

Watermarking Java Programs

Akito Monden¹ Hajimu Iida² Ken-ichi Matsumoto¹ Katsuro Inoue^{1,3} Koji Torii¹

¹Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma,
Nara 630-0101, Japan

²Information Technology Center,
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma,
Nara 630-0101, Japan

³Graduate School of Engineering and Science,
Osaka University
1-3 Machikaneyama-cho, Toyonaka,
Osaka 560-8531, Japan

{akito-m@is, iida@itc, matumoto@is, k-inoue@is, torii@is}.aist-nara.ac.jp

ABSTRACT

Java programs distributed through Internet are now suffering from program theft. It is because Java programs can be easily decomposed into reusable classfiles and even decompiled into source code by program users. In this paper we propose a technique that discourages program theft by embedding Java programs with a digital watermark. Embedding a program developer's signature as a watermark in Java classfiles may help the developer to ensure their ownership rights of the classfiles. Our embedding method is indiscernible by program users, yet enables us to identify an illegal program that contains stolen classfiles.

KEYWORDS

copyright protection, digital watermark, JAVA, program theft

INTRODUCTION

Java technology is widely regarded as evolutionary by means of its portability: an idea that the same program should run on many different kinds of computers, consumer gadgets, and other devices. Java enables us to let computers and devices communicate with one another much more easily than ever before.

However, this evolutionary property of Java brought us a security problem against *program theft* [9]. Java applets put

on the Internet sites and Java applications sold to users are now suffering from program theft. It is because Java programs can be easily decomposed into reusable classfiles, and investigated by using class viewer or class editor. Moreover, program users can obtain source code of a classfile by using decompilers, such as *Mocha*, *Source Again*, etc[7]. In such situation, the Java program developer's intellectual property will be infringed if a program user steals developer's classfile and builds it into his/her own program. We call this copyright infringement a program theft.

This paper proposes a technique that discourages program theft by embedding Java programs with a digital watermark. We claim that embedding a program developer's signature as a watermark into Java classfiles will discourage program theft. Below we describe major features of our watermarking method:

1. Program users can hardly know the location of watermark, thus, erasing and/or tampering with the watermark is very hard for them.
2. Watermarks in programs will be kept unaware to program users even if our embedding method or embedding tool was opened to the public.
3. Even in case only a part of a program was stolen and was built into other program, the watermark is easily decoded

wherever it may exist in the program.

4. Watermarks in programs do not reduce the execution efficiency.

The remainder of the paper first describes cases where watermark is needed. Next, describes current researches on watermarking technique for computer programs, then discusses important properties that watermarking method should satisfy. Then we propose a watermarking method for Java programs including the watermark encoding procedure and the decoding procedure. Finally the paper describes the conclusion.

CASES WHERE WATERMARK IS NEEDED

Proving of program theft

Even in case an illegal program was found, it is often not easy to say who is the true developer of that program. A program thief who stole anyone else's classfile may insist, "I have originally developed this program." In this case, we need a *proof* to explode the thief's claim.

For that situation, a signature previously embedded in the program as a watermark authenticates original program developer's ownership claims and protects proprietary interests. The watermark in the program has an effect of proving the fact of program theft. Decoding the watermark from a suspicious program will make clear of who is the original developer of that program.

Finding of illegal program

It is not easy for Java program developers to find an illegal program that contains stolen classfiles. For example, we may not be able to find out if it is an illegal program or not by executing the program, because illegal programs often do not resemble original programs in their specifications if the stolen piece is small. This difficulty in finding the illegal program is a crucial problem for Java program developers.

We believe that watermark is effective to find the illegal program in Internet situation. Illegal programs may be easily detected by employing a watermark-decoding agent

(robot) that goes through Internet. This agent goes through Internet and finds Java programs, then checks the watermark in them. If a program turned out to be an illegal program that contains a watermark of an original program developer, the agent will inform it to the developer. By using this agent, program developers may automatically find the illegal program, so that they can protect their programs from program thieves.

