Graphs Resemblance based Software Birthmarks through Data Mining for Piracy Control

[1] *Sohail Sarwar <sohail.sarwar@seecs.edu.pk>*

[1] *Zia .Ul .Qayyum <zia@aiou.edu.pk>*

[2]*Muhammad. Safyan<msafyan@gcul.edu.pk>*

[3,4]*Muddesar Iqbal <miqbal@lsbu.uk>*

[1]*Yasir. Mahmood <yasir@iqra.edu.pk>*

[1] *Department of Computer Science, University of Gujrat, Pakistan*

[2] *Department of Computing, GC University Lahore, Pakistan*

[3] *School of Engineering, London South Bank University, England*

[4] *School of Computer Science and Electronic Engineering University of Essex, England*

Abstract. The emergence of software artifacts greatly emphasizes the need for protecting intellectual property rights (IPR) hampered by software piracy requiring effective measures for software piracy control. Software birthmarking targets to counter ownership theft of software by identifying similarity of their origins. A novice birthmarking approach has been proposed in this paper that is based on hybrid of text-mining and graph-mining techniques. The code elements of a program and their relations with other elements have been identified through their properties (i.e code constructs) and transformed into Graph Manipulation Language (GML). The software birthmarks generated by exploiting the graph theoretic properties (through clustering coefficient) are used for the classifications of similarity or dissimilarity of two programs. The proposed technique has been evaluated over metrics of credibility, resilience, method theft, modified code detection and self-copy detection for programs asserting the effectiveness of proposed approach against software ownership theft. The comparative analysis of proposed approach with contemporary ones shows better results for having properties and relations of program nodes and for employing dynamic techniques of graph mining without adding any overhead (such as increased program size and processing cost).

Keywords: Intellectual Property Rights (IPR); Software Ownership Theft; Software Birthmarking; Graph Mining; Credibility and Resilience;

For citation: Sarwar S., Qayyum Z.U., Safyan M., et al, Graphs Resemblance based Software Birthmarks through Data Mining for Piracy Control. *ISP RAS*, vol. 1, issue 2, 2016. pp. 3-4 (in Russian). DOI: 10.15514/ISPRAS-2016-1(2)-33

1. Introduction

The revolutionary impact of software engineering has changed the course of technological advancements through innovative ideas. Implementation of these ideas has entailed in a paradigm shift with concepts like mobile apps, smart technologies (smart homes, offices and cities etc), computational paradigms (grid and cloud computing) and Technology enhanced Learning (TeL) and Internet of Things (IoT) etc. These innovative software developments based on novelty of ideas (i.e. intellectual property), are facing potential threats. Few of these threats to intellectual property rights (IPR) are software piracy, ownership theft, reverse engineering and software copying (or copying of software parts) etc. One of the studies asserts that more than 50% of technology consumers are working on pirated software [1]. A major reason behind this widespread violation of piracy laws is inherent nature of software products that can be replicated and redistributed quite easily contrary to products from other industries. The software products are purchased by customers while acquiring the rights only for using those solutions with no entitlement for subsequent alteration or distribution of programs (or software) in any capacity. However, these ethical bindings are violated by illegally redistributing the software entailing in huge loss to IoT vendors due to software piracy. Software vendors secure the copyrights of their products while keep in view the best practices [6] so that personnel having authorization may use the software or customize if required. Almost all the aspects of software are concealed to prevent it from unauthorized selling. So the ultimate objective of product seller would be to prevent and protect theft of pirated software [7].

A number of techniques [2] [3][4][5] have been developed to control software piracy. These techniques target to counter the increasing number of pirated software, reverse engineering, software tampering and software theft. Examples of these techniques are:

- Watermarking: used to prove software ownership.
- Tamper-proofing: program is destroyed on illicit use.
- Obfuscation: targets to obscure the software for countering revere engineering.
- Encryption: secures the software by encoding through public/private keys.
- Birthmarking: theft detection of software based on its unique properties.

All the techniques except software birthmarking may add different decision statements in the code; consequently, the size of the code increases entailing in performance degradation of the and hence the reusability. Also, software birthmark *identifies unique characteristics of a program (termed as intrinsic properties) such as elements of variables, loops, decisions and assignments etc* [8]. It is not a trivial task to alter these program constructs and easily substantiate [7], [9] while identifying software theft (or ownership theft).

Keeping above in view, a novel software birthmarking technique is proposed by exploiting graph mining constructs. The graph theory [8] and network methods [9] have been used to detect birthmarks of each method and class to prove their respective ownership. The proposed method based birthmarking technique is static in nature that extracts syntactic structure of program. This structure is used to compute property value for each program element and relations among those elements. Subsequently, the property values are transformed into graphs. These graph theoretic properties (through clustering coefficient) compare birthmarks (in graph forms) of two programs. Hence, two programs are classified as similar or dissimilar (identify if a program has been copied or a single class or a method(s) of the class is pilfered).

