JavaScript Theft Detection using Birthmark and Subgraph Isomorphism

Snehal N. Nayakoji, PG student, Dept. of Computer Science & Engineering, Walchand College of Engineering, Sangli, India
S. P. Sonavane, Associate professor, Dept. of Computer Science & Engineering, Walchand College of Engineering, Sangli, India

Abstract
In the web community, JavaScript has become one of the widely used languages. The facility of “view source code”, provided by many browsers, made easy to copy the JavaScript source code of any webpage. It leads to a serious threat of IPR law infringement. This problem of JavaScript theft can be addressed effectively with the help of novel scheme of deriving a birthmark which is the signature drawn from the program. As software birthmark represents unique characteristics of the program, this can be used to detect the theft of a JavaScript program. This paper proposes a method to derive birthmark from a JavaScript program and method to compare two birthmarks to detect the copy of the original program or portion of it in the suspicious program. The paper also gives a comparison between node.js and chromium browser with respect to the heap snapshot functionality.

Keywords: Software birthmark, heap graph, sub-graph isomorphism, IPR, theft detection

Introduction
According to ninth annual BSA and IDC global software piracy study in 2011, 57% of the world’s personal computer users admit that they pirate software [1]. It is very common for large companies to make hundreds or thousands of illegal copies. It leads to infringement of intellectual property rights (IPR) of software developers.

One can see, JavaScript is becoming more popular as a programming language in the community of web developers. Many of the browsers provide “view source code” function, through which one can easily obtain copies of source code. Hence, to protect IPR of software developers, it is vital to detect the theft or copy of a source code. Right from the beginning of the software era, several techniques are being used to protect the software. Out of them, watermarking and code obfuscation were used extensively.

1) Software watermark: Hiding a trademark or identification for the purpose of determining ownership is called as watermarking [2], [3]. It is the oldest and popular approach to detect the piracy of the software, where extra code is inserted into the program code for claiming the possession of the program. The hidden watermark can be recognized or extracted, at a later date, by the use of a recognizer or extractor to prove ownership of the stolen software. However, watermarking requires extra overhead of embedding watermark into the program prior to its release and it has been proved that watermarks can easily be defaced by an assailant.

2) Code obfuscation: Software obfuscation is a way of protecting software from unauthorized modification [4]. It translates a program into another one which is semantically equivalent, but is hard for attackers to understand and analyze, although obfuscated code is difficult to understand by humans, it can be copied and used.

To overcome these problems of watermarking and code obfuscation, a relatively new technique, called as software birthmark has been introduced. It is used to prove the originality of the software or the program [5-7]. Software birthmark depends solely on the intrinsic characteristics of a program. This paper proposes a software birthmark which is based on objects and references produced by the JavaScript program under execution. A heap graph represents a software birthmark of a program. By using subgraph isomorphism and software birthmark together, theft of a JavaScript program can be detected. For theft detection purpose, the software birthmark of the original program is extracted 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.

Definitions
The definitions of software birthmark and its types are given in this section [8].

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 the program are extracted from a program source code only, it is referred as static birthmark. Let p, q be two programs or program components and f(p) be a 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:

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

2) **Dynamic birthmark**: When unique characteristics of the program are extracted from a program in execution, it is referred as dynamic birthmark. Let p, q be two programs or program components, I be an input to p and q and \( f(p; I) \) be a 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:

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

A. **Graph isomorphism**

Two graphs are said to be isomorphic (equivalent) if they have identical behavior 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.

![Graph](image)

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

1) **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. 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 \( f : n \rightarrow N \) such that

\[
(u, v) \in e \iff (f(u), f(v)) \in E.
\]

The subgraph isomorphism problem is; deciding whether the pattern graph exists in target graph. Pattern graph may be a subgraph of target graph [11].

B. **Software birthmark based on heap graph**

The graph generated from the JavaScript program’s objects and its references, collected in heap memory, is called as the heap graph. The heap graph is denoted as HG. Let two programs or program components be p and q. Let I be the input to p and q. HGp and HGq be the heap graphs of the program runs with input I for p, q respectively. A subgraph of HGp is denoted by HGBp[5]. HGBp is also called as the pattern graph and HGq is called as target graph. By searching pattern graph in target graph and if multiple patterns are matched then it is possible to detect the theft of source code.

**Literature Review**

“A dynamic birthmark for java”, proposes API birthmark which is based on how a program interacts with objects from Java Standard API at runtime [10]. 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.

“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 [11].

A system call dependence graph based software (SCDG) birthmark reflects unique behavioral characteristics of a program [7]. 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 behavior characteristic of the program. A SCDG birthmark may not work if the program does not involve any system calls or has very few system calls. It is also not applicable to the programs which do not have unique system call behaviors.

Tamada et al. [8] proposed two kinds of dynamic software birthmarks based on API calls. Their approach was based on the capacity to understand the hidden truths that it was difficult for opponent to alter the API calls with other equivalent ones and that the compiler did not make the effective use of the APIs themselves. Runtime information of API calls was used as a strong signature of the program. Tamada et al. proposed two types of birthmarks: EXESEQ birthmark and EXEFREQ birthmark. The proposed birthmarks have quite strong tolerance against the automatic program optimization. This promising result motivated the researches to work on dynamic birthmarks based on API calls.
Threat Model

Fig 2: Threat model

This section highlights the situation where birthmark system can be used to solve the conflict of code theft.

1) A is the owner of program P.
2) B is another developer, who wants to develop a program Q, which will work similarly as program P works.
3) B copies the part of the program P.
4) Later A comes to know that program Q, developed by B works similar to his own program P.
5) Dynamic birthmark system can help A in following manner:
   a) Run program P and get its heap based birthmark, HG(P). Select subgraph of HG(P), HGB(P).
   b) Run program Q and get its heap based birthmark, HG(Q).
   c) Search birthmark of program P, HGB(P) against the birthmark of program Q, HG(Q). If found, theft will be detected.

