논문 15-40-04-11 The Journal of Korean Institute of Communications and Information Sciences '15-04 Vol.40 No.04
http://dx.doi.org/10.7840/kics.2015.40.4.700

난독화에 강인한 안드로이드 앱 버스마킹 기법

김 동 진◆, 조 성 제°, 정 영 기*, 우 진 운**, 고 정 욱***, 양 수 미****

Android App Birthmarking Technique Resilient to Code Obfuscation

Dongjin Kim◆, Seong-je Cho°, Youngki Chung*, Jinwoon Woo**, Jeonguk Ko***, Soo-mi Yang****

요 약

소프트웨어 버스마크는 한 프로그램이 보유한 고유한 특징으로 해당 프로그램을 식별하는데 사용될 수 있다. 소프트웨어 버스마크 기반으로 자바 프로그램의 도용을 탐지하는 연구들이 진행되어 왔다. 안드로이드 앱의 경우, 앱 보호를 위해 난독화 방법이 제공되고 있다. 그러나 공격자들도 자신이 도용한 프로그램을 감추기 위해 난독화를 적용하기도 한다. 특정 앱에 난독화를 적용하면 앱의 특징정보가 변경될 수 있다. 따라서 난독화를 고려한 버스마크 기반의 앱 도용 탐지 기법에 대한 연구가 필요하다. 본 논문에서는 난독화에 강인한 안드로이드 앱 버스마크 및 이에 기반한 앱 도용 탐지 기법을 제안한다. 몇몇 난독화 도구들을 분석하여 효과적인 버스마크로 메서드의 매개변수 및 반환값의 자료형을 선정하였고, 비교 대상 앱들로부터 해당 버스마크를 추출하여 이들 간의 유사도를 측정하였다. 여러 앱들을 대상으로 난독화 적용 전/후의 앱 유사성을 분석한 결과, 제안한 버스마크가 난독화가 적용된 앱에 대한 도용 탐지에도 효과적임을 확인하였다.

Key Words : Android app, Software birthmark, Code obfuscation, Software theft, Resilience

ABSTRACT

A software birthmark is the set of characteristics of a program which can be used to identify the program. Many researchers have studied on detecting theft of java programs using some birthmarks. In case of Android apps, code obfuscation techniques are used to protect the apps against reverse-engineering and tampering. However, attackers can also use the obfuscation techniques in order to conceal a stolen program. A birthmark (feature) of an app can be alterable by code obfuscations. Therefore, it is necessary to detect Android app theft based on the birthmark which is resilient to code obfuscation. In this paper, we propose an effective Android app birthmark and app theft detection through the proposed birthmark. By analyzing some obfuscation tools, we have first selected parameter and the return types of methods as an adequate birthmark. Then, we have measured similarity of target apps using the birthmarks extracted from the apps, where some target apps are not obfuscated and the others obfuscated. The measurement results show that our proposed birthmark is effective for detecting Android app theft even though the apps are obfuscated.

※ 본 연구는 문화체육관광부 및 한국저작권위원회의 2014년도 저작권 기술개발사업과 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음 (IITP-2015-H8501-15-1012)
◆ First Author : Dankook University Department of Computer Science, kdjorang@dankook.ac.kr, 학생회원
° Corresponding Author : Dankook University Department of Computer Science & Engineering, sjcho@dankook.ac.kr, 정회원
* Dankook University Department of Computer Science, youngki.chung99@gmail.com
** Dankook University Department of Computer Science & Engineering, jwwoo@dankook.ac.kr
*** Hancom Inc., newlife8837@gmail.com
**** University of Suwon Department of Information Secrecy, smyang@suwon.ac.kr, 정회원
논문번호 : KIC2015-03-063, Received March 23, 2015; Revised April 13, 2015; Accepted April 13, 2015

Ⅰ. 서 론

최근 스마트폰의 대중화와 함께 모바일 애플리케이션(이하 '앱') 마켓도 급성장하였다. 이에 따라 스마트폰 앱 불법 도용 및 표절에 따른 저작권 피해와 분쟁이 급증하고 있다. 특히, 안드로이드 앱은 Java 기반의 프로그램으로 역공학에 의한 소스코드 추출이 비교적 쉬워, 앱 도용 문제가 매우 심각하다. 실제로 2013년 기준 95%의 안드로이드 게임이 표절 혹은 도용의 위험이 있으며, 스마트폰 앱 도용으로 인한 피해가 매우 심각한 수준으로 보고되었다[1,2].

안드로이드 앱은 APK(Android application Package) 형태로 배포되며, 내부에 바이트 코드 수준의 실행파일인 DEX(Dalvik EXecutable)이 포함되어 있다. DEX는 Java 클래스 포맷을 안드로이드에 적합하도록 재배치 및 최적화한 것으로 알려져 있다. 따라서 바이너리 실행파일과 비교하여 역공학과 소스코드 추출이 용이하다는 Java 프로그램의 단점이 그대로 존재한다. 실제로 Java 프로그램에 대한 역컴파일 및 역공학 도구와 언어들 대부분이 DEX에도 그대로 적용된다. 이러한 특징을 악용하여, 안드로이드 코드를 도용 및 표절하고 재패키징(repackaging)하여 불법 앱 또는 악성 앱이 제작 및 배포된다.