Moreover, the proposed technique of “Graph based Static Birthmarking” has potential to spot if a software program has been altered. The technique calculates software birthmark based upon relationship of intrinsic *characteristics* of each method in program. These *coupled characteristics* have a key role in carrying out the functional requirements of each method. So all the method level birthmarks in program classes are exploited to construct birthmark of a program. These *coupled class relations* incorporated in program are used in generating the program birthmarks. All the birthmarks of shortlisted methods are used to develop a statistical view of resemblance by comparing the generated birthmarks. These statistical calculations provide a measure of similarity or dissimilarity among different programs. The aspects of proposed technique have been evaluated over the metrics of credibility and resilience at different levels of granularity while keeping track of overheads involved.

Rest of the paper has been organized as given: Section II highlights the basic concepts of software birthmarking and overview of prevalent techniques. Section III provides a detailed insight to the birthmarking technique presented in this work. The evaluation of proposed approach has been discussed in section IV. Section V concludes the presented work with a view of future directions.

2. Literature Survey

In this section, a review of existing piracy control techniques is presented in general and birthmarking techniques specifically.

The increasing volume of software products catalyzed by internet, has given rise to phenomenon of software piracy [10].

Software piracy has become a global issue as depicted by a figure of $62.7 in the form of unlicensed software in 2013 [1]. Similar studies indicate 42- 43% increase in piracy just in 2014 [2]. The piracy may be divided into two main domains i.e. software illegal distribution and reverse engineering. Although, plenty of developments have been made for minimizing software piracy such as obfuscation, tamper-proofing, watermarking, hashing and control flow monitoring, but their effectiveness seems to have more room for improvement. Obfuscation transforms the program to make it less intelligible while preserving its semantics [11][12]. Tamper-proofing [7] introduces checks for protection [13] against piracy with added code overhead. So these techniques degrade performance of software program, requiring special execution environment such as customized JVM [14].

Software watermarking is a technique for shielding the intellectual property of an application program. Its concept has same notion as that of digital watermarking in which unique identifier are placed in text, imagery, or audio/video data in a way that cannot be detected by humans [19] [20] [21][22] [23].

Some of the researchers have used knowledge engineering techniques for classification of plagiarism [24]. Intrinsic plagiarism, external plagiarism and authorship may be detected [25] using text mining. In [26], a practical and effective technique for the detection of pirated software using metaphoric analysis is discussed. Multi diminution attributes have been used for detection of software theft.

In some of the approaches, hybrid of software birthmarks and watermarks has been presented to counter software piracy [22][24] [25]. Since birthmark identifies only the intrinsic properties of the software program. So combining watermark and birthmark may be an effective counter piracy measure without compromising software performance [15].

Beside detection of copy-theft the software birthmarks have been used for malware detection [27, 28] in software.

In order to minimize the underutilized reuse of hardware designs, hardware birthmarks have been used for predicting reusable hardware designs [29].

The issues of prevalent techniques for malware classification in android based applications have been discussed in [35]. Subsequently, a system named FalDroid, based on sub-graph analysis for malware family classification, shows a better degree of accuracy in malware classification.

The classification of software Birthmarks and its prevalent research endeavors are given in the following:

2.1 Software Birthmark

The phenomenon of software birthmarks refers to unique set of properties in a computer program or in its component. Every birthmark of a program possesses two properties i.e. credibility and resilience for transformation [7], [15], [16], [9]. It may be exemplifies with following variables:

Suppose '*a', 'b'* refer to two different code snippets or modules of code and '*z'* refers to code snippet for extracting set of unique properties of '*a'* and '*b'* to have a birthmark *z(a, b).* Here *z(a)* refers to the unique characteristics of '*a' and z(b)* refers to unique properties of '*b'*. These birthmarks have been categorized into two genres namely static birthmarks and dynamic birthmarks as elaborated below [15]

2.2. Software Birthmarks: Static

The variation of birthmarks named static birthmarks is described here with baseline work in [8].

Let '*a', 'b'* are the code snippets from different programs and 'z' refers to the program component extracting unique set of properties from certain method(s). The birthmark of 'a' exists *if and only if:*

1. '*z(a)'* may be extracted from snippet 'a' (by not having data from other components of program), and
2. '*b'* is a copy of '*a'* => *'z(a)' = 'z(b)'*

2.3. Software Birthmarks: Dynamic

The other variation of birthmarks is the dynamic birthmarking given below with foundational work in [15].

Let *'a', 'b'* is the code snippets with 'c' as console input to these code snippets.