RELATED WORKS

There were many studies done for watermarking images, sounds, texts, etc[1][2]. On the other hand, a few methods for watermarking computer programs have been proposed[3][5].

Hirose et al. proposed a method for embedding C source programs with user identification number[3]. In their method, one single change in the target program represents one bit information. This change in the program includes addition of dummy variables, rearrangement of variable declaration statements, changes in coding style (such as 'n=n+1' to 'n++'), and so on. In the decoding phase, the original program and the marked program are compared so that the changes in the program will be identified. However, if program thieves applied the same method to the watermarked program, original watermark will be easily erased. Also, this method is not applicable to the binary (executable) program.

Kitagawa also proposed a method for watermarking Java programs[5]. In this method, new variables are appended to a program; and, watermark codewords (values) are set to those variables. In decoding phase, program developers need to replace a specific classfile in the program with a special classfile (called *detection-classfile*), and execute the program. This detection-classfile outputs the values of the variable in which watermark was embedded. However, this method is not useful from the purpose of defending classfiles from program thieves. Because, when we want to check whether a target program contains a stolen program or not, we must find out which classfile is to be replaced with the detection-classfile. If we couldn't find the classfile

that should be replaced, or simply there is no classfile that should be replaced, we can not decode the watermark even if a watermark exists in the target program. It is natural that program developers want to easily check whether a target program contains a stolen program or not. From this viewpoint, watermarks should be automatically decodable from a target program by using a decoding tool.

IMPORTANT PROPERTIES OF WATERMARKING METHOD

Here we discuss the properties that the goal in the design of the watermark coding method should satisfy. We claim that the coding method should satisfy following six criteria.

(c1) Encoding watermarks does not change the specifications of the program.

This property must be satisfied in any encoding method for computer programs.

(c2) A person who encoded a watermark can decode a watermark automatically.

It is natural that program developers want to easily check whether a target program contains a stolen program or not. Program developers do not want to read or analyze the program for finding a watermark. From this viewpoint, watermarks should be automatically decodable from a target program. The design of the coding method should consider the ease of decoding.

(c3) A watermark in a program is decodable wherever it may exist in the program.

Program thieves may steal a program component of various sizes, and build it into their programs in various ways. In this situation, the decoding algorithm should not depend on the location of the watermark.

(c4) A method for erasing and/or tampering with watermark can not be derived from the watermark encoding method.

It is not avoidable that program thieves may know the encoding method as we describe the method in this paper.

(c5) Watermarks in programs should not reduce the execution efficiency.

Since some programs are very sensitive to the reduction of the execution efficiency, this property should be considered.

(c6) Watermarks should survive program conversions.

Program thieves may disassemble and re-assemble the program, decompile and re-compile the program, optimize the program, or use various program conversion tools such as obfuscator, scrambler, etc[6]. Watermarks should be reliably decodable even after these program conversions were applied.

WATERMARKING METHOD

Encoding procedure

Our watermark encoding procedure consists of the following three phases (see Figure 1):

(Phase 1) Dummy method injection

In the first phase of watermarking, a dummy method (of a class), which will never be executed, is appended to a target Java source program. This dummy method is a *space* for watermark codeword. Content of the dummy method is not cared, but it should have enough size for watermark injection.

Next thing we should do is to append a dummy method invocation in the original Java program. Below we show an example of the invocation:

```
if (Condition) Dummy_Method();
```

'Condition' is a expression that will never become true. So, actually, dummy method is never invoked. If this expression (formula) is complex enough, it is difficult for program users to become aware of the dummy method[4].

