Dalvik명령어 유사도 비교를 통한 안드로이드 표절 탐지 기법

최성하, 황나현, 박희완
한라대학교 정보통신방송공학부
e-mail: shoutme1991@nate.com, nahyun66@naver.com,
heewanpark@halla.ac.kr

Android plagiarism detection through Dalvik instruction similarity comparison

Sung-ha Choi, Na-hyun Hwang, Heewan Park
Dept of Information Communication Broadcasting Engineering
Halla University

요 약

스마트폰 애플리케이션 중에서 안드로이드 앱은 자바를 기반으로 한다. 따라서 자바 프로그램과 마찬가지로 디컴파일러 도구를 활용하여 원본 소스 코드를 얻어낼 수 있기 때문에 코드 도용에 대해서 매우 취약하다. 본 논문에서는 안드로이드에 대한 코드 도용과 표절을 막기 위한 기법을 제안한다. 효과적인 코드 도용 및 표절 여부를 탐지하기 위한 방법으로서, 안드로이드 달빅(Dalvik) 코드에 대해서 요약 단계를 거친 후 유사도를 측정하는 방법을 사용한다. 기존의 안드로이드 유사도 비교 연구에서는 달빅 코드가 정확하게 일치해야만 유사도가 높게 측정될 수 있었지만, 요약 단계를 통해서 변환된 달빅 코드를 비교하면 코드 도용시 일부 코드의 의도적인 수정이 있더라도 유사도가 높게 측정된다. 그 결과, 본 논문에서 제안하는 표절 탐지 기법이 기존 연구와 비교하여 표절에 대한 탐지 능력이 우수함을 확인하였다.

1. 서론

일반적으로 소프트웨어는 역공학 분석이 어려운 바이너리(exe, dll, ocx) 상태로 배포되기 때문에 프로그램으로부터 원본 소스를 얻어내는 것은 매우 어렵다. 그러나 자바는 가상머신 환경에서 동작하는 클래스 형태로 배포된다. 또한 자바 클래스로부터 역으로 소스를 얻을 수 있는 다양한 디컴파일러[1,2]가 개발되었기 때문에 클래스 파일로부터 자바 소스를 얻어낸 후 일부 코드를 그대로 도용하거나 일부 수정하는 작업을 거쳐서 표절하는 행위가 가능하다.

스마트폰 애플리케이션 중에서 안드로이드 앱은 자바를 기반으로 한다. 따라서 자바 프로그램과 마찬가지로 쉽게 디컴파일러 도구를 활용하여 원본 소스 코드를 얻어낼 수 있어 코드 도용에 대해서 매우 취약하다.

본 논문에서는 안드로이드 앱의 코드 도용과 표절을 막기 위한 방법으로써 앱 사이의 유사도를 측정하는 도구를 제안하고 구현한다. 원본 프로그램과 도용 및 표절된 프로그램은 유사도가 높게 측정될 것임을 예상할 수 있기 때문에 유사도 측정 프로그램은 코드 도용이나 표절을 확인하는 용도로도 활용 가능하다. 본 논문에서 제안하는 표절 탐지 도구는 안드로이드 달빅(Dalvik) 바이트 코드를 특성에 따라서 분류하고 요약한 후, 요약된 코드에 대한 유사도를 측정한다. 따라서 기존 유사도 비교 방법에서 사용하던 비교 기법과는 차이가 있다.

2. 관련 연구

코드 도용 및 표절에 대한 탐지 기법은 지속적으로 연구되어 왔으며 전통적인 소스 코드 표절 탐지 기법은 Attribute Counting 과 Structure Metric 방법으로 나눌 수 있다[3,4].

1) Attribute Counting 방법

Attribute Counting 방법은 사용된 단어나 키워드의 유사성 및 사용 빈도를 비교한다. 따라서 문서 길이에 영향을 받지 않고, 단락의 순서가 변경되어도 상관없다. 그러나 부분적 표절 탐지가 어렵다.

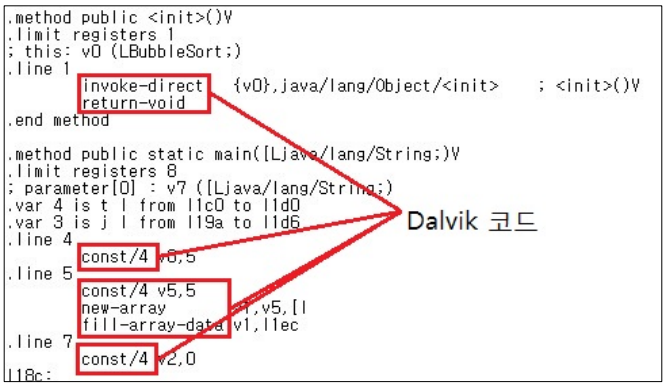
2) Structure Metric 방법

Structure Metric 방법은 토큰 문자열의 유사성을 계산하는 방식이다. 먼저 소스 코드를 토큰 문자열로 변환하고, 변환된 토큰 스트링을 비교한다. 이 방법은 문서의 길이와 순서에 영향을 받는다. 부분적인 표절 탐지가 쉽다.

