Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 1, January 2014, pg.470 – 476

RESEARCH ARTICLE

Code Birthmarks and Graph Isomorphism for Theft Detection

Snehal N. Nayakoji, S. P. Sonavane

Department of Computer Science & Engineering, Sangli, India

Department of Computer Science & Engineering, Sangli, India

sneh514@gmail.com; shefali.sonavane@walchandsangli.ac.in

Abstract— JavaScript is becoming more and more popular as client-side scripting language in the web community. However, it is easy to copy the source code of JavaScript program with the help of additional functionalities provided by the browser. Hence, the intellectual property right of web application developers is at risk. To address this issue a new software theft detection technique, called as software birthmark; is introduced. Software birthmark results from the intrinsic characteristics of the program which could be used to determine the similarity between two programs. The proposed paper demonstrates the way to extract code signature and to design software birthmark along with the idea of using subgraph isomorphism to detect the source code theft of JavaScript programs.

Keywords— Software Birthmark; Heap Graph; Subgraph Isomorphism; Intellectual Property Rights (IPR); Theft Detection

I. INTRODUCTION

Because of the advance techniques in web development, JavaScript has become a very good platform to design various applications for web users. As the source code of JavaScript program can easily be copied with the help of 'view source page' function of a browser, it is essential to detect the theft for protecting the IPR of web application developers.

For computer scientists it has become important to protect the software or program source code over years. There are several methods to detect the code theft and hence to protect the IPR of developers. One of the earliest approaches is watermarking. Hiding a trademark or identification for the use of determining ownership is called as watermarking [1], [2]. Watermarking helps in claiming authorization of software. However watermarking is time consuming process as it requires owner to embed watermark into the software and static watermarks are susceptible to attacks such as translation, optimization or obfuscation. Therefore some JavaScript developers use obfuscation technique to convert code into some other format so that it is difficult for humans to understand the syntax of code. If the code is obfuscated then it is difficult for user to reverse engineer it, but one can easily copy the code.

Another emerging technique for detecting source code theft is software birthmark. It is used for identifying the originality of software [3]-[5]. Software birthmark could be extracted from the inherent characteristics of the program. It requires no additional information except for the program itself. Software birthmark system consists of following functions:

$$\text{Extract}(p) \rightarrow b_p$$
$$\text{Extract}(q) \rightarrow b_q$$
$$\text{Compare}(b_p, b_q) \rightarrow [0, 1]$$

Where p, q are programs or program segments and b_p, b_q are software birthmarks of P and Q respectively.

Software birthmark can be categorized into two types, static birthmark and dynamic birthmark. Software birthmark acts as a signature to identify the originality of any software or program. This paper proposes a software birthmark which is based on graph and is used to identify theft of source code. When the JavaScript program runs in the Chrome browser [12], the heap snapshot is taken with the help of 'Heap profiling' function which is provided by 'Chrome Developer Tools'. The objects and references from the snapshot are filtered to get useful objects only. From the filtered objects and references object graph is constructed, which will act as a software birthmark for that particular JavaScript program. For theft detection purpose, software birthmark of original program is constructed and it is searched in the object graph of suspected program. If the software birthmark is found in suspected program's object graph, then theft will be detected.

The paper is organized as follows:

Section II : Definitions [3], [6]
Section III: Survey of related work
Section IV: Methodology to design software birthmark with results of subgraph isomorphism
Section V : Remark

II. DEFINITIONS

A. *Software birthmarks*

A software birthmark can be defined as collection of distinctive characteristics extracted from a source code of a program.

1) *Static birthmark:* When unique characteristics of a program are extracted from a program source code only, it is referred as static birthmark.

p, q: Two programs or program components
f(p) : Set of program characteristics extracted from the source code of p

f(p) is a static birthmark of p only if both of the following criteria are satisfied:

- f(p) is obtained only from p itself
- Program q is a copy of p if f(p) = f(q)

2) *Dynamic birthmark:* When unique characteristics of a program are extracted from a program in execution, it is referred as dynamic birthmark.

p, q : Two programs or program components
I : Input to p and q
f(p; I) : Set of characteristics extracted from p when executing p with input I

f(p; I) is a dynamic birthmark of p only if both of the following criteria are satisfied:

- f(p; I) is obtained only from p itself when executing p with input I
- Program q is a copy of p if f(p; I) = f(q; I).

