International Journal of Computer Applications (0975 – 8887)
Volume 97– No.12, July 2014

Intellectual Property Right Protection of Browser based Software through Watermarking Technique

Yogesh Awasthi
Assistant Professor
Shobhit University
Meerut,U.P.,India

R.P. Agarwal
Professor
Shobhit University
Meerut,U.P.,India

B.K. Sharma
Professor
AKGEC
Ghaziabad,U.P.,India

ABSTRACT

With the explosive growth of internet technology, many innovative web applications have been launched. For designing web applications Java has become a very popular programming language. Java bytecode technique makes it with high portability. However, it also poses large dangers to malicious users. Many techniques have been proposed for software copyright protection. They are useful for stand-alone Java applications. Due to special characteristics of Java web applications, copyright protection faces new challenge. In this paper, a copyright protection scheme for browser based Java applications have been introduced. The core of this scheme is a software watermarking algorithm. Code tamper-proofing and code obfuscation are also applied to watermarked Java code for better protection. The experimental results demonstrate the feasibility of the scheme.

Keywords

web applications, watermarking, code obfuscation, temper proofing, bytecode.

1. INTRODUCTION

Software piracy has been causing enormous losses for software vendors. The dramatically increased use of the Internet makes the situation worse because the illegal software can be spread through the Internet much easier than before. The survey released by the Business Software Alliance [1] showed that 36% of software was being used illegally in 2010 and software vendors suffered huge losses.

Java is one of the most popular programming languages for software development, especially for developing web applications. Java bytecode technique and Java virtual machine make Java having high portability - compile once, run everywhere. However, the standard form of bytecode provides detailed information about the code and make it easy to be decomposed into reusable class files and even decompiled into source code by malicious program users [2][12].

Watermarks can be classified as either static or dynamic: static watermarks are embedded in the code and/or data of a computer program [8][9][10], whereas dynamic watermarking techniques store a watermark in a program's execution state [3][11]. Figure 1 shows a conceptual diagram of a simple static watermarking system and Figure 2 shows a conceptual diagram of a simple dynamic watermarking system.