Let 'z' is a code snippet for extracting unique properties of programs *'a'* and *'b'*.

Then it may be asserted that *z (a, c)* is birthmark of 'a' *if and only if*:

1. *z(a, c)* is obtained from 'a' only after execution of snippet *'a'* and giving the console input of *'c'*. and
2. *'b'* is a copy of *'a' => z(a, c) = z(b, c)*.

Some other variations of software birthmarks have also been found in [8, 9, 15 and 16] with four prominent types specifically targeting java class files as given in the following:

a) Constant values in the field variables (CVFV),
b) Sequence of method calls (SMC),
c) Inheritance structure (IS)
d) Used classes (UC).

These birthmarks lack the requisite reliance and are trivial in nature hence easily prone to transformation [15]. The birthmark approach based on whole program path (WPP) proposed in [16] may suffer vulnerabilities such as loop transformations or method in-lining attacks [17]. Also, WPP may not rightly differentiate different code snippets as given in Fig. 1 where two different codes are declared as "copied". Another dynamic birthmarking approach, more resilient than WPP, is based on call sequence of Application Program Interface (API) [9]. However, a window of short method calls may cater a limited set of API calls. Dynamic birthmarking techniques for multithreaded applications have been proposed in [17] [18] exploiting thread-related system calls. Birthmark generation for detecting plagiarism in multithreaded programs is a challenging task. A dynamic birthmarking technique named TOB-PD has been proposed in [36] that exhibits a resilient performance.

Keeping in view the shortcomings of existing work and research challenges, a novel and effective software birthmark technique has been proposed. The proposed approach, as given in section 3, is a hybrid of text mining and graph mining techniques employed to identify the software birthmarks. The graph mining mechanics, such as clustering co-efficient and clique [30, 31], have been used to identify the modified code in addition to detecting birthmarks. Graphs are typically helpful and crucial to the domain [32] such as relationship among elements within method as well as classes of software program (or program) for detecting its origin.

3. Proposed Approach

The proposed "*Graph based Static Birthmarking Technique*" works on intrinsic properties of program code by identifying elements in method(s), element properties and their relation with other elements in other software codes. The elements of a program (within a class or method) represent the properties and relations among elements. These elements are the method variables, repetitions/loops, assignments and decision statements. The relation among elements with their properties is transformed into graph. Nodes in graph represent method elements and edges represent the relations among elements (as shown in Fig 4 and 5). It implies that in code snippets, all the method elements should have minimum one connection (or relation) with other elements [8]. This phenomenon may be explained for graph '*G'* having '*N'* number of nodes with '*E'* number of edges as given in the following:

$G = \{N, E\}$

$N = \{n1, n2, n3, \ldots nk\}$

$E = \{e1, e2, e3, \ldots em\}$

For any node *i* and *j*, there must be a relation between them.

$n_i \rightarrow n_j$ for $i \neq j$

In order to make a birthmark more resilient, code snippet (or the method) comprises of multiple number of elements and properties (instead of 1-1 mapping of nodes and edges). This result in a complex transformation of *intrinsic characteristics* of methods called program birthmarks. Such characteristic (or birthmarks) classify if the code snippet as a copy or original one.

The examples of relation among elements for birthmark may be viewed in Table 1.

Table.1

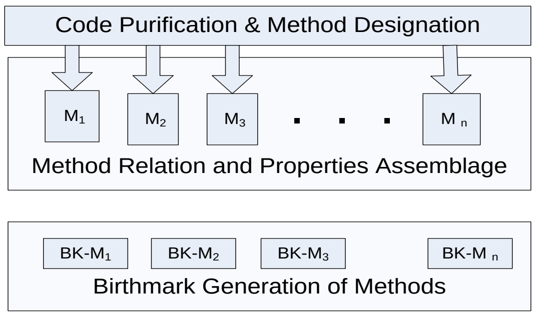
Intrinsic Characteristics: Properties and Relations in Elements

Element Name	Properties				
	Global variable	Local variable	Loop	Condition	Data
Global variable	assign	assign	Used	Used	Assign
Local variable	assign	assign	Used	Used	Assign
Loop	used	Used	Sibling	Used	Used
Condition	exit	Used	Used	Sibling	Used
Data	assign	Assign	Used	Used	Assign

Here, the different types of relations (subsumption or generalization) exist among loops and sibling relations with other loops. Similar types of relationship can be observed between global and local variables. These combinations among relations and properties of method elements are essential for birthmark classification.

