American Journal of Engineering Research (AJER)
e-ISSN : 2320-0847 p-ISSN : 2320-0936
Volume-03, Issue-03, pp-181-187
www.ajer.org

Research Paper Open Access

Software Theft Detection Using Birthmark Alg.

Dheeraj Kumar[1], Ms. Prince Marry[2]
[1]*PG Scholar (M.E. /CSE),* [2]*Faculty of computer science*
Department of Computer Science Sathyabhama University, Chennai, Tamil Nadu

Abstract: - Due to rapid development 0f internet technologies java script has become an important part of the software development process. Though the copy of java script program can be easily done and poses a serious threat to the intellectual property theft. In order to help protect and prove the ownership of the source code there are two technique such as software watermarking and software code obfuscation technique are being used. However watermarking needs extra effort to implement to prove ownership of the source code and it has been found that it can be defaced by a deliberate attempt, whereas code obfuscation makes the code hard to be understand by a human but does not prevent it from being copied. In this paper our attempt is to implement a considerably new technique called software birthmark to which takes the security of codes to another level. Birthmark technique is defined as the intrinsic characteristic of a source code and does not require any additional effort to implement. In order to implement birthmark we extend two recent birthmark that extract the birthmark at runtime.

Keywords: - *code theft detection, software birthmark, Software protection, Software code obfuscation.*

I. INTRODUCTION

Today I won't be wrong if i would say that we live in the era of information technology and due to the existence of web 2.0 and HTML5 it is more relevant to say that everything wat we can imagine can be done with the help of virtual environment i.e, internet. It has been found under various studies being conducted that today java script is proving to be one of the most popular programming language and will continued to be so. According to a survey conducted by Evans Data in the year 2008, it has been found that over 60% of the developers uses java script and that has outraged all scripting languages use, including Java[1]. However the codes of java script programs can be obtained very easily as it is an interpreted language and almost all browsers provide handful of methods to obtain the source code of the web page. This makes it very difficult to protest intellectual property right of the software developer.

Software security has already been a topic of interest for the computer scientist an will continue to be so. In order to protect and prove ownership of the software code their is a earliest and most popular technique being used called watermarking technique. It is an well-known and easiest technique used to detect software piracy. In this approach watermark is embedded into the source program which proves the ownership just like a signature [2]-[5]. However it has been found by studies that water marking can be defeated by a fairly deliberate attempt of a hacker [4]. However it requires an extra effort from the developer to embed watermark in the software prior to the release. Due to these reasons many developers avoid using watermarking and use code obfuscation technique before the release of their product. This technique is used to obfuscate codes which makes it difficult to understand by human in other words the original code is encrypted. code obfuscation is actually a semantics-preserving transformation of source code which makes it difficult to understand and reverse engineer [5]. Though it does not prevent the pirates from copying of the code.

There is another technique available which is relatively less popular but i believe is more powerful and handy for the software theft detection. This technique is called as birthmark. The term birthmark was first used by a computer scientist Derrick Grover in 1989. He explained birthmark as the intrinsic characteristic occurring in a program by chance which could be used to aid program identification. It was first used by IBM to sue the pirates for their PC-AT ROM by showing the register push and pooped pattern. As it is clear from the definition birthmark does not require any code being added to the program it depends solely on the intrinsic

characteristic of a program to determine similarity between two programs [6]-[13]. It was shown in [7] that a birthmark could be used to identify software theft even if watermark transformation is destroyed. According to Wang et al.[6] birthmark is unique characteristic a program possesses. To identify the software theft first the birthmark of the protected code is extracted and then the suspected program is compared against the birthmark if the search is found match then the suspected program has the high possibility that it is a copy of the original program. Birthmark can be looked under two categories i.e. static and dynamic birthmark. Static birthmark can be extracted from the syntactic structure of the program [10],[12],[13]. Whereas dynamic birthmark is extracted from the dynamic behaviour of the program at run-time [6]-[9],[11]. As we know that the technique like code obfuscation semantic-preserving transformation which only modifies the syntactic structure of a program but not the dynamic behaviour of the program, hence birthmark is more efficient and robust with respect to them.

However we know that birthmark is a better technique for securing software codes but still there is certain variations found in the to improve its performance. Earlier dynamic birthmarks made use of the complete control flow trace, the API call trace, or the system call trace obtained during the execution of the program [6]-[9],[11]. Birthmark based on these factors exhibits certain flaw such as birthmark based on control flow trace may still be exposed to the obfuscate attack as loop transformation. Similarly birthmark based on API call trace may suffer may suffer from the problem of not having enough system call to make unique identification. Recently, another technique to identify birthmark on the basis of run-time heap has been proposed [14],[15]. However evaluation of these birthmark is based on small number of tiny programs. Moreover birthmark comparison algorithm proposed in [14] does not scale up well and limits the size of birthmark. The graph isomorphism algorithm used in [15]makes birthmark vulnerable to reference injection attack.

