Software Theft Detection for JavaScript Programs Based on Dynamic Birthmark Extracted from Runtime Heap Graph

Somanath Janardan Salunkhe[1], Umesh Laxman Kulkarni[2]

[1]PG Student, Department of Computer Science Engineering, DYPCET, Shivaji University, Kolhapur, India

[2]Professor, Department of Computer Engineering, Mumbai University, Mumbai, India

Abstract: ***Software's, programs are valuable assets to developer companies. However, the source code of programs can be theft and JavaScript programs whose code is easily available which is a serious threat to the industry. There are techniques like watermarking which prove the ownership of the program but it can be defaced and encryption which changes source code but it may decrypted and also it cannot avoid the source code being copied. In this paper, we use a new technique, software birthmark, to help detect code theft of software or program. A birthmark is a unique characteristic of a program which is used to identify the program. We extract the birthmark of software from the run-time heap by using frequent sub graph mining and search the same in suspected program.***

Keywords: Heap graph, software birthmark, frequent sub graph mining.

1. Introduction

Software theft is a serious issue in the industry. Software theft detection is very important for software industry. The watermarking is a solution used to prove ownership. Another approach is encryption in which a source code is transformed insuch way that it becomes more difficult to understand. A new technique for software theft detection is software birthmark which does not insert any code to the software and it depends on the characteristics of the programs. There are two types of software birthmarks, static birthmarks and dynamic birthmarks. Static birthmarks are extracted from the syntactic structure of a program.Dynamic birthmarks are extracted from the dynamic behavior of a program at run-time.We use dynamic birthmark approach for software theft detection.

A birthmark is used to identify software theft, to detect software theft; the dynamic birthmark of the original program is first extracted by using frequent sub graph mining. The same software birthmark will be search in suspected program. If the birthmark is found, the suspected program is a copy of the original program.

2. Related Work

1) **A. Monden, H. Iida, K. I.Matsumoto, K. Inoue, and K. Torii** have proposed Watermarking based software theft detection method for java programs.
2) **C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Step** have proposed Dynamic path-based software watermarking for software theft detection.
3) **X. Wang,Y.-C. Jhi, S. Zhu, and P. Liu** have proposed dynamic behavior based software theft detection
4) **G.Myles and C. Collberg** have detected software theft via whole program path birthmarks.
5) **D. Schuler, V. Dallmeier, and C. Lindig** have shown robust dynamic birthmark for java program software theft detection.
6) **H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. I. Matsumoto** have designed and evaluated dynamic software birthmarks based on API Calls.
7) **P. Chan, L. Hui, and S. Yiu** have proposed dynamic JavaScript birthmark based on the run-time heap(JsBirth).
8) **P. P. F. Chan, L. C. K. Hui, and S.M. Yiu** have used heap memory analysis for dynamic software birthmark formation and comparison for java programs.

3. Proposed Algorithm

Based on above discussion proposed work is to implement and analyzebirthmark based software theft detecting system for JavaScript programs. Specifically the proposed work is stated below.

1. Design and implementation of the graph generator and filter-The JavaScript heap profiler will run a JavaScript program and takes multiple heap snapshots in the course of its execution.The graph generator and filter will traverse the objects in the heap snapshots and builds heap graphs out of them. Heap Graph generates for each heap snapshot. Heap snapshot consist arrangement of objects which are created runtime. Heap graph contains various types of objects and edges. Graph filter is used to filter unwanted objects and edges. Object of type OBJECT and CLOSURE are considered and remaining types of objects are discarded. References of type ELEMENT and PROPERTY are only included other are discarded.

2. Design and implementation of the graph merger –The graph merger will merge the filtered heap graphs together to form one single graph. According to object id all heap graphs are combined together. We merge all the graphs one by one by taking the union set of the nodes and edges of the two graphs being merged. In order to make the resulting superimposition graph also connected, it need to ensure that there is at least one object in common (with the same object ID) in two graphs before superimposing them.

3. Design and implementation of modified Graph Selector- The subgraph selector will select a subgraph from the heap graph to form the birthmark of the originalprogram.Themodified Graph Selector will use frequent subgraph mining to get the frequent subgraph that appears in all the heap graphs that are extracted from the program by graph generator and filter. The proposed modified Graph Selector will be tested for its suitableness in using it as the more representative birthmark of the program.