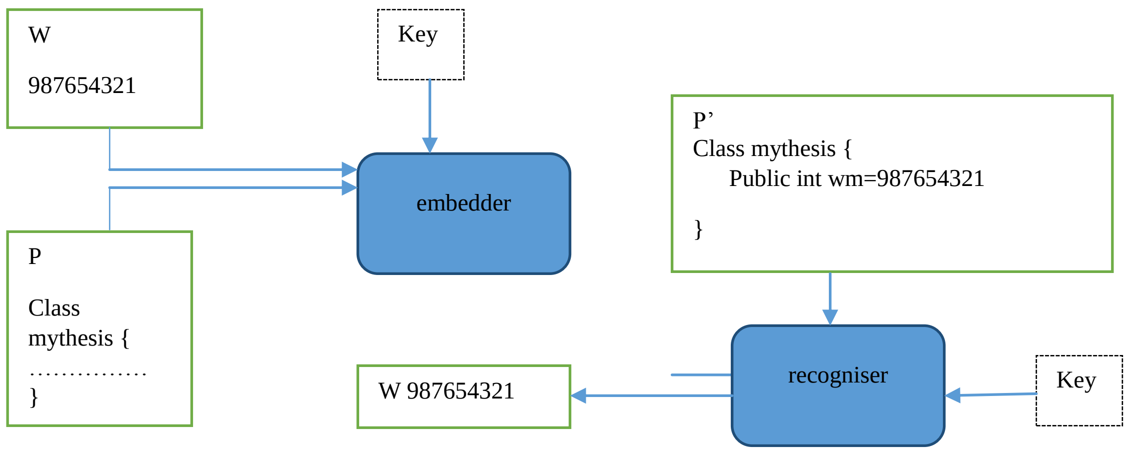

Figure 1: A Static Watermark System

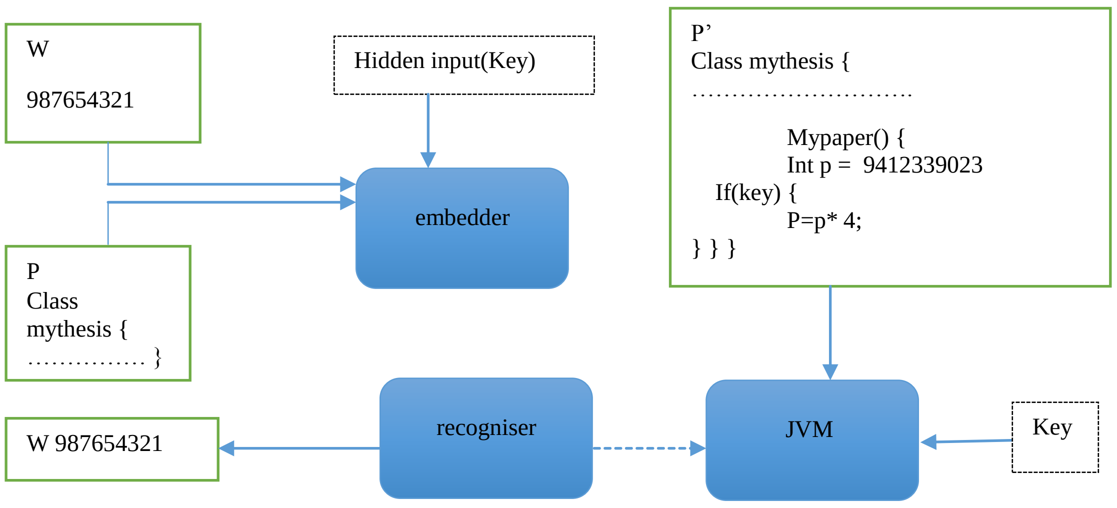

Figure 2: A Dynamic Watermark System

A variety of techniques, including hardware and software techniques, have been proposed for software copyright protection. In this paper only focus on software techniques. Three techniques fall into this category: code obfuscation and code tamper-proofing, and software watermarking.

Obfuscation technique applies semantics-preserving code transformations in an attempt to make the code as complex and confusing as possible. It makes the code hard to be decompiled by removing some information from the bytecode, and hard to be analyzed by transforming the code to irregular form. Tamper-proofing is the technique that prevents unauthorized use of or modification to a program. The mechanism is to put a guard or several guards in the code to check authorization so as to prevent the program from operating properly if any change is made in it [4]. Software watermarking involves embedding ownership information represented by a watermark in an executable program and the watermark can be retrieved latter by inputting a predefined secret key [5][7].

In this paper, A new watermarking scheme for J2EE web applications have been introduced. This scheme consists of watermark embedding, key hiding, watermark tracing, code tamper-proofing and code obfuscation. The watermarking algorithm provide ways to solve the following two key issues: 1) hide the input key even it is input through web pages; and 2) return the watermarks stealthily via HTTP or TCP/IP protocols.

2. EMBEDDING WATERMARKING OF JAVA BASED BROWSER APPLICATION

Suppose “YBKRP” is the watermark would be embed, it can hide it among the following exceptions that represent authentication and validation results in a specific page in the web application:

```
string excep1 = "a is not allowed in user name";
string excep2 = "e is not allowed in user name";
string excep3 = "i is not allowed in user name";
string watermark = "YBKRP";
string excep4 = "o is not allowed in user name";
string excep5 = "u is not allowed in user name";
string excep6 = "A is not allowed in user name";
string excep7 = "E is not allowed in user name";
string excep8 = "I is not allowed in user name";
string excep9 = "O is not allowed in user name";
string excep10 = "U is not allowed in user name";
```

The main modularization concept in Java programming language is class. A class is an abstract data type that encapsulates data and methods. Collberg et al. [6] suggested that the complexity of a class grow with its distance from the root in the inheritance hierarchy, and the number of its direct descendants. Following above proposed argument of an object-oriented language obfuscation technique is to embed the watermark. There are three steps for watermark embedding.

