Java 프로그램에 적용가능한 소프트웨어 워터마킹

신 원*

A Software Watermarking for Java Programs

Shin, Weon*

요 약

컴퓨터 소프트웨어의 불법 복제는 정보화사회의 가장 큰 위협 중 하나이다. 이를 해결하기 위한 수많은 불법복제 방지 기술이 등장하였지만 급속히 발전하는 인터넷을 통한 배포를 막기에는 현실적으로 다양한 어려움이 존재한다. 본 논문에서는 인터넷 환경에서 많이 사용되는 Java 프로그램을 대상으로 저작권 보호를 위한 소프트웨어 워터마킹 방안을 제안한다. 제안 방안은 효율적인 구현이 가능하고 실행 파일 저작권 보호를 위해 강건한 소프트웨어 워터마크를 제공한다.

ABSTRACT

The illegal copy of computer software have been one of the most serious threats in information society. But a number of protection schemes to solve it have proposed, there are a lot of difficulties for prevention from illegally distributing software through emerging Internet environments. In this paper, we propose a software watermarking scheme for Java programs which it have been popular on the Internet. The proposed scheme can be efficiently implemented and provide a robust software watermark for the protection of the copyright of Java programs.

키워드

소프트웨어 워터마킹, 디지털 워터마킹, 자바 프로그램, 저작권

Key word

Software watermarking, Digital watermarking, Java program, Copyright

* 동명대학교 정보보호학과 (shinweon@tu.ac.kr)

접수일자 : 2010. 03. 12
심사완료일자 : 2010. 09. 03

Ⅰ. 서 론

미디어 및 정보통신 기술의 발달로 인터넷을 통하여 텍스트, 이미지, 사운드, 동영상 등의 멀티미디어 콘텐츠와 이를 실행할 수 있는 소프트웨어를 누구나 쉽게 얻을 수 있고 이를 다시 복제, 배포할 수 있는 환경이 되었다. 이는 모든 미디어가 컴퓨터가 처리할 수 있는 디지털 데이터로 가공됨으로써 더 이상 원본과 복사본의 구분이 불가능하다는 것을 의미한다. 원본과 동일한 품질의 디지털 데이터를 거의 무한정으로 복제하고 인터넷을 통하여 빠른 시간에 배포할 수 있다는 것은 정보화 사회에서의 가장 큰 장점인 동시에 콘텐츠와 소프트웨어의 저작권에 대해 새로운 위협이 되고 있다. 국내외 저작권법 논쟁, 반복되는 소프트웨어 불법 복제는 이러한 문제점을 극명하게 보여주는 실례이며 근본적인 해결 방안이 등장하지 않는 한 이와 같은 문제가 얼마든지 반복될 수 있다.

최근에는 콘텐츠의 복제는 허용하도록 하고 사용 권한에 제한을 두어 원작자의 권리를 사전에 보호하는 방식인 디지털저작권 관리[1]와 콘텐츠에 삽입되어 있는 특정 정보를 추적하여 원작자 또는 불법 배포자를 가려내는 사후 검출 방식인 디지털 워터마크[2] 기술이 구현되어 디지털 저작권을 보호하기 위한 목적으로 사용되고 있다. 그러나 디지털 콘텐츠를 보호하기 위해서는 다양한 방법이 동원되고 있으나 디지털 콘텐츠를 구동하고 제어하기 위한 실행 파일에 대한 보호기술은 전무할 뿐만 아니라 인식조차도 희박한 상태이다. 최근 해킹, 악성코드, 저작권 위협 등의 다양한 사례를 살펴보면 디지털 콘텐츠 뿐만 아니라 소프트웨어 실행 파일을 보호하기 위한 연구도 함께 진행되어야 한다. 또한 실행 파일의 저작권을 보호하기 위한 연구뿐만 아니라 실행 파일의 무결성을 함께 보장할 수 있는 방안도 마련되어야 한다.

본 논문에서는 Java 프로그램에 적용가능한 소프트웨어 워터마킹 방안을 제안하고자 한다. 먼저 2장에서는 기존 연구와 소프트웨어 워터마킹에 대하여 살펴보고, 3장에서는 Java 언어에 안전한 소프트웨어 워터마킹 방안을 제안한다. 4장에서는 제안 방안을 분석한 후 응용 분야를 살펴보고, 마지막 5장에서 결론을 맺는다.