The process of Birthmark assemblage and generation has been illustrated in Fig 1. The foremost step is *code purification and designation* that scans the program code and purifies it by pruning the impurities such as non-referred instances or code comments. Subsequently, method properties of program are identified followed by identification of method elements and relations with designated methods. These elements and their properties are *assembled* for having method-level graph birthmark. The extracted birthmark assists in distinguishing the two codes (implemented in isolated environments) from each other based on method properties and elements. Birthmark 'P' (BMP) and Birthmark 'Q' (BMQ) have been extracted from code snippets 'P' and 'Q' respectively as shown in Fig 1.

Generally, software birthmarks are employed for detection of original code when two codes have same origin. However, original program out of two programs can be distinguished through graph based relations where graph having strong and realistic relation refers to original program.



Code Purification & Method Designation
M1
M2
M3
M n
Method Relation and Properties Assemblage
BK-M1
BK-M2
BK-M3
BK-M n
Birthmark Generation of Methods

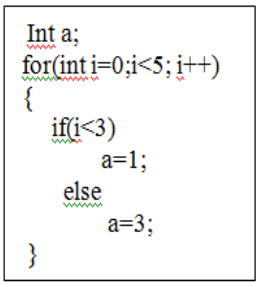
Birthmarks of two programs are classified as similar if their properties and relations match provided the credibility conditions are satisfied. The credibility condition of birthmarks states that all the codes developed in isolated environments must be distinct having no similarity with each other. Furthermore, if content of the program copied is altered, the transformation should also be the disclosed by birthmark (termed as resilience of birthmark) [7, 8, 9, 10, 11]. The proposed technique comprehends both the properties of birthmarking as discussed in section IV-(A). The contemporary techniques may not detect the modified birthmarks. However, proposed technique detects both the program modifications and transformations through method based copy-detection ability.
The sample code in Fig 2 exhibits similar behavior through conventional birthmarking as given in [7, 8, and 9]. Hence, the reliability property is not satisfied. The logic in two sample programs is different that may be identified by the applied birthmarking technique. The proposed technique has capacity to rightly identify both the programs as different from each other. This capacity is due to individual element identification along with their properties as shown in Table 1.

3.1. Birthmark Extraction

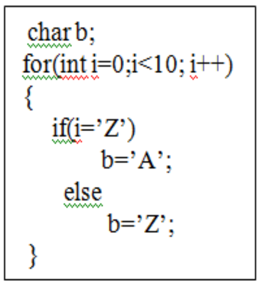
The process for birthmark extraction from certain method code is given in the following:

- Read method(s) and identify elements in the method(s).
- Identify properties in method elements.
- Identify relationship of elements and calculate edge-weight
- Transform relations into graphs based on edge-weight.
- Generate birthmark based on relations and properties.

The birthmark generated comprises of different program constructs such as method elements, properties of elements and relationship between identified elements. The generated birthmark has been employed for classifying the degree of match or disparity among methods under consideration.

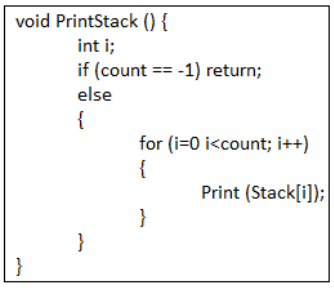


```
Int a;
for(int i=0;i<5; i++)
{
   if(i<3)
          a=1;
     else
          a=3;
}
```



```
char b;
for(int i=0;i<10; i++)
{
   if(i='Z')
          b='A';
     else
          b='Z';
}
```

A sample code form hypothetical program "A" has been used for explanation of proposed technique



```
void PrintStack () {
        int i;
        if (count == -1) return;
        else
        {
                for (i=0 i<count; i++)
                {
                        Print (Stack[i]);
                }
        }
}
```

3.2. Similarity Extraction

The process for similarity extraction birthmarks (graphs) of the two codes is given in the following

- Consider both the programs claimed as original one and suspected one.
- Identify *similar* elements from both graphs (birthmark).
- Compare clustering coefficient and property value of *similar* elements.
- Computing the degree of similarity among elements of code snippets.

The process of Birthmark similarity extraction (and their comparison) may entail in one of the four possibilities as given below:

a. *Full Similarity*: When all the elements of generated birthmark for method(s) share similarity of properties and relations, it implies that considered methods in code are "full-copy" of original method as given in Fig 3.

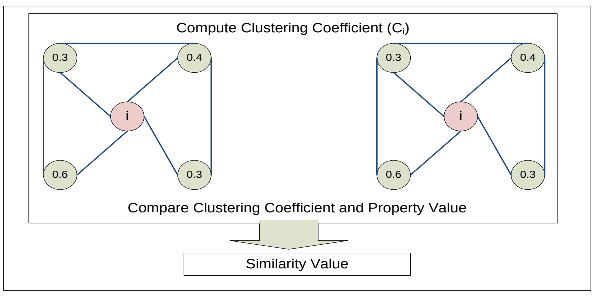
b. *Modified Similarity:* If all the properties of two programs are copied, it is called modified-copy. Here, methods are duplicated for illegally customizing and disguising the program.

c. *Suspected Similarity:* If some of the relations and properties are found similar, it is called modified copy. This case is challenging to detect as program may or may not be proved as copy.

d. *No Similarity:* If there is no similarity in properties and relations of a birthmark, the program may be asserted to have different origins.