B. *Graph Isomorphism*

Two graphs are said to be isomorphic (equivalent) if they have identical behaviour in terms of graph-theoretic properties. Let G and G' are two graphs. G and G' are said to be isomorphic to each other if there is one-to-one correspondence between their vertices and between their edges such that the incidence relationship is preserved.

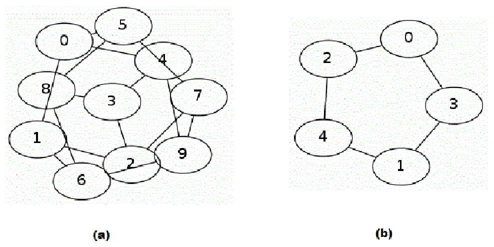


Fig 1: (a) Graph (b) one of the subgraphs of (a)

C. *Subgraph Isomorphism*

A graph g is said to be a subgraph of a graph G if all the vertices and all the edges of g are in G, and each edge of g has the same end vertices in g as in G. A subgraph isomorphism from a graph g=(n,e) to a graph G=(N,E) is an injective function f:n→N such that(u, v)∈e↔(f(u),f(v))∈E.

D. Subgraph Isomorphism Problem

Deciding whether the pattern graph exists in target graph is known as subgraph isomorphism problem. Pattern graph may be a subgraph of target graph [9].

E. Software birthmark based on heap graph

The graph generated from the JavaScript program's objects and its references which are collected in heap memory, is called as the heap graph. The heap graph is denoted as HG.

p, q	: Two programs or program components
I	: Input to the p and q
HG_p, HG_q	: Heap graphs of the program runs with input I for p, q respectively

A subgraph of HG_p is denoted by HGB_p [3]. HGBp is also called as the pattern graph (software birthmark) and HG_q is called as target graph. By searching pattern graph in target graph, it is possible to detect the theft of source code.

III. LITERATURE SURVEY

"A dynamic birthmark for java" [7], proposes API birthmark which is based on how a program interacts with objects from Java Standard API at runtime. To capture a program's API usage, the birthmark observes method calls that are issued by objects from the program and received by objects from the API. This birthmark system may not work if the software does not have many API calls.

Reference [8], "Whole Program Path (WPP) Birthmarking", uniquely identifies a program based on a complete control flow trace of its execution. WPP is constructed by collecting a trace of the path executed by the program. WPP birthmarks can be corrupted by code obfuscation i.e. they are susceptible to various loop transformations.

A system call dependence graph based software (SCDG) birthmark proposed in [5] reflects unique behavioural characteristics of a program. In a SCDG, system calls are represented by vertices, data and control dependences between system calls by edges. A SCDG shows the interaction between a program and its operating system and the interaction is an essential behaviour characteristic of the program. A SCDG birthmark may not work if the program does not involve any system calls or it has very few system calls. It is also not applicable to the programs which do not have unique system call behaviours.

IV. PROPOSED METHODOLOGY

Fig 2 represents the in detail input and output of the proposed work to detect code theft.