이러한 문제를 해결하기 위해, 소프트웨어 버스마크(software birthmark)[3-10]를 기반으로한 안드로이드 앱 도용 탐지 연구들이 수행되고 있다. 소프트웨어 버스마크(이하 '특징정보'와 동일)는 프로그램의 고유한 특징정보(feature)로, 특징정보 간의 유사성 비교를 통해 프로그램 도용 탐지 및 식별이 가능하다. 기존 소프트웨어 버스마킹 기법은 Java 프로그램과 C/C++로 개발된 Windows 프로그램들을 대상으로 연구되어 왔고, 최근에는 안드로이드 앱을 대상으로한 연구들도 진행되고 있다[11-13].

코드 난독화(code obfuscation)는 프로그램 보호를 위해, 역공학이 어렵도록 코드를 변형하는 기법이다[14]. 하지만 프로그램 도용 및 악성코드 탐지 우회를 위해 공격자가 코드 난독화를 악용하기도 한다. 안드로이드 앱 보호를 위한 다양한 무료/상용 난독화 도구[15-17]들이 존재하는데, 이 역시 앱 도용 탐지를 우회하기 위해 사용될 수 있다. 하지만 기존 버스마킹 기반 안드로이드 앱 도용 탐지 연구들은 코드 난독화에 의한 영향을 고려하지 않고 있으며, 이로 인해 난독화가 적용된 앱을 대상으로 한 유사도 측정 결과의 신뢰성을 보장하기 어렵다.

본 논문에서는 코드 난독화에 강인한 안드로이드 앱 버스마크 및 도용 탐지 기법을 제안한다. 기존 안드로이드 난독화 도구들을 분석하여 난독화에 강인한 버스마크를 선정한다. 실험을 통해 효과적인 버스마크로, DEX 파일의 고유정보들 중에서 메서드(method)의 매개변수(parameter) 및 반환값(return value)의 자료형을 선정하였다. 각 메서드의 매개변수 및 반환값의 자료형(type)을 버스마크로 사용하여 두 앱을 구성하는 메서드들 간의 유사성을 비교한다. 각 메서드들 간의 유사도 측정 결과를 종합하여, 앱들 간의 도용 여부를 판단한다. 난독화에 대한 강인성 및 효율성을 위해, 추출한 메서드들의 정보 중에서 API 정보, 광고 및 외부 라이브러리의 메서드 정보들을 제외한다. 난독화에 대한 강인성, 도용 탐지의 효율성 및 신뢰성 등을 검증하기 위하여, 난독화 적용 전/후의 동일 앱들 간의 유사성과 동일 카테고리에 포함된 서로 다른 앱들 간의 유사성을 비교하였다. 이를 통해, 제안 기법이 난독화가 적용된 앱 도용 탐지에도 매우 효과적임을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 앱 난독화 도구 및 기법과 안드로이드 앱 도용 탐지 연구에 대해 알아본다. 3장에서는 난독화에 강인한 본 논문의 제안 기법에 대해 기술한다. 4장에서는 실험 및 평가에 대해 설명하고, 마지막으로 5장에서 결론 및 향후 연구를 제시한다.

Ⅱ. 관련 연구

2.1 안드로이드 앱 난독화 기법

Java 프로그램에 대한 대표적인 난독화 기법들로는 식별자 변환(renaming), 제어흐름, API 은닉(API hiding), 문자열 암호화 기법 등이 있다. 식별자 변환 난독화는 클래스, 필드, 메소드 식별자를 의미 없는 문자로 대체하여 역컴파일로 추출한 소스코드의 분석을 어렵게 만드는 기법이다. 제어 흐름 난독화는 분기 명령어들을 추가 및 수정하여 제어 흐름 분석을 어렵게 만든다. API 은닉 난독화는 API 호출 정보에 대한 분석을 차단하는 방법으로 프로그램의 주요 기능을 파악하기 어렵게 한다. 문자열 암호화는 문자열을 암호화하여 정적 분석이 불가능하도록 만드는 기법이다[18].

안드로이드 앱 보호를 위한 난독화 도구는 Proguard[15], Dexguard[16], Dex Protector[17]이 대표적이다. 표 1은 가장 많은 난독화 기법을 제공하는 Dexguard를 기준으로 다른 난독화 도구들을 비교한 결과이다.

Proguard는 구글에서 기본 제공하는 난독화 도구로

표 1. 안드로이드 난독화 도구 비교
Table 1. Comparison of Android obfuscation tools

Obfuscation method	Proguard	Dex Protector	Dexguard
Shrinking	o	o	o
Optimization	o	o	o
Renaming	o	o	o
String encryption	x	o	o
Class encryption	x	o	o
API hiding	x	o	o
Native library encryption	x	x	o

써, 가장 기본적인 식별자 변환 및 shrinking, 최적화(optimization)를 제공한다. 먼저 소스코드를 컴파일 할 경우, 사용되지 않는 클래스 혹은 메소드, 필드를 찾아 삭제하고, 최적화 과정을 진행한다. Dex Protector와 Dexguard는 안드로이드 상용 난독화 도구로 Proguard의 기능을 제공함과 동시에 API 은닉, 문자열 및 클래스 암호화를 제공한다.

