Journal of Information Technology and Engineering
Vol. 3 No. 2 (December, 2018)

A Hybrid Approach for Mobile Agent Security using Reversible Watermarking

ABID KHAN[1], Xia-Mu NIU[2] and Yong ZHANG[3]

[1]Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, China
E-mail:abidkhan_hit@yahoo.com

[2]Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, China
E-mail: xiamu.niu@hit.edu.cn

[3]Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, China
E-mail: zhangyong@hitsz.edu.cn

Received: 15th April 2017 Revised: 10th May 2017 Accepted: 17th January 2018

Abstract: *Reversible Watermarking has recently drawn a lot of attention for content authentication. An additional advantage of this technique is that one can always get the original content after the authentication of watermark. At present reversible watermarking is being used only for protecting digital media like images, audio and video. Protecting Mobile Agents from malicious hosts have been investigated for some time now. In this paper we present a framework for protecting mobile agent's results from tampering and also suggest a way to get back the original values of results. Our framework uses both reversible watermarking and dynamic graph based software watermarking in a hybrid way.*

Key words: *Mobile agent Security, Reversible watermarking, Malicious hosts, Dynamic graph based watermarking.*

1. INTRODUCTION

In the past few years, we have seen some tremendous changes in distributed and client-server computing. Earlier software applications were limited only to a few nodes in computer networks. But with the advent of Mobile agents [1] this reality is likely to change. A Mobile agent (MA) can be simply defined as a program that can act in a computer network on behalf of a user or an application. MA has already been employed in a variety of applications with great effect such as information retrieval, workflow management systems, e-commerce applications and Network Management [2, 3, 4, 5, 6]. Despite all these promises there are some issues which needs to be address properly by research community before mobile agents can be used widely. The security being the most serious issue among others needs a lot of attention. Security in mobile agents System (MAS) can be divided into two major types. The first type of security is the security of the platform from malicious mobile agents and the second one is the security of the mobile agent from a malicious platform. The second type is our focus here in this paper.

This paper introduces the idea of using reversible watermarking and graph based software watermarking for mobile agent security. The digital content in our case is the results carried by mobile agents while visiting a number of hosts in its itinerary. Reversible Watermarking is used to watermark the results. The watermarked results are then represented by a dynamic graph structure. The advantage of our approach is twofold. Firstly, by using reversible watermarking we can restore the original content after authentication. Secondly, by using software graph based watermarking it is very difficult for an attacker to get some useful information without a lot of effort. We will be using a dynamic heap allocated data structure to represent the watermark results.

The remaining of the paper is organized as follows: In Section 2 we discuss the Notion of Malicious host and enumerate some existing approaches to solve the problem. In Section 3 we discuss the idea of Graph based Reversible watermarking and give an example of potential application scenario where this approach is being implemented. Section 4 discusses some attacks against our proposed idea. Conclusion is given in Section 5.

2. MALICIOUS HOST AND EXISTING APPROACHES

Mobile agent execution platform is responsible for providing the necessary environment and resources in order to successfully execute a mobile agent. Thus the proper goal and function of a mobile agent can not be achieved without cooperation from its platform. The manipulation attacks performed by a malicious host are difficult to detect and are considerably expensive [16]. Even if some attacks are detected it is not possible to get the original content (data). In [18] the idea of a time limited black box security was presented using obfuscation as a key technique. The problem with this technique is that how long or how short should be that protection time interval. In [11, 15] the idea of encrypted computing was given as a way to execute mobile code with integrity and privacy. However there are some limitations as there are few functions that can be executed in an encrypted way. In [13] the idea of cryptographic traces for mobile agents was presented based on the execution tracing and cryptography. It allows the "detection of attacks against the code, state and execution flow for mobile agents". These facts can be used to punish the attackers using a Trusted Third Party (TTP) [12]. However it has some limitation and cost as well. In [14] the idea of using reference states to protect mobile agent was presented. "A reference state is a state of mobile agent that is produced by non-attacking hosts/ reference host". The idea of using watermarking as a protection mechanism was presented by [16,17]. This method attempts to detect manipulation attacks performed during agent's execution. The agent's execution creates marked results. When the agent returns to the origin host, these results are examined in order to locate the watermark. If the mark has been changed this proves a manipulation of results and malicious behavior of the host. The problem with this approach is that a trusted third party is needed in order to punish the malicious hosts. Colluding attacks are shielded by using fingerprinting where a different watermark is embedded instead of a same watermark for each host. In [7] the idea of software watermarking was given. As compared to [7] we have used reversible watermarking so the advantage is twofold. Firstly we are using graph to represent the result so because of the aliasing effect it is very difficult to understand the graph representation. Secondly we are using reversible watermarking so the original content/results can be retrieved after the authentication of the watermark.