다른 사람의 소스 코드를 도용하거나 표절한 사람은 도용 및 표절 사실을 숨기기 위한 노력을 기울일 것이다. 즉, 원본 소스와 도용된 소스 사이의 유사도를 낮추기 위한 방법을 고민할 것이다. 실행 결과는 동일하지만 대체 가능한 다른 명령어를 사용하거나, 실행 결과에 영향을 끼치지 않는 명령어들에 대해서 순서를 바꾸는 방법을 적용할 수도 있을 것이다. 코드 도용 및 표절 탐지 도구는 이러한 상황에서도 도용 및 표절을 탐지할 수 있어야 한다.

3. 안드로이드 달빅(Dalvik) 코드와 요약

3.1 안드로이드 달빅(Dalvik) 코드



(그림 1) 안드로이드 어셈블리의 Dalvik 코드 구성

안드로이드 프로그램은 달빅 가상머신[5]에서 실행 가능한 달빅 명령어들로 구성되어 있다. (그림 1)은 안드로이드 앱으로부터 추출한 달빅 코드의 예이다.

만일 어떤 사람이 코드 도용을 하여 프로그램을 개발하였다면 도용된 코드는 두 프로그램에 공통적으로 포함되어 있을 것이고 달빅 바이트 코드에서도 공통된 부분이 발견될 것이다. 그러나 만일 도용한 사람이 일부 코드를 대체 가능한 코드로 변경하였다면 완전히 동일한 코드가 발견되지 않을 수도 있다.

3.2 달빅(Dalvik) 코드의 요약

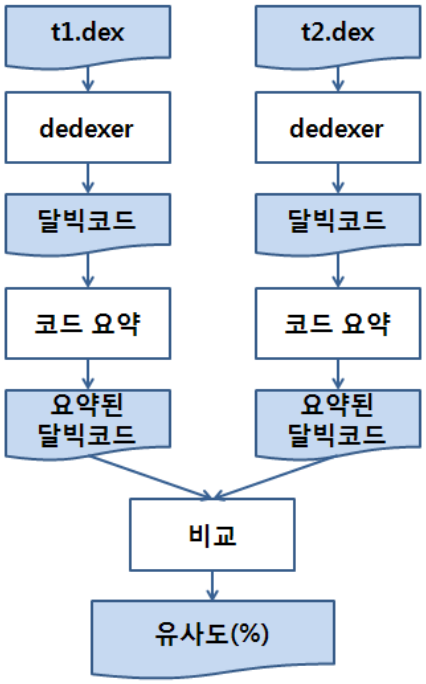
<표 1> 달빅 코드에 대한 요약 테이블

원본 코드	요약된 코드
move	move
move/from16	
move/16	
move-wide	
…	
const	const
const/4	
const/16	
const/high16	
…	
invoke-kind	invoke
invoke-virtual	
invoke-super	
invoke-direct	
…	
if-test	if
if-eq	
if-ne	
…	
…	

안드로이드 달빅 코드는 총 233개로 구성되어있다. 비슷한 역할의 달빅 코드들을 그룹화 하여 <표1>과 같이 요약된 코드로 변환하였다.

달빅 코드를 요약하는 과정을 거치는 이유는 도용된 코드에 대한 의도적인 변경에 대해서도 높은 유사도를 얻기 위해서이다. 예를 들어, 반복문이나 조건문은 적은 노력으로도 다양한 형태로 변형시킬 수 있다. 이렇게 의도적으로 변형된 코드에 대해서도 높은 유사도를 얻는 방법이 필요하다. 코드 요약 단계를 거치고, 요약된 코드를 비교한다면 이러한 간단한 코드 변경에 대해서 높은 유사도를 얻을 수 있다.

3.3 달빅 코드 요약을 통한 표절 탐지 도구



(그림 2) 달빅 코드 요약을 통한 표절 탐지 도구 동작 흐름도

(그림 2)는 달빅 코드 요약을 통한 표절 탐지 도구의 동작 흐름도이다. 비교 대상인 안드로이드 dex 파일로부터 달빅 코드를 추출하기 위해서 dedexer 도구[6]를 사용하였다. 추출된 달빅 코드에 대해서 3.2절에서 설명한 코드 요약 단계를 거치고, 요약된 달빅 코드에 대한 유사도를 계산한다.