위의 분석 결과, 안드로이드 앱 난독화 도구들에 의해 API, 문자열 등 기존 소프트웨어 버스마크 연구들에서 사용되던 주요 특징정보들이 달라지며, 이에 따라 해당 특징정보 기반 도용 탐지 결과의 신뢰성을 보장할 수 없게 된다. 본 논문에서는 표 1의 다양한 난독화 기법들 중에서 기존 연구들에서 고려하지 않았던, 식별자 변환(renaming) 및 API 은닉(API hiding) 난독화에 강인한 버스마킹 기법에 대해 제안한다. 다른 난독화 기법과 달리, 문자열 암호화, 클래스 암호화, Native library 암호화 기법은 정적분석 자체를 차단하는 기법이기 때문에 본 연구에서는 고려하지 않는다.

2.2 기존 안드로이드 앱 유사성 비교 연구

현재, 안드로이드 DEX 파일들의 유사성 비교 연구는 PC 프로그램에 비해 부족한 수준이다. J. Ko[12]는 DEX 파일에 포함된 문자열과 메서드들의 정보를 특징정보로 DEX 파일간 유사성 비교 연구를 수행하였다. 하지만 문자열은 앱 도용시 매우 쉽게 변조될 수 있고, 문자열 암호화 난독화에 의해 정적분석만으로는 추출이 어려워질 수 있기 때문에 특징정보로써 부적합하다. J. Ko가 추출한 메서드 정보는 특징정보로써 효과적이지만, 앱에서 사용하는 모든 API와 광고 및 외부 라이브러리들이 포함하고 있다. API 정보는 API 은닉 난독화에 의해 추출이 어려워지며, 난독화에 대한 강인성 및 신뢰성이 떨어지게 된다. 또한 API 및 광고 라이브러리를 포함하기 때문에 서로 다른 앱, 특히 기능이 유사한 앱들 간의 유사도가 실제 유사도 보다 높게 측정될 수 있다는 문제점이 존재 한다.

H. Park[13]은 API k-gram 기반 안드로이드 앱 유사성 비교 기법을 제안하였다. DEX에서 API 정보를 추출하여 특징정보로 사용하였으며, 이름이 동일한 오버로딩(overloading)된 API명을 구분하기 위해 메서드의 매개변수 및 반환값의 정보도 함께 사용하였다. 이 연구도 API 정보를 특징정보로 사용하기 때문에 API 은닉 난독화에 대한 강인성이 떨어질 수 있다. 실제로 이 연구[13]에서는 API 은닉 난독화 기법을 제공하지 않는 Proguard로만 난독화에 대한 강인성 실험을 수행하였다. 또한 사용자 정의 메서드의 경우, 식별자 변환 난독화에 의해 메서드명이 의미 없는 알파벳 등으로 변경되기 때문에 난독화에 대한 강인성이 더 떨어질 수 있다.

위와 같이, 기존의 안드로이드 앱 유사성 비교 연구들에서는, 기존의 C/C++로 개발된 PC 프로그램간의 유사성 비교에 효과적인 특징정보로 알려진 API 정보를 기반으로 연구를 수행하였다. 하지만 Java 프로그램 및 안드로이드 앱의 경우 API 은닉과 같은 더 강력한 난독화 기법에 의해 기존 기법들은 난독화에 대한 강인성이 낮아질 수 있고, 이로 인하여 앱 도용 탐지 시 우회될 수 있다. 본 논문에서는 기존 연구들의 문제를 개선하기 위해, 난독화에 더 강인한 특징정보를 기반으로한 안드로이드 앱 버스마크 기법을 제안한다.

Ⅲ. 제안 기법

본 논문에서 제안하는 난독화에 강인한 버스마크 기법은 안드로이드 앱에 포함된 메서드 정보를 특징정보로 사용한다. 메서드 정보 중에서 메서드명은 제외하고 매개변수 및 반환값의 자료형만을 사용한다. 또한 기존 기법들과는 달리 API 정보와 광고 등의 외부 라이브러리에 포함된 메서드들은 특징정보에서 제외시켜, API 은닉 난독화에 대한 강인성을 높인다. 결과적으로 사용자 정의(user-defined) 메서드의 정보만을 특징정보로 사용하며, 이들 간의 유사도를 측정하여 앱들 간의 도용 탐지 여부를 판단한다.

기존의 버스마크 연구들에서는 각 프로그램 또는 앱 별로 1개의 버스마크를 생성한다. 하지만 기존 연구들과는 달리, 제안 기법은 각 메서드 간의 유사도