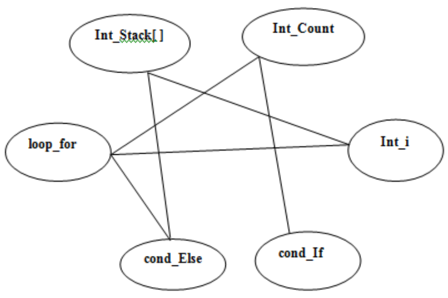
First and last cases are pretty straightforward and can be validated quite easily contrary to second and third cases having overlapped modification or transformation.

In view of mentioned scenarios, four levels of element similarity have been defined. Level 0 implies zero level of likeness among elements, level 1 represents a suspected similarity, level 2 hints that alteration has been made and level 3 means full similarity.


Compute Clustering Coefficient (Ci)
0.3
0.4
i
0.6
0.3
Compare Clustering Coefficient and Property Value
Similarity Value

Suspected and modified similarities are calculated by exploiting the decided threshold. If ε represents a specified threshold, then similarity may be calculated as given in eq(i).

$$P(pi) \,||\, P(qi) > \varepsilon > P(qi) \,||\, P(pi) \qquad eq(i)$$



The similarity level 2 and level 3 can be distinguished by measuring the difference (or distance) between element properties. The distance between properties of elements of p_i and elements of q_i is calculated through eq (ii). If defined threshold appears between distance of elements (p_i, q_i) and (q_i, p_i) then the objects are classified on level 2 i.e. modified similarity. When similarity for every element of entire method is calculated for both programs, distance between properties *p* and *q* is calculated. The calculated distance for all the elements in method classifies the similarity at level 3 i.e. suspected similarity.

e

$$\sum_{j=1}^{m} P(p_i| \quad) \vee P(q| \quad |i)$$

4. Analysis and Outcomes

A similarity percentage between different programs was computed through weighted-clustering-coefficient of various nodes in birthmark graph. The proposed graph based birthmarking technique is evaluated over the metrics of credibility, resilience, modified code detection and self-copy detection for an overall performance measurement. Moreover, a comparative analysis has been made with several attacks while focusing on code transformations and modification. The results of transformation and modification attacks were evaluated through confusion matrix for object methods as discussed in section IV-D. Each of the evaluation aspects has been discussed in the following.

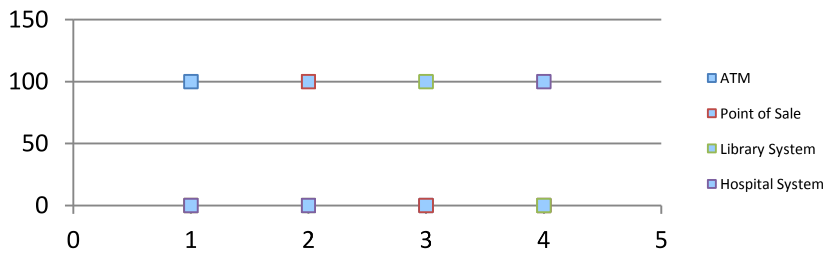
4.1. Credibility and Resilience

Credibility property of birthmark prevents it from identifying the independent programs to be similar. Resilience property identifies the modified and transformed programs as similar. The resilience property is important when programs are modified to attack the existing birthmark relations. For example, two independently developed code snippets may be asserted as dissimilar with zero percent similarity; on the other hand the code snippets having 100% similarity indicates that both codes are perfect copy of each other. The percentage has been divided into 10 equal levels, level 1 shows similarity between 0% to 10%, level 2 shows 11% to 20% whereas level 10 shown 90% to 10% similarity in similar order.
The Table 2 shows results for similarity classification with focus on credibility and reliability for selected programs. The similarity classification has been computed among different programs and same (self) program. The results show that self-programs has been classified as perfect copy where as other programs show 0% classification results.

Table 2. Similarity Classification for Credibility and Reliability

Java Software Packages	**ATM**	**Library System**	**Point of Sale**	**Hospital System**	**Kmeans Algo.**
ATM	100	0	0	0	0
Library System	0	100	0	0	0
Point of Sale	0	0	100	0	0
Hospital System	0	0	0	100	0
K-means Algo.	0	0	0	0	100

The results furnished in table II have been plotted in Fig 6. The horizontal axis shows ATM, Point of Sale, Library System and Hospital System. ATM comparison with ATM is 100% while ATM comparison with rest of the programs is 0%. Each of the system has login management system, since every program is independently developed; therefore no similarity classification has been computed among them.