4. Design and implementation of the detector –The detector will search for the birthmark of the original program in the heap graph of the suspected program.

5. Application and analysis of above mentioned scheme for its usefulness, robustness and accuracy in protecting the intellectual property rights of JavaScript developers.

4. Simulation Results

Module 1:

1) Graph generator and filter

Algorithm:

Input: JavaScript heap profile from JavaScript heap profiler

Output: A set of filtered heap graphs captured at different points of time with the annotates nodes in the heap by its object ID.

Graph generator:

For each snapshot taken using the Chromium browser, perform a depth first search traversal of it and print out the heap graph with annotated nodes by its object ID (objects) and edges (reference) and then pass it to a filter.

Graph filter:

Graph Filter working is as follows.

For each JavaScript Objects we do following:

1) If Object is of type **INTERNAL** or **ARRAY** or **STRING** or **CODE** discard the object.
2) If Object is of type **OBJECT or CLOSURE** include only *ELEMENT* and *PROPERTY* references and
3) Filter out every other reference.

Module 2:

2) Graph merger:

Algorithm:

Input: Multiple labelled connected heap graphs G from Graph generator and filter

Output: A superimposition of multiple labelled connected heap graphs G into a one single graph M that includes all the nodes and edges appearing in the input heap graphs.

The above algorithm superimposes all the graphs one by one by taking the union set of the nodes and edges of the two graphs being merged. In order to make the resulting superimposition graph also connected, it need to ensure that there is at least one object in common (with the same object ID) in two graphs before superimposing them.

3) Subgraph Selector

Algorithm:

Input: A set of filtered heap graphs captured at different points of time with the annotated nodes in the heap by its object ID (obtained from module 2- graph generator and filter).

Output: the frequent subgraphs to be used as the birthmark

The modified Graph Selector uses frequent subgraph mining to get the frequent subgraph that appears in all the heap graphs that are extracted from the program by graph generator and filter. For frequent subgraph mining gSpan algorithm is used which is complete frequent subgraph mining algorithm on labelled graphs. gSpan builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the depth first search strategy to mine frequent connected subgraphs efficiently.

This module uses java library ParSeMiS (Parallel and Sequential Graph Mining Suite). The library searches for frequent, interesting substructures in graph databases. It has implementation of gSpan algorithm for frequent subgraph mining.

5. Experimental Result

We proposed five modules in this paper,three modules are implemented and result is described in following section. Subgraph detector and analysis are not implemented. The results of first three modules are mentioned

We took first heap snapshot of www.google.com and passed it to the Graph Generator. It creates Heap graph by considering all types of object and all types of edges the result of first heap snapshot is as follows