3. GRAPH BASED REVERSIBLE WATERMARKING

The basic idea of the graph based reversible watermarking is to watermark the results that a mobile agent gathers from a host. The watermarked results are represented by a graph, using java object serialization. This graph can travel along with mobile agent as it traverses different nodes in its itinerary. The graph is a heap allocated data structure that is built at runtime of the mobile agent. This heap allocated data structure can be a PPCT or a list as described by [7]. Beside that we can use any dynamic data structure like graph/list for encoding and decoding. When a mobile agent returns to home platform it delivers the necessary information to it and from this information the graph can be deserialized. Now we can get the watermarked results from the graph. We can finally get the original results by extracting the watermark from the results represented by the graph after the watermarks are found to be authentic. Because of the undecidability of pointer aliasing [8,9] it is difficult to analyze the code that builds the graph structure. So there are two lines of defense for any attacker to break. First is the difficulty of analyzing the heap allocated data structure and the second is the reversible watermarked data. Since we are watermarking the data not the code it is easy to add tamper proofing to the code segment that embed the watermark and builds up the graph [7].

4. ALGORITHM DESCRIPTIONS

In order to implement the idea we have selected a typical Airline Ticket Booking scenario as shown in Fig.1. We have used IBM Aglet [10] for implementation.

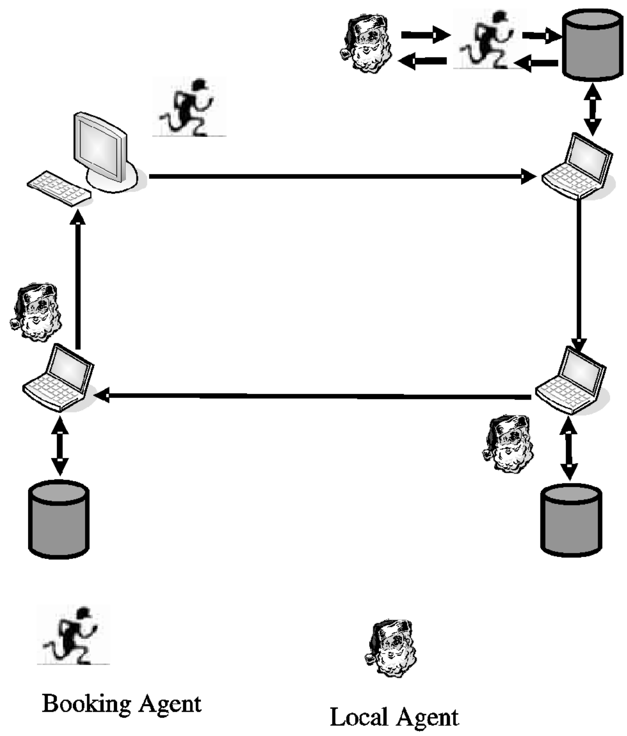

Figure 1: Airline Ticket Booking System.

