2001 년 한국정보처리학회 추계 학술발표논문집 제 8 권 제 2 호

자바 클래스 보호를 위한 동적 워터마킹

조 익°, 이수현
창원대학교 전자계산학과
e-mail : chowing@pl.changwon.ac.kr

Dynamic Watermarking for Java Class Protection

Ik Cho, Su-Hyun Lee
Dept. of Computer Science, Changwon National University

요 약

최근 저작권 보호를 위한 워터마킹 기술이 활발하게 연구되고 있지만, 소프트웨어에 대한 워터마킹 기술은 멀티미디어 워터마킹에 비하여 상대적으로 부족하다. 인터넷의 발달로 인하여 자바로 프로그램을 작성하는 경우가 증가 되었으나, 자바 프로그램은 인터넷에서 쉽게 얻을 수 있고, 역-컴파일이 쉽고, 또한 재 사용 가능한 클래스 파일로 쉽게 분해되기 때문에 불법 사용이 가능하다. 본 논문에서는 소프트웨어의 저작권 보호를 위한 소프트웨어 워터마킹 기술에 대한 연구와 자바 프로그램 워터마킹 시스템을 제안 한다. 본 시스템은 워터마크를 힙 영역에 저장하기 때문에 왜곡 공격에 강한 장점이 있다.

1. 서론

자바는 한번 작성되면 자바 가상기계가 있는 모든 곳에서 실행될 수 있기 때문에 개발과 유지보수에 많은 장점을 가지고 있지만, 자바 애플릿이나 응용프로그램은 인터넷에서 쉽게 얻을 수 있고, SourceAgain[1], Jad[2], Mocha[3] 등으로 쉽게 역-컴파일(decompile) 하여 소스 코드를 얻을 수 있기 때문에 불법 사용이 가능하다. 이러한 이유로 많은 소프트웨어 개발사들은 자바로 개발하는 것을 주저하고 있다[4].

이에 대한 해결 방법으로 저작권을 삽입하는 워터마킹 기술에 대한 연구가 활발하게 진행되고 있다[4,5,6]. 워터마킹 기술을 적용할 경우 저작권자는 저작권이 있는 소프트웨어의 불법 사용과 유통을 막을 수 있고, 더 나아가 컨텐츠에 대한 정당한 대가를 보장 받을 수 있다.

현재 멀티미디어 데이터에 대한 워터마킹 연구는 많은 기술이 존재하는 반면, 소프트웨어에 대한 워터마킹 기술은 상대적으로 부족한 실정이다. 워터마킹 기술은 복사를 사전에 방지하는 것이 아니라, 불법복제가 발생했을 경우, 이를 검출 및 추적할 수 있는 기능을 제공하는 사후 검출 기술이며[7], 소프트웨어 워터마킹은 소프트웨어에 저작권 정보 또는 구매자 정보를 삽입하는 기술로서, 텍스트 파일이나 실행파일에 저작권 정보를 삽입하게 된다. 이러한 소프트웨어 워터마킹은 악의 있는 사용자가 프로그램의 알고리즘이나 모듈을 자신의 것이라고 주장할 때, 우리는 포함되어 있는 워터마크를 이용하여 저작권을 주장할 수 있다.

기존의 자바 프로그램에 대한 워터마킹 시스템은 SandMark[8], JavaWiz[9], Jmark[10]가 있다. SandMark 와 JavaWiz 는 패키지 단위의 워터마킹 시스템이며, Jmark 는 난독기(obfuscator)와 같은 공격에 대하여 충분하게 강인하지 못하다는 단점이 있다. 본 논문에서는 여러 가지 공격에 대한 충분한 강인성을 가지는 클래스 단위의 자바 프로그램 워터마킹 시스템을 제안한다. 본 시스템은 워터마크를 힙 영역에 저장하기 때문에 왜곡 공격에 강한 장점이 있다.