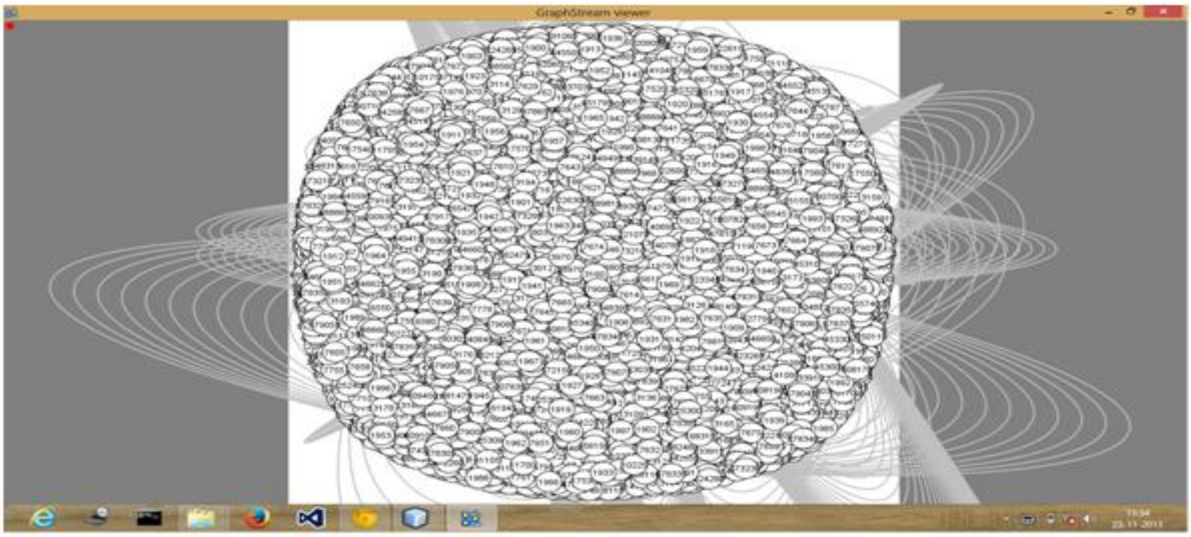

Figure 1: Results of Graph Generated by first module from sample firstheapdump

We took second heap snapshot of www.google.com and passed it to the Graph Generator. It creates Heap graph by considering all types of object and all types of edges the result of second heap snapshot is as follows

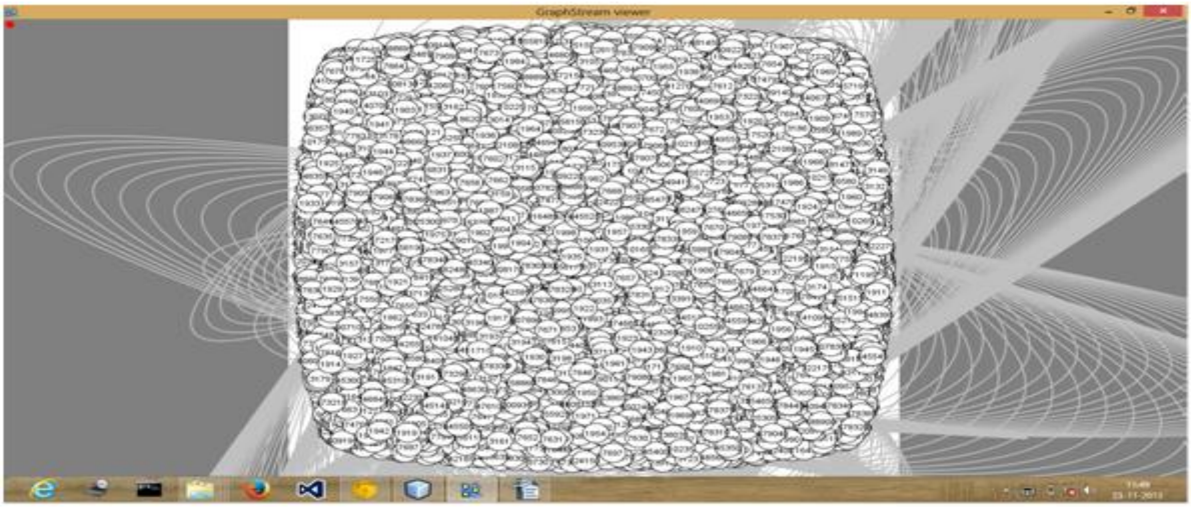

Figure 2: Results of Graph Generated by first module from sample secondheapdump

First heap snapshot of www.google.com passed to the Graph Generator which generates heap graph as shown in figure 1.Then resulting graph passed as input to the Graph Filter which only consider Object of type OBJECT and CLOSURE and remaining types of objects are discarded. References of type ELEMENT and PROPERTY are only included other are discarded. The resultant filtered graph is as follows