It's basically an information system consisting of various mobile agents communicating with each other to achieve their designated tasks. In the following discussion we will use the word agent and aglet interchangeably. Every Airline has its own local representative agent. The user agent which we call Booking Agent (BA) is provided with necessary information before being dispatched. Mobile Agent visits various hosts in its itinerary before returning to its home platform. Mobile agent is supposed to give the result of its computation to the home platform at the end. The information provided by the user to this BA includes its itinerary plan (The number of hosts and the order in which they will be traversed by BA), a user query to be executed on a remote airline host. The query specifies the information which includes departure city, destination city, and the date etc about the ticket that a user want to reserve. Now the booking agent is given this information (in the form of an SQL query) and its itinerary plan before being dispatched. Each airline has its own database which includes information about various flights schedules. In addition to that each airline host is responsible for providing the necessary resources to the incoming agents from remote hosts. The resources include the Agent's execution environment, database connectivity etc. The airline host communicates with the booking agent via a Local Agent (LA). This LA is responsible for providing the necessary information to the BA. In other words the LA is basically representing the airline in this multi-agent information system.

The information provided by LA in our scenario is the Data Source Name (DSN) and the name of the driver (Oracle/SQL/MYSQL) used by the airline hosts. This information is necessary for the booking agent to successfully connect itself to the database. After establishing connection successfully with the database, the booking agent (BA) can communicate directly without exposing the query to any other entity. A similar implementation scenario can be found also in [4].After getting results from a host airline database the mobile agent does the following things before dispatching itself to the next host in its itinerary. First of all it uses wavelets based reversible watermarking [23] to embed a watermark in the results. Secondly after the results are being watermarked they are represented by a dynamic data structure in the form of a list. In order to get the results all we only need is the head of the list only. By serializing the head of every list for each host we can get the result of each host. When the mobile agent returns to the home platform the correctness of its results is checked by authenticating the presence of watermark. If the watermark is not present that means the results were modified by a malicious host.

Since we are using reversible watermarking we can get the original results after authentication. If the watermark is destroyed by an attacker then the correct results can not be retrieved. For every pair *(x, y)* in the results we have to calculate the average value *l* and difference value *h* using the following transformation respectively

$$l = \lfloor x + y \rfloor / 2 \quad (1)$$

$$h = x - y \quad (2)$$

The modified values of *x* and *y* are calculated by using the following transformation

$$x' = l + \lfloor h' + 1 \rfloor / 2 \quad (3)$$

$$y' = x' - h' \quad (4)$$

The watermark is embedded into the binary representation of *h* at the location right after the most significant bit (MSB). Where h' is the new difference number after embedding the watermark into *h*. The algorithm's steps are explained in the following pseudocode below.

Step 1: Get results from database
Step 2: Convert results to ASCII values.
Step 3: Make pairs of ASCII values i-e. (x, y)
Step 4: If $x > y$ go to step 5 else go to step 6
Step 5: Swap the values of *x* and *y*.
Step 6: Update the flags record for *x* and *y*
Step 7: Calculate the values of *l* and *h*.
Step 8: Embed watermark bit $\boldsymbol{w} = \{0, 1\}$.
Step 9: Calculate x' and y'.
Step 10: Construct the graph representation of x' and y'
Step 11: Serialize the graph (only the head is necessary to serialize)
Step 12: If itinerary completed to home else dispatch next host

In the following table we show some sample output of the algorithm for watermark embedding process. The table show the ASCII values of x and y before embedding the watermark and new modified values (x' & y') after the watermark bit is embedded.

In this table h' is the value of *h* after embedding the watermark. Whereas $(h)_2$ represents the binary value of *h*. similarly $(x')_2$ and $(y')_2$ is the binary representation of x' and y' respectively.

Now we discuss how the watermark is extracted. After the Booking mobile agent reach the home platform it delivers all the results and flags set to a Home Agent. This whole process is explained by the following pseudo code.