Ⅱ. 소프트웨어 워터마킹 개요

2.1 소프트웨어 워터마킹의 정의

소프트웨어 워터마킹은 “소프트웨어에 비밀리에 정보를 삽입하는 절차”이다[3]. 여기서, 정보는 소프트웨어의 소유권을 구별할 수 있는 정보[3]로, 저작권자가 소프트웨어에서 비밀 정보를 추출함으로써 허가되지 않은 소프트웨어 복제를 증명하는 수단으로 사용된다. 즉, 기본 개념은 디지털 콘텐츠의 저작권을 보호하기 위해 삽입하는 디지털 워터마킹[2]과 거의 유사하지만, 워터마킹을 적용하는 방식과 워터마크를 삽입하는 절차는 완전히 다르다.

또한, 소프트웨어 워터마킹은 응용방식에 따라 단순히 저작권을 증명하는 목적으로 사용하는 것 이외에도 소프트웨어 실행 파일에 직접 적용한다면 소프트웨어 실행 파일에 대한 무결성 검증에 응용할 수도 있다. 이와 같이 소프트웨어 워터마킹은 저작권 보호는 물론 소프트웨어 실행에 대한 무결성 보장, 소프트웨어 실행 파일에 대한 접근제어 등에 응용할 수 있으며 복잡한 암복호화 기술 적용이 힘든 모바일 환경 및 임베디드 환경에 적절하게 적용할 수 있는 방안이라 할 수 있다.

2.2 기존연구

디지털 워터마킹은 어떤 정보에 특정 비밀 정보를 삽입하는 기술로, 멀티미디어 콘텐츠의 저작권을 보호하기 위하여 많이 사용되어 왔다. 즉, 워터마크는 디지털 콘텐츠의 저작권 보호를 목적으로 쉽게 감지하기 어렵도록 디지털 이미지, 오디오 또는 비디오 신호에 저작권 정보를 삽입하여 멀티미디어 데이터에 대한 소유권을 보호할 수 있으는 기술이다[2]. 그러나 소프트웨어 워터마킹은 C/C++, Java 등 특정 언어로 개발된 프로그램에 저작권 정보를 삽입하여 프로그램의 소유권을 주장하는 방식으로 기존 디지털 콘텐츠 위주의 워터마킹과 그 개념은 유사하지만 적용 방식과 내부 동작은 매우 다르다.

Collberg 등[5]은 프로그램 실행 상태에서 동적 워터마크(Dynamic Watermark)가 저장되는 방안을 제안하였으나, 하나의 소스로 구성된 단일 프로그램에만 적용할 수 있고 Java와 같이 다양한 클래스로 이루어진 언어에

서는 적용할 수 없는 문제점을 가진다. Jamal 등[4]은 동적 워터마킹을 워터마크 삽입 방식에 따라 구분하고, 다양한 기존의 동적 워터마킹 방안들을 분석하였다. Zhang 등[6]은 전통적인 워터마킹과 차별화하여 해쉬 기반의 알고리즘을 제안하였다. 워터마크를 삽입하기 위해 프로그램을 분할하고, 파라메터에 따라 다른 워터마크를 삽입하는 방식이다. 제안 방안은 의미상의 분석만을 통하여 특정 정보를 삽입, 추출할 수 있는 단점을 가진다. Monden 등[7]은 Java 바이트코드에 워터마크를 삽입하는 방법을 제안하였다. Java 소스에 더미 메소드를 삽입하고 이를 컴파일한 후 바이트코드의 더미 메소드에 해당하는 부분을 수정하여 워터마크를 삽입하는 방식이다. 이 방안은 Java 언어에만 적용가능한 정적 워터마크 방식이다.

Ⅲ. 새로운 소프트웨어 워터마킹 방안

3.1 안전한 소프트웨어 워터마킹의 요구사항

안전한 소프트웨어 워터마킹을 위한 구체적인 요구사항은 다음과 같다.