This paper proposes a redesigned heap graph based birthmark algorithm for java script as well as the design code such as HTML, and JSP to make it a scalable and robust technique to detect software theft.

II. RELATED WORKS

There are many works has already been carried out on this topic in order to help insure and improve the performance of the birthmark selected. The works related to this topic are:

A. *Dynamic Path Based Software Water [4],this paper is mainly focused on the watermarking technique to provide security to the software codes. According to this paper the watermarking is done by inserting certain bit stream into the software code to make it identifiable but this paper has also introduced about the short falls of the watermarking technique and described how it can be destroyed by deliberate attempt of the attacker. This paper introduces path-based watermarking, which is a new approach to software watermarking based on the dynamic branching behaviour of a program. Experimental results, using both Java byte code and IA-32 native code, indicate that even relatively large watermarks can be embedded into programs at reasonable cost.*

B. *Detecting Software Theft via Whole Program [7],According to this paper Every rule of the grammar is constructed by a non-terminal and a series of symbols which the non-terminal represents. To create DAG, a node is added for every unique symbol. For every rule an edge is added from the non-terminal to each of the symbols it represents. The DAG is the Whole Program path.. The WPP birthmark is built i similar manner as The WPP with the exception of the DAG in the stage. As our interest lies in regularity only hence we eliminate all terminal nodes in the DAG. It is the internal nodes which will be more difficult to modify through program transformation.*

C. *A Dynamic Birthmark for Java[8], this paper deals with the issues related with the creation of birthmark for java codes. As we know that java has been a widely accepted language used for coding by most prominent companies. same has been the reason for the concern for them as they believe code as the important resource of their organisation and consider code as a core asset. A birthmark can help them to detect code theft by identifying intrinsic properties of a program. Two programs with the same birthmark are likely to share a common origin. Birth marking works in particular for code that was not protected by tamper-resistant copyright notices that otherwise could prove ownership. We propose a dynamic birthmark for Java that observes how a program Uses objects provided by the Java Standard API. Such a birthmark is difficult to foil because it captures the observable semantics of a program. In an evaluation, our API Birthmark reliably identified XML parsers and PNG readers before and after obfuscating them with state-of-the-art obfuscation tools. These rendered existing birthmarks ineffective, such as the Whole-Program-Path Birthmark by Myles and Collberg.*

D. *Dynamic Software Birthmarks to Detect the Theft of Windows Applications [11], This paper introduces dynamic software birthmarks which can be obtained while execution of Windows applications. Birthmarks are unique and native characteristics of software. For a pair of software p and q, if q has the same birthmarks as p's, q is suspected as a copy of p. security analyse of this paper shows that the proposed birthmark possess good result against different types of program transformation attacks.*

There are many other works being carried out in order to enhance and increase the robustness and performance of the birthmark algorithm.

III. EXISTING SYSTEM

For the purpose of insuring the integrity and security of the software codes mainly there are two techniques being used such as software watermarking and code obfuscation. Where as another relatively new technique being used and has become a matter of huge research among the computer scientist is birthmark. Software birthmark algorithm is getting huge acceptance and still a lot of work is still required to be carried out to make it more robust.

Fig 2. Existing system with watermarking and code obfuscate algorithms.

The problem existed with the existing system was that there were no guarantee that the system was full proof and and requires extra effort in order to implement.

IV. PROPOSED SYSTEM

Here in this paper we propose a relatively new technique called birthmark technique. Here in our proposed system we extend two types of birthmark i.e., static and dynamic birthmark.

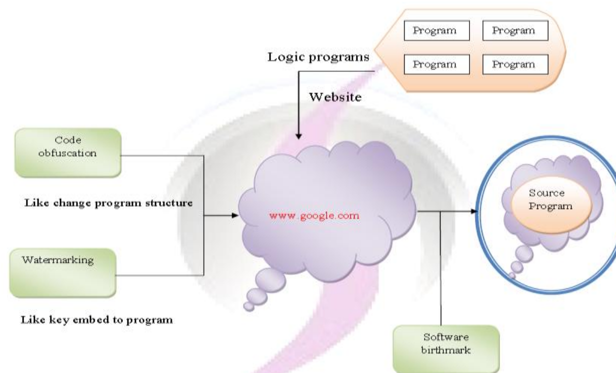

Fig 1. Proposed system with birthmark algorithm used.