4. 실험 및 평가

달빅 코드 요약을 통한 표절 탐지 도구의 성능 평가를 위해서 널리 알려진 BubbleSort, HeapSort 프로그램을 이용하여 실험하였다. 먼저 요약하지 않은 상태로 달빅 코드의 유사도를 측정하고, 이어서 요약된 달빅 코드에 대해서 유사도를 측정하였다.

4.1 BubbleSort.java 표절에 대한 실험

```
int N = 5;
int data[] = { 7, 5, 3, 2, 1 };
        for (int i = 0; i < N; i++) {
                for (int j = 1; j < (N - i); j++) {
                        if (data[j - 1] > data[j]) {
                                int t = data[j - 1];
                                data[j - 1] = data[j];
                                data[j] = t;    }
                                }
                        }
```

(그림 3) BubbleSort.java 원본 소스

```
long total = 5;
long input[] = { 7, 5, 3, 2, 1 };
int out = 0;
        while (out < total) {
                int in = 1;
                        while (in < (total - out)) {
                        if (!(input[in] >= input[in - 1])) {
                        long tmp = input[in - 1];
                        input[in - 1] = input[in];
                        input[in] = tmp;  }
                in++;   }
        out++;   }
```

(그림 4) BubbleSort.java의 표절된 소스

(그림 3)의 원본 소스는 5개의 숫자를 버블 정렬하는 코드이다. 이 코드를 표절하여 (그림 4)와 같은 유사 코드를 만들었다. 이제 두 java파일을 컴파일하여 먼저 class 파일을 만들고 안드로이드 SDK에 포함된 dx 도구를 사용하여 class 파일을 dex 파일로 변경하였다.

(그림 3)의 소스로부터 생성된 달빅코드	(그림 4)의 소스로부터 생성된 달빅 코드
invoke-direct return-void	invoke-direct return-void
const/4 const/4	const-wide/16 const/4
const/4 new-array	const/4 new-array
new-array fill-array-data	new-array fill-array-data
fill-array-data const/4	fill-array-data const/4
const/4 if-lt	const/4 int-to-long
…	…

(그림 5) 원본과 표절된 프로그램에서 추출된 달빅 코드

dex로부터 달빅 코드를 추출하기 위해서는 디스어셈블러인 dedexer 도구를 사용하였다. 이렇게 추출된 달빅 코드는 (그림 5)와 같다.

달빅 코드의 유사도를 계산하기 위해서 k-gram기법[7]을 사용하였다. k-gram 기법은 전체 프로그램 소스를 k 크기의 조각으로 나누고, 전체 조각 중에서 일치하는 조각의 개수의 비율을 계산한다.

k-gram 유사도 비교 결과 **31.03%**로 측정되었다. 원본 소스 코드와 표절된 소스를 비교했음에도 불구하고 매우 낮은 유사도가 측정되었다.

(그림 3)의 달빅 코드를 요약한 결과	(그림 4)의 달빅 코드를 요약한 결과
invoke return	invoke return
const const	const const
const new	const new
new filled-array	new filled-array
filled-array const	filled-array const
const if	const type-conv
…	…

(그림 6) 원본과 표절된 프로그램에서 추출된 달빅 코드를 요약하여 비교

(그림 6)은 원본 프로그램과 표절된 프로그램에서 추출한 달빅 코드를 요약 테이블에 기반하여 요약한 코드이다.

이렇게 요약된 코드에 대해서 k-gram 유사도를 측정한 결과 유사도 값은 **72.73%**로 2배 이상 상승한 우수한 결과를 보여주었다.

4.2 HeapSort.java 표절에 대한 실험

```
int list[] = { 3, 1, 4, 5, 2 };
    //….
    for (v = size / 2 - 1; v >= 0; v--) {
     //…..
       while (w < size) {
             if (w + 1 < size)
             if (list[w + 1] > list[w])
             //…..
             if (list[v] >= list[w])
             //….. }   }
    while (size > 1) {
    //…..
       while (w < size) {
          if (w + 1 < size)
             if (list[w + 1] > list[w])
              //…..
          if (list[v] >= list[w])
             //….. }   }
```

(그림 7) HeapSort.java 원본 소스

```
int H[] = { 3, 1, 4, 5, 2 };
    //….
    while (test >= 0) {
    //….
        for (www = 2 * test + 1; www < S; www++) {
            if (!(H[test] <= H[www]))
                //….  }
            //…. }
        for (S = 4; S >= 0; S--) {
        //….
            for (www = 2 * test + 1; www < S; www++) {
                if (!(H[test] <= H[www]))
                //….  }   }
```

(그림 8) HeapSort.java의 표절된 소스

두 번째 실험 대상으로 HeapSort에 대해서 표절 탐지를 수행하였다. (그림 7)의 원본 프로그램을 (그림 8)과 같이 결과 값은 같지만 반복문의 형태와 변수 명들을 바꾸는 표절을 한 후, 유사도 비교 실험을 하였다.