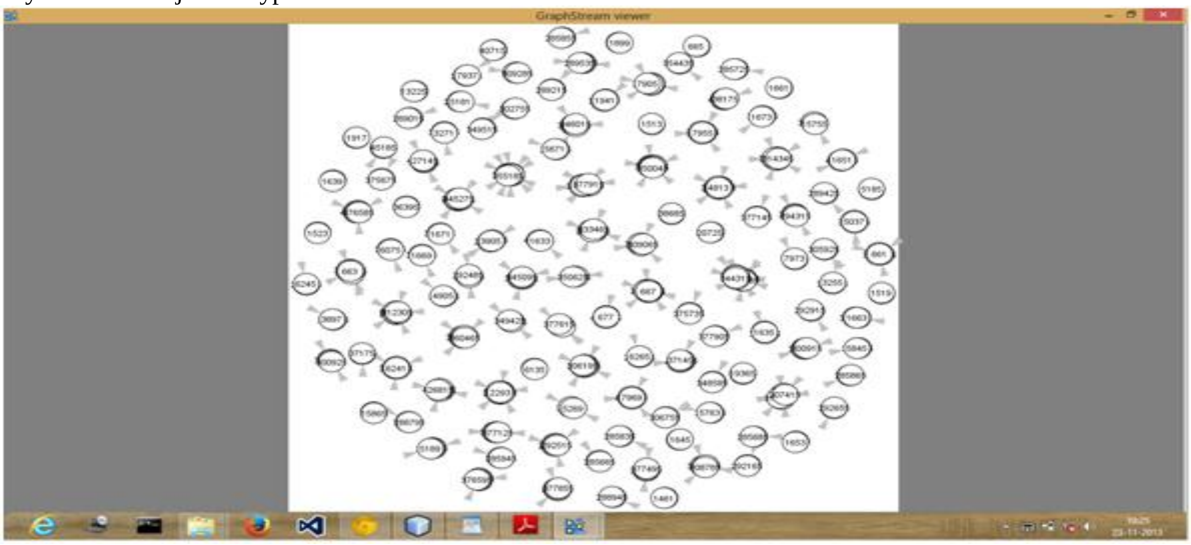

Figure 3: Results of Graph filtered of graph from fig.1 by second module

Second heap snapshot of www.google.com passed to the Graph Generator which generates heap graph as shown in figure 2.Then resulting graph passed as input to the Graph Filter which only consider Object of type OBJECT and CLOSURE and remaining types of objects are discarded. References of type ELEMENT and PROPERTY are only included other are discarded. The resultant filtered graph is as follows

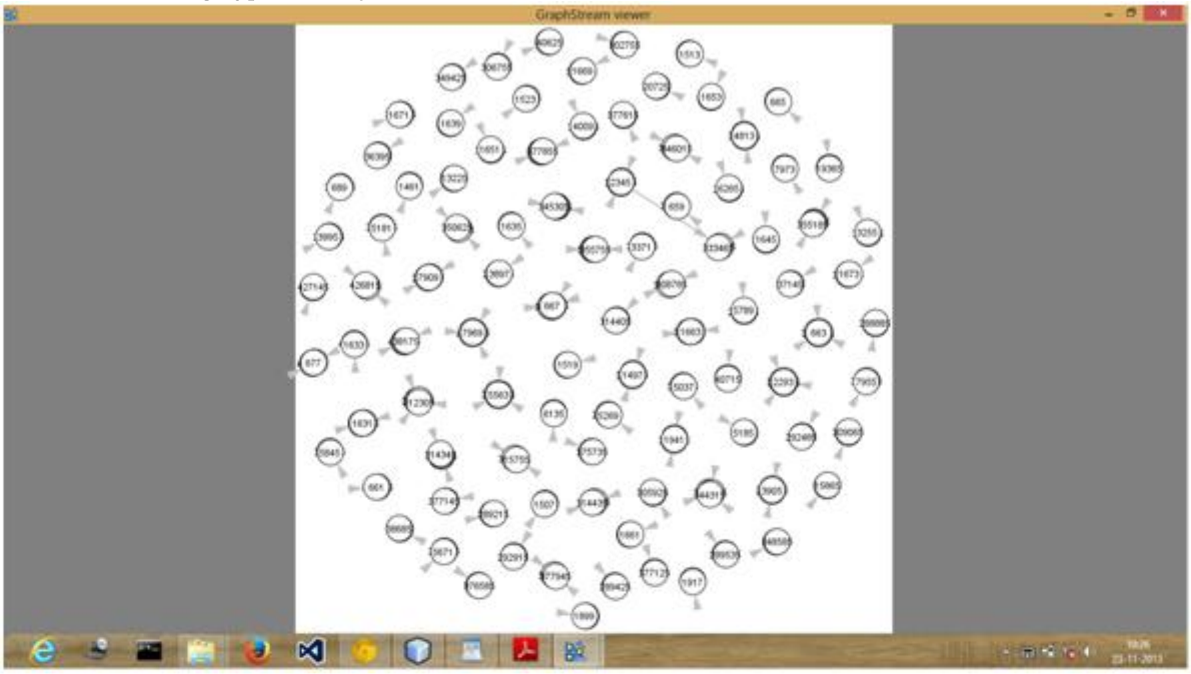

Figure 4: Results of Graph filtered of graph from fig.2 by second module.