2. 관련 연구

2.1 소프트웨어 보호

소프트웨어를 보호하는 방법에는 암호화를 이용하는 방법과 소프트웨어 사용자의 목록을 만드는 방법이 있다. 사용자 목록을 만드는 방법은 등록이 되어 있는 사용자에게만 사용 허가를 주어 불법 사용자로부터 소프트웨어를 보호하는 것으로 프로그램 사용자가 목록에 존재하지 않으면 그 사용자는 불법 사용자로 간주한다[5]. 자바 프로그램의 보호 방법은 난독기를 이용하여 사용자가 역-컴파일 했을 때 소스 코드

를 알기 어렵게 하는 방법이 있다. 애플릿에 대한 보호는 인증을 이용하는 방법이 있지만 이 방법은 클래스 파일과 인증 부분이 분리되어 있기 때문에 실제로 자바 프로그램을 불법 사용자로부터 보호 하지는 못한다.

2.2 소프트웨어 워터마킹

워터마크를 삽입하는 가장 간단한 방법은 초기화되는 데이터 부분에 저작권을 삽입하는 것이다. 기존의 소프트웨어에 대한 워터마킹은 정적 워터마킹으로 (그림 1)과 같이 정적인 코드에 Copyright 나 고유번호 등을 삽입하는 방법을 사용하였다. 이러한 방법은 소스 코드에서 쉽게 발견될 수 있기 때문에 불법 사용자로부터의 보호가 어렵다.

```
/* My Software Copyright */
System.out.println("My Software");
```

(그림 1) Copyright 표현

Collberg 와 Thomborson 은 실행 상태(힙 영역)에 워터마크를 삽입하는 동적 워터마킹 기술을 제안하였다[6]. 동적 워터마킹은 소스 코드에서는 일반적인 코드처럼 보이지만 특정한 입력이 있으면 삽입된 워터마크가 나타난다. 대표적으로는 Easter Egg 가 있다[11]. 이것은 개발자가 프로그램 내부에 의도적으로 숨겨놓은 코드를 말한다. Easter Egg 는 쉽게 드러나기 때문에 제거가 쉽다.

힙이나 스택에 정보를 숨기는 것은 더 좋은 해결책이 될 수 있다. 자바는 이러한 영역에 정보를 숨기는 것은 쉬우나 자바는 여러 개의 클래스로 분리가 가능하기 때문에 불법 사용자가 특정 클래스만 사용하게 되면 워터마크를 숨기는 것은 어렵게 된다[6].

2.3 워터마킹의 조건

워터마킹은 다음과 같은 조건을 갖춰야 한다.

강인성: 워터마크의 제거가 어려워야 한다.
은밀성: 워터마크의 위치를 알 수 없어야 한다.
삽입비율: 상대적으로 작은 원본 메시지에 많은 워터마킹 정보를 포함할 수 있어야 한다.
투명성: 삽입된 워터마크는 프로그램의 특성을 변경시키면 안된다.

2.4 워터마킹 시스템에 대한 공격

불법 사용자는 워터마크가 포함된 프로그램에 대하여 워터마크를 제거하려고 할 것이다.

공격의 형태는 다음과 같은 것들이 있다.

감산 공격(subtractive attack): 워터마크의 위치를 발견하여 워터마크를 제거할려고 함.
왜곡 공격(distortive attack): 프로그램을 변경하여 워터마크 정보를 알아보지 못하게 함.
추가 공격(additive attack): 별도의 워터마크를 프로그램에 삽입하여 저작권을 주장함.
공모 공격(collusive attack): 서로 다른 워터마크가 포함된 프로그램을 가진 두 사용자가 공모하여 공통적인 부분을 찾아 워터마크를 알아내고자 함.

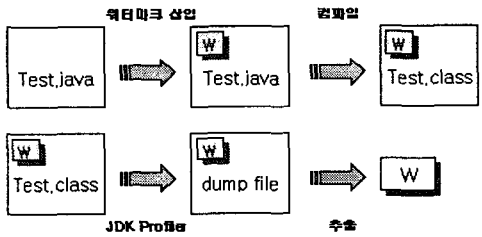
따라서, 워터마킹 시스템을 구성함에 있어서 불법 사용자의 다양한 공격에 대한 방어를 고려해야 한다.