A. Software birthmark:

It is a group of unique characteristics obtained from a program. software birthmark can be divided into two categories: static birthmark and dynamic birthmark.

Definition 1: Dynamic Birthmark: dynamic birthmark is defined as the unique characteristics obtained by chance by a program at runtime.

Let two a and b be two program components and let I be an input given to the programs a and b. Now let f(a, I) be the dynamic birthmark of program a if and only if the two of the conditions are satisfied:

1) f(a, I) comes out only when p is executed with input I.
2) Program a is copy of b if a=>f(a, I) = f(b,I).

B. Subgraph Monomorphism:

Definition 2: A graph monomorphism from a graph G=(N,E) to a graph G' = (N', E') is a bijective function f(N) -> N' such that (u,v) belongs to E=>(f(u),f(v)) belongs to E'.

V. V. SYSTEM DESIGN

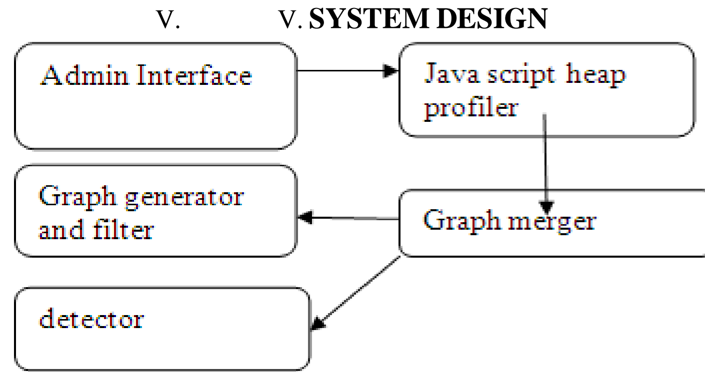

Fig 3. System design with modules.

Fig 3 shows the complete design module of our proposed system. It outlines the steps that original program and the suspected program undergoes in our system.

A. Admin Interface Design: in our proposed system admin interface module is one of the important part. It gives interface to the user and also provides primary level security by checking the authenticity of the user. Admin interface design is basically used as the normal user login authentication purpose. It also provides the admin and normal user access to the teo separate windows with the relevant capability.

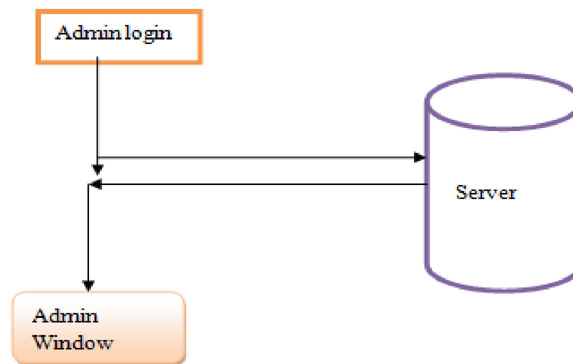

Fig. 4 Admin Interface Design.

B. ***Java Script Heap Profiler:***

Java script heap profiler in our paper is another module where the java script codes are stored. Being an interpreted language we know that java script allows for the creation of objects at anytime. And on the other shide we know that java script heap keeps changing due to the creation of object and garbage collection but abter some time it has been found that hai peap of the java script becomes constant.

Therefore to exploit full benefit of the behaviour of the heap , we try to obtainevery object that occurs in the heap.there for we use java heap profiler in our system.

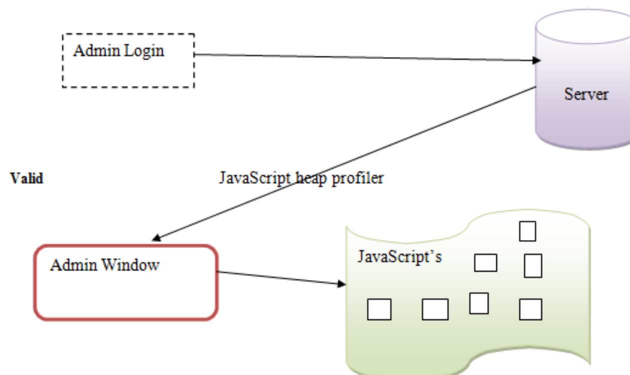

Fig 5. Java Script Heap Profiler

The fig5 shows the structure and the worke flow of the java script heap profiler module.

