안드로이드를 위한 난독화 도구 프로가드(Proguard) 성능 평가

박희완*, 박희광*, 고광만**, 최광훈***, 윤종희****
*한라대학교 정보통신방송공학부, **상지대학교 컴퓨터정보공학부
***연세대학교 컴퓨터정보통신공학부, ***강릉원주대학교 컴퓨터공학과
e-mail:{heewanpark@halla.ac.kr, parkhk2468@naver.com,
kkman@sangji.ac.kr, kwanghoon.choi@yonsei.ac.kr, jhyoun@gwnu.ac.kr}

An Evaluation of the Proguard, Obfuscation Tool for Android

Heewan Park*, Heekwang Park*, Kwangman Ko**, Kwanghoon Choi***,
Jonghee Youn****
*School of Info. & Comm., Broadcasting Engineering, Halla University
**Dept. of Computing Information & Engineering, Sangji University
***Computer&Telecommunication Engineering Division, Yonsei University
****Dept. of CS & Engineering, Gangneung-Wonju National University

요 약