3. 제안 시스템

3.1 시스템의 구조

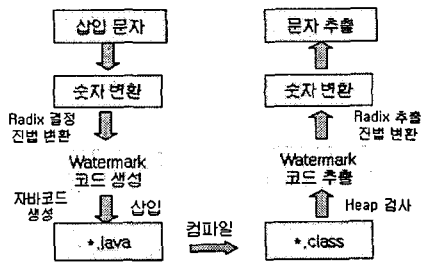
정적 데이터나 일반적인 데이터에 특정 워터마크를 숨기면 난독기에 의해 쉽게 변형 되기 때문에 본 논문에서 제안한 시스템은 워터마크 정보를 힙 영역에 저장한다. 힙 영역에 워터마크를 저장하는 방법은 기존에도 있지만[8,9], 기존의 기술은 응용프로그램 전체에 대하여 워터마크를 삽입하는 반면, 제안된 시스템은 하나의 특정 클래스 파일에 대하여 워터마크를 삽입한다.

(그림 2)는 제안 시스템의 흐름도이다. 자바 파일에 워터마크를 삽입하고 컴파일 한 후 워터마크가 삽입된 클래스 파일을 일반 프로그램 사용자에게 배포한다. 만약 악의 있는 사용자가 이 프로그램을 불법으로 사용하여 저작권을 주장하면 개발자는 프로그램의 실행 상태의 힙 영역을 분석하여 워터마크를 추출할 수 있다.



(그림 2) 제안 시스템 흐름도

(그림 3)은 제안 시스템의 전체적인 구조이다. 특정의 문자를 숫자로 변환 후, 워터마크 코드를 생성하고 이 코드를 자바 소스 파일에 삽입한다. 추출 시 힙 영역을 검사하여 숫자를 추출하여 문자로 변환 시킨다.



(그림 3) 제안 시스템 구조

3.2 워터마크 삽입과 추출

워터마크를 자바 파일에 삽입하는 방법은 먼저 특정 문자를 임의의 숫자로 변경하고, 이 숫자를 자바

파일에 삽입한다. 많은 문자는 그 문자를 표시하기 위해 많은 정보가 필요 하지만 숫자로 변경하여 어떤 수식을 적용하면 적은 숫자로 많은 정보를 나타낼 수 있다. 본 논문에서는 진법 변환에 의해서 큰 숫자를 몇 개의 숫자로 나누어 자바 파일에 삽입하였다.

진법 변환에 사용할 기수(Radix)의 결정 방법은 다음과 같다.

$$R = (r-1)r^{(r-1)} + (r-1)r^{(r-2)} + \ldots\ldots + (r-1)r^{0} \quad (\text{식 } 1)$$

여기에서 R 은 워터마크로 삽입할 숫자보다 크거나 같은 최소의 수이고 이때의 r 을 기수로 한다.

(그림 4)는 문자를 숫자로 변경하여 삽입하는 예를 보여주고 있다.

```
"mark" => 230193515
radix 결정 => 9
진법 변환 => 531131122(9)
삽입 코드 => 5 3 1 1 3 1 1 2 2
```

(그림 4) 워터마크 코드 생성

이와 같이 변환된 워터마크 정보는 (그림 5)과 같이 자바 파일에 기록 된다.

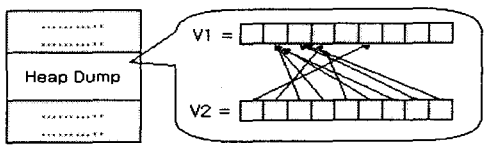
```
Vector v1, v2;
void insertWatermark(int code[]){
    // for example, code[] = {5, 3, 1, 1, 3, 1, 1, 2, 2}
    for (i=0; i<code.length; i++) {
        v2.addElement(v1.elementAt(code[i]));
    }
}
```