C. Graph Generator And Filter:

Graph generator and filter is another important module in our system which is mainly used for providing the security of the code.infact graph generator and filter is a module where code obfuscation is implemented to make the software pirates difficult to understand the code. Here symentic preserving transformation is being used which transforms the original code with the encrypted code but it does not have any effect on the runtime behaviour of the program.

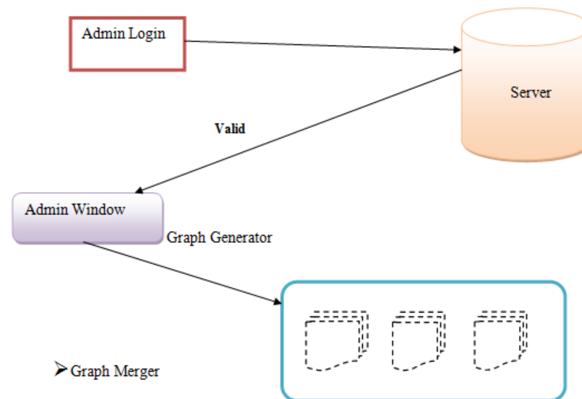

Fig6. Graph Generator And Filter.

We perform a death first search traversal of and print out the heap graph with nodes and edges that pass a filter. We describe such a filter in details as follows.

Objects in the V8 JavaScript heap are divided by six categories: *INTERNAL*, *ARRAY*, *STRING*, *OBJECT*, *CODE*, and *CLOSURE*. We need not to include in our heap graph objects which comes under *INTERNAL*, *ARRAY*, *STRING*, and *CODE* categories. Because the design decision are as follows: *INTERNAL* objects are virtual objects fand are used for housekeeping purpose which can not be accessed from the program code. *Array* objects, represent an array of elements objects. Though, our observation says that arrays are actually represented by an object of the type *OBJECT* with name "*Array*" and the references from the array are coming out from that object. Hence, *ARRAY* objects are also not included. *STRING* and *CODE* objects, we do not find any reference coming out of them. Hence, they are excluded as well. To count up, we only include *OBJECT* and *CLOSURE* objects in our heap graph.

References between objects in the V8 JavaScript heap are
divided into 4 categories: *CONTEXT_VARIABLE*, *ELEMENT*,
PROPERTY, and *INTERNAL*. We do not include references that belongs to *CONTEXT_VARIABLE* and
INTERNAL
categories. The reasons behind this design decision are as follows: *CONTEXT_VARIABLE* is a variable in a function context, accessible by its name from inside a function closure.Therefore, it is not accessible by objects outside that function and it is automatically created by V8 for housekeeping purpose. *INTERNAL* references are properties added by the JavaScript virtual machine. They are not accessible from JavaScript code. Therefore, we only include in our heap graph *ELEMENT* and *PROPERTY* references. *ELEMENT* references are regular properties with numeric indices, accessed via [] (brackets) notation and *PROPERTY* references are regular properties with names, accessed via the . (dot) operator, or via [] (brackets) notation. There are some objects created by the JavaScript engine that exist not just for one program. For example, the *HTML Document* object can be found in the heap graphs of all the JavaScript programs we studied. Therefore, we need to filter such objects out as they dilute the uniqueness of the heap graph. Basically, the filtered objects include objects created to represent the DOM tree and function closure objects for JavaScript built-in functions.

D. Graph Merger:

Graph merger is another module in our system which is basically implemented in order to show verify the result of the project. Graph merger is the module which is basically used in another application such as in the suspected program which is used to reconstruct the original code from the obfuscated code which is obtained as the output of the graph merger and filter. unique ID is assigned to every object in the JavaScript
heap by the V8 JavaScript engine. However, the ID of an object is fixed even after the multiple dumps and therefore, can be used to identify the object. The Graph Generator and Filter also uses every node in the heap graph with its object

ID. Hence, we can tell whether or not two nodes in two heap
graphs points to the same object. The graph merger takes multiple heap graphs as input and which is basically the output from graph generator and outputs a one single graph that includes all the nodes and edges appearing in the input heap graphs. The algorithm of graph merger is shown in
Algorithm 1.
Algorithm 1 Calculate single graph of a set of labeled
connected graphs
Require: connected graphs , labeling
function where and is a positive integer
Ensure: is connected and is a superimposition of
graphs in G with labeling function where
and is a positive integer
for all where **do**
if where **then**
Combine mapping and
end if
end for
In short, it merges all all the graphs one by one by
taking the union of the nodes and edges of the two graphs
being merged. In order to make the resulting single graph also connected, we need to ensure that there should be at least
one object in common (with the same object ID) in two graphs
before merging them.