Table 1
Watermark Embedding Process

x	*y*	*l*	*h*	$(h')_2$	h'	x'	y'	$(x')_2$	$(y')_2$
90	72	81	18	100010	34	98	64	1100010	1000000
57	55	56	2	100	4	58	54	111010	110110
51	49	50	2	100	4	52	48	110100	110000
52	48	50	4	1000	8	54	46	110110	101110
49	45	47	4	1000	8	51	43	110011	101011
54	45	49	9	10001	17	58	41	111010	101001
50	48	49	2	100	4	51	47	110011	101111
54	48	51	6	1010	10	59	49	111011	110001
49	48	48	1	10	2	49	47	110001	101111
58	48	53	10	10010	18	62	44	111110	101100
56	53	54	3	101	5	57	52	111001	110100
52	48	50	4	1000	8	54	46	110110	101110

Step 1: get a list of graph heads and deserialize each graph
Step 2: Get the value of x' and y'
Step 3: Calculate new l' *and* h' using (5) & (6).
Step 4: Autheticate watermark.If the watermark bit is present or not?
Step 5: Extract watermark if authenticated.
Step 6: Calculate the original values of *x* and *y* using the transformation in equation (7) & (8). *h* is the value of h' after watermark bits are removed
Step 7: Check the flags accordingly to swap the values if necessary.
Step 8: Get the final result(*x* and *y*)

$$l' = \lfloor x' + y' \rfloor / 2 \quad (5)$$

$$h' = x' - y' \quad (6)$$

$$x = l' + \lfloor h + 1 \rfloor / 2 \quad (7)$$

$$y = x - h \quad (8)$$

Table 2
Watermark Extraction

x'	y'	l'	h'	$(h)'_2$	$(h)_2$	h	x	y
98	64	81	34	100010	10010	18	90	72
58	54	56	4	100	10	2	57	55
52	48	50	4	100	10	2	51	49
54	46	50	8	1000	100	4	52	48
51	43	47	8	1000	100	4	49	45
58	41	49	17	10001	1001	9	54	45
51	47	49	4	100	10	2	50	48
59	49	51	10	1010	110	6	54	48
49	47	48	2	10	1	1	49	48
62	44	53	18	10010	1010	10	58	48
57	52	54	5	101	11	3	56	53
54	46	50	8	1000	100	4	52	48

The watermark results are represented by a graph data structure. The following piece of Java code shows the graph data structure representation.

```
class WatermarkGraph implements Serializable{
      private int selfReference;
      private WatermarkGraph nextNode;
      private WatermarkGraph head;
}
```

This data structure can be used to represent our watermark graph. This data structure has three fields. The first field self Reference is of type integer and it is used to represent the number of zeros. For every 1 in the result we draw a new node. The nextNode is a reference to the next node in the graph structure. The nextNode Null value represent that we have reached to the end of the input. The third field is the head and it used to keep track of the head of the graph.

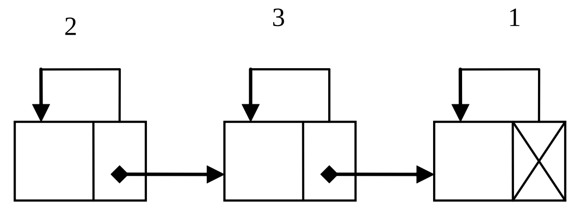

Figure 1: Graph Representation of the Results.

The graph above encodes the binary value of 100100010.

5. ATTACKS AND SECURITY ANALYSIS

There are many possible attacks to prevent Mobile agent from getting the results without being tampered. The attacks that we discuss for our approach are also common against program transformation attacks.

Before giving a security analysis of our approach we would like to stress that no technique or algorithm can guarantee unresponsiveness against all attacks and often we have to choose a tradeoff between some factors to achieve the desired level of security.

The approach proposed by this paper has also some limitations.

5.1 Reverse Engineering of Code

One of the shortcomings is that an attacker can still capture the code and makes as many copies of it as he wishes. That means the attacker can decompile the code also. There are so many decompilers publicly available [19, 20, 21]. So getting the mobile agent code is not a difficult problem for an attacker. So generally for mobile agent it is always assumed that the attacker will always have a copy of the code. So if we can not stop the attackers to capture the code we can make it difficult for him to understand the code and extract some useful information out of it.