(그림 5) 워터마크를 삽입한 자바 소스 코드

추출은 삽입의 반대로 삽입된 숫자를 추출하여 다시 문자로 변환후 삽입된 문자를 인식한다. (그림 6)는 워터마크가 삽입된 프로그램을 JDK 에 포함되어 있는 -Xrunhprof 옵션(JDK1.2 이전버전은 -prof)을 이용하여 heap 영역을 조사한 것이다. 프로그램을 이 옵션을 사용하여 실행하면 결과물로 분석 정보가 있는 파일이 생성된다. 파일명은 기본적으로 java.hprof.txt 이며, 파일명은 -Xrunhprof:file=<파일명>으로 지정할 수 있다.[12] 분석된 파일의 HEAP DUMP 부분에서 사용된 영역을 조사하면 (그림 6)과 같은 영역이 발견된다. 이 영역을 그림으로 보면 (그림 7)과 같다. 이것은 $531131122_{(9)}$ 를 나타낸다.

```
OBJ 8153c78 (sz=24, trace=456, class=java.util.Vector@80bf3e0)
elementData 8153cc8
ARR 8153cc8 (sz=56, trace=457,  .....)
[0]     81540f0
[1]     8154190
    .....
OBJ 8153d60 (sz=24, trace=458, class=java.util.Vector@80bf3e0)
elementData 8153e08
ARR 8153e08 (sz=56, trace=459,  .....)
[0]     8154250
[1]     81541f0
.....
```

(그림 6) java.hprof.txt



(그림 7) heap file 분석

(그림 8)은 실제 워터마크를 삽입하는 모습이다. "chowing@pl.changwon.ac.kr"이라는 문자를 Test.java 소스 파일에 삽입시키면 기존의 자바 파일은 텍스트 파일로 복사되고 워터마크가 삽입된 새로운 자바 파일이 생성된다.

```
[@ cwm] % cwm -i "chowing@pl.changwon.ac.kr" Test.java
        watermarking ...........
        radix =>  35L
        insert watermark ..........
        java file edit ..................................
        Rename Java source ......Test.txt

        Test.java  watermark insert success......
[@ cwm] %
```

(그림 8) 워터마크 삽입

사용자가 자바 파일에 포함되어 있는 워터마크 코드를 발견하게 되더라도 그 코드와 대응되는 워터마크 정보를 알기는 어렵다.

(그림 9)는 추출하는 모습이다. -Xrunhprof 옵션으로 먼저 dumpfile 을 생성한다.

java –Xrunhprof:file=dumpfile Test

시스템은 이 파일을 조사하여 워터마크를 추출하게 된다.

```
[@ cwm] % java -Xrunhprof:file=dumpfile Test
Dumping Java heap ... allocation sites ... done.
[@ cwm] %
[@ cwm] % cwm -e dumpfile Test
watermark code find .....................
WATERMARK..........

        chowing@pl.changwon.ac.kr
```

(그림 9) 워터마크 추출

기존의 시스템인 SandMark 와 JavaWiz 는 다른 클래스를 사용하여 객체를 생성한 후 힙 영역에 워터마크를 삽입하는 반면, 본 시스템은 Vector 를 이용하여 자바 소스 파일에 직접 삽입함으로써 클래스 단위의 워터마킹이 가능하게 하였다.

4. 실험 및 고찰

기존의 여러 가지 자바 프로그램 워터마킹 시스템을 여러 가지 공격 가능한 형태로 워터마크가 삽입된 프로그램에 대하여 실험을 하였다. 공격의 형태는 난독기를 이용하는 방법, 최적화시키는 방법, Decompile 과 Recompile 을 이용 하는 방법에 대하여 실험 하였다. 실험된 자바 프로그램은 일반적인 몇 개의 프로그램을 사용 하였고, 난독기는 KlassMaster[13]를, 최적화

는 Bloat[14]을 이용하였고, Decompile 과 Recompile 은 Jad 와 JDK1.3 을 사용하였다.

실험에 사용된 기존의 자바 워터마킹 시스템은 다음과 같다.

SandMark – Radix 방법으로 힙 영역에 그래프를 삽입시킨다.
JavaWiz – 그래프의 형태 Enumeration 을 힙 영역에 삽입시킨다.
Jmark – 소스 파일에 dummy method 를 삽입하여 클래스 파일에서 워터마크를 삽입시킨다.