Methodology

Fig 3 represents the outline of the proposed work to detect the code theft.

The following steps indicate the methodology of the proposed work:

a) Get heap snapshots of running JavaScript program from chrome browser.
   b) Filter the objects from the snapshot to get required objects only.
   c) Create graph using available objects and references.
   d) Select a subgraph for original program only (subgraph for suspected program is not needed).
   e) Search subgraph of original program in graph of suspected program.
   If able to find the subgraph, then theft will be detected.

Fig 3: work diagram

Steps 1 to 5 are carried out by Birthmark generator and 5th step is done by Birthmark comparison.

Figure 4 gives the overview of birthmark generator. Steps are carried out using:

Step 1 - JavaScript heap profiler.
Step 2 and 3- Graph generator and filter.
Step 4- Subgraph selector.

If multiple heapsnapshots are taken for one single program then graph merger is used to merge all graphs and build a single graph.

Fig 4: Birthmark generator
• **Birthmark generator**: It summarizes the processes that are required to detect the theft of JavaScript. JavaScript program is given as an input to a **JavaScript heap profiler** and heap snapshots are taken in the course of its execution. The **graph generator and filter** then filters out the inefficacious objects, references and builds the graph out of it. The **graph merger** merges all the heap graphs to form one single graph. The **subgraph selector** can then selects a subgraph which will act as a birthmark for that program.

• **Birthmark comparison**: Two birthmarks, one from original program and other from suspicious program, are given to the birthmark comparison module. Subgraph isomorphism is the technique used for this purpose. To detect pattern graph in target graph, directedLAD version2 library is used [13]. By using this library one can search birthmark of original program into the suspected program’s graph for detection of theft.

**Experimental Setup And Results**

Heap snapshots of a JavaScript program are taken with the help of **node.js** [14] and **chromium browser** [15]. The heap snapshots obtained are then analyzed and are filtered to get the useful objects and references. With the help of these useful objects and references heap graphs are constructed which will act as a birthmark for that particular JavaScript program.

1) **Comparison between node.js and chromium browser**: Figure 5 and 6 represents the comparison between node.js and chromium browser with respect to the number of objects and references generated. The X axis in the graph denotes the name of the test sample program and Y axis denotes number of objects and references in node comparison and edge comparison respectively.

<table>
<thead>
<tr>
<th>Name of program</th>
<th>Node.js</th>
<th>Chromium browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison between node.js and chromium browser concludes that the number of objects and references generated in node.js are less than generated in chromium browser. The reason behind this is, in chromium browser there exist objects and references which are generated by the browser for housekeeping functions. These objects and references are not generated in node.js.

The obtained snapshot is then forwarded to the graph generator and filter where it filters out the objects and references which are not required for processing purpose. The output of graph generator and filter is a object graph which is given to the graph merger if there exist multiple object graphs of single program. Then object graph is given to subgraph selector. Subgraph selection is applied to plaintiff program only, as part of plaintiff program is searched against whole suspicious program.

2) **Subgraph isomorphism results**:

Table 1 gives the subgraph isomorphism results where each pattern graph (software birthmark of original program) is searched against target graph (software birthmark of suspected program) and if found; theft will be detected. The target graph used for this purpose is shown in figure 1 (a). Knowing the actual fact which is the ground truth, the number of matches of pattern graph to a target graph, which is called as “frequency” is calculated. For the chosen sample which is small in size with ten nodes target graph, the frequency observed is 60 or above, if theft case is detected, then results 100% accuracy in the theft detection results. However, if the number of nodes in target graph gets increased; another threshold which is the frequency of matching can be set and results can be checked accordingly.
Table 1: Subgraph Isomorphism Results

<table>
<thead>
<tr>
<th>Pattern graph</th>
<th>Ground truth</th>
<th>Frequency of matching</th>
<th>Theft detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern1</td>
<td>YES</td>
<td>60</td>
<td>YES</td>
</tr>
<tr>
<td>pattern2</td>
<td>YES</td>
<td>120</td>
<td>YES</td>
</tr>
<tr>
<td>pattern3</td>
<td>NO</td>
<td>0</td>
<td>NO</td>
</tr>
<tr>
<td>pattern4</td>
<td>NO</td>
<td>0</td>
<td>NO</td>
</tr>
<tr>
<td>pattern5</td>
<td>NO</td>
<td>0</td>
<td>NO</td>
</tr>
<tr>
<td>pattern6</td>
<td>YES</td>
<td>120</td>
<td>YES</td>
</tr>
<tr>
<td>pattern7</td>
<td>YES</td>
<td>120</td>
<td>YES</td>
</tr>
<tr>
<td>pattern8</td>
<td>YES</td>
<td>120</td>
<td>YES</td>
</tr>
<tr>
<td>pattern9</td>
<td>YES</td>
<td>240</td>
<td>YES</td>
</tr>
<tr>
<td>pattern10</td>
<td>YES</td>
<td>120</td>
<td>YES</td>
</tr>
</tbody>
</table>

7. REMARK

As JavaScript theft has become easier for anyone in the web community, intellectual property rights of web application developers are at risk. To protect IPR laws of developers software birthmark proves to be an efficient idea. Software birthmark system helps to detect the copy or theft of a code. The paper gives the idea of using sub-graph isomorphism and software birthmark together to detect the software piracy. It also gives the comparison between node.js and chromium browser to analyze objects and references generated during JavaScript execution. Comparison indicates node.js is better to use for taking heap snapshots of a JavaScript program.

As the issue of copy theft is under discussion, it also aware programmers and professionals for registering their IPR and software development.

References