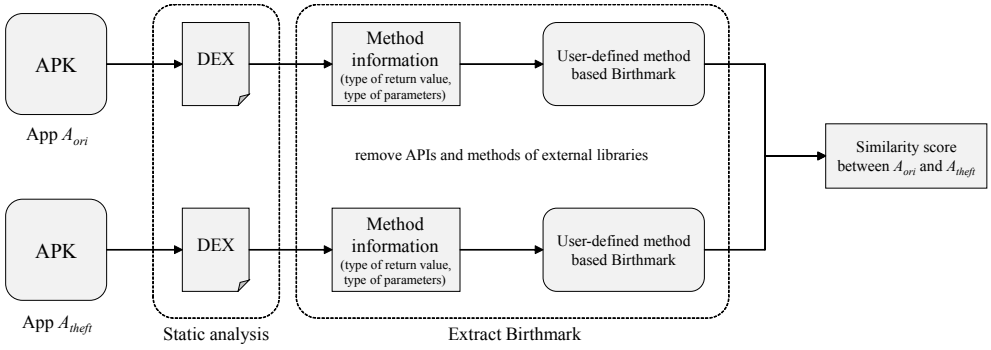

그림 1. 시스템 개요
Fig. 1. System overview

측정을 위해, 메서드별로 버스마크를 생성한다. 즉 반환값 및 매개변수의 자료형들이 각 메서드의 버스마크이며, 두 앱 간의 최종 유사도는 각 메서드들의 유사도를 종합하여 산출한다. 세부적인 유사도 측정 방법은 3.2절에서 설명한다.

앱 내의 메서드 수준 특징정보를 사용하므로, 앱 전체 수준의 특징정보 비교할 때에 비해 특징정보의 유일성(uniqueness)은 낮아질 수 있다. 예를 들어, 특징정보가 특정 메서드와 동일 혹은 매우 유사한 메서드가 동일 앱 또는 비교 대상 앱에 다수 존재할 수 있다. 하지만, 두 앱의 최종 유사도는 모든 메서드들의 유사도를 함께 고려하기 때문에 유일성의 저하는 본 제안 기법의 신뢰성에 영향을 미치지 못한다.

기존 연구들에서 사용한 특징정보들과 비교하여 본 논문에서 제안하는 특징정보의 장점은 다음과 같다. 첫 번째, API 정보를 사용하지 않기 때문에 API 은닉 난독화에 강인하다. 두 번째, 메서드명을 특징정보로 사용하지 않기 때문에 식별자 변환 난독화에도 강인하다. 세 번째, API 기반 특징정보를 사용할 경우, 서로 다른 앱이지만 기능이 유사한 앱 간의 유사도가 실제 유사도 보다 높게 측정될 수 있는데, API 정보를 제외함으로써 이 문제를 개선한다. 또한 광고 등의 외부 라이브러리를 제외함으로써, 오탐(false-positive)률을 낮출 수 있다.

본 논문에서 제안하는 버스마크 기법의 전체 개요는 그림 1과 같다.

3.1 버스마크 추출

원본 앱 A_{ori}과 도용이 의심되는 앱 A_{theft}으로부터 버스마크를 추출하기 위해, 먼저 두 APK 파일을 언패킹(unpacking)하여, 실행파일인 classes.dex를 추출한다. 그 후에 DEX에 포함된 'method_ids'와 'proto_ids'를 분석하여, 모든 메서드 정보, 즉 메서드명과 각 메서드의 반환값 및 매개변수의 자료형을 추출한다.

메서드 정보들 중에서 난독화에 대한 강인성을 저하시킬 수 있는 API 정보들과 오탐률을 높일 수 있는 광고 등의 외부 라이브러리에 포함된 메서드 정보들을 제거한다. 메서드 중에서 이름이 'java' 및 'andorid'로 시작하는 것들을 API로 판단하고 제거한다. 또한 표 2의 광고 및 외브 라이브러리들에 포함된 메서드들과 'onCreate()', 'onDestroy()'와 같이 모든

표 2. 광고 및 외부 라이브러리
Table 2. Advertisement and external libraries

Category	Library list	
Advertisement libraries	Android.tapit	Mobfox
	mopub	Admob
	millenniamedia	Richmedia
	Com.google.ads	adsmogo
External libraries	PhoneGap	jraf
	Org.apache .cordova	net.londatiga
	Org.apache .commonss.io	com.antiy
	uk.co.senab	Android.vender .billing
	sonyericsson	tencent mm
	mobeta	org.acra
	kankan	de.greenrobot
	cn.sharesdk	org.taptwo
	org.achartengine	com.lamerman
	com.tokaracamara	com.ijinshan.kingmob
	com.appsflyer	viewpagerindicator
etc.	Android.support.v4	Android.support.v13

앱에 포함된 메서드들을 제거하였다. 표 2는 다양한 앱 분석을 통해 도출하였다.

최종적으로 남은 사용자 정의 메서드들의 반환값 및 매개변수의 자료형만을 특징정보로 사용한다. 반환값의 자료형은 각 메서드별로 1개씩만 존재하기 때문에 직접 비교하지만, 매개변수의 개수는 정해져 있지 않기 때문에 각 메서드별 매개변수들의 자료형을 중복 허용된 집합 형태로 추출한다. 메서드의 매개변수의 경우, 각 메서드의 기능과 관련 있기 때문에 제거 및 추가하기는 어렵지만, 순서를 변조하는 것은 어렵지 않다. 이에 대한 강인성을 높이기 위해 서열이 아닌 집합 형태로 추출한다. 반환값의 자료형이 'int'이고, 3개 매개변수의 자료형이 'int', 'String', 'int'인 메서드의 특징정보의 예는 그림 2와 같다.