4.2. Detection of Modification and Transformation

The programs are modified and transformed in order to hide the existing birthmarks. The proposed technique classifies both the transformed as well as modified programs as similar through its predictive analysis. Since each of the elements in the birthmark has property value based on relation among elements (nodes), so they are used for similarity classification in modified and transformed codes. In the Table 3, confusion matrix for similar as well as non-similar objects has been shown. The accuracy computed for similar as well as non-similar objects in the methods is 0.90. The class method objects have been chosen without transformation but with modification with in method codes as explained in section 4-C.

Table 3. Confusion Matrix Similarity Calculation for Modified Programs

Classes	Similar Objects'	Non-Similar Objects'	Total
Similar Objects	974	36	1010
Non-Similar Objects	164	826	990
Total	1138	862	2000

4.2. Attack analysis

The birthmarks were extracted from program methods as relations between elements of program along with element properties. In such scenarios, the code (i.e. methods) needs to be re-written completely for penetrating the proposed technique. It increases the time required for modifying the code exceeding time required to develop a new program. In view of current scenario, the transformation of small sized programs seems trivial but program transformations for larger programs are difficult. Also, modifying single or a few methods may not be useful for the attacker since each of the methods have dependencies.

In another scenario of attacking the proposed technique successfully, code may be changed by adding extra blocks in program such as adding outer loops, outer conditions or changing *for-loop* to *while-loop* etc. Such code changes are rightly detected but asserted as suspected-copy instead of full-copy. Another known shortcoming in proposed technique is inability to detect single variable transformation to 2-variables. For example, a variable x is distorted with two new variables y and z to maintain the value of variables and for further processing. Such transformations are difficult to be identified. However, such transformations increase the size of the code and hence processing time. Further such transformations are time consuming, tedious and time for modification or transformation may exceed the actual development time.

Example

int a;	int a = 10;
a =10;	int b = a*a;
Transformation	int c= b/a;

Table 4. Self Copy Detection for the Chosen Program

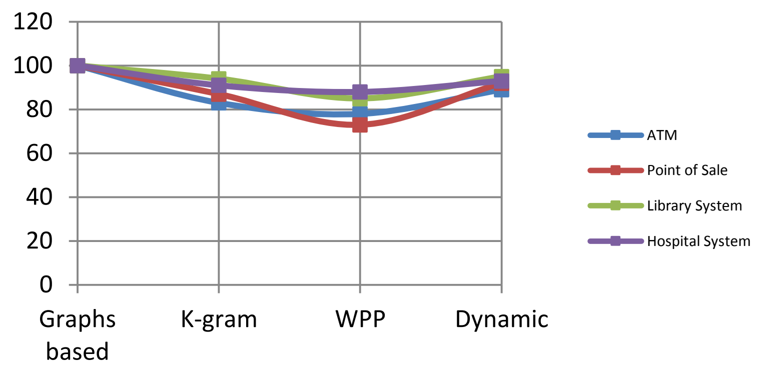
Software Packages	Graphs	N- gram	WPP	Dynamic
ATM	100	83	8	89
Point of Sale	100	87	73	92
Library System	100	94	85	95
Hospital System	100	91	88	93

Since proposed approach extracts method-level birthmarks, so certain method have been identified as similar but overall percent similarity of entire program was too low. So the generation of method-level birthmark is envisioned as a handy to counter methods copy and hence piracy in broader spectrum.

4.4. Comparison with Prevalent Approaches

A comparative analysis of proposed approach is presented with prevalent techniques such as Whole Program Path (WPP) birthmark [15], K-gram based birthmark [16] and dynamic birthmark [9]. Four different and independent programs were selected from sorceforge.net [34] for comparison. Every program was compared with itself and with remaining three programs for calculating the degree of similarity. The comparison of approaches has been illustrated in Fig 7. As evident from the results, the proposed *Graph based* technique with similarity classification is better than rest of the techniques when program code is compared with itself. The reason for better performance of proposed approach is its consideration of relations and properties of each of the nodes in a graph (birthmark).

Further different methods were copies into the code for classifying method similarity in the program code considered. The experimental results are shown in the Table 5. Method classification was not possible with rest of the static birthmarking techniques.