(그림 7)의 소스로부터 생성된 달빅코드	(그림 8)의 소스로부터 생성된 달빅 코드
const/4 const/4 new-array const/4 new-array fill-array-data new-array fill-array-data const/4 fill-array-data const/4 const/4 const/4 const/4 add-int/lit8 const/4 add-int/lit8 if-gez …	const/4 const/4 new-array const/4 new-array fill-array-data new-array fill-array-data const/4 fill-array-data const/4 const/4 const/4 const/4 add-int/lit8 const/4 add-int/lit8 if-gez …

(그림 9) 원본과 표절된 프로그램에서 추출된 달빅 코드

이때의 k-gram 유사도는 **56.10%**로 측정되었다.

(그림 7)의 달빅 코드를 요약한 결과	(그림 8)의 달빅 코드를 요약한 결과
const const new const new filled-array new filled-array const filled-array const const const const add const add if …	const const new const new filled-array new filled-array const filled-array const const const const add const add if …

(그림 10) 원본과 표절된 프로그램에서 추출된 달빅 코드를 요약하여 비교

요약된 코드를 통한 k-gram 유사도 실험 결과는 **73.68%**로 측정되었다.

이와 같이 소스 코드를 표절한 사람이 표절 사실을 숨기기 위해서 변수 타입이나 이름 또는 조건문이나 반복문을 조금이라도 수정한다면 모든 달빅 코드를 정확하게 비교하는 방법을 사용할 경우 낮은 유사도를 얻게 된다. 그러나 본 논문에서 제안하는 요약 과정을 거친 코드를 비교한다면, 간단하게 변경 가능한 코드인 경우에 요약된 코드가 일치할 확률이 높기 때문에 비교적 높은 유사도를 얻을 수 있다.

그러나 요약 과정을 거치더라도 표절된 코드에 대한 유사도 값은 70% 정도 수준이므로 이 수치를 더욱 높이기 위한 방법을 고려해야만 한다.

5. 결론 및 향후 연구 과제

본 논문에서는 코드 도용 및 표절에 대처하는 방법으로서, 안드로이드에서 사용하는 바이트 코드인 달빅 코드를 먼저 요약하고, 요약된 코드 사이의 유사도를 측정하는 방법을 제안하였다.

본 논문에서 제안한 방법을 평가하기 위하여 정렬 프로그램에 대하여 유사도 측정 실험을 하였다. 실험 결과, 달빅 코드를 그대로 비교하는 방법보다 요약 과정을 거칠 경우 유사도가 높아지는 것을 확인하였다.

이러한 실험 결과들을 통해서 달빅 코드 요약을 통한 표절 탐지 도구가 불법적인 코드 도용을 탐지하는데 유용하게 활용될 수 있을 것임을 기대한다.

향후 연구로서, 다양한 프로그램 변형 기법에 대해서 유사도를 더욱 높이는 방법에 대한 연구를 진행할 예정이다.

참고문헌

[1] "Jad-the fast Java Decompiler,“ http://www.kpdus.com/jad.html.

[2] "Mocha, the Java Decompiler," http://www.brouhaha.com/~eric/software

[3] Kristina L. Verco and Michael J. Wise, Software for Detecting Suspected Plagiarism : Comparing Structure and Attribute - Counting Systems, St. Australian Conf. on Computer Science Education, Sydney, Australia, July, 1996

[4] 한소정, 용환승 “오픈 소스코드 표절 탐지 기법”, 제30회 한국정보처리학회 추계학술발표대회 논문집 제15권 제2호.

[5] Dalvik VM Bytecode, "http://source.android.com/ tech/dalvik/ dalvik-bytecode.html

[6] "dedexer, disassembler tool for DEX files," http://dedexer.sourceforge.net.

[7] G. Myles and C. Collberg. "k-gram Based Software Birthmarks, " In Proceeding of the 2005 ACM Symposium on Applied Computing, pp. 314-318. Santa Fe, New Mexico, USA, 2005.