Comparative Analysis of Technical Methods for Detecting Software Thefts

Hyun-Il Lim*
Department of Computer Engineering, Kyungnam University, Changwon, Gyeongsangnam-do, Republic of Korea.

* Corresponding author. Tel.: +82-55-249-2650; email: hilim@kyungnam.ac.kr
Manuscript submitted January 10, 2015; accepted March 8, 2015.
doi: 10.17706/jcp.11.1.26-32

Abstract: Software plays an important role in current computing environments, and it is protected as intellectual property of its author. However, the cases of software thefts are increasing every year. To deal with the problem, there have been researches on detecting software thefts. In this paper, we introduce technical methods for detecting software thefts, and compare characteristics of the methods according to several performance evaluation criteria.

Key words: Plagiarism detection, software birthmark, software theft detection, software watermark.

1. Introduction

In current computing environments, software is used in various applications. To improve the current computing environments, it is essential to develop useful software. Software is protected as intellectual property of its developers. For example, software is protected by software license or copyright. However, it is required much effort to develop new software. So, the cases of software thefts are increasing every year. Such cases of software thefts are major obstacle to development of software industry.

To deal with software theft problems, there have been several researches to detect software thefts. In this paper, we introduce technical methods for detecting software thefts, such as source code plagiarism detection method, software watermark, and software birthmark. Then, we compare the methods according to performance characteristics, such as limitation, target, credibility, resilience, and overhead. In the comparison results, the methods have pros and cons according to evaluation criteria. So, the methods will be more effective in detecting software theft if the methods are applied in proper environments according to the evaluation criteria.

2. Source Code Plagiarism Detection

The source code of software can represent all the information about software, because the software is originated from the source code. So, when the source code of software is available, it is preferable to analyze the source code to detect software thefts. Source code plagiarism detection methods try to find plagiarized program regions through comparisons of specific structures analyzed from source code of software, including texts in the software, structural syntax or programming styles of source code.

Such source code plagiarism detection methods are generally performed in the following procedures:

1) Source code of software is parsed and converted into lists of token strings.

2) The lists of token strings are compared in their specific ways to identify the suspicious contents of software.

In considering that most of commercial programs are only distributed in the form of binary executables, these approaches are limited to apply in environments where their source codes are available.

Among source plagiarism detection methods, Sim [1] is one of such methods to calculate similarity between computer programs written in C language. The approach compares programs as the following procedures:

1) Sim tokenizes the source code of programs into sequences of tokens.
2) From sequences of tokens, it constructs parse trees of the programs.
3) The parse trees are compared as strings by aligning their sequences to calculate the similarity.
4) In aligning two strings, their matching score is calculated by counting the scores of matches, mismatches, or gaps between tokens of the two strings.

From the matching score of alignment, the similarity between two programs P and Q is calculated as follows:

$$sim(P,Q) = \frac{2 \times score(P,Q)}{score(P,P) + score(Q,Q)},$$

where $score(P,Q)$ denotes the matching score calculated by alignment between two programs P and Q.

3. Software Watermark

3.1. The Concepts of Software Watermark

Software watermark is an approach to detecting the instances of software thefts by confirming previously embedded identifiers. So, this approach can be applied only to programs that are watermarked before releasing the software. Fingerprint is a way of embedding a different identifier for each program, so it can trace the source of illegal redistribution. Given a program P, a watermark w, and a key k, software watermark is performed by the following two functions [4]:

$$\begin{cases} embed(P,w,k) \rightarrow P', \\ recognize(P',k) \rightarrow w. \end{cases}$$

The embed() function embeds a watermark w with a key k into an original program P, and then generates a new program P' that is a watermark-embedded version of P.

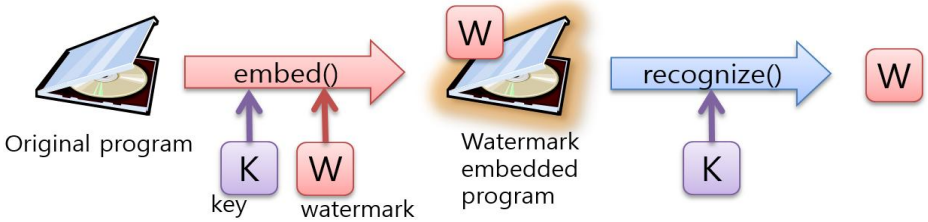

Fig. 1. The procedure for applying software watermark for detecting software thefts.