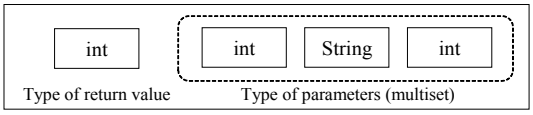

그림 2. 메서드 특징정보의 예
Fig. 2. An Example of feature of a method

3.2 유사도 측정

A_{ori}와 A_{theft}의 유사도를 측정하기 위해, A_{ori}에서 추출한 메서드들을 기준으로 A_{theft}의 메서드들과의 특징정보간 유사도를 측정하고, A_{ori}의 각 메서드별로 유사도가 가장 높은 A_{theft}의 메서드를 찾는다. 이렇게 찾은 A_{ori}의 각 메서드별 유사도의 평균값을 A_{ori}와 A_{theft}의 최종 유사도로 사용한다. 세부적으로 A_{ori}와 A_{theft}가 각각 사용자 정의 메서드 $\{p_1, p_2, p_3, ..., p_n\}$와 $\{q_1, q_2, q_3, ..., q_m\}$로 구성된다고 했을 때, 기본적으로 $n \times m$ 모든 경우의 수에 대해 메서드 간의 유사도를 측정하고, A_{ori}의 메서드를 기준으로 $n \times m$ 유사도 조합 중에서 총합이 가장 큰 측정 결과의 평균을 두 앱 간의 최종 유사도로 사용한다. 하지만 위의 유사도 측정 방식의 경우, 모든 경우의 조합에 대해 유사도를 측정해야하며, 측정 결과를 바탕으로 총합이 가장 큰 조합을 찾아내는 오버헤드는 O(n!)으로 매우 높다. 이를 개선하기 위해, 본 논문에서는 A_{ori}의 어떤 메서드 p와 A_{theft}의 메서드 $q_i(1 \le i \le m)$를 순차적으로 비교하다가, p와 q_i의 유사도가 임계치인 0.75 이상이면 동일한 메서드로 판단하여 $q_j(i < j \le m)$와의 비교는 수행하지 않는다. 또한, q_i를 A_{theft}의 메서드 리스트에서 제거한다. 기존 프로그램 간 유사성 비교 연구들에서는 최적의 유사도 임계치를 0.6~0.8 값들로 제시하고 있다[10,19,20]. 본 연구에서는 다양한 임계치별 실험 결과를 바탕으로 임계치를 0.75로 선정하였다.

원본 앱과 비교 대상 앱에 포함된 메서드가 각각 M_1, M_2, 이들의 매개변수 자료형 집합을 P_1, P_2, 반환값의 자료형을 R_1, R_2라고 했을 때, M_1, M_2의 유사도는 수식 (1)을 사용하여 계산한다. 매개변수의 자료형 집합 간의 교집합 크기에 반환값의 자료형이 서로 동일한 경우에만 1을 더한다. 그리고, 원본 앱 메서드의 매개변수 자료형 집합의 크기에 반환값의 자료형 개수인 1을 더하여 나눈 값을 최종 유사도로 사용한다.

$$Similarity(M_1, M_2) = \frac{|P_1 \cap P_2| + \alpha}{|P_1| + 1} \quad (1)$$
$$where\ \alpha = 1\ \ if\ R_1 = R_2,\ otherwise\ \alpha = 0$$

Ⅳ. 실험 및 분석

본 논문에서 제안한 버스마크 기법이 난독화에 강인함을 검증하기 위해, 다양한 오픈소스 앱을 주요 기능 카테고리별로 수집하고 이를 대상으로 실험을 진행하였다. 실험 대상 앱은 표 3와 같다.

본 논문의 제안기법이 효과적임을 검증하기 위해, 기존 연구에서 제안한 API k-gram 기반 정적 버스마크와 문자열 기반 정적 버스마크 기법과 비교 실험하였다. 기법 1의 경우, 기존 연구[14]를 참고하여 k 값을 3으로 선정하였다.

기법 1 : API k-gram 기반 정적 버스마크
(k = 3)

표 3. 실험대상 앱
Table 3. Target apps

Category	Target apps	Source code size (kb)
Bluetooth	Bluetooth Chat	39.8
	Bluetooth Car	32
Notepad	Notepad	93.8
	Mini-Noteapd	34.5
Keyboard	AnySoftKeyboard	823
	SoftKeyboard	55.7
Weather	WeatherList Widget	77
	ADG Weather	108
Battery	Battery-indicator (7.0.6)	258
	Current-Widget (0.38)	156
SSH	ConnectBot (1.7.1)	418
	SSHTunnel (1.5.5)	304

기법 2 : 문자열 기반 정적 버스마크
기법 3 : 제안 기법

4.1 난독화에 대한 강인성

제안 기법의 난독화에 대한 강인성(resilience)을 확인하기 위해, 각 앱을 Proguard와 Dex Protector로 난독화하고 난독화 적용 전/후의 유사도를 측정하였다. 먼저, 원본 앱(난독화가 적용되지 않은 앱)을 기준으로 Proguard 도구를 적용한 앱 간의 유사도 측정 결과가 표 4에 나타나 있다. 기법 1과 제안 기법인 기법 3은 난독화 적용 전과 비교하여, 유사도가 최소 0.2에서 최대 0.18 감소하였다. 이를 통해 Proguard에서 제공하는 식별자 변환 등의 비교적 단순한 난독화에 대해서는 기법 1, 3 모두 강인함을 확인할 수 있다. Proguard의 경우, API 은닉 난독화를 제공하지 않기 때문에 API 정보 기반의 기법 1도 난독화 후의 유사도가 크게 떨어지지 않은 것으로 파악된다. 기법 2의 경우, 버스마크로 사용된 문자열 정보 중에 변수의 식별자 정보들도 포함되기 때문에 식별자 변환 난독화에 의해 유사도가 매우 크게 떨어졌다.