5.2 Static Analysis of Code

Now the attacker can perform analysis of the code in order to understand the code and in particular the part/routine of the code that build the watermark structure. So what can be done then? Fortunately for the last few years there have been many techniques developed to make a program difficult to understand against such attackers. Obfuscation being the most popular and strongest of all, the goal of an obfuscator is to produce a program that has the exact same function but a lot more difficult for an attacker to understand and reverse engineer [22].

5.3 Cropping Attacks

For an attacker to be successful locating the watermark is the first job. In order to locate the watermark the code that construct and embeds the watermark must be analyzed. This task can be made difficult by using some of obfuscation techniques [22]. That means the attacker can apply some sort of deobfuscation to remove the effects of the obfuscator. After the analysis of code, the heap allocated data structures must be access to locate and remove the watermark. The code that builds a heap allocated data structure is difficult to analyze for an attacker and generally it is believed that given two pointer it is undecidable to find whether they refer to the same memory location [8, 9].

The fact that the results are watermark and not have the actual values makes the task of a malicious host even more difficult although not impossible. Another reason is that all the attacks must be carried out in a way without affecting the function of the program. The routine that actually extract watermark from the mobile agent results is not part of the mobile agent code so there is little information in the executable code about the exact location of the watermark. The Home Aglet (HA) is responsible for authentication and extraction of watermark in order to get the results.

5.4 Code Optimization Attacks

A clever attacker may use some code optimization techniques to remove the code that builds the dynamic graph structure. The code optimizer will recognize the graph that build the graph structure as dead code and try to remove it. One of the strength of proposed approach is that since all the watermark is constructed inside a heap allocated dynamic data structure so the strong typing feature of the java can be used as a relying factor to check the authentication of the embedded watermark.

6. CONCLUSION

This paper has introduced a new method for mobile agent security using graph based reversible watermarking. Mobile Agent Security is a challenging problem to solve because of the diverse nature of the execution environment of mobile agents. By using mobile agents in any client/server application we can perform a lot of task more conveniently and efficiently but there are always some security vulnerabilities as well. On one hand it is easy to capture and reverse engineer the mobile agent code. we can use obfuscation to make it difficult for an attacker to understand the code. On the other hand we can use heap allocated dynamic data structures to hide the watermark results of a mobile agent. Using reversible watermarking for authentication of the mobile agent is a new idea. It guarantees the recovery of the original untampered results after the watermark has been found to be authentic.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Project Number: 60671064), the Science Foundation of Guangdong Province (Project Number: 05109511), the Science and Technology Plan of Guangdong Province (Project Number: 2006B37430001), the Foundation for the Author of National Excellent Doctoral Dissertation of China (Project Number: FANEDD-200238), the Multidiscipline Scientific Research Foundation of Harbin Institute of Technology (Project Number: HIT.MD-2002.11), the Scientific Research Foundation of Harbin Institute of Technology (Project Number: HIT.2003.52), the Foundation for the Excellent Youth of Heilongjiang Province, the Program for New Century Excellent Talents in University (Project Number: NCET-04-0330), and the Chinese national 863-Program (Project Number: 2005AA73120).

REFERENCES

[1] Todd Sundsted, "An Introduction to Agents." *www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html*

[2] Dag Johansen, "Mobile agent Applicability", *Proceedings of the Mobile Agents 1998*, Berlin, Springer-Verlag, Lecture Notes in Computer Science; vol. 1477, ISBN 3-540-64959-X,(1998), pp. 9-11 September, 1998.

[3] Pattie Maes, Robert H. Guttman and Alexandros G. Moukas, "Agents That Buy and Sell ", *Communication of ACM*, Vol. 42, no. 3, pp. 81-91, March 1999.

[4] Stavros Papastavrou, George Samaras and Evaggelia Pitoura, "Mobile Agent for WWW Distributed Database Access", *Proceedings of IEEE International Conference on Data Engineering (ICDE99)*, 1999.