In the heap graph all objects are assigned a unique id. Two snapshots which are shown in figure 3 and figure 4 are passed to the Graph Merger which produces single graph and merging done on the basis of object id. The resultant single graph is s follows

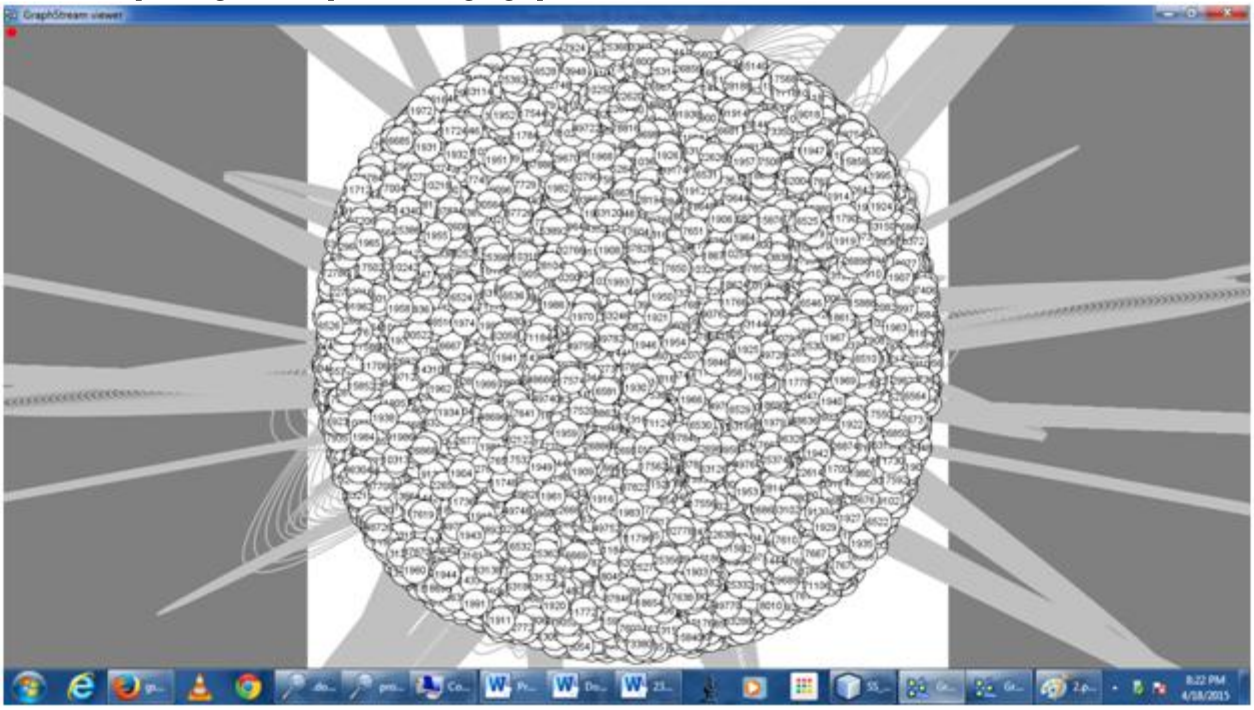

Figure 5: Results of merging of two Graphs from fig.3 and fig 4

Frequent sub graph Mining is applied on single graph generated by Graph Merger as shown in figure 5 which searches subgraph with maximum occurrence in the largest graph. The resultant subgraph of graph displayed in figure 5 is as follows