When it is required to confirm previously embedded watermark, the watermark can be identified from the program P'. Fig. 1 shows the procedure for applying software watermark for detecting software thefts. The recognize() function extracts previously embedded watermark w from a watermarked program P' with its key k. So, a recognition of previously embedded watermark w from a suspicious programs can be a strong evidence of software thefts.

Static watermarks make use of software's features available at static time. Static watermarks are stored in software itself and recognized without executing the software. They can be encoded in several ways,

such as basic block reordering or register reallocation. The advantage of this approach is that it is simple to embed a watermark without affecting the semantics of software. However, static watermarks can be easily modified or broken by software modifications or semantics preserving transformations.

On the other hand, dynamic watermarks are stored in execution state of a program, rather than in the program code itself. So, the recognition of watermark requires executing the program. Dynamic watermarks are embedded so that the identifier can be extracted with a predefined input sequence at specific run-time states. Because it is difficult to modify the dynamic states of a program, such as dynamic traces or data structures, without affecting the semantics of the program, this approach is more robust than static one.

3.2. Static Software Watermark

Data watermark [5] is the most simple and naive watermark. It can be embedded in any part of software, such as data section, text section, or images used in the software. For example, copyright notice can be embedded in a program as follows:

```
String watermark = "(c) Copyright 2008 This program is ... ";
String fingerprint = "Customer ID is ... ";
```

Because this watermark is easily caught by users, it is trivial to remove or modify the identifier. Moreover, modifications, such as obfuscation or optimization, can easily break such static data, leading loss of its watermark information.

Code watermark is embedded by using redundant information that is carried by object code of a program. For example, if there is no data or control dependency between two adjacent statements S_1 and S_2, the statements can be flipped in either order. A watermark can also be encoded in such a way that the lexicographic order of statements can represent the encoded bits of the watermark.

Code watermark is embedded using behind information of a program. However, encoded information can be easily affected by modifications because watermark is not tightly encoded in a program, that is, the modification of watermark information is not likely to affect the semantics of a program.

3.3. Dynamic Software Watermark

Easter egg watermark [5] is the simplest approach in dynamic watermarking. The watermark is an embedded code of action activated by a highly unusual input sequence. The code may display a copyright message or an unexpected image on screen, or activate some secret stage in a game program. Because such action can be immediately noticed by users, the watermark can be easily removed or disabled by using some debugging techniques.

Dynamic data structure watermark [6] is a method to embed a watermark within the run-time state of a program. For a particular input sequence, the watermark is encoded in predefined data structure, such as global variable, heap, stack, array, or linked list. After executing the program with the specified input, the recognition of the watermark is accomplished by examining the current values of the data structure which embeds the watermark.

Data structure watermark has no output even though the embedded watermark is constructed. So, it is difficult to perceive the existence of watermark. Thus, this approach is stealthier than Easter egg watermark. However, data structure watermark is also susceptible to attacks by obfuscating transformations. For example, obfuscation may split one variable into several variables or merge several variables into one.

4. Software Birthmark

4.1. The Concepts of Software Birthmark

A software birthmark refers to program's inherent characteristics that can be used to identify the program. If two programs have the same or similar birthmarks, one is likely to be an illicit copy of the other. For example, comparing the strings in two programs can be a naive birthmarking technique. Software birthmark consists of two functions as follows:

$$\begin{cases} extract(P) \rightarrow bm_P, \\ compare(bm_P, bm_Q) \rightarrow [0,1] \end{cases}$$

The extract() function extracts the inherent characteristics bm_P from a program P through static or dynamic software analysis. Then, the compare() function compares the extracted characteristics of programs to calculate similarities between the programs.

