Some other java obfuscated tools are loco/Diabo [5], klass master [6], and Thicket [7] are used to obfuscate the programs.

The existing static analysis based plagiarism detection methods techniques are related to the code detection and malicious intention. A technique called program dependence graph (PDG) by Chao Liu and a tool GPLAG to find plagiarism by mining the programs. Based on control and data flow of the program PDG is constructed. The GPLAG is more effective and efficient to find the plagiarism within a few seconds. The core parts of the program are detected by the tool instead of computing whole program so it called as core part plagiarism [8]. String based: In plagiarism detection each statement in a program are defined as the string. Sequences of string are called as programs. The programs are compared based on strings, if strings are matched between two programs then can say two sequences of strings are identical. Main drawback is strings based are weak at finding or identifying renaming [9], [10].AST based: Abstract syntax tree (AST) is constructed by paring the program with exact values and variable names discarded. From two abstract syntax trees of two programs are compared for finding duplicated subtrees and code corresponding to duplicate subtrees and the duplicate subtrees are plagiarism. AST are strong on renaming and weak at data flows and statements reordering [11], [12]. Token-based: In token based the program symbols like keywords, identifiers are tokenized. Tokenized programs sequence is checked for plagiarism in between two programs. This approach is strong on renaming, weak at statement reordering and code insertion [13].

The dynamic analysis plagiarism detection method is a run time execution of the program. One of the existing approaches in dynamic analysis is birthmark. Ginger Myles proposed a whole program path (WPP) which is a dynamic birthmark. Birthmark is used to find similar characters, watermarking and fingerprinting are the techniques used in birthmark. The dynamic control flow of the programs represents the WPP. The WPP is strong at obfuscation techniques but weak at loop unwinding and semantic preserving transformation. Haruaki Tamada proposed two types of birthmark sequence API function call (EXESEQ) and Frequency API function call (EXEFREQ) which is recorded during the execution of the programs [14]. Code similarity measurement methods like binary similarity detection, clone detection, and software plagiarism detection helps to find the duplicated codes.

2.1 Longest common subsequent

Lannan Luo proposed a binary-oriented, longest common subsequent of semantically equivalent basic block. Based on fuzzy matching it combines exact program semantics with longest common subsequence. To design program semantics at three different levels: path, whole program and basic block. To design the basic block the symbolic execution and set of formulas are used based on input and output relation. The two basic blocks are compared on output variables or registers and theorem prover. The plaintiff blocks are used to calculate the overall percentage of the blocks. In path level the longest common subsequent algorithms are used to compare two paths of both suspected path and plaintiff path. CoP is based on a new method which is used to evaluate the levels, CoP will measure obfuscation resiliency, accuracy, and scalability [15].

3 Design

To identify the similar code for an obfuscated whole-program is checked for plagiarism, where the plagiarizer copies the plaintiff source code and modifies the interface. In the core part plagiarism the plagiarizer’s copies only core part or module of the source code. The VaPD is used to measure the both core part and whole part of the source code. The plaintiff program will provide the information about the source code. The suspect program is modified with obfuscated tools. In value based plagiarism detection uses core part of the programs. The core values are taken for both plaintiff and suspect programs. The core values are the output values for desired input values, in core values the number of inputs are given for the same program.

3.1 Value Sequence Extraction

From the execution of the core values the value sequence extracted. The outputs in core values are added in value sequence. After adding to value sequence the output values are updated in value updating instruction for very execution of the source code. The state update instruction is used to update for every single execution of the source code. For value extraction the program is executed by adding start extraction ( ) and stop extraction ( ).

3.2 value sequence refinement

In value sequence refinement the values are analyzed deeper in core values and capability of fragmenting the large programs or source code. In value sequence refinements are classified in Sequential refinement, optimization-based refinement and address removal.

3.4 similarity metrics

The similarity metrics is based on the longest common subsequence (LCS). Given \( v_p \), is a value refined sequence of a plaintiff program and \( v_s \) is the sequence values of a suspect program, similarity value for the suspect program on the plaintiff program is defined as:

\[
\text{Sim} \left( v_p, v_s \right) = \frac{| \text{LCS} \left( v_p, v_s \right) |}{| v_p |}
\]

Fig 1: value based plagiarism detection process

4 Conclusion

The automated obfuscation techniques are used to manipulate the program. The VaPD is more effective in finding the plagiarism in source core. The VaPD uses longest common subsequent algorithms to identify similarity. This technique is strong at data and control flow abstraction of obfuscated techniques. The proposed system is on directly identifies executable files without using suspicious programs or source code.
References