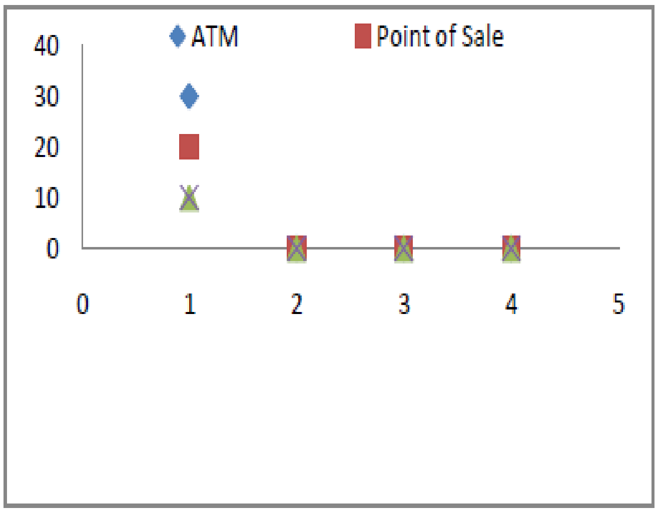


For example, 30% of the methods in the *ATM Software Package* have been classified as *'Copied'*. Contrary to the proposed approach, method copying for same example is shown 0% by rest of the techniques. So it can be asserted that proposed technique is an effective measure against software piracy for method copy detection and complete program copy detection.

Table 5. Method Copy Detection Comparison

Software Packages	Graphs	gram	PP	Dynamic
ATM	30	0	0	0
Point of Sale	20	0	0	0
Library System	10	0	0	0
Hospital System	10	0	0	0

Graph based technique exploits the method structure with existing relations among program elements which has been discussed in section 3. Method special code transformation (as stated in section0 Attack Analysis) of smaller methods, into similar code is comparatively easy. Therefore proposed technique may not be effective measure for programs where smaller methods exist and special code transformation is easy. In case of large methods, the special code transformation is cumbersome and time required for transforming every element of method may exceed that of developing new method.



5. Conclusion and Future Directions

A novel graph-based birthmarking approach has been proposed for detecting ownership theft in IoT software programs. The unique characteristics of code (method elements) with respective relationships have been transformed into graphs for classifying similarity or dissimilarity of programs. The method-level birthmarks complement rest of the methods for similarity classification. Two birthmarks are compared and similarity-factor is assigned to birthmark element based on their depth. These similarity calculations have been used to assign the similarity classification to code out of four categories. The proposed technique is compliant with software birthmarking principles of reliability as well as resilience. Further modified elements in the code can also be detected through proposed approach. The proposed approach has been compared with prevalent techniques asserting that proposed classification technique is superior to rest of the techniques even if flow of code was similar but code logic has been transformed.

In future, we look forward to experiment the dynamic solution for the detection of software theft using hybrid of graph-based birthmarking and software watermarking. Also, the experimental setup will be improved by having larger code repositories and programs.

References

[1]. S. Alliance (BSA), Ninth Annual BSA Global Software 2013 Piracy Study, Business Software Alliance(BSA), 2013.

[2]. Anckaert B., De Sutter, D. Chanet, and Bosschere K. Steganography for executables and code Transformation Signatures, *Information Security and Cryptology,* pp. 425–439, 2005.

[3]. Fu B., Richard G., and Chen Y. Some New Approaches for Preventing Software Tampering, 44th annual Southeast regional conference, pp. 655–660, 2006.

[4]. Collberg S., Thomborson C., Watermarking, tamper-proofing, and obfuscation-tools for software protection, *IEEE Transactions on Software Engineering*, Vol. 28, pp. pp. 735–746, 2002.

[5]. Udupa S. K., Debray K., and Madou M., Deobfuscation: Reverse engineering obfuscated code, 12th Conference on *Reverse Engineering,* pp. 10–19, 2009.

[6]. Palsberg J., Krishnaswamy S., Kwon M. Experience with software watermarking, 16th *Computer Security Applications Conference,* pp. 308–316, 2003.

[7]. Bai Y., Sun X., Sun G., Deng X., and Zhou X. Dynamic k-gram based software birthmark, *19th Australian Conference on Software Engineering*, pp. 644–649, 2008.

[8]. Mahmood Y., Pervez Z., Sarwar S., and Ahmed H. F. Similarity Level Method Based Static Software Birthmarks, *International Symposium on High Capacity Optical Networks and Enabling Technologies,* pp. 205–210, 2008

[9]. Schuler D., Dallmeier V., Lindig C. A dynamic birthmark for java, *22nd IEEE/ACM international conference on Automated software engineering*, pp. 274–283, 2007

[10]. Nazir S., Shahzad S., Binti N., Alias, and Anwar S. A Novel Rules Based Approach for Estimating Software Birthmark, *The Scientific World Journal*, vol. 2015, 2015.