	제안시스템	SandMark JavaWiz	Jmark
실행단위	클래스	패키지	클래스
난독공격	강함	강함	약함
최적화	강함	강함	중간
Decompile Recompile	강함	강함	강함

(그림 10) 제안 시스템과의 비교

난독공격과 최적화, Decompile-recompile 공격은 제안된 시스템에게 영향을 주지 않는다. 불법 사용자는 최적화나, Decompile 할 수도 있고, bogus 분기나, 임의의 단계를 추가할 수도 있다. 그러나 프로그램이 충분히 크면, 일반적인 분석 도구를 사용하지 않고는 워터마크의 위치를 알 수 없기 때문에 모든 부분을 조사 할 것이고, 이것은 비용이 많이 든다.

제안 시스템은 동적 워터마킹 기술을 이용하였다. 따라서 소스 코드의 필드 변경으로는 워터마크를 제거 할 수 없기 때문에 난독기와 최적화에 충분히 강하다. 그리고, 워터마크의 삽입 위치를 알 수 없어도 프로그램을 실행 시켜 실행시의 힙 영역을 조사하여 워터마크를 추출 할 수 있다.

5. 결론

최근 불법 소프트웨어의 사용이 증가하고 있고, 이에 따라 저작권 보호문제가 대두 되고 있다. 본 논문에서는 소프트웨어의 저작권을 주장하기 위한 방법으로 워터마크 기술을 연구하였다. 그러나 이러한 워터마크 기술은 멀티미디어에 대한 것은 많이 존재하는 반면 소프트웨어에 대한 것은 아직 미약하다.

본 논문에서는 쉽게 역-컴파일 가능한 자바 프로그램에 대한 워터마크 삽입기술을 제안 하였다. 논문에서 제안된 자바 프로그램 워터마킹 방법은 특정한 클래스에 적용이 가능 하고, 실행시에 워터마크가 추출되기 때문에 난독기와 최적화에 강하다. 그러나 워터마크의 기술만으로는 자바 프로그램을 완전하게 보호할 수 없다. 이러한 기술은 난독과 Tamperproofing 과 같은 보호 기술과 병행되어 사용 되어야 한다.

참고문헌

[1] Ahpah Software, SourceAgain
http://www.ahpah.com/sourceagain

[2] Kouznetsov, P. jad –the fast Java Decompier
http://www.geocities.com/SiliconValley/Bridge/8617/jad.html

[3] Vliet, H., Mocha –the Java Decompiler,
http://www.brouhaha.com/~eric/computers/mocha.html

[4] A. Monden, H. Iida, K. Matsumoto, K. Inoue, K. Torii, "A Practical Method for Watermarking Java Programs", The 24th Computer Software and Application Conference (compsac2000), Taipei, Taiwan, Oct. 2000

[5] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Yi Zhang, "Experience with Software Watermarking" 16th Annual Computer Security Applications Conference Dec. 11-15, 2000

[6] C. Collberg, C. Thomborson, "Software watermarking: Model and dynamic embeddings," The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages(POPL'99), San Antonio, Texas, Jan.1999

[7] 임양규, 국상진, 전종민, 원동호, "워터마킹 기술의 평가 기준에 관한 연구", 한국 정보 처리 학회 춘계 학술발표 논문집 제 8 권 제 1 호, 2001

[8] SandMark: Software Watermarking for Java
http://www.cs.arizona.edu/sandmark

[9] JavaWiz Watermarking System
http://www.cs.purdue.edu/homes/madi/wm

[10] Jmark : Digital watermarking tools for Java class files
http://tori.aist-nara.ac.jp/jmark

[11] The Easter Egg Archive ,http://eeggs.com

[12] Jack Shirazi "Java Performance Tuning", o'relly

[13] Zelex KalssMaster - Java code obfuscator and obfuscation. <http://www.zelix.com/ >

[14] The Bytecode-Level Optimization and Analysis Tool <http://www.cs.purdue.edu/homes/hosking/bloat/>