Copyright Protection of J2EE Web Applications through Watermarking

Feiyuan Wang, Jiying Zhao, and Abdulmotaleb El Saddik
School of Information Technology and Engineering, University of Ottawa
800 King Edward Ave., Ottawa, Ontario, K1N 6N5, Canada
Email: {fwang, jyzhao, elsaddik}@site.uottawa.ca

Abstract

Three techniques – code obfuscation, software watermarking, and code tamper-proofing – have been proposed to protect the copyright of software. But they face challenges when they are applied to web applications, in which servers and clients are connected through Internet protocols, such as HTTP and TCP/IP. These protocols are intrusion prone. The proposed research aims at protecting the originality and copyright of web applications through watermarking. The presented technique includes watermark embedding, key obfuscation, and watermark retrieval through Internet protocols and Java based web application techniques in a stealthy way. Experimental tests demonstrate the feasibility of the algorithm.

Keywords *— Software copyright protection, watermarking, Java web applications.*

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.

Java has become a popular Internet programming language for developing web applications because of its portability – compile once, run everywhere. However, its bytecode technique makes Java easy to be decomposed into reusable class files and even decompiled into source code by malicious program users [1].

Three techniques have been used for software copyright protection: code obfuscation, software watermarking and code tamper-proofing. Code obfuscation is the process of transforming bytecode to a less human-readable format, making it hard to be decompiled or analyzed even it is decompiled [2]. It includes stripping all unnecessary information, such as line numbers, local variable names and source file names used by debuggers from the classes, and renaming classes, interfaces, fields and method identifiers to make them meaningless. Software watermarking is to embed watermarks into an executable program and these watermarks can be retrieved at a later stage by inputting a predefined key. Two types of software watermarking have been proposed: static and dynamic watermarking. Tamper-proofing mechanism puts a guard in the software to check authorization and to prevent the program from operating properly if a change is made to it.

These three techniques are useful for Java applications copyright protection, but they face new challenges for Java web applications, such as popular J2EE E-Commerce applications, based on two reasons. First, most of these web applications are zipped in jar files and then delivered to the host. In this case, a malicious host may unzip these files and reuse part of them to escape from the constraint of code obfuscation and tamper-proofing. Second, in web applications, servers and clients are connected by Internet protocols, such as HTTP/HTTPS and TCP/IP. It is more difficult to hide a secret key when it is input to trace the watermark and return watermarks to the retriever stealthily, since they need to be transmitted over these vulnerable protocols.

To our best knowledge, up to now there is no research that addresses the issues of web application watermarking. In this paper, we propose a new watermarking scheme for J2EE web applications.

2 J2EE Web Application Watermarking

Web applications are usually based on the multi-tier architecture. A tier is a package of software that implements corresponding functions and contacts with other tiers through interfaces. Each tier can be developed, tested and maintained independently [3].

A typical J2EE application consists of four tiers: client tier, web tier, business tier and Enterprise Information System (EIS) tier. The web tier implemented by Servlets and JSPs communicates with the client tier by accepting requests from clients and forwarding the request to the business tier. In another direction, it accepts appropriate responses from the business tier and sends to the client tier. The business tier handles the business logic of the application. It is accessed by clients via the web tier and provides the interfaces to the underlying business modular components, called EJBs (Enterprise JavaBeans). Each EJB is responsible for a defined area of the business logic. The EIS tier is responsible for the enterprise information systems, including database systems, transaction processing systems and legacy systems.

There are two significant differences between regular Java-based stand-alone application watermarking and Java-based web application watermarking. First, for a Java stand-alone application, we have sufficient space to input the key; but for web applications, we can only input the key through limited web pages. Second, in Java stand-alone applications, watermarks can be deposited, generated and retrieved in many ways. For example, we can check the heap or a variable value and then simply display by "System.out.print" command. On the other hand, for web applications, we can only retrieve and display the watermark via HTTP, TCP/IP or a few other protocols. Under these

protocols, the transferred data is transparent and easy to be monitored. As a result, the main challenge in web application watermarking is to return the watermark to the retriever stealthily, rather than the complexity of algorithm of embedding the watermark.