E. Detector:

The detector takes the sub graph from the original program and the entire heap graph of the suspected program as inputs and checks whether the selected sub graph of the original program can be found in the heap graph of the suspected program. Similar to the the sub graph selector, it takes sub graph of the objects under the Window objects from the suspected program and uses sub graph monomorphism to check whether the sub graph of the original program is present in suspected program. Once there is a match found, the detector raises an alert and reports where the match is found.

VI. CONCLUSION

In order to discuss the performance of the work carried out in this paper we created two prototype of the java application and we checked the result on about 200 wesites and found much improved results whith almost zero false-positive output. We believe that the system proposed is best to our knowledge and can be implement to be used with effective results.

VII. FUTURE WORK

In this section we want to discuss the sections we wish we could do something better in order to improve the results.
1). *Improved Graph Selector:* Currently, we are using the largest object subgraph as the birthmark of our program. As of now we do not know if there exists a good way then this to do it. One primary idea can be of using frequent subgraph mining which gives the frequent subgraph which appears in all the heap graphs we extracted from the program. This can make the birthmark more lively and effective of the program. Though, the running time
of frequent subgraph mining on large graphs is slow and there should be some performance tuning in order for it to be practical.
2) Faster Detector: Due to the theoretical running time limit of graph monomorphism algorithm we used in our detector, we need to limit the size of the heap graphs in order to control the running time of the detector.
3) Implementation in design codes: currently in our prototype we implement birthmark on the java script codes but we believe that the design codes such as JSP,HTML are equally important.

REFERENCES

[1] E. Data, JavaScript Dominates EMEA Development Jan. 2008 [Online]. Available: http://www.evansdata.com/press/viewRelease. php?pressID=127

[2] C. Collberg and C. Thomborson, “Software watermarking: Models and dynamic embeddings,” in *Proc. Symp. Principles of Programming Languages (POPL’99)*, 1999, pp. 311–324.

[3] 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.
[4] 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.
[5] C. Collberg, C. Thomborson, and D. Low, A Taxonomy of Obfuscating Transformations Tech. Rep. 148, Jul. 1997 [Online]. Available: http://www.cs.auckland.ac.nz/~collberg/Research/Publications/ Collberg ThomborsonLow97a/index.html
[6] X. Wang,Y.-C. Jhi, S. Zhu, and P. Liu, “Behavior based software theft detection,” in *Proc. 16th ACM Conf. Comput. and Commun. Security (CCS ’09)*, New York, 2009, pp. 280–290, ACM.
[7] 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.
[8] 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.
[9] 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.
[10] H. Tamada, M. Nakamura, and A. Monden, “Design and evaluation of birthmarks for detecting theft of java programs,” in *Proc. IASTED Int. Conf. Software Eng.*, 2004, pp. 569–575.
[11] H. Tamada, K. Okamoto, M. Nakamura, and A. Monden, “Dynamic software birthmarks to detect the theft of windows applications,” in *Proc. Int. Symp. Future Software Technol.*, Xian, China, 2004.
[12] G. Myles and C. Collberg, “ -gram based software birthmarks,” in *Proc. 2005 ACM Symp. Appl. Computing (SAC ’05)*, New York, 2005, pp. 314–318, ACM.
[13] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. I. Matsumoto, Detecting the Theft of Programs using Birthmarks, Graduate School of Information Science, Nara Institute of Science and Technology, Tech. Rep., 2003.
[14] 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.
[15] 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.
[16] P. C. Team, Prototype JavaScript Framework [Online]. Available: http://prototypejs.org/
[17] V. Proietti, Mootools JavaScript Framework [Online]. Available: http://mootools.net/
[18] Google Chromium Project [Online]. Available: http://code.google. com/chromium/
[19] Google v8 JavaScript Engine [Online]. Available: http://code.google.com/p/v8/
[20] VFLib [Online]. Available: http://www-masu.ist.osaka-u.ac.jp/~kakugawa/ VFlib/
[21] Content Scripts [Online]. Available: http://code.google.com/chrome/ extensions/content_scripts.html
[22] Jasob 3 [Online]. Available: http://www.jasob.com/
[23] L. Cordella, P. Foggia, C. Sansone, and M. Vento, “Performance evaluation of the vf graph matching algorithm,” in *Proc. Int. Conf. Image Anal. and Processing*, 1999, pp. 1172–1177.
[24] M. Sosonkin, G. Naumovich, and N. Memon, “Obfuscation of design intent in object-oriented applications,” in *Proc. 3rd ACM Workshop on Digital Rights Manage. (DRM ’03)*, New York, 2003, pp. 142–153, ACM.