Step-1 A class is split into an interface, a set of abstract class and a concrete class. They are organized in a hierarchical tree where the concrete class extends abstract classes and the top-level abstract class implements the interface.
Step-2 Write a set of methods that contain the watermark or the exception messages shown above.
Step-3 Spread these methods to the abstract classes and the concrete class generated in the first step.
For example, the methods could be as follows:

```
public String exception1(int d) {
String excep1 = "a is not allowed in user name";
return excep1;
}
public String exception2(int d){
String excep2 = "e is not allowed in user name";
return excep2;
}
.....
public String watermark(int d) {
String watermark = "YBKRP";
return watermark;
}
.....
```

Through these three steps, the watermark is hidden among a number of exceptions and spread in a hierarchical tree, making it is hard to be distinguished.

3. KEY OBFUSCATION

When user want to retrieve the embedded watermark, user have to input a key into the web application. As it is already mentioned that the essential issues of browser's application watermarking are to protect the key from attacks and return the watermark to retrievers in a stealthy way. In this section, a technique that combines Java code and native code written in C language to protect the key when it is input from the web page to trace the watermark have been propose.

The web page propose to input the key is the login page in which the user input the user ID and password. User can also select other pages to input the key. However, in login page, authentication and different validation results (exceptions) will be undertaken, so it provides more spaces to hide the key among these results and makes it hard for attackers to distinguish. For example, assume the user ID cannot contain any vowels such as a,e,i,o,u or A,E,I,O,U . An exception will be thrown if such vowels exists in user ID. The key is unique and easy to distinguish. In order to achieve stealthiness, other languages can be employ, such as C language to obfuscate the key and other input. Unlike Java code, C language code will be compiled to machine language directly and cannot be decompiled to source code. It can only be analyzed from assembly language. The method written in such language as C language is called native method. J2SE provides the Java Native Interface (JNI) to access the native methods.

In the native code, a random number in a designed range is generated. Each vowel generates a random integer in a given range. The watermark embedded among these integers and return an integer to the business delegate class. Table 1 shows the encoding format. Because only an integer, which may represent authentication, validation result or the key, returns from the native code program to the business delegate class, it is impossible for attackers to distinguish which integer represents the key. The following pseudo-code implemented by native code (C-code) shows such a process. As a result a random integer is returned to business delegate class.

```
……………………
…………………

integer genvalue;
if useID contains "a"
genvalue = random(600,999);
else if userID contains "e"
genvalue = random(1200,1600);
else if userID = key
genvalue = 1978;
else if userID contains "i"
genvalue = random(1979,2500);
…………….
……………..
```

Table 1: Vowels and corresponding range of integers

Vowels	A	E	I	O	U	Key	A	E	I	O	U
Integers	600-999	1200-1600	1979-2500	2550-2998	3001 -3550	1978	3600-4150	4200-4500	4550-5100	5200-5660	5700-6300

4. WATERMARK RETRIEVAL

In order to retrieve the embedded watermark, user have to input the pre-defined key in the login page. In our algorithm, the key is an integer, which equals to 1978. After the user input a username or the pre-defined key in the login page, Java program will validate the input and then return an integer according to different exceptions that includes the case of inputting an illegal username or the key. This is achieved by a piece of C language code that is introduced in above section. According to the returned integer, different error messages or the embedded watermark will be returned to the user. The following is a piece of pseudo-code that shows how an error message or the watermark is picked up and sent to the client side according to the returned integer.