[11]. Jorge E. N., Pirmez L., Costa O., Boccardo R., Bento M., Tiny Watermark: a code obfuscation-based software watermarking framework for wireless sensor networks, in *ICWN'14-The 2014 International Conference on Wireless Networks*, 2014.

[12]. Nayakoji N., Sonavane S. JavaScript Theft Detection using Birthmark and Subgraph Isomorphism, *Journal of Engineering Computers & Applied Sciences*, vol. 3, pp. 1–5, 2014.

[13]. Che S. and Wang Y. A Software Watermarking Based on PE File with Tamper-proof Function, *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, pp. 1012–1021, 2014.

[14]. Zhu F. Concepts and techniques in software watermarking and obfuscation, ResearchSpace@ Auckland, 2007.

[15]. Myles G., and Collberg C., Detecting software theft via whole program path birthmarks, *Information security*, Springer, 2004, pp. 404–415.

[16]. Myles G. and Collberg C. K-gram based software birthmarks, in Proceedings of the 2005 ACM symposium on Applied computing, 2005, pp. 314–318.

[17]. Tian Z., Liu T., Zheng Q. A new thread-aware birthmark for plagiarism detection of multithreaded programs, *38th International Conference on Software Engineering Companion*, 2016, pp. 734–736.

[18]. Tian Z., Liu T., Zheng Q. Exploiting Thread-Related System Calls for Plagiarism Detection of Multithreaded Programs, *Journal of Systems and Software*, 2016.

[19]. Bhattacharya S., Survey on Digital Watermarking–A Digital Forensics & Security Application, *International Journal on Piracy Control*, vol. 4, 2014.

[20]. Khan A., Siddiqa A., Munib S. A recent survey of reversible watermarking techniques, *Information Sciences*, vol. 279, pp. 251–272, 2014.

[21]. Zhou W., Zhang X., and Jiang X., AppInk: watermarking android apps for repackaging deterrence, *8th ACM SIGSAC symposium on Information, computer and communications security*, 2013, pp. 1–12.

[22]. Ren C., Chen K., and Liu P. Droidmarking: resilient software watermarking for impeding android application repackaging, *29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 635–646.

[23]. Guang S., Xiaoping F., Sha F., Yingjie S., Software Watermarking in the Cloud: Analysis and Rigorous Theoretic Treatment, *Journal of Software Engineering*, vol. 9, pp. 410–418, 2015.

[24]. Rubini P., Leela S. A Survey On Plagiarism Detection In Text Mining, *International Journal of Research in Computer Applications and Robotics*, vol. 1, p. 117, 2013.

[25].Oberreuter G., and VeláSquez J. D. Text mining applied to plagiarism detection: The use of words for detecting deviations in the writing style, *Expert Systems with Applications*, vol. 40, pp. 3756–3763, 2013.

[26].Rana H., Stamp M., Hunting for Pirated Software Using Metamorphic Analysis, *Information Security Journal: A Global Perspective*, vol. 23, pp. 68–85, 2014.

[27].Costa M., Gong Z. Web structure mining: an introduction, *Information Acquisition, 2005 IEEE International Conference on Software Security*, 2005, p. 6–pp.

[28].Vemparala S., Di Troia F., Corrado V., Austin H., and Stamo M. Malware Detection Using Dynamic Birthmarks, *2016 ACM- International Workshop on Security And Privacy Analytics*, 2016, pp. 41–46.

[29].Zeng K., and Athanas P., A q-gram birthmarking approach to predicting reusable hardware, *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 1112–1115.

[30].Bogdanov P., Baumer B., Basu P., As Strong as the Weakest Link: Mining Diverse Cliques in Weighted Graphs, *Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 525–540.

[31].Getoor L., Diehl P. Link mining: a survey, *ACM SIGKDD Explorations Newsletter*, vol. 7, pp. 3–12, 2005.

[32].Kavitha D., Rao M., Babu K. A Survey on Assorted Approaches to Graph Data Mining, *International Journal of Computer Applications*, vol. 14, pp. 43–46, 2011.

[33].Tamada H., Nakamura M., Monden A., Java Birthmarks–Detecting the Software Theft, *IEICE transactions on information and systems*, vol. 88, pp. 2148–2158, 2005.

[34].Sorceforge.net, accessed on Nov 2, 2017.

[35].Fan M., Liu J., Luo X. & et al., Android Malware Familial Classification and Representative Sample Selection via Frequent Sub-graph Analysis, IEEE Transactions on Information Forensics and Security, vol. 13, pp. 1890-1905, 2018.

[36].Tian Z., T. Liu, Zheng Q. and et al., Reviving Sequential Program Birthmarking for Multithreaded Software Plagiarism Detection, IEEE Transactions on Software Engineering, Vol. 44, pp. 491-511, 2018.