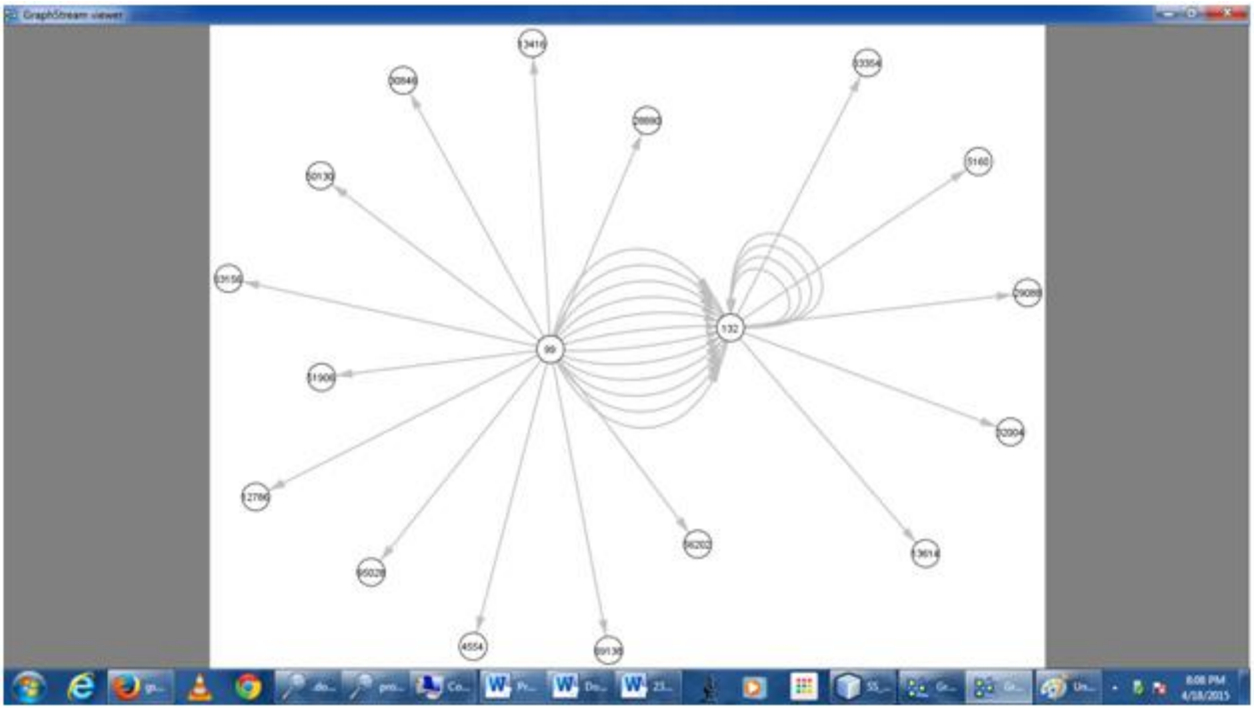

Figure 6: Subgraph selected by subgraph selector module as a software birthmark

6. Conclusion and Future Work

we proposed a robust heap graph based software birthmark system for JavaScript programs. We evaluated our birthmark system using 50 large-scale websites which gives us birthmark through frequent subgraph mining. We will implement modified subgraph detector which will search birthmark of original program into suspected program

References

[1] 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.
[2] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp, "Dynamic path-based software watermarking," in Proc. ACM SIGPLAN 2004 Conf. Programming Language Design and Implementation (PLDI '04), New York, 2004, pp. 107–118, ACM.
[3] X. Wang,Y.-C. Jhi, S. Zhu, and P. Liu, "Behavior based software theft detection," in Proc. 16th ACM Conf. Comput.andCommun. Security (CCS '09), New York, 2009, pp. 280–290, ACM.
[4] 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.
[5] 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.
[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] P. Chan, L. Hui, and S. 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.
[7] P. P. F. Chan, L. C. K. Hui, and S.M. Yiu, "Dynamic software birthmark for java based on heap memory analysis," in Proc. 12th IFIP TC 6/TC 11 Int. Conf. Commun. and Multimedia Security (CMS'11), Berlin, Heidelberg, 2011, pp. 94–106, Springer-Verlag.
[8] Yan, X.,Jiawei Han, "gSpan: graph-based substructure pattern mining," in 2002 IEEE International Conference on Data Mining, 2002. ICDM 2003.