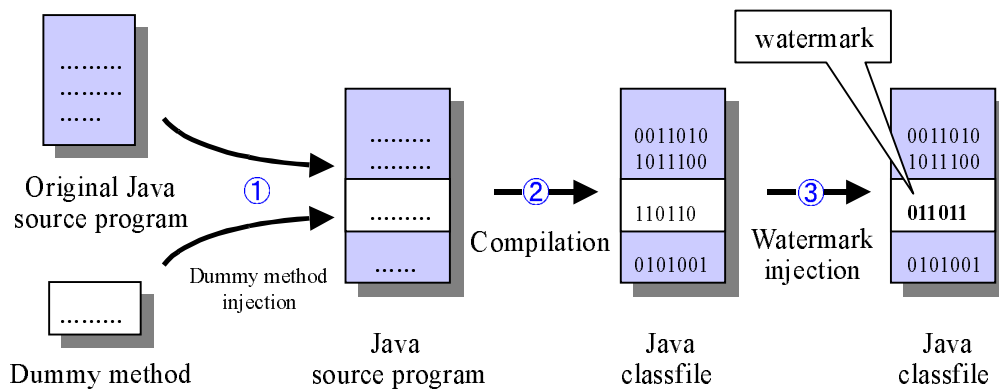


Figure 1. Overview of watermark encoding procedure

(Phase 2) Compilation

In the second phase, the Java source program, in which dummy method was injected, is compiled with a Java compiler. We can use common Java compiler for this compilation.

(Phase 3) Watermark injection

In this phase, we need to take account of the *bytecode verifier*[8]. When we execute a Java applet, bytecode verifier checks the syntactical rightness and type consistency of the program. In order to keep syntactical correctness and type consistency, we use following two approaches:

(i) Overwriting numerical operands

One simple way to keep syntactical correctness is to limit the place to overwrite. A numerical operand of an opcode that pushes a value to the stack, and of an opcode that increases a value on a stack, can be overwritten without syntactical incorrectness and type inconsistency. For example, an operand 'xx' of the opcode '*iinc xx*' and '*bipush xx*' can be overwritten into any single byte. On the other hand, most of other operands that indicate a position or an index of class tables or local variables of a method cannot be overwritten because of syntactical incorrectness. For example, an operand such as '*getfield xx*' and '*putfield xx*' cannot be overwritten. However, this limitation in operands drastically reduces the place that can be overwritten.

(ii) Replacing opcodes

In order to increase the place for watermark injection, we replace some of the opcodes, such as *iadd*, *ifnull*, and *iflt*, into other kind of opcode. For example, an opcode replacement from *iadd* to *isub* does not violate syntactical correctness and type consistency. Moreover, the opcode *iadd* can be replaced to anything among *isub*, *imul*, *idiv*, *irem*, *iand*, *ior*, and *ixor*. This indicates that above eight opcodes *iadd*, *isub*, ..., and *ixor* can be replaced mutually. By using this ability of mutual replacement, we can encode 3 bits information into these opcodes. For example, we may assign 000_2 to *add*, 001_2 to *isub*, 010_2 to *imul*, ..., and 111_2 to *ixor*. Whichever the above opcode appeared in the dummy method, we will replace them into one of the above eight opcodes according to the bits we want to encode. Such information assignment and opcode replacement can be also done in other opcodes. Table 1 shows the example of information assignment to opcodes that can be replaced mutually.

Table 1. Bit assignment to opcodes

bytecode	mnemonic	assigned bits
0E	dconst 0	0
0F	dconst 1	1
C6 xx xx	ifnull	0
C7 xx xx	ifnonnull	1
9B xx xx	iflt	00
9C xx xx	ifge	01
9D xx xx	ifgt	10
9E xx xx	ifle	11

address	bytecode	mnemonic	watermark
1000	02	iconst_m1	00 A
1001	60	iadd	000
1002	60	iadd	000
1003	68	imul	B 010 C
1004	3E	istore_3	-
1005	84 01 21	iinc 01 21	00100001 D
1008	1C	iload_2	-
1009	10 90	bipush 90	10010000 E