난독화 후에 기법 3의 유사도가 조금 낮아진 이유는 최적화로 인해 실제 사용되지 않는 메서드들이 삭제되었기 때문이다. 기법 1의 경우, 앞의 이유와 함께 난독화 과정에서 새로운 API가 추가되기 때문에 유사도가 낮아진 것으로 분석된다.

표 5는 API 은닉 난독화에 대한 각 기법의 강인성을 평가하기 위해, 각 앱에 대해 Dex Protector의 API 은닉 및 식별자 난독화를 적용하고 적용 전과의 유사도를 측정한 결과이다. 표 4와는 달리, API 기반 기법 1의 최대 유사도가 0.14에 불과하였고, 그 외에는 0.1 미만으로 매우 낮음을 확인할 수 있다. 이는 API 은닉 난독화에 의해 기존의 API들이 Java Reflect API로 대체되었기 때문이다. 하지만, 제안 기법(기법 3)의 경우, 표 4와 표 5의 실험 결과가 동일하였는데 API 은닉 난독화에 영향을 받는 API들을 특징정보에서 모두 제외시켰기 때문이다. 이를 통해, 본 논문의 제안 기법이 난독화에 가장 강인함을 확인할 수 있다.

표 4. Proguard 난독화 도구에 대한 강인성 분석
Table 4. Analysis of resilience against obfuscation by Proguard

Target apps	Similarity		
	Method 1	Method 2	Method 3
BluetoothChat	0.90	0.52	0.91
BluetoothCar	0.91	0.21	0.92
NotePad	0.92	0.58	0.93
Mini-NotePad	0.84	0.24	0.85
AnySoft Keyboard	0.82	0.25	0.82
Soft Keyboard	0.98	0.28	0.99
WeatherList Widget	0.95	0.25	0.98
ADG Weather	0.90	0.22	0.95
Battery-indicator	0.88	0.25	0.90
Current-Widget	0.87	0.26	0.89
ConnectBot	0.88	0.39	0.90
SSHTunnel	0.89	0.33	0.91

표 5. Dex Protector 난독화 도구에 대한 강인성 분석
Table 5. Analysis of resilience against obfuscation by Dex Protector

Target apps	Similarity		
	Method 1	Method 2	Method 3
BluetoothChat	0.04	0.30	0.91
BluetoothCar	0.02	0.18	0.92
NotePad	0.05	0.40	0.93
Mini-NotePad	0.04	0.17	0.85
AnySoft Keyboard	0.14	0.18	0.82
Soft Keyboard	0.07	0.33	0.99
WeatherList Widget	0.04	0.19	0.98
ADG Weather	0.04	0.17	0.95
Battery-indicator	0.03	0.20	0.90
Current-Widget	0.04	0.20	0.89
ConnectBot	0.04	0.25	0.90
SSHTunnel	0.05	0.19	0.91

4.2 신뢰성

제안 기법의 신뢰성(Credibility)을 검증하기 위해, 동일 카테고리에 포함된 서로 다른 앱 간의 유사도를 측정하였다. 기능이 유사하지만 서로 독립적으로 개발된 앱의 경우, 유사도가 낮을수록 신뢰성이 높다. Dex Protector 난독화 적용 전의 두 앱간 유사도와 적용 후의 유사도를 각각 측정하였으며, 검증을 위해 원본 소스코드 간의 유사도를 Stanford 대학의 Moss[21]를 통해 비교하였다. 결과는 표 6와 같으며, 카테고리내의 두 앱의 유사도를 기준을 변경하여 각각 실험한 결과이다. 예를 들어, 'Notepad'의 유사도는 'Notepad'를 기준으로 'Mini-Notepad'를 비교한 결과이며, 'Mini-Notepad'의 유사도는 그 반대이다.

난독화 전의 각 기법별 신뢰성 평가 결과, 본 논문에서 제안하는 기법 3의 결과가 평균 0.06으로 소스코드 평균 유사도 0.08과 가장 가까웠다. 기법 1의 경우

표 6. 동일 카테고리의 서로 다른 앱 간의 유사성 비교
Table 6. Similarity comparison between different apps of same category