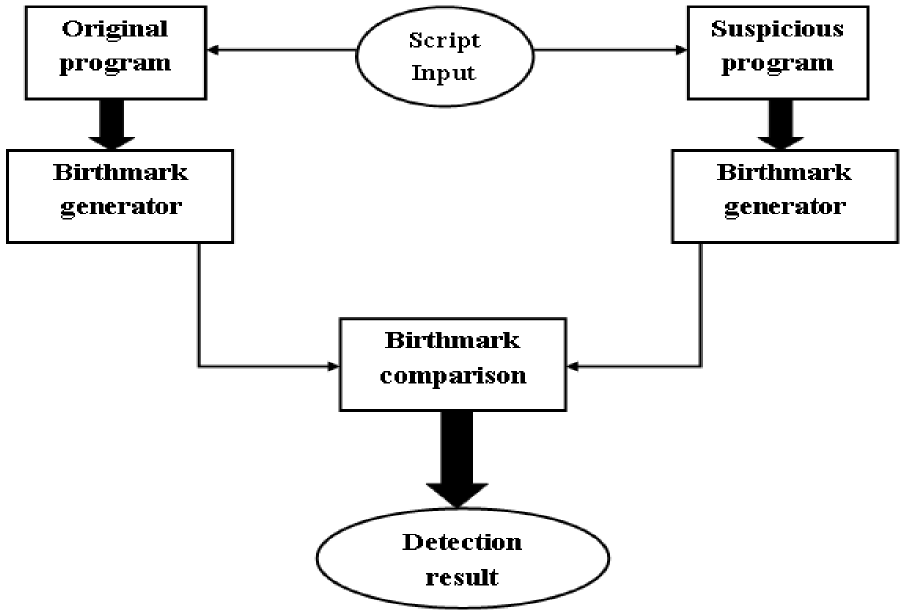


Fig. 2 Input Output System

System steps:

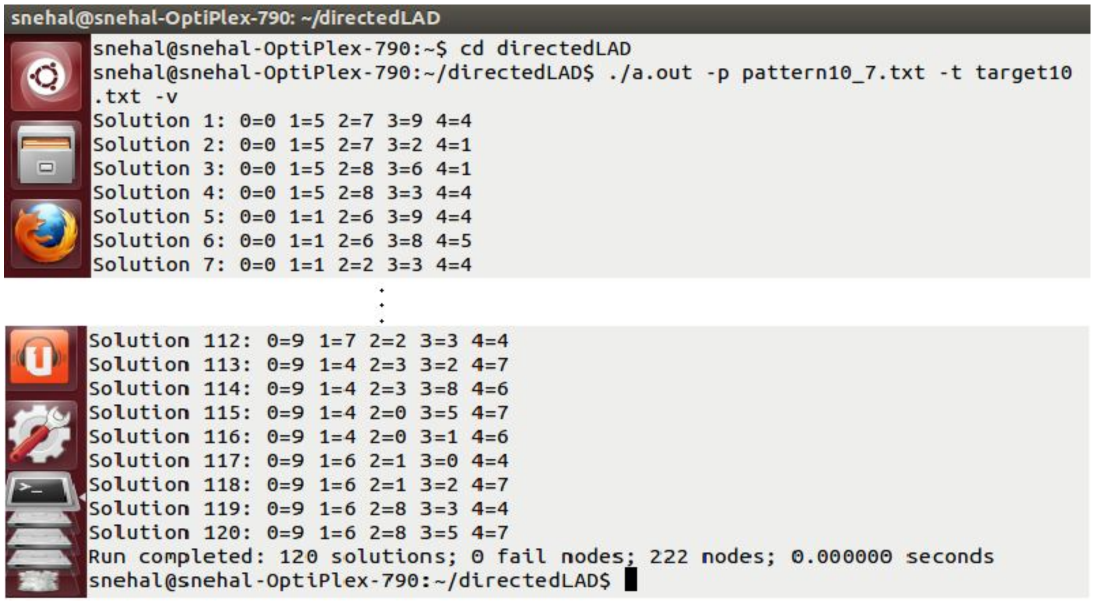
1) Take heap snapshots of running JavaScript program from chrome browser.
2) Filter the objects from the snapshot to get required objects only.
3) Create graph using available objects and references.
4) Select a subgraph for original program only (subgraph for suspected program is not required).
5) Search subgraph of original program in graph of suspected program.
   If the subgraph has been found, theft will be detected.