[5] Gian Pietro Picco and Mario Baldi, "Evaluating Tradeoffs of Mobile Code Design Paradigms in Network Management Applications", *Proceedings of 20th International Conference on Software Engineering*, ICSE98, Kyoto, Japan IEEE CS Press, 1998.

[6] D.B. Lange and M. Oshima, Seven Good Reasons for Mobile Agents. *Communication of the ACM*, 42(3): 8-91, March 1999.

[7] Christian Collberg, Clark Thomborson, "Software watermarking: Models and Dynamic Embeddings". In *Symposium of Principles of Programming Languages* (POPL), pages 311-324, 1999

[8] G. Ramalingam. The Undecidability of aliasing. ACM Toplas, 16(5): 1467-1471, September 1994.

[9] Rakesh Ghiya and Laurie J. Hendren. Is it a tree, a DAG, or a Cyclic Graph? A Shape Analysis for Heap-directed Pointers in C. In *POPL'96*, pages 1-15, St. Petersburg Beach, Florida, 21-24 January 1996.

[10] D.B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley.

[11] T. Sander and C.F. Tschudin. Protecting Mobile Agents against Malicious Hosts. In Mobile Agents and Security, volume 1419 *LNCS. Springer-Verlag*, 1998.

[12] O. Esparza, M. Soriano, J.L. Mu˜noz, and J. Forn´e. Host Revocation Authority: A way of Protecting Mobile

Agents from Malicious Hosts. *International Conference on Web Engineering* (ICWE 2003), *LNCS. Springer-Verlag*, 2003.

[13] Vigna J., Cryptographic Traces for Mobile Agents, in: Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1419, 1998**,** *Springer*, pp 137-153

[14] Fritz Hohl. A Framework to Protect Mobile Agents by using Reference States. *In International Conference on Distributed Computing Systems*, pages 410-417, 2000.

[15] Hyungjick Lee, Jim Alves-Foss and Scott Harrison. "The use of Encrypted Functions for Mobile Agent Security". In the *Proceedings of 37th Hawaii International Conference on System Sciences* 2004.

[16] O. Esparza, M. Fernandez, M. Soriano, J.L. Munoz, and J.Forne. "Mobile Agent Watermarking and Fingerprinting: Tracing Malicious Hosts". In *Database and Expert System Applications* (DEXA 2003), volume 2736 of LNCS. Springer-Verlag, 2003.

[17] O.Esparza, M.Soriano, J.L.Munoz, and J.Forne. "Detecting and Proving Manipulation attacks in Mobile Agent System". In *First International Workshop, MATA* 2004, Florianopolis, Brazil, October 20-22, 2004. Proceedings, Volume 3284 of LNCS. Springer Verlag, 2004.

[18] Fritz Hohl, "Time Limited Black box Security: Protecting Mobile Agents from Malicious Hosts", LNCS, Volume 1419, Springer-Verlag London, UK 1998.

[19] Pavel Kouznetsov. Jad-the fast java decompiler, version 1158e for Linux on Intel platform. Avaliable *http://kpdus.tripod.com/jad.html*, 5 august, 2001.

[20] Jerome Miecznikowski. Dava Decompiler, part of SOOT, a Java Optimization Framework, Version 7.1.09. Avaliable *http://www.sable.mcgill.ca/software/soot/* 17 December, 2003.

[21] Peter Ryland. Homebrew decompiler, version 0.2.4. Avaliable http://www.pdr.cx/projects/hbd/, 15 february, 2003. Available *http://www.pdr.cx/projects/hbd/.*

[22] Christian Collberg, Clark Thomborson, and Douglas Law. Breaking Abstraction and unstructuring data structures. In IEEE international conference on computer languages, ICCL'98, Chicago, IL, May 1998.

[23] Tian, Jun. "Wavelet-based Reversible Watermarking for Authentication". Proc. SPIE Vol. 4675, p. 679-690, Security and Watermarking of Multimedia Contents IV, Edward J. Delp; Ping W. Wong; Eds.