The following is a piece of pseudo-code that shows how an error message or the watermark is picked up and sent to the client side according to the returned integer.

```
............
............
genvalue = nativeCode.validate(input)
if genvalue is between 600 and 999
outputString = generating_result.exception1();
else if genvalue is between 1200 and 1600
outputString = generating_result.exception2();
......
else if genvalue is 1978
outputString = generating_result.watermark();
......
```

Methods exception1( ), exception2( ), and watermark( ) are defined above. They are spread among a hierarchy of abstract classes and concrete class. In order to improve the stealthiness, more dummy methods could be added in the hierarchy tree. Utilize the integer which will not be returned from the native code to establish the connection to these never-executed methods.

```
public String exception11() {
String s="Your choice may be correct";
return s;
}
public String exception12() {
String s="Your choice may be more correct";
return s;
}
```

An always-false prediction is used to connect to these two methods in other classes:

```
if genvalue is between 1000 and 1100
outputString=exception11();
if genvalue is between 1601 and 1700
outputString=exception12();
```

Since integer 1000-1100, 1601-1700 will never be generated, these methods will never be executed, but attackers cannot know it. They have to spend more time to analyze and try to find the watermark among these strings. After the watermark is retrieved, it will be sent back to the client via certain communication protocol, such as HTTP, TCP/IP, remote method invocation, JavaMail or even with techniques like XML and XSLT.

5. CODE PROTECTION

The proposed software watermarking algorithm embedded the copyright information in the Java source code. Since Java byte code is easy to be decompiled back to source code, obfuscation technique and tamper proofing technique to protect the Java code from reverse engineering and illegal modifications of the code have been employ.

5.1 Code tamper-proofing.

The working theory is that a sign of the program is set up in advance, and at running time, a new sign is generated and compared with the original one. Any difference between these two signs can prove the program is modified. The sign could be the length of the file, a checksum created by using message digests, or a guard provided by cryptography techniques.

5.2 Code obfuscation

Employ this technique to watermarking algorithm in order to achieve that: 1) the class files cannot be decompiled; 2) the code from decompiling tool is hard to read; and 3) the code from decompiling tool cannot be recompile. As a result, it is much more difficult for attackers to attack the watermarked application.

6. PERFORMANCE EVALUATION METRIC FOR WATERMARK WEB APPLICATION

A performance of a web application is based on its space, speed and time. In order to evaluate the performance of watermark web application, the mathematical theory of relativity have been introduced as follows:

6.1 Size Cost

A web application server hosts a lot of web applications. The size of the watermarked web application should not increase significantly. Let Size(X)(true value) be the size of a web application X and Size(W(X))(change value) be the size of the watermarked web application W(X), the approximation cost Sc(W(X),X) of an web application is given as:

Approximation cost S_c(W(X) , X)= ((True Value – Observed Value))/ True Value

S_c(W(X) , X) = ((Size(X) - Size(W(X))) / Size(X)

If value of Sc(W(X) , X) is +ve then cost of size is improved else not improved.

6.2 Running Cost

The running cost Rc measures the extra main memory size and running time for the watermarked web application compared with the original web application. Let path P = {P0, P1, · · · , Pn} be a possible path in a web application X. Let XRt(pi) be the running time of the methods in path Pi, XM(pi) be the average size of memory that all methods used in Pi in application X. Let WRt(pi) be the running time of the methods in path Pi, WM(pi) be the average size of memory the methods used in Pi in the watermarked web application W(X). The running-time cost Rc(W,A) is given by:

Approximation running cost R_c(W(X) , X)= (True Value – observed Value)/ True Value

$Rc(W(X),X) = \sum_{p=0}^{n} X\, R_t(p_i) \times X\, M(p_i) - \sum_{p=0}^{n} W\, R_t(p_i) \times W\, M(p_i) / \sum_{p=0}^{n} X\, R_t(p_i) \times X\, M(p_i)$

If value of Rc(W(X) , X) is +ve then running cost is improved else not improved.