A. Experimental set up

As shown in figure 2, same input is given to the original program as well as to the suspicious program.

1) *Birthmark Generator:* The first four steps which are described above are carried out by this module. By analysing heap snapshot of a JavaScript program, the constructive objects and references are extracted and object graph is generated. The output of birthmark generator is an object graph of original program and suspicious program. The object graph generated from the original program is referred as software birthmark (pattern graph) for original program and an object graph generated from suspicious program is called as software birthmark (target graph) for suspicious program.

2) *Birthmark Comparison:* For comparing the birthmarks, subgraph isomorphism is used. To detect pattern graph in target graph, directedLAD version2 library [10] is used. By using this library one can search birthmark of original program into the suspected program's graph. If the birthmark is found, theft can be detected. Figure 1 shows a graph and one of its subgraphs. The graphs, shown in figure 1 are generated with the help of Graphviz [11], graph generation tool. Let graph (a) be the target graph and graph (b) be the pattern graph. In this case subgraph isomorphism problem denotes, finding pattern graph in target graph. Figure 3 shows the result of this problem.

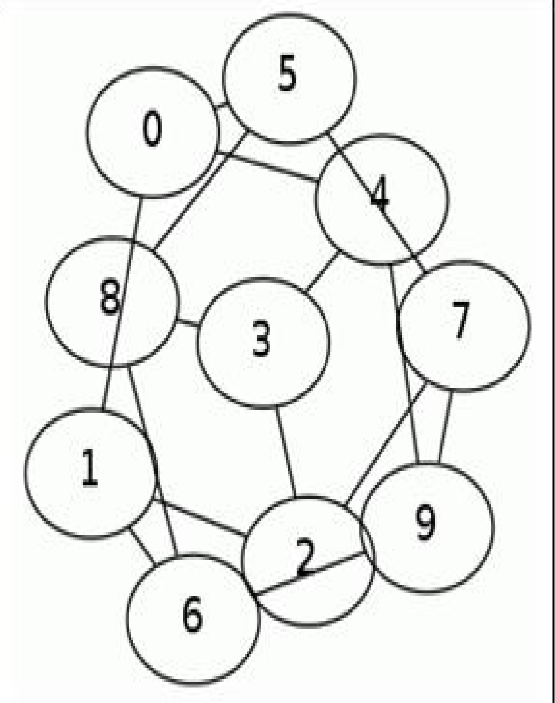
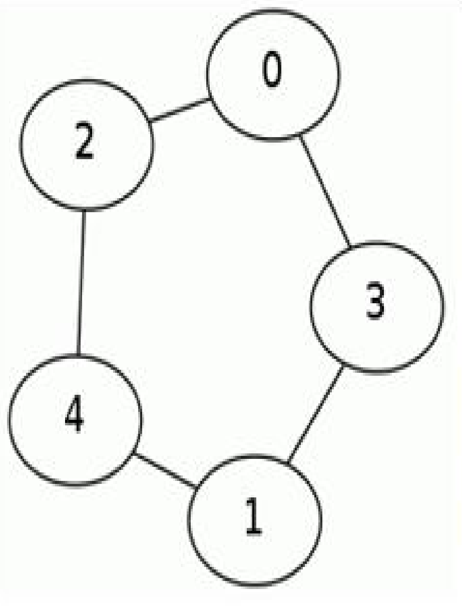


```
snehal@snehal-OptiPlex-790: ~/directedLAD
snehal@snehal-OptiPlex-790:~$ cd directedLAD
snehal@snehal-OptiPlex-790:~/directedLAD$ ./a.out -p pattern10_7.txt -t target10
.txt -v
Solution 1: 0=0 1=5 2=7 3=9 4=4
Solution 2: 0=0 1=5 2=7 3=2 4=1
Solution 3: 0=0 1=5 2=8 3=6 4=1
Solution 4: 0=0 1=5 2=8 3=3 4=4
Solution 5: 0=0 1=1 2=6 3=9 4=4
Solution 6: 0=0 1=1 2=6 3=8 4=5
Solution 7: 0=0 1=1 2=2 3=3 4=4
                         .
                         .
                         .
Solution 112: 0=9 1=7 2=2 3=3 4=4
Solution 113: 0=9 1=4 2=3 3=2 4=7
Solution 114: 0=9 1=4 2=3 3=8 4=6
Solution 115: 0=9 1=4 2=0 3=5 4=7
Solution 116: 0=9 1=4 2=0 3=1 4=6
Solution 117: 0=9 1=6 2=1 3=0 4=4
Solution 118: 0=9 1=6 2=1 3=2 4=7
Solution 119: 0=9 1=6 2=8 3=3 4=4
Solution 120: 0=9 1=6 2=8 3=5 4=7
Run completed: 120 solutions; 0 fail nodes; 222 nodes; 0.000000 seconds
snehal@snehal-OptiPlex-790:~/directedLAD$
```