○ 비가시성 (Invisibility)
가장 중요한 요구사항으로 워터마크가 삽입되더라도 원본의 품질에 중대한 영향을 미쳐서는 안된다. 즉, 워터마크 삽입 전후를 비교하여 사용자가 인지할 수 없어야 한다.

○ 견고성 (Robustness)
워터마크는 여러 가지 비의도적 공격이나 의도적 공격에 강해야 한다. 비의도적 공격은 컴파일 환경에서 발생할 수 있는 최적화, 포맷 변경 등을 포함하고, 의도적 공격은 불법적으로 도용하기 위한 목적으로 시도되는 데이터 삽입, 고의적인 조작 등을 포함한다. 다양한 종류의 공격에도 불구하고 워터마크는 여전히 검출되어야 한다.

○ 워터마크 삽입량 (Capacity)
소유자, 고유번호, 제작날짜 등 워터마크의 저작권에 대한 정보를 충분히 삽입할 수 있어야 한다. 많은 정보를 수용하기 위해서는 큰 삽입량을 확보해야 하지만, 견고성을 만족시키기 위해서는 삽입량이 가능한 작아야 한다.

○ 검출률 (Detection Rate)
검출률은 정확하게 워터마크를 검출할 확률을 나타내는 것으로, 워터마크를 제대로 탐지하지 못하는 비율인 FNR(False Negative Rate)과 워터마크가 아닌데 워터마크로 판정하는 비율인 FPR(False Positive Rate)로 나뉜다. 특히, FPR은 삽입된 값이 아닌 다른 값을 워터마크로 오인할 확률로 워터마크의 진위를 가리는데 매우 중요하므로 작은 값을 갖도록 설계되어야 한다.

○ 복잡성 (Complexity)
워터마크는 충분히 복잡한 특성을 가지고 있어서 신뢰할만한 통계적 성질을 제공하고, 제 3자에 의한 워터마크의 검출이 어려워야 한다.

3.2 안전한 소프트웨어 워터마킹 방안의 제안

본 연구에서는 프로그램 내에서 특정한 패턴매칭을 이용하는 "구문적 변환"과 같은 의미의 다른 코드로 변경하여 소스 코드의 해석이 어렵도록 하는 "의미론적 변환"을 사용하여 위의 요구사항을 만족하는 새로운 방안을 제안한다. 제안 방안의 소프트웨어 워터마크 생성과 검출은 다음과 같이 구성된다.

워터마크 생성 :

$$w = W(k)$$

w는 비밀키 k를 워터마크 생성 함수 W에 적용하여 생성된 워터마크

$$wp = E(op, w)$$

워터마크가 적용된 프로그램 wp는 워터마크 삽입 함수(Embedding Function) E를 적용하여 원래 프로그램 op에 워터마크 w를 삽입

워터마크 검출 :

$$w = X(wp)$$

워터마크 w는 워터마크가 적용된 프로그램 wp에서 워터마크 추출 함수(Extraction Function) X를 적용하면 추출 가능

$$D(wp, k) = true \text{ or } false$$

또한, 워터마크 검출 함수(Detection Function) D는 워터마크가 적용된 프로그램 wp에서 워터마크 w와 비밀키 k가 있는지 검출하는 함수

특히, 제안 방안은 다음의 성질을 만족한다. 첫째, 악의가 있는 사용자에 의해 의미론적 변환이 행해진다라고 해도 삽입한 워터마크는 제거되지 않고 남아야 한다. 둘째, 가령 구문적 변환이 행해졌다 하더라도 저작권자는 삽입된 워터마크를 추출하는 것이 가능하다.