6.3 Speed Cost

Speed is essential for web applications. No one likes to stay at a slow web site. The speed cost Pc measures the response speed of the watermarked web application compared with the original web application. Let path P = {P0, P1, · · · , Pn} be a possible path in a web application X. Let XRe(pi) be the response time of the web pages in path Pi in the web application X. Let WRe(pi) be the response time of the web pages in path Pi in the watermarked web application W(X). The speed cost Pc(W,X) is given by:

Approximation speed cost P_c(W(X) , X)= (True Value – observed Value) / True Value

$P_c\,(W(X),X) = \sum_{p=0}^{n} A\, R_t(p_i) - \sum_{p=0}^{n} W\, R_e(p_i) / \sum_{p=0}^{n} A\, R_t(p_i)$

If value of Pc(W(X) , X) is +ve then speed cost is improved else not improved.

After applying the proposed watermarking algorithm and metric measure, improvement have been observed in size, speed and running cost shown in figure 3.

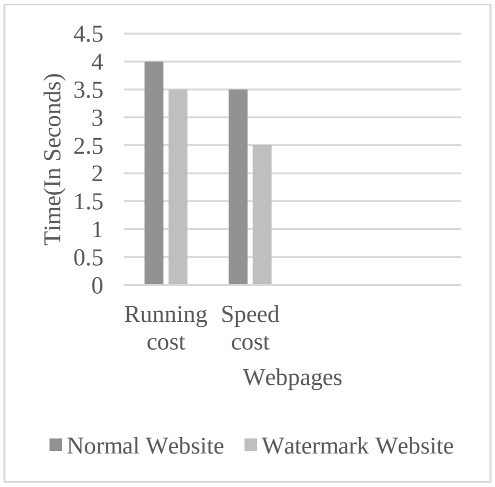

Figure 3: Performance measurement of webpages

7. CONCLUSION

In this paper, a watermarking scheme to protect copyright of browser based Java applications have been proposed. In our scheme, the watermark is hidden among a set of exceptions that represent validation and authentication result. All these exception are spread in a hierarchy tree to add the complexity of analysis. The key for watermark retrieval is also obfuscated by a native method written in C language. Code tamper-proofing and code obfuscation techniques are employed to protect the watermarked Java code. Mathematical metric showed the robustness of the watermarking scheme and its little impact in performance. With this scheme user can protect the copyright dispute of commercial web applications.

For future research, it is necessary to implement watermarking techniques to various scripting language which is used to implement web applications.

8. REFERENCES

[1] Business software alliance. http://www.bsa.org/.

[2] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii. A practical method for watermarking java programs. In Proceedings of the 24th Annual International Computer Software and Applications Conference, pages 191–197, Taipei, Taiwan, October 2000.

[3] C. Collberg and C. Thomborson. On the limits of software watermarking. Technical Report 164, August 1998.

[4] M. Atallah, H. Chang, T. Korb, and J. Rice. Software tamperproofing. CERIAS Conference, April 2001.

[5] D. Curran, N. J. Hurley, and M. O’Cinneide. Securing java through software watermarking. In Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java, volume 42, pages 145 – 148, Kilkenny City, Ireland, 2003.

[6] C. Collberg, C. Thomborson, and D. Low. Breaking abstractions and unstructuring data structures. In Proceedings of 1998 International Conference on Computer Languages, pages 28–38, Chicago, IL, USA, may 1998.

[7] G. Naumovich and N. Memon. Preventing piracy, reverse engineering, and tampering. Computer, 36(7):6471, 2003. ISSN 0018-9162.

[8] B. K. Sharma, R.P. Agarwal & R. Singh “An IPR of software codes using Watermarking for Modular Programming”, IJMCS, January- June -2010, PP 55 - 58

[9] R. Davidson and N. Myhrvold, “Method and system for generating and auditing a signature for a computer program,” Jun. 1996, Microsoft Corporation, US Patent 5559884.

[10] P. R. Samson, “Apparatus and method for serializing and validating copies of computer software - US patent 5287408 full text,” 1994. [Online]. Available: http://www.patentstorm. us/patents/5287408/fulltext.html

[11] C. Collberg and C. Thomborson, “On the limits of software watermarking,” Tech. Rep. 164, Aug. 1998.

[12] A. kalinovsky, Covert Java: Techniques for Decompiling, Patching, and Reverse Engineering, Sams Publishing, 2004.