Figure 2. Example of watermark

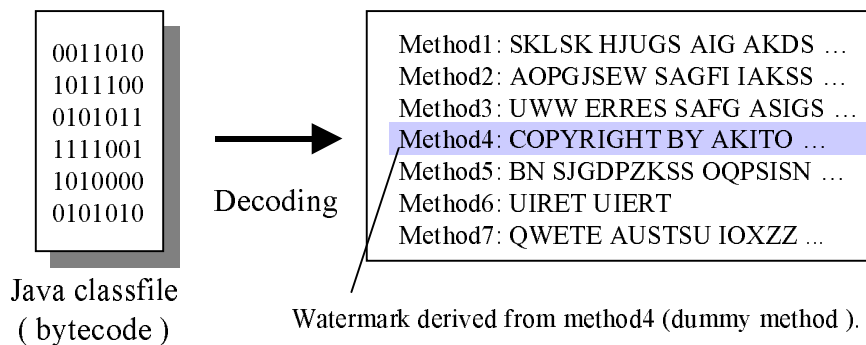


Figure 3. Example of watermark decoding

In case we need to encode a sentence, such as ‘COPYRIGHT 1999 BY AKITO MONDEN’, first we need to translate the sentence into a bit sequence, then we encode the bit sequence into the program. Figure 2 shows an example of a sentence encoding. In this example a word ‘ABCDE’ has been encoded into bytecode. At first, we have assigned ‘A’ to 00000_2 , ‘B’ to 00001_2 , ‘C’ to 00010_2 , ‘D’ to 00011_2 , and ‘E’ to 00100_2 , then we have encoded a bit sequence ‘0000000001000100001100100’ to the program.

Decoding procedure

In the watermark decoding phase, there is an assumption that we know the relation between bytecodes and their assigned bits, and also the relation between bit sequences and alphabets. The decoding algorithm is very simple. We simply do the exactly opposite procedure of watermark injection procedure, from top of every method. Both opcodes and operands in each class method should be

replaced into bit sequence, then, they should be replaced into alphabet sequence. After that, watermark will appear from the dummy method (Figure 3). This decoding procedure can be automated, so that even in case only a part of a program was stolen and was built into other program, the watermark is easily decoded wherever it may exist in the program.

SUMMARY

We have proposed a digital watermarking method applicable to the present Java programs. In most cases, Java programs are stolen in blocks of classfiles. In our method, program developers can discourage the program theft by embedding a watermark into their important classfiles. Our method is powerful and practical, because watermarks can be easily decoded from the suspicious program wherever the stolen classfile was built in.

We have implemented our method. Now we are planning to evaluate the strength of the watermark against program

conversions.

ACKNOWLEDGMENTS

The authors wish to acknowledge helpful suggestions from Dr. Yuuji Ichisugi of Electrotechnical Laboratory.

REFERENCES

1. Berghel, H. Watermarking cyberspace. *Communications of the ACM*, 40, 11 (1997), 19-24.
2. Craver, S., Memon, N., Yeo, B. and Yeung, M.M. Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications. *IEEE Journal on Selected Areas in Communications*, 16, 4 (1998), 573-586.
3. Hirose, N., Okamoto, E., Mambo, M. A proposal for software protection. *The 1998 Symposium on Cryptography and Information Security*, SCIS'98-9.2.C (Jan. 1998).
4. Ichisugi, Y. Watermark for software and its insertion, attacking, evaluation and implementation methods. *Summer Symposium on Programming*. (July 1997), 57-64.
5. Kitagawa, T. Digitalwatermarking method for Java programs. *Master's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology*, NAIST-IS-MT9751041 (Feb. 1999).
6. Leininger, K.E. The Java developer's tool kit. *McGraw-Hill* (1997).
7. McGraw, G. and Felten E. Java security: hostile applets, holes, and antidotes. *John Wiley & Sons* (1997).
8. Meyer, J. and Downing, T. Java virtual machine. *O'Reilly & Associates* (1997).
9. Monden, A., Hajimu, I., Matsumoto, K., Torii, K. and Ichisugi, Y. A watermarking method for computer programs. *The 1998 Symposium on Cryptography and Information Security*, SCIS'98-9.2.A (Jan. 1998).