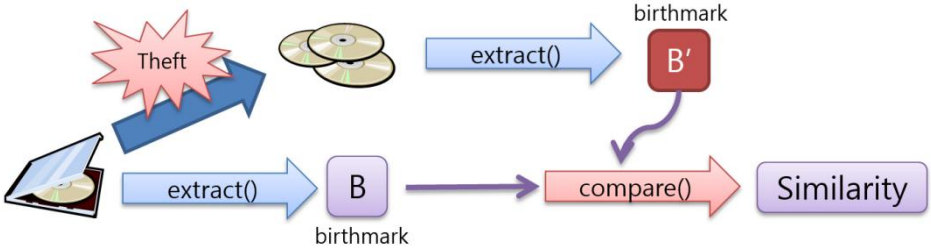

Fig. 2. The procedure for applying software birthmark for detecting software thefts.

Fig. 2 shows the procedure for applying software birthmark for detecting software thefts. From the comparison results, the similarity values of extracted birthmarks is an evidence of identifying software thefts.

Static birthmark is extracted from program itself without executing the program, while dynamic birthmark is extracted from observable program behaviors during program execution.

4.2. Static Software Birthmark

Tamada birthmark [7] is the first practical application of static software birthmark for Java programs. The approach consists of four individual birthmarks as follows:

- Constant Values in Field Variables: Classes in Java programs have field variables to store the values of instantiated objects. The constant values in field variables are used as distinguishing characteristics of Java programs.
- Sequence of Method Calls: Java class library provides lots of classes and methods to support primitive functions. Method calling sequences may be different according to each program.
- Inheritance Structure: Every Java program has class hierarchy beginning from java.lang.Object. The class hierarchy of a program is designed differently according to class design strategy and development environments.
- Used Classes: Java programs are modularized in terms of classes, and each program may import external classes to use necessary functions. The used classes may be also different according to individual programs.

These four birthmarks can be used individually, but they are more reliable when combined. The birthmarks are based on the underlying structures of a Java program, so the birthmarks can only be extracted from an entire class. Therefore, it is inappropriate to use in cases when a class is embedded in a larger program.

The k-gram based birthmark [3] is a static birthmark based on instruction sequences. A k-gram denotes a

sequence of k consecutive opcodes in a program. The set of all k-grams in a program is used as a birthmark of the program. The similarity between two programs P and Q is calculated by comparing k-grams in two birthmarks as follows:

$$\text{similarity(P, Q)} = \frac{|kgrams(P) \cap kgrams(Q)|}{|kgrams(P)|} \times 100$$

where $kgrams(P)$ denotes the set of k-grams in programs P. The similarity assumes that P is an original program, while Q is a target program. So, the similarity measures how many k-grams of an original program is contained in a suspicious program. The k-gram based birthmark is highly credible because the sequence of opcode is significantly different with each other. However, the birthmark may be susceptible to program transformation, such as optimization or obfuscation, because such transformation can change the sequence or the order of opcodes.

4.3. Dynamic Software Birthmark

Dynamic API Birthmark [2] is a dynamic birthmark based on API call sequences in Windows applications. A sequence and frequencies of API function calls during program execution are used as birthmarks. The similarity is calculated by comparing birthmarks through string matching tools, such as diff and CCFinder.

The whole program path birthmark [3] is an approach to using the notion of whole program paths (WPP). A WPP represents a graph representation obtained by compressing a dynamic trace of a program. At first, the control flow graph of a program is constructed, and a dynamic trace of the program is extracted during program execution. The dynamic trace is then compressed into a context-free grammar using SEQUITUR algorithm. Finally, the grammar is represented as a directed acyclic graph, which is a WPP. In order to use the WPP as a software birthmark, all terminal nodes and their corresponding edges are removed from the WPP because terminal nodes may be easily affected by run-time environments or program modifications. The similarity between two WPP birthmarks is obtained by using graph distance based on the maximum common subgraph as follows:

$$similarity(P, Q) = \frac{|mcs(WPP(P), WPP(Q))|}{|WPP(P)|}$$

where $WPP(P)$ denotes the WPP birthmark of program P, and $mcs(G, H)$ denotes the maximal common subgraph between graphs G and H. $|G|$ denotes the sum of numbers of edges and vertices in a graph G. Because graph comparison is very expensive operation, this birthmark is not appropriate for large programs.

5. Comparative Analysis

In this section, we compare the characteristics of methods for detecting software thefts. We compare the characteristics according to several performance criteria, such as limitation, target, credibility, resilience, and overhead. Table 1 shows the summarized comparison results of comparison.

Table 1. Characteristic Comparison Results of Methods for Detecting Software Thefts

	Source code plagiarism detection	Software watermark	Software birthmark
Limitation	Availability of source code	Previous embedding of watermark	No
Target	Source code	Watermark embedded software	Source code or binary code
Credibility	High	Very high	Medium
Resilience	Medium	High	High
Overhead	Low	High	Medium

In comparing limitation of methods, source code plagiarism detection method has limitation that source

code of software should be available, because the method only can analyze source code of software. Software watermark also has limitation that software watermark should be embedded before release to recognize watermark from suspicious software. So, the target of software watermark is limited only to previously watermark embedded software. On the other hand, software birthmark has no limitation to apply in detecting software thefts. So, software birthmark can be applied to source code or binary programs in detecting software thefts without limitation.

In comparing credibility of methods, source code plagiarism detection methods has high credibility, because source code of software has all the information about the software. So, the similarity of source code can be an obvious evidence of software thefts. Software watermark has very high credibility, because the recognition of previously embedded watermark is a confident evidence of software theft. On the other hand, software birthmark has medium credibility, because birthmarks extracted from suspicious programs may be accidently similar although programs are different programs.

In comparing resilience of methods, source code plagiarism detection method has medium resilience, because source code can be easily understood and modified to hide the fact of software theft. On the other hand, software watermark and software birthmark have high resilience, because the two approaches analyze binary programs, which are hard to be understood or modified.

In comparing overhead of methods, source code plagiarism detection method has low overhead, because the method do not have any requirement in detecting software thefts except for source code. Source code is the easiest and simplest representation of software. So, the method has low overhead in detecting software theft. Software watermark has high overhead, because the method requires to embed watermark before releasing the software. In addition, the method should be able to analyze suspicious programs to recognize previously embedded software watermark. So, the method has overhead in embedding and recognizing software watermark. Software birthmark do not require any operation before releasing software. However, the method should be able to analyze binary programs to detect software thefts. Binary program is hard to analyze as compared to source code, so software birthmark has medium overhead.

6. Conclusion

In current computing environments, software plays an important role, and it is protected by software license and copyright. However, the cases of software thefts are increasing every year. Such cases of software thefts are major obstacle to development of software industry. To deal with software theft problems, there have been several researches to detect software thefts.

In this paper, we introduced the concepts and applications of technical methods for detecting software thefts, such as source code plagiarism detection method, software watermark, and software birthmark. To apply the methods in practical applications, we compare the characteristics of the methods according to limitation, target, credibility, resilience, and overhead. In the comparison results, the methods have pros and cons according to each evaluation criteria. So, the methods will be more effective in detecting software theft if the methods are applied in proper environments according to the evaluation criteria.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0024658).

References

[1] Gitchell, D., & Tran, N. (1999). Sim: A utility for detecting similarity in computer programs. *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (pp. 266–270).

[2] Haruaki, T., Akito, M., *et al.* (October 2004). Dynamic software birthmarks to detect the theft of windows applications. *Proceeding of International Symposium on Future Software Technology*.

[3] Ginger, M. M. (2006). Software theft detection through program identification. PhD thesis, Department of Computer Science, The University of Arizona.

[4] Ginger, M., & Christian, C. (2006). Software watermarking via opaquepredicates: Implementation, analysis, and attacks. *Electronic Commerce Research, 6(2)*, 155–171.

[5] Christian, C., & Clark, T. (January 1999). Software watermarking: Models and dynamic embeddings. *Principles of Programming Languages*. Antonio, TX.

[6] Christian, C., & Clark, T. (August 1998). On the limits of software watermarking. Technical Report 164, Department of Computer Science, University of Auckland.

[7] Tamada, H., Nakamura, M., Monden, A., & Matsumoto, K. (2005). Java birthmark–detecting the software theft. *IEICE Transactions on Information and Systems*, 2148–2158.

Hyun-Il Lim received his B.S., M.S. and Ph.D. degrees in computer science from KAIST, Republic of Korea, in 1995, 1997, 2009, respectively. He is currently an assistant professor in the Department of Computer Science and Engineering, Kyungnam University, Republic of Korea. His current research interests include software security, software protection, watermarking, and program analysis.