3.3 제안 방안의 구현

제안 방안을 Java 프로그램으로 구현하는 경우, 일반적인 방식을 사용하면 역컴파일을 통하여 워터마크를 쉽게 열람할 수 있으므로 코드 난독화(Code Obfuscation)[5][8]를 적용하여 이를 막을 수 있다. 여기서, 코드 난독화란 프로그램 변환의 일종으로 코드를 읽기 어렵게 함으로서 역공학(Reverse Engineering)을 통한 공격을 막기 위한 목적을 가지고 있다[5][8]. 결과적으로 프로그램 동작과 해석을 매우 어렵도록 원래의 소스와는 판이하게 변경시키지만 실행 결과는 동일하다. 구체적인 예는 그림 1과 같다.

```
int[] w = new int[1000];

w[0]=5; w[1]=9; w[2]=5; w[3]=3;
```

⇓

```
int[] w = new int[1000];
int i;

w[0]=5; w[1]=24; w[2]=67; w[3]=32;

for(i=2; i>=(w[0] %4); i--)
    w[i] = i + 1;

w[w[2]-w[1]] = w[0]*w[2]-6;
w[(w[1]-w[0])/2] += w[0]/2;
w[w[1]/3] = w[2]/w[0]*3;
```

그림 1. 코드 난독화 적용
Fig. 1 An example of code obfuscation

먼저 워터마크 삽입 알고리즘을 통하여 배열에 임의의 워터마크 w를 삽입한다.

워터마크 삽입 알고리즘
① 키를 적용한 워터마크 생성 $w = W(k)$
② 충분한 크기의 배열을 준비
③ 배열의 특정 위치에 워터마크 삽입

제안 방안에서는 프로그램의 배열에 워터마크 정보를 삽입하고 비밀리에 보관한다. 그러나, 단순히 배열에 워터마크 정보를 삽입한다면 공격자에 발견될 가능성이 있기 때문에 코드 난독화를 적용해서 워터마크의 역할을 하는 배열을 은닉한다. 만일, 배열 내용이 변경된 경우에는 프로그램이 바르게 동작하지 않도록 한다. 다음은 이를 반영한 워터마크 추출 알고리즘이다.

워터마크 추출 알고리즘
① 임의의 값 v를 입력
② 워터마크 검출 함수 호출 준비
③ if($D(wp, k) == v$){
  배열 내 워터마크 위치 정보 읽기
  워터마크 w 추출 후 출력
}

이후 컴파일을 통하여 실행 파일을 제작하고 배포하여 원래의 목적대로 사용한다. 만약 저작권 문제가 발생하면 워터마크 검증을 통하여 저작권을 위배하는지 아닌지를 판단한다. 실행 파일과 함께 배포되는 워터마크 검증 알고리즘은 다음과 같다. 실제 구현시에는 코드 난독화를 적용한다.

워터마크 검증 알고리즘 I
① 임의의 값 v 입력
② 비밀키 k가 v와는 항상 다른 값이 되도록 설정
③ 따라서, 항상 다음을 수행
  if($v \neq k$) return “false”
※ 별도의 클래스로 제작하여 배포

워터마크 검증 알고리즘 II
① 임의의 값 v 입력
② 비밀키 k가 v와는 항상 같은 값이 되도록 설정
③ 따라서, 항상 다음을 수행
if($v == k$) return "true"
※ 별도의 클래스로 제작하여 검증 시 교환

즉, 일반 사용시에는 검증 모듈이 활성화되어 있지 않으므로 워터마크를 추출할 수 없다. 그러나 워터마크를 검증할 때, "워터마크 검증 알고리즘 I"을 구현한 모듈을 "워터마크 검증 알고리즘 II"를 구현한 모듈로 교체함으로써 검증 모듈이 활성화되고 삽입한 워터마크를 추출할 수 있게 된다. 그러나, 배포될 것이 아니므로 굳이 코드 난독화를 적용할 필요는 없다.