Category	Target apps	Similarity						
		Source code	Before obfuscating			After obfuscating		
			Method 1	Method 2	Method 3	Method 1	Method 2	Method 3
Bluetooth	Bluetooth Chat	0.03	0.03	0.05	0.05	0.22	0.01	0.03
	Bluetooth Car	0.22	0.16	0.46	0.14	0.88	0.40	0.12
Notepad	Notepad	0.09	0.07	0.48	0.12	0.86	0.43	0.09
	Mini-Notepad	0.08	0.06	0.06	0.08	0.24	0.01	0.07
Keyboard	AnySoftKeyboard	0.01	0	0.07	0	0.34	0.01	0.0
	SoftKeyboard	0.15	0.06	0.42	0.07	0.87	0.38	0.06
Weather	WeatherList Widget	0.14	0.05	0.56	0.12	0.85	0.45	0.12
	ADG Weather	0.02	0	0.03	0.01	0.29	0.01	0.02
Battery	Battery-indicator	0.02	0.01	0.09	0.01	0.25	0.05	0.01
	Current-Widget	0.09	0.07	0.42	0.06	0.88	0.24	0.05
SSH	ConnectBot	0.04	0.04	0.50	0.02	0.26	0.38	0.01
	SSHTunnel	0.09	0.08	0.59	0.08	0.89	0.42	0.07
Average		0.08	0.05	0.31	0.06	0.57	0.23	0.05

에도 기법 3과 매우 유사한 결과를 보였다. 기법 2의 경우, 최대 0.59, 평균 0.31의 유사도로 신뢰성이 가장 낮았다.

Dex Protector를 통해 각 앱에 난독화를 적용한 후에 유사도를 비교한 결과, 난독화에 영향을 가장 적게 받는 기법 3의 경우 난독화 전과 거의 유사한 결과를 보였다. 하지만, 기법 1의 경우, API 은닉 난독화의 영향으로 최대 0.89, 평균 0.57로 높은 유사도를 보였으며, 이를 통해, 난독화된 경우 기법 1의 신뢰성이 낮음을 알 수 있다. 난독화 후에 기법 1의 유사도가 높아진 이유는 API 은닉을 위해 기존의 API들이 Java Reflect API로 대체되었고, 결국 서로 다른 앱들에 동일 또는 유사한 API(Java Reflect API)가 증가했기 때문이다. 기법 3의 경우, 난독화 전과 비교하여 평균 유사도 0.01 차이로 크게 달라지지 않았다. 난독화 전/후의 실험 결과를 종합하면, 본 논문에서 제안하는 기법 3의 신뢰성이 가장 높다.

V. 결 론

안드로이드 플랫폼의 경우, 앱 역공학이 비교적 쉽기 때문에 앱 도용 및 표절 분쟁이 많이 발생하고 있다. 이를 해결하기 위해, 소프트웨어 버스마크 기반의 안드로이드 앱 도용 및 표절 탐지 연구들이 진행되고 있다. 기존의 앱 도용 탐지 연구들은 안드로이드 앱에 대한 난독화 기법들을 충분히 고려하지 않았다.

본 논문에서는, 먼저 3가지 안드로이드 앱 난독화 기법을 분석하였고, 난독화에 강인한 앱 버스마크 기반의 도용 탐지 기법을 제안하였다. 제안 기법에서는 앱 실행파일(DEX)에 포함된 메서드의 매개변수 및 반환값의 자료형을 추출하여 특징정보로 사용한다. 또한 도용 탐지의 강인성 및 신뢰성을 저하시킬 수 있는 API 정보, 광고 등의 외부 라이브러리에 포함된 메서드들을 제거하였다. 그리고 추출된 메서드 수준의 특징정보를 기반으로 메서드 간의 유사도를 측정하여 앱 도용 여부를 판단하였다. 마지막으로 잘 알려진 안드로이드 앱 난독화 도구인 Proguard 및 Dex Protector와 소스가 공개된 앱을 사용하여, 기존 앱 도용 탐지 기법들과의 강인성 및 신뢰성 면에서 비교 실험을 수행하였다. 실험 결과, 본 논문의 제안 버스마크 기법이 기존 기법들과 비교하여, 난독화에 더 강인하고, 신뢰성이 높음을 확인하였다.

References

[1] C. Davies, *95% Android game piracy experience highlights app theft challenge*, Retrieved May, 15, 2013, from http://www.slashgear.com/95-android-game-piracy-experience-highlights-app-theft-challenge-15282064/

[2] D. Seo, *Smart phone apps plagiarism warning*, Retrieved Mar. 22, 2010, from http://www.etnews.com/201003190142

[3] H. Park, S. Choi, S. Seo, and T. Han, "Analyzing differences of binary executable files using program structure and constant values," *J. KIISE : Software and Appl.*, vol. 35, no. 7, pp. 452-461, Jul. 2008.

[4] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. Matsumoto, “Dynamic software birthmarks to detect the theft of windows applications,” in *Proc. Int. Symp. Future Software Technol. 2004 (ISFST 2004)*, vol. 20, no. 22, Oct. 2004.

[5] D. Schuler and V. Dallmeier, “Detecting software theft with API call sequence Sets,” in *Proc. 8th Workshop Software Reengineering*, May 2006.

[6] G. Myles and C. Collberg, “Detecting software theft via whole program path birthmarks,” in *Proc. Inf. Security Conf.*, vol. 3225, pp 404-415, Sept. 2004.

[7] X. Zhou, X. Sun, G. Sun, and Y. Yang, “A combined static and dynamic software birthmark based on component dependence graph,” in *Proc. 4th Int. Conf. Intell. Inf. Hiding and Multimedia Signal Process. (IIH-MSP)*, pp. 1416-1421, Aug. 2008.