Fig. 3 Snapshot of Results

Figure 3 represents the result of one of the subgraph isomorphism problems described above for figure 1. To analyse the result, consider f(I) as a node matching function, where 'I' represents node of a pattern graph. The solution gives matching between nodes of a pattern graph to the nodes of a target graph.

TABLE 1: Subgraph Isomorphism Solutions

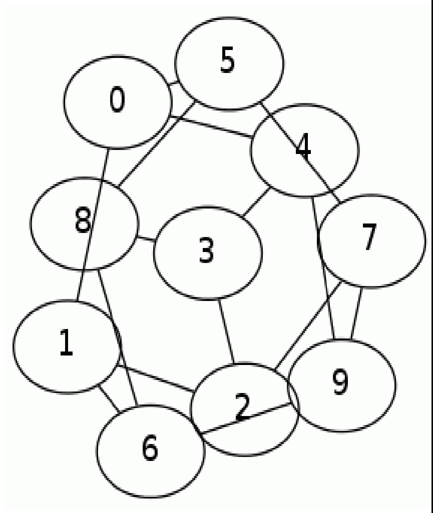
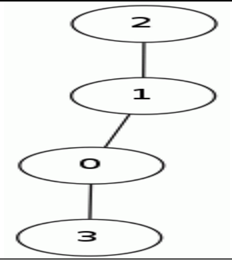
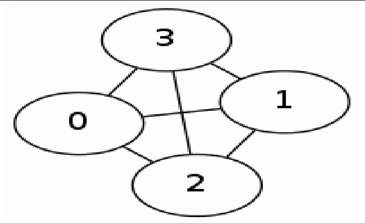
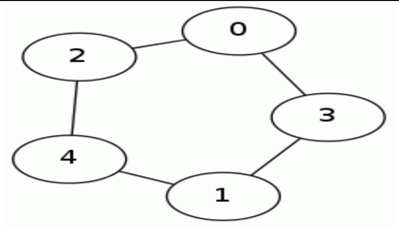
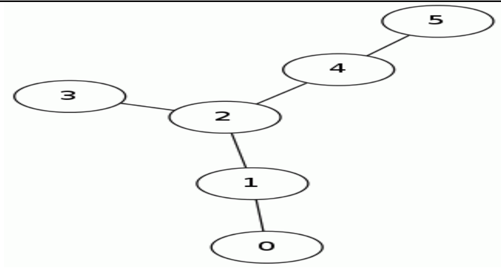
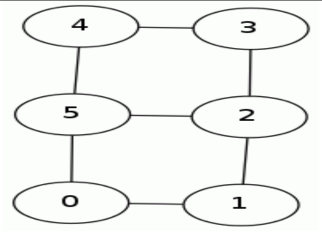
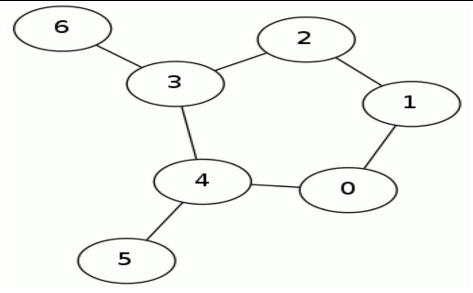
Graph (a)[Traget graph]	Subgraph (b)[Pattern graph]	subgraph isomorphism between (a) and (b)
		solution 1: f(0)=0 f(1)=5 f(2)=7 f(3)=9 f(4)=4 solution 2: f(0)=0 f(1)=5 f(2)=7 f(3)=2 f(4)=1