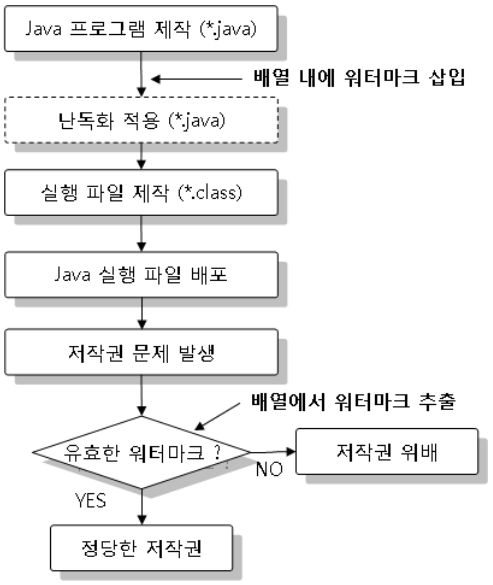

그림 2. 제안 방안
Fig. 2 The proposed scheme

그림 2는 제안 방안의 전체 과정을 보여준다. 그 중 워터마크 검증에 대해 자세히 설명하면 다음과 같다. 배포된 파일에서 워터마크 검증을 위해 모듈을 교체하여 활성화한다. 활성화된 검증 모듈을 실행하여 그 결과를 확인한다. 이때 검증 모듈은 실행 파일의 특정한 배열에 포함된 워터마크를 추출하고 그 결과를 출력한다. 저작권자가 삽입한 워터마크와의 일치여부를 확인하여 유효한 워터마크인지 아닌지를 판단한다. 단, 이 워터마크 검증 과정은 저작권 관련 이해당사자 합의 하에 공개된 방식으로 수행하고 그 결과를 확인한다. 그림 3은 세부 과정을 보여준다.

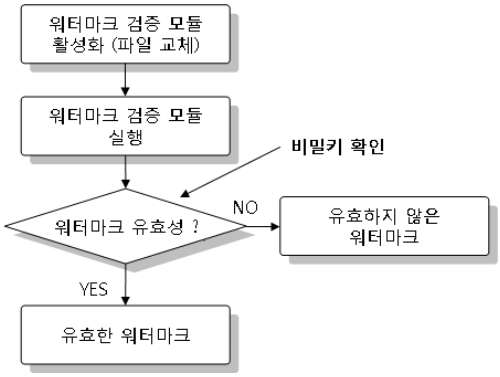

그림 3. 소프트웨어 워터마크 검증
Fig. 3 Verify software watermark

Ⅳ. 제안 방안의 분석과 응용

4.1 제안 소프트웨어 워터마킹 실험

제안 방안을 적용하여 실험하기 위해서 JDK (Java Development Kit)[9]에 포함된 예제 프로그램 5개를 이용하였고, 실험 결과는 표 1과 같다.

표 1. 실험 결과
Table 1. The results of experiments

소스	①	②	③
FileChooserDemo	12,259	12,331	12,673
Notepad	12,000	12,090	12,423
SampleTree	6,056	6,127	6,468
Stylepad	5,303	5,388	5,720
TableExample	5,704	5,775	6,116

여기서, 크기는 소스 파일(.java)을 컴파일하고 난 후

생성된 클래스 파일(.class)의 Byte 단위 크기로, ① 수정 없이 컴파일한 후 클래스 파일의 크기, ② 워터마크를 삽입하고 컴파일한 후 클래스 파일의 크기, ③ 워터마크를 삽입하고 난독화를 적용하여 컴파일한 후 클래스 파일의 크기이다. 원래의 클래스 파일 ①에 비해 워터마크만 삽입한 ②의 경우 0.5～1.2% 범위로 0.1KB 이내의 크기가 증가하고 난독화까지 함께 적용한 ③의 경우 3.4～7.9% 범위로 0.5KB 이내의 크기가 증가함을 알 수 있다. 따라서, 크기 증가가 미미하므로 실제 사용에 있어서도 큰 문제가 없다 할 수 있다.

4.2 제안 소프트웨어 워터마킹의 안전성

제안 소프트웨어 워터마킹에 대한 공격으로는 다음과 같은 공격이 있을 수 있다[10][11]. 워터마크가 삽입된 소프트웨어에 새로운 워터마크를 삽입하여 저작권자가 삽입한 원래의 워터마크로부터 소유권을 주장할 없도록 하는 “삽입 공격(Additive Attack)”, 워터마크가 삽입된 소프트웨어의 워터마크를 해당 소프트웨어와의 상관 관계에 영향을 미치지 않고 워터마크를 없애는 “제거 공격(Subtractive Attack)”, 저작권자에 의해 워터마크가 추출되는 것을 방지하지만 소프트웨어의 기능은 그대로 유지하도록 워터마크를 수정하는 “왜곡 공격(Distortive Attack)”, 잘못된 결과를 출력하도록 워터마크 검출기 또는 입력값을 수정하거나 사용하지 못하도록 하는 “인지 공격(Recognition Attack)”이 있다. 각각의 공격에 대한 제안 방안의 안전성은 다음과 같다.