[8] D. Kim, S. Cho, S. Han, M. Park, and I. You, “Open source software detection using function-level static software birthmark,” *J. Internet Services and Inf. Security (JISIS)*, vol. 4, no. 4, pp. 25-37, Nov. 2014.

[9 D. Kim, Y. Han, S. Cho, H. Yoo, J, Woo, Y. Nah, M. Park, and L. Chung, “Measuring similarity of windows applications using static and dynamic birthmarks,” in *Proc. ACM Symp. Applied Computing (2013 SAC)*, pp. 1628-1633, Mar. 2013.

[10] G. Myles and C. Collberg, “k-gram based software birthmarks,” in *Proc. ACM Symp. Applied Computing (2005 SAC)*, pp. 314-318, Mar. 2005.

[11] J. Ko, H. Shim, D. Kim, Y. Jeong, S. Cho, M. Park, S. Han, and S. Kim, “Measuring similarity of android applications via reversing and K-gram birthmarking,” in *Proc. Research in Adaptive and Convergent Syst. (2013 RACS)*, pp. 336-341, Oct. 2013.

[12] J. Ko, S. Kang, J. Moon, D. Kim, and S. Cho, “A study on comparing similarity of android applications based on dex,” *J. The Korea Software Assesment and Valuation Soc.*, vol. 9, no. 1, Jun. 2013.

[13] H. Park, “An android birthmark based on API k-gram,” *KIPS Trans. Comput. Commun. Syst. (KTCCS)*, vol. 2, no. 4, pp. 177-180, Apr. 2013.

[14] C. S. Collberg and C. Thomborson, “Watermarking, tamper-proofing, and obfuscation-tools for software protection,” *IEEE Trans. Software Eng.*, vol. 28 no. 8, Aug. 2002.

[15] *Proguard*, http://developer.android.com/tools/help/proguard.html

[16] *Dexguard*, https://www.saikoa.com/dexguard

[17] *Dex Protector*, http://dexprotector.com

[18] Y. Piao, J. Jung, and J. Yi, “Structural and functional analysis of ProGuard obfuscation tool,” *J. KICS*, vol. 38, no. 08, pp. 654-662, Aug. 2013.

[19] H. Lim, “Comparing binary programs using approximate matching of k-grams,” *J. KIISE : Computing Practices and Lett.*, vol. 18, no. 4, pp. 288-299, Apr. 2012.

[20] Y. Bai, X. Sun, G. Sun, X. Deng, and X. Zhou, “Dynamic k-gram based software birthmark,” in *Proc. 19th Australian Conf. Software Eng. (ASWEC 2008)*, pp. 644-649, Mar. 2008.

[21] *A system for detecting software plagiarism - MOSS*, [online] http://theory.stanford.edu/~aiken/moss

김 동 진 (Dongjin Kim)

2009년 2월 : 단국대학교 컴퓨터과학과 이학사

2011년 2월 : 단국대학교 컴퓨터학과 공학석사

2011년 3월~현재 : 단국대학교 컴퓨터학과 박사과정

<관심분야> 컴퓨터 보안, 소프트웨어 지적재산권 보호, 스마트폰 보안, 시스템 소프트웨어 등

조 성 제 (Seong-je Cho)

1989년 : 서울대학교 컴퓨터공학과 공학사
1991년 : 서울대학교 컴퓨터공학과 공학석사
1996년 : 서울대학교 컴퓨터공학과 공학박사
2001년 : 미국 University of California, Irvine 객원연구원
2009년 : 미국 University of Cincinnati 객원연구원
1997년 3월~현재 : 단국대학교 소프트웨어학과/컴퓨터학과 교수
<관심분야> 컴퓨터보안, 소프트웨어 지적재산권 보호, 시스템 소프트웨어, 스마트폰 보안 등

정 영 기 (Youngki Chung)

1981년 : BS in Industrial Engineering Columbia University, NY USA
1989년 : MS in Computer Science New York University, NY USA
1986년 : Computer Network Systems
1989년 : AT&T Bell Labs
1997년 : Hyundai Electronics Industry
1999년 : Motorola
2005년~현재 : Samsung Electronics
2013년~현재 : 단국대학교 컴퓨터학과 박사과정
<관심분야> 스마트폰 보안 등

우 진 운 (Jinwoon Woo)

1980년 : 서울대학교 수학교육과 (학사)
1989년 : 미국 University of Minnesota 전산학과 (박사)
1989년~현재 : 단국대학교 컴퓨터학부 교수
<관심분야> 분산 및 병렬처리, 알고리즘, 자료구조

고 정 욱 (Jeonguk Ko)

2013년 2월 : 제주대학교 컴퓨터교육과 졸업
2015년 2월 : 단국대학교 컴퓨터학과 석사
2015년 3월~현재 : (주)한글과컴퓨터
<관심분야> 시스템 및 웹 보안, 컴퓨터공학

양 수 미 (Soo-mi Yang)

1985년 2월 : 서울대학교 컴퓨터공학과 졸업
1987년 2월 : 서울대학교 컴퓨터공학과 석사
1997년 2월 : 서울대학교 컴퓨터공학과 박사
1988년 3월~2000년 9월 KT 연구소 선임연구원
2004년 9월~현재 : 수원대학교 정보보호학과 교수
<관심분야> 정보보호, 시스템 보안, 네트워크 보안