The table 1 reveals the description of result given in figure 3. As described in table 1, for this particular problem, there are total 120 solutions.

The table 2 reflects the results of subgraph isomorphism. In a given target graph multiple pattern graphs are searched and the output is given in table 2. Table 2 describes number of solutions obtained and indicates whether the pattern graph is present in target graph or not i.e. whether the subgraph has been found in target graph or not.

V. Remark

TABLE 2: Experimental Results of Subgraph Isomorphism

Target graph	Pattern graph	No. of solutions	Subgraph found (yes/no)
		120	YES
		0	NO
		120	YES
		240	YES
		0	NO
		120	YES

The literature review elaborates the idea of software birthmark for code theft detection. It finds the scope in creation and design of software birthmark. The paper discussed, demonstrates the graph based technique for designing software birthmark which can be used to decide authentication of software or a program. Code birthmark and graph isomorphism, collectively address the issue of code theft. Consequently it leads to the protection of IPR laws of application developers. This project can be supported and extended further with designing software birthmarks and detecting code theft for other programming languages. At the time of comparison; certain condition filters can be applied which will be useful to undertake various security applications and can check for the robustness of the system.

REFERENCES

[1] C. Collberg and C. Thomborson , "Software watermarking: Models and dynamic embeddings," in Proc. Symp. Principles of programming Languages (POPL'99),1999, pp. 311-324

[2] A. Monden, H. Iida, K. I. Matsumoto, K. Inoue, and K. Torii, "Watermarking java programs," in Proc. Int. Symp. Future Software Technol., Nanjing, China, 1999.

[3] P. P. F. Chan, L. C. K. Hui, and S.M. Yiu, "Heap Graph Based Software Theft Detection," in IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL 8, NO.1, JANUARY 2013, pp-101-110.

[4] P. P. F. Chan, L. C. K. Hui, and S.M. Yiu, "Jsbirth: Dynamic JavaScript birthmark based on the run time heap ," in Proc. 2011 IEEE 35th Annu. Comput. Software and Applicat. Conf.(COMPSAC), Jul. 2011, pp 407-412.

[5] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Behaviour based software theft detection", in Proc. 16th ACM Conf. Comput. And Commun. Security (CCS '09), New York, 2009, pp. 280-290, ACM .

[6] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. I. Matsumoto, "Design and Evaluation of Dynamic Software Birthmarks based on API Calls," Nara Institute of Science and Technology, Tech. Rep., 2007.

[7] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for java," in Proc. 22nd IEEE/ACM Int. Conf. Automated Software Eng.(ASE '07), New York, 2007, pp. 274–283, ACM.

[8] G.Myles and C. Collberg, "Detecting software theft via whole program path birthmarks," in Proc. Inf. Security 7th Int. Conf. (ISC 2004), Palo Alto, CA, Sep. 27–29, 2004, pp. 404–415.

[9] C. Solnon, "AllDifferent-based Filtering for Subgraph Isomorphism," Universite de Lyon 1, CNRS, LIRIS, UMR5205 CNRS, F-69622, France.

[10] directedLAD version2- [Online]. Available: http://liris.cnrs.fr/csolnon/LAD.html [referred on 21 November 2013]

[11] graphviz 2.34.0-1 [Online]. Available: http://www.graphviz.org/ [referred on 19 November 2013]

[12] Chrome browser [Online]. Available: https://www.google.com/intl/en/chrome/browser/ [referred on 2 September 2013]