제거 공격과 왜곡 공격을 수행하기 위해서는 워터마크의 구조와 위치를 알고 있어야만 가능한데, 제안 방안에서 배열을 다양하게 사용하고 임의의 위치 값을 워터마크 값으로 사용한다면 2가지 공격에 대해 안전하다. 또한 코드 난독화를 도입하면 역컴파일에 의해서도 해당 워터마크를 찾아내는 것은 현실적으로 어렵다. 삽입 공격은 워터마크의 구조와 위치를 모르더라도 임의의 값을 삽입하여 진행하는데, 제안 방안에서 워터마크 배열의 이름과 위치를 임의로 사용한다면 이를 이용한 공격도 역시 어렵다. 인지 공격은 워터마크를 추출하는 조건문에 대해 공격을 수행하거나 저작권자가 가지는 검증 모듈을 교체하면서 이루어진다. 조건문 및 검증 모듈은 코드 난독화를 수행하여 일차적으로 보호하도록 하고, 특히 검증 모듈은 저작권자가 안전하게 보관하여 불법 수정을 할 수 없게 한다면, 인지 공격도 쉽게 막을 수 있다.

4.3 제안 방안의 개선과 응용

제안 방안을 기본으로 안전성을 더 강화하거나 다른 응용 분야에 적용하기 위하여 다음과 같이 수정하거나 응용할 수도 있다.

첫째, 제안 방안은 워터마크 검증을 위하여 별도의 검증 모듈을 두는데, 이것이 어려운 경우 사용자 입력을 이용하여 워터마크를 추출하는 방안을 생각할 수 있다. 즉, 사용자가 정해진 특정값을 입력하면 이에 반응하여 워터마크를 출력하는 방식이다. 여기서, 출력값을 확인하는 조건문은 배포본에 포함되므로 코드 난독화 등을 사용하여 알기 어렵도록 구현되어야 한다. 그림 4는 사용자가 비밀키 ″123457″을 입력하면 워터마크를 출력하는 예이다. 둘째, 제안 방안에서 워터마크를 삽입할 때, 배포본마다 각각 다른 워터마크를 삽입하면 추후 저작권 문제 발생시 누가 유출하였는지 알 수 있다. 이것이 핑거프린팅(Fingerprint)[2]인데, 각 소프트웨어 배포본에 지문과 같은 핑거프린트를 삽입하면 소프트웨어 불법 복제에 대해 역추적할 수 있다. 이 경우 암호학적인 일방향 해쉬 함수[12]를 사용하면 안전성을 보장할 수 있다. 셋째, 클라이언트/서버 환경에서 각 클라이언트에 소프트웨어를 배포하고 주기적으로 해당 워터마크 검증 결과를 서버로 전송하게 되면 서버에서는 클라이언트에 설치된 소프트웨어의 변조 유무를 확인할 수 있다. 즉, 중간에 의도적 또는 비의도적으로 소프트웨어가 변경된다면 워터마크 결과가 다르게 나올 것이므로 이를 중앙에서 알 수 있다. 각 클라이언트에 설치되는 중요 소프트웨어의 경우 매우 효과적으로 소프트웨어의 무결성을 검증할 수 있게 된다. 넷째, 제안 방안은 Java 언어를 이용하여 쉽게 구현할 수 있으므로 일반 PC 환경은 물론 Java 기반의 스마트폰에서도 사용할 수 있다. 특히, 안드로이드(Android)[13] 운영체제는 Java에 기반하므로 거의 수정없이 사용할 수 있다. 스마트폰 기반의 어플리케이션에서 저작권 문제가 대두되는 경우, 제안 방안은 스마트폰 환경에서도 효율적으로 사용할 수 있다.

```
int value = 123457;
Scanner sc = new Scanner(System.in);
int input = sc.nextInt();

if( input == value )
  워터마크 추출 후 출력
```