In this section, we will give the definition and metrics of web application watermarking, propose our web application watermarking algorithm including the watermark embedding, key obfuscation, and watermark retrieval.

2.1 Definition

Based on the definition of software watermarking [4], we define web application watermarking as follows:
Let A be a web application, let W(A) be the watermarked web application with an embedded watermark wk , let K be the input key accepted by W(A) in a specific web page, let res(A,I) be the responding content from A on input I and let res(A,K) be the responding content from A on input K, the following identities must hold:
res(A, I) = res(W(A),I) and wk = res(W(A), K).

2.2 Metrics

In order to evaluate the performance of a web application watermarking algorithm, we propose following metrics:

- Running cost

The running cost R_c measures the extra main memory size and running time for the watermarked web application compared with the original web application.

Let path $P = \{P_0, P_1, \cdots, P_n\}$ be a possible path in a web application A. Let $AR_t(p_i)$ be the running time of the methods in path P_i, $AM(p_i)$ be the average size of memory that all methods used in P_i in application A. Let $WR_t(p_i)$ be the running time of the methods in path P_i, $WM(p_i)$ be the average size of memory the methods used in P_i in the watermarked web application W(A). The running-time cost $R_c(W, A)$ is given by:

$$R_c(W,A) = \frac{\sum_{p=0}^{n} WR_t(p_i)\times WM(p_i) - \sum_{p=0}^{n} AR_t(p_i)\times AM(p_i)}{\sum_{p=0}^{n} AR_t(p_i)\times AM(p_i)} \quad (1)$$

- Speed cost

Speed is essential for web applications. No one likes to stay at a slow web site. The speed cost S_c measures the responding speed of the watermarking web application compared with the original web application.

Let path $P = \{P_0, P_1, \cdots, P_n\}$ be a possible path in a web application A. Let $AR(p_i)$ be the response time of the web pages in path P_i in the web application A. Let $WR(p_i)$ be the response time of the web pages in path P_i in the watermarked web application W(A). The speed cost $S_c(W, A)$ is given by:

$$S_c(W,A) = \frac{\sum_{p=0}^{n} WR(p_i) - \sum_{p=0}^{n} AR(p_i)}{\sum_{p=0}^{n} AR(p_i)} \quad (2)$$

The running cost $R_c(W, A)$ and speed cost $S_c(W, A)$ should be less than a factor of δ.

2.3 Watermark Embedding

In this section, we propose a scheme for watermark embedding. In this scheme, a static watermark "Site, MCR-Lab" is embedded among sorts of exception contents, such as "No '!' in use ID please!" or "No '@' in user ID please!".

The Chidamber metric [5] argued that the complexity of a Java class grows with its depth (distance from the root) in the inheritance hierarchy, and the number of its direct descendants. Following this metric, we propose an algorithm for watermark embedding. First, a hierarchical tree that consists of an interface, a number of abstract classes and a concrete class are set up. Second, a set of methods that contain the exception or the watermark are written. Third, these methods are spread in the hierarchy tree with other dummy methods.

2.4 Key Obfuscation

As we mentioned, the essential issues of web application watermarking are to protect the key from attack and to return the watermark in a stealthy way. In this section, we will propose a technique that combines Java code and native code written in C language to protect the key.

In typical Java based web applications, such as J2EE E-Commerce web applications, there are limited places to input the key because only Servlets in the web tier handle all inputs from the client tier (e.g. an HTML form from browser). Due to the fact that the key is unique, it is quite easy for attackers to detect it in these Servlet classes and to disable the watermarking attached to it.

The web page we propose to input the key is the Login page where a user inputs the user ID and password, because in this page, authentication and different validation results (exceptions) will be undertaken. It provides more spaces to hide the key among these results and makes it hard for attackers to distinguish. For example, we assume the user ID cannot contain special symbols such as !, @, #, etc. When the user input his user ID, the login Servlet will send it to a native code (C-code) program via a business delegate class. In the native code, a random number in a designed range is generated. Each symbol generates a different random integer in a given range. We can embed the watermark among these integers and return such an integer to the business delegate class. Table 1 shows the encoding format.

TABLE I
EMBED THE KEY TO A SET OF INTEGERS.

Symbol	!	@	#	2909 (key)
Integer	300-800	999-1621	1822-2708	2909
Symbol	%	<	>	+
Integer	3101-3576	3877-4050	4122-4900	5129-5801

Because only an integer, which may represent authentication, validation result or 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 the process. As a result, outputValue, a random integer, is returned to business delegate class.

```
integer outputValue;
if useID contains '!'
  outoutputValue = random(300,800);
else if userID contains '#'
  outoutputValue = random(1822,2708);
else if userID = 2909
  outoutputValue = 2909;
else if userID contains '%'
  outoutputValue =random(3101-3576);
```

2.5 Watermark Retrieval

Several J2EE techniques are available to return watermarks to the retriever. In this section, we explain how to apply these methods to return the watermark embedded in the application to the retriever.

• Servlet/JSP

Servlets/JSPs play the main communication role between the server and clients, so we can utilize them to return the watermark to the retriever. In Section 2.4, we have used a set of integers to represent authentication result, different validation exceptions and our watermark.

According to the returned integer, the string (exception content or watermark) from corresponding method in the hierarchical tree will be picked up and sent to client via Servlet/JSP.

Following piece of pseudo-code shows the string (the exception or the watermark) hidden in hierarchical tree is picked up and sent to clients:

```
outputValue=nativeCode.validate(input);
 if outputValue is between 300 and 800
 outputString=generating_result.exception1();
 // exception: No '!' in user ID please!
 else if outputValue is between 999 and 1621
 outputString =  generating_result.exception2();
 // exception:  No '@' in user ID please!
 else if outputValue is 2909
 outputString =  generating_result.watermark();
 // watermark: Site, MCRLab
 else if outputValue is between 3101 and 3576
 outputString =  generating_exception3();
 // exception:  No '%' in user ID please!
```

Methods exception1(), exception2(), and watermark() are spread among abstract classes and concrete class.

In order to add the difficulty of analyzing, more dummy methods, which contain useless context, are added in the hierarchy tree. We utilize the integer which will not be returned from native code to establish the connection to these never-executed methods. For example, two dummy methods are added into a class:

```
public String m0009() {
 String s="Stateless session bean";
 return s;
}
public String m0010() {
 String s="stateful session bean";
 return s;
}
```

A always-false predicts is used to connect to these methodsfrom other classes:

```
if outputValue is between 810 and 850
 outputString=m0009();
if outputValue is between 3601 and 3776
 outputString=m00010();
```

Since interger 810-850, 3601-3776 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.

• JavaMail

JavaMail is an API provided by J2EE platform. This API provides a platform independent and protocol independent framework to build Java based email applications. We can apply this technique to send the watermark to retriever when the key is input in some page in the application.

• TCP/IP socket

A socket is an endpoint for communication between two machines. We can set up a dynamic TCP/IP socket between the suspicious server and the watermark retriever's machine when the key is input to the former machine from the latter, and then the watermark can be sent to the retriever. TCP/IP socket is widely supported by different languages, e.g., Java or C language.

In our implementation, the server of socket is set up at the retriever's site:

```
 socket = server_socket.accept();
 input=new BufferedReader(new
 InputStreamReader(socket.getInputStream()));
 String watermark=input.readLine();
 if (message==null) break;
  System.out.println("You get the watermark!"
   + watermark);
```

And the dynamic client of socket is set up at the suspicious host when the key is received:

```
 socket=new Socket(site, port);
 String outString=watermark;
 output=new
   PrintWriter(socket.getOutputStream(),true);
 output.println(outString);
```

3 Code Protection

In the previous sections, we proposed a scheme for watermarking Java based web applications. All the code is written in Java bytecode. Obviously, it is fragile. Just like

Java-base application, code tamper-proofing and obfuscation techniques should be applied to protect the watermark and corresponding code.

3.1 Code Tamper-proofing

Software tamper-proofing technique provides us with a useful way to protect the watermark and its corresponding code in the applications. The working theory of tamper-proofing technique is that a sign of the program, such as checksum or guard is set up in advance, and at running time, a new sign is generated and compare to the original sign. Any difference between these two signs can prove the program is modified and the developer may take specific actions to the program.

Relatively, software tamper-proofing is a mature technique and many methods, including software and hardware methods are available. In this paper, we are only interested in protecting watermark code by software methods. The following are the possible ways that we can use in our web applications:

- File lengths comparison

We calculate the length of the class that contains watermarking code and save the length. At the running time, we load the file and calculate its length again. If two lengths are different, we can assume that the file has been changed.

- Message digests comparison

Similar to the first method, we calculate and store a checksum in the local machine, and calculate the checksum of the same file and compare both to decide if the file is changed.

3.2 Code Obfuscation

Obfuscation provides a powerful way to protect the Java code from reverse engineering, so that attackers have to spend more time, even give up to analyze the watermarking and tamper-proofing code. There are many commercial software obfuscation tools available. The use of these tools will add the difficulty of code analyzing. After obfuscated, we expect to achieve: 1) The class files cannot be decompiled; 2) The code from decompile tool is hard to read; 3) The code from decompile tool cannot be recompile, to increase the difficulty for attackers if they change the code. From our experiments, the second and third target are achievable.

4 Experimental Results

In order to evaluate the resilience, running cost, and speed cost of our web application watermarking scheme, following experiments are conducted to the two versions of our fictive online shopping application. The first version, called SITE1, has no watermarks and the second one, called SITE2, is protected by watermarking, tamper-proofing and obfuscation.

- Resilience

Ad-Aware by Lavasoft and Microsoft AntiSpyware by Microsoft are used to attack SITE2. all three watermarks survived. When using DashO to transform SITE2. Two out of three watermarks survived. Returning the watermark through e-mail failed.

- Running cost

From the experimental data, the running cost is 15.63 %.

- Speed cost

From the same experiments above, the speed cost is 7.1 %. The experiments show that the watermarking scheme has some impact to the performance compared with the original web application.

5 Conclusions

In this paper, we proposed a new scheme for watermarking Java based web applications. Although some obfuscators are proposed on web applications, we have not seen any work on web application watermarking. 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 is obfuscated by a native method. Servlet class returns the content of exception or watermark according to an integer from the native method, rather than the plain text from web page. Except for Servlet, we also proposed several ways to return the watermark to retriever, such as JavaMail and TCP/IP socket. When JavaMail method is used, the watermark triggered by input key will be sent to the retriever via an e-mail using JavaMail API. In the socket method, when the secrete key is input, a dynamic socket between the suspicious host and retriever's machine will be set up and the watermark from the host will be sent to the retriever. The experiments have shown that the scheme works and there is no big impact on the performance. Furthermore, we can apply tamper-proofing and obfuscation tools to protect the watermark and corresponding code to add robustness. Compared with Java based application watermarking, Java based web application watermarking is much more difficult, but we believe that it is possible.

References

[1] A. Monden, H. Iida, and K. Matsumoto, *"A Practical Method for Watermarking Java Programs," The 24th Computer Software and Applications Conference*, 2000.

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

[3] S. Allamaraju, A. Longshaw, and L. Kim, *Professional Java Server Programming J2EE Edition,* Wrox publishing, 2000.

[4] T.C. Collberg and C. Thomborson, *"Software Watermarking: Models and Dynamic Embeddings," ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2000.

[5] T.C. Collberg, C. Thomborson, and D. Low, *"Breaking Abstractions and Unstructuring Data Structures," International Conference on Computer Languages*, 1998.