A Kind of Static Software Birthmark Based on Control Flow

Yalin Zhao¹,³,⁴ and Juchao Lei²

¹,²Department of Computer Science and Engineering, XI’AN Technological University, Located at No. 2 Xuefu Middle Road, Weiyang District, Xi’an City, Shaanxi Province.
³This work was supported by The Education Reform Project of Xi’an Technological University (No.18jgz01).
⁴Email: zhaoyalin6987@qq.com

Abstract. In traditional static k-gram birthmark algorithm, the result of plagiarism detection is inaccurate. It is a new method for plagiarism detection that using the software birthmark based on program control flow in this paper. First, it is used to the result of static that analysis of the Java program as meta information, analyze meta information to get byte stream instruction in method. Then, a byte instruction stream in the method to get the control flow structure of the program has been analyzed, and replaced the external references with external control flow structures, using this control flow structure as a software birthmark. Finally, it is calculated that the similarity between two birthmarks by VF2 algorithm, thus determining whether there is plagiarism between the two programs. It is shows that birthmark which prese...
is obtained by a birthmark comparison model based on VF2 [7] algorithm, and then use the similarity to determine two softwares are plagiarized.

2. Traditional Software Birthmark Algorithm

The concept of software birthmarks was proposed by Grover [8]. A complete software birthmark plagiarism detection model consists mainly of two functions:

\[
\begin{aligned}
B_p &= \text{extract}(P) \\
\text{similarity} &= \text{compare}(B_p, B_Q), \text{similarity} \in [0, 1]
\end{aligned}
\]  

(1)

Where \( B_p \) represents the birthmark of program \( P \), \( \text{extract} \) represents the feature extraction function of the software birthmark, and \( \text{similarity} \) is in the interval \([0, 1]\). Formula (1) shows that software birthmark is a product of one or more aspects of software feature extracted by \( \text{extract} \) algorithm.

The criterion for software plagiarism is as shown in formula (2): set the threshold as \( \alpha \), \( P \) and \( Q \) are two programs, \( B_p, B_Q \) are the birthmarks of the two programs.

\[
\text{comp}(B_p, B_Q) = \begin{cases}
\geq 1 - \alpha, P, Q \text{ are plagiarism} \\
\leq \alpha, P, Q \text{ are non-plagiarism} \\
r \text{other, uncertain}
\end{cases}
\]

(2)

Software birthmarks will be judged from two aspects: credibility and resilience.

**Definition 1.** Credibility: Let \( P \) and \( Q \) be two programs of non-plagiarism, \( \text{extract} \) is the extraction function of software birthmark, \( B_p \) and \( B_Q \) are software birthmarks of two software respectively, \( B_p = \text{extract}(P), B_Q = \text{extract}(Q) \), if \( B_p \neq B_Q \), then \( B_p \) and \( B_Q \) has credibility.

**Definition 2.** Resilience: Program \( P' \) is the plagiarism program of program \( P \), \( \text{extract} \) is the extraction function of software birthmark, \( B_p \) and \( B_{P'} \) are software birthmarks of two programs respectively, \( B_p = \text{extract}(P), B_{P'} = \text{extract}(P') \), if \( B_p = B_{P'} \), \( B_p \) has resilience.

Software Birthmark Evaluation System can also evaluated by Precision, Recall, and F-Measure.

In the sample, let \( T \) be plagiarized test results was right, \( F \) be plagiarized test results was wrong. \( T_p \) indicates that the pirated program pair is detected as a pirated program pair, \( T_n \) detects the non-pirated program pair as a non-pirated program pair, \( F_p \) indicates that the non-pirated program pair is detected as a pirated program pair, and \( F_n \) indicates that the pirated version is detected as a non-pirated program pair.

**Precision:** The correct ratio of detection result was plagiarism

\[
\text{Precision} = \frac{T_p}{T_p + F_p}
\]

**Recall:** The ratio of pirated programs to detected

\[
\text{Recall} = \frac{T_p}{T_p + F_p}
\]

**F-Measure:**

\[
F_{\text{measure}} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\]

It is inaccurate to evaluate the quality of the plagiarism detection model by the correct rate of the test results. Only when the plagiarized model both have high Precision and recall rate, the detection result of the plagiarism detection model is trustworthy.

3. Improved Software Birthmark Algorithm

The k-gram software birthmark was proposed by Myles et al. [9]. The main idea is to extract k-gram fragments from the program instructions and use all the extracted fragments as a set to represent the software birthmark. When extracting k-gram fragments, they extracted in the form of sliding windows,
that is, divided according to their physical neighboring addresses. Only physical relationships are considered regardless of their logical relationship or internal control flow conditions.

This paper proposes a software birthmark based on program control flow, use this birthmark detected plagiarism has the following three steps:

First, the Java source program $P$ and the Java suspicious program $Q$ obtain their corresponding control flow feature by analyzing the corresponding class files, then generate corresponding birthmarks by corresponding control flow feature; Second, constructed a control flow feature static birthmark comparison model based on VF2 algorithm, then calculating the similarity between two birthmarks as the similarity between source and suspicious program; Finally is set threshold $\alpha$, use formula (2) to detect whether two programs are plagiarized.

The flow chart is shown in figure 1.

![Figure 1 Plagiarism detection process.](image-url)

3.1. Static Birthmark Based on Control Flow Structures

As a compiled intermediate file, the Class file provides all the necessary symbol information for JVM in runtime. The most important information in a Class file is the Code Attribute in the Method array. This attribute table records the array of instructions generated by the source program after compile, JVM executes a method by traversing this instruction array at runtime. Since the bytecode instruction system of the JVM is a stack-based variable length instruction system, this paper uses the structure of the map to store the control flow information of the program;

The steps for getting control flow birthmark of method $M$ in this paper are as follows:

1. Mark jar file as $J$, iteration depth as $R_{max}$, project as $P$ and set $R_{temp} = 1$.

2. According to the given method signature $M_{sign}$ and class file $C$, obtain instruction array of method $M$ in project $P$, recorded as $code$, first element of $code$ must be an instruction.

3. Traversing $code$, according to the virtual machine bytecode instruction specification (hereinafter referred to as the instruction specification), the information of $code$ will be translated into $instruction$.

4. Traversing $instruction$, initializing the predecessor and successors of all elements not null, in particular, elements with no predecessor are recorded as $start$, with no successors are recorded as the terminating node $end$. Record $instruction$ after the initialization of the predecessor is $feature_{struct}$.

5. Traversing $instruction$, $i$ is the instruction being traversed, Check whether $i$ is an invoke instruction, if $i$ is an invoke instruction and $R_{temp} \leq R_{max}$, check whether the method $M'$ called by $i$ exists in $P$, if $M'$ exists in $P$, then generate a signature $M'_{sign}$ for $M'$, and mark the class file containing $M'$ as $C'$, $R_{temp} = R_{temp} + 1$. Repeat Step (2) to generate a control flow feature $instruction'$ of $M'$ with $M'_{sign}$ and $C'$ as input, then add the first element of $instruction'$ to the successor of $i$. After the traversal of $instruction$ is completed, $R_{temp} = R_{temp} - 1$.

The birthmark extracted using birthmark extraction algorithm proposed in this paper for the sample program shown in figure 2 is shown in figure 3.
3.2. Reduction of Control Flow Birthmark
This paper proposes a method of reducing control flow birthmarks:
(1) Initializes the state of all elements of instruction in birthmark $B$ to MEANINGLESS.
(2) Iteratively traversing the instruction in birthmark $B$, Set the state of all elements whose precursors or successor is not only one to MEANINGFUL. In particular, if elements $i$ has more than one successor, then set the state of each successor of element $i$ to MEANINGFUL.
(3) Iteratively traversing the instruction in birthmark $B$, mark the element being traversed as $i$, if the state of $i$ is MEANINGLESS, then add the only successor $suc$ of $i$ to the successor set of $i$’s predecessor, $i$ is deleted from the successor set of $i$’s predecessor, then delete $i$.

3.3. Using Control Flow Birthmark for Plagiarism Detection
This paper proposes a method for plagiarism detection using control flow birthmark:
(1) mark source project as $P_{src}$ and suspicious project as $P_{d}$; source class as $C_{src}$ and source method signature as $M_{src}^{sign}$; suspicious class as $C_{d}$ and suspicious method signature as $M_{d}^{sign}$; iteration depth as $R_{max}$; detection threshold as $\alpha$; net input amplification as $\theta$ and net input offset as $bias$ .
(2) Generate source birthmark $B_{src}$ and suspicious birthmark $B_{d}$ according to the method described in section 2.1 and the parameters in step 1.
(3) According to the method described in section 2.2, \( B_{src} \) and \( B_d \) generate corresponding reduce birthmarks \( B_{src}^{\text{reduce}} \) and \( B_d^{\text{reduce}} \).

(4) Select the one with fewer nodes in \( B_{src}^{\text{reduce}} \) and \( B_d^{\text{reduce}} \) as \( G_{\text{query}} \). If the number of nodes in the two graphs is the same, select \( B_{src}^{\text{reduce}} \) as \( G_{\text{query}} \), and use VF2 algorithm to match the maximum isomorphic subgraph of \( B_{src}^{\text{reduce}} \) and \( B_d^{\text{reduce}} \), denoted as \( G_{\text{common}} \).

(5) Calculate the similarity \( \text{sim}_{src,d} \) of the method \( M_{src} \) and \( M_d \) using formula (3), and give the final conclusion according to the formula (2) and \( \alpha^{[10]} \).

\[
\text{sim}_{src,d} = (1 + e^{\frac{\theta N_{\text{common}} - \text{bias}}{N_{src} + N_d}})^{-1}
\]

4. Experimental Results

This paper selects four open source testing and analysis tools in wide used-Junit; JMeter; selenium and find bugs, set \( R_{\text{max}} = 3 \), \( \alpha =0.3 \), \( \theta =10 \), \( \text{bias} = -2 \).

This experiment will test the experimental subjects in two cases, case 1: corresponding method between different versions of the same program; case 2: random picked method between different programs. For case 1, when a software evolving from version 1 to version 2, more classes and methods will be retained, the default result are plagiarized, we marked this kind of result as positive. For case 2, the default result is independent development, marked this kind of result as negative.

In order to verify the credibility of the birthmarks presented in this paper, 8 pairs of positive programs and 4 pairs of negative programs were used, result shown in table 1 and figure 4.

| Test program pair       | Similarity of this paper | Fit actual | SKB | Fit actual |
|-------------------------|--------------------------|------------|-----|------------|
| Assert Equals(Junit4.0&4.3) | 0.97                     | Y          | 0.94| Y          |
| Run Bare(Junit4.1&4.2)   | 0.93                     | Y          | 0.92| Y          |
| Make Alias(Jmeter3.1&3.2) | 0.95                     | Y          | 0.90| Y          |
| Write Header(Jmeter3.2&3.3) | 0.86                     | Y          | 0.83| Y          |
| Get Current(selenium3.11&3.12) | 0.68                    | N          | 0.65| N          |
| Merge(selenium3.12&3.14) | 0.93                     | Y          | 0.93| Y          |
| Parse(findbugs4.4&4.6)   | 0.90                     | Y          | 0.90| Y          |
| Perform(findbugs4.5&4.7)  | 0.73                     | Y          | 0.68| N          |
| RunBare_junit4.2&makeAlias_Jmeter3.1 | 0.26                 | Y          | 0.28| Y          |
| WriteHeader_Jmeter3.2&merge_seelenium3.12 | 0.19 | Y | 0.23 | Y |
| GetCurrent_selenium3.12&parse_findbugs4.4 | 0.31 | N | 0.34 | N |
| Perform_findbugs4.5&assertEquals_Junit4.0 | 0.17 | Y | 0.21 | Y |
Figure 4 compare control flow birthmark with SKB.

It can be seen from Table 1 that the birthmarks presented in this paper have 2 misjudgments for the 14 pairs of procedures tested, correct rate is 85.7%, precision rate is 87.5%, recall rate is 87.5%, and F-measure is 87.5; while SKB has three misjudgments on 14 pairs of procedures, the correct rate is 78.6%, precision is 85.7%, recall rate is 75%, and F-Measure is 0.8.

In order to verify the resilience of the birthmarks proposed in this paper, 50 methods are randomly searched from four software, and 200 subjects are confused by proGuard\(^{11}\). The 200 pairs of programs are tested for plagiarism using the birthmarks proposed in this paper. The results are shown in Table 2:

To verify resilience of the birthmarks presented in this paper, find 50 methods randomly from 4 softwares, then use proGuard to confuse a total of 200 subjects, use birthmarks presented in this paper to perform plagiarism tests on these 200 pairs of programs, the results are shown in table 2:

Table 2 Verification of resilience of software birthmark.

| Source code | Mean similarity | Precision | F-Measure |
|-------------|----------------|-----------|-----------|
| Junit       | 0.88           | 0.90      | 0.95      |
| Jmeter      | 0.78           | 0.86      | 0.92      |
| Find Bugs   | 0.73           | 0.84      | 0.91      |
| Selenium    | 0.91           | 0.94      | 0.97      |

Cause this test is a resilience test, therefore, the actual results of subjects are positive, there is no FP and FN. There are 23 pairs of misjudgments for a total of 200 pairs of confusing programs, overall correct rate is 88.5%. This result shows that the birthmark proposed in this paper has resilience.

5. Conclusions
This paper obtains the control flow birthmark through the instruction flow by analysis program, then use the VF2 algorithm to calculate the similarity of two birthmark and compare the credibility and resilience of birthmark proposed in this paper. The experimental results show that birthmarks presented in this paper are more credible than traditional SKB birthmarks. Cause the birthmark proposed in this paper is based on intermediate instruction analysis, so it has more applicability than SKB.

6. Acknowledgement
This work was supported by the Education Refore Project of Xi’an Technological University (No.18jgz01).

7. References
[1] Christian Collberg, Clark Thomborson. Software Watermarking: Models and Dynamic Embeddings [A]. Proceedings of Symposium on Principles of Programming Languages [C], 1999: 311-324.
[2] Haruaki Tamada, Masahide Nakamura. *Detecting the Theft of Programs using Birthmarks* [R]. Information Science Technical Report NAIST-IS-TR2003014 ISSN 0919-9527, Graduate School of Information Science, Nara Institute of Science and Technology, 2003.

[3] Lin C, Fen-Lin L, Bin L U, et al. *Static Software Birthmark Based on k-gram Frequency*[J]. Computer Engineering, 2011, 37(4):46-48.

[4] Xin Xie, Fenlin Liu, Bin Lu, Lin Chen. *A Software Birthmark based on Weighted K-gram*[C]. IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2010), Xiamen, China, 2010:400-405.

[5] David Schuler, Valentin Dallmeier, Christian Lindig. *A Dynamic Birthmark for Java*[A]. Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering [C]. 2007: 274-283.

[6] Heewan Park, Seokwoo Choi, Hyun-il Lim, Taisook Han, *Detecting Java Theft Based on Static API Trace Birthmark* [A]. Proceedings of International Workshop on Security (IWSEC) [C]. Springer LNCS, 2008, 5312: 121-135.

[7] Cordella L P, Foggia P, Sansone C, et al. *A (Sub) Graph Isomorphism Algorithm for Matching Large Graphs*[J]. IEEE Trans Pattern Anal Mach Intell, 2004, 26(10):1367-1372.

[8] Derrick Grover. *Program Identification* [M]. The Protection of Computer Software - Its Technology and Applications, Cambridge University Press, Cambridge, 1989: 119-150.

[9] Ginger Myles, Christian Collberg. *K-gram Based Software Birthmarks* [A]. Proceeding of ACM Symposium on Applied Computing [C]. ACM, 2005: 314-318.

[10] Wang S, Zhao P, Sun J. *Software birthmark extraction algorithm based on multiple features*[J]. Journal of Computer Applications, 2018.

[11] Kim D, Gokhale A, Ganapathy V, et al. *Detecting plagiarized mobile apps using API birthmarks*[J]. Automated Software Engineering, 2015:1-28.