그림 4. 사용자 입력값으로 워터마크 출력
Fig. 4 Print watermarks by user's input

Ⅴ. 결론

최근 인터넷 및 컴퓨터의 보급으로 수많은 소프트웨어가 사용되고 있다. 이러한 소프트웨어의 배포 방식으로 네트워크를 사용한 소프트웨어 배포가 사용되고 있지만 불법 복제가 사회적인 문제가 되고 있다. 즉, 소프트웨어는 디지털 데이터이기 때문에 원본과 사본의 구분이 불가하므로 복제 자체를 원천적으로 방지하는 것은 매우 어려운 문제이다. 그러나 소프트웨어에 디지털 워터마크를 삽입하여 불법 복제가 발견되었을 경우, 저작권자는 소프트웨어에 삽입되어 있는 워터마크를 추출하여 부정 여부를 판정할 수 있다.

지금까지는 이미지, 음성, 동영상 등 디지털 콘텐츠를 중심으로 워터마크가 연구되어 왔으나 그 개념을 소프트웨어에도 적용가능하다. 그러나 소프트웨어는 디지털 콘텐츠에 비교하여 중복성(Redundancy)이 낮고, 프로그램의 동작과 제어에 영향을 끼치면 안되기 때문에 종래의 디지털 워터마크 기술을 그대로 적용하는 것은 불가능하다. 본 논문에서는 Java 프로그램에 적용가능한 디지털 워터마크 방안을 제안하고 구현 방안을 제시하였다. 제안 방안은 컴퓨터 상에서 동작 가능한 소프트웨어는 물론 Java 프로그램이 사용되는 스마트폰에서도 안전한 소프트웨어 배포와 저작권 확인에 효과적으로 적용할 수 있다.

참고문헌

[1] 강호갑, DRM 최신 국제표준 기술사양분석 및 세계 유명제품 동향과 전망에 관한 연구, 한국소프트웨어진흥원, 2004.

[2] 오병택, 문병주, 이동일, "디지털 워터마킹 기술동향 및 전망", 전자통신동향분석 제17권 제6호, pp. 155-162, 2002

[3] C. Collberg, S. Jha, D. Tomko and H. Wang. "UWStego: A General Architecture for Software Watermarking", Technical Report(Aug. 31, 2001), available on http://www.cs.wisc.edu/ hbwang/ watermark/TR.ps on Nov. 20 , 2004

[4] S. Jamal H. Zaidi and Hongxia Wang, "On the Analysis of Dynamic Software Watermarking", International Journal of Computer Science and Security, vol.4 issue 2, 2010

[5] Christian S. Collberg, C. Thomborson, "Watermarking, tamper-proffing, and obfuscation: tools for software protection", IEEE Transactions on Software Engineering, v.28 n.8, p.735-746, August 2002

[6] X. Zhang, F. He and W. Zuo. "Hash Function Based Software Watermarking". In Proc. of ASEA 2008, IEEE Computer Society Conference, pp. 95-98, 2008

[7] A. Monden, H. Iida, K. Matsumoto, K. Inoue, K. Torii, "A Practical Method for Watermarking Java Programs", Computer Software and Applications Conference 2000, pp. 191-197, 2000.

[8] 이병용 최용수, "Obfuscation 기술의 현황 및 분석과 향후 개발 방향", 보안공학연구논문지, Vol. 5, No. 2, pp. 43-52, 2008

[9] Developer Resources for Java Technology, http://java.sun.com/

[10] W. Zhu1, C. Thomborson and F. Wang, "A Survey of Software Watermarking", Lecture Notes in Computer Science, Vol 3495, pp. 454-458, 2005

[11] W. F. Zhu, "Concepts and Techniques in Software Watermarking and Obfuscation", Ph.D Thesis, The Department of Computer Sciences, The University of Auckland, 2007

[12] A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, Boca Raton, FL, 1996
[13] Android at Google I/O, http://www.android.com/

저자소개

신원(Shin, Weon)

2005.3～현재 동명대학교
정보보호학과 조교수
2002.3～2005.1 (주)안철수연구소
선임연구원

※관심분야 : 악성코드 확산, 디지털 포렌식, 소프트웨어 보안, 암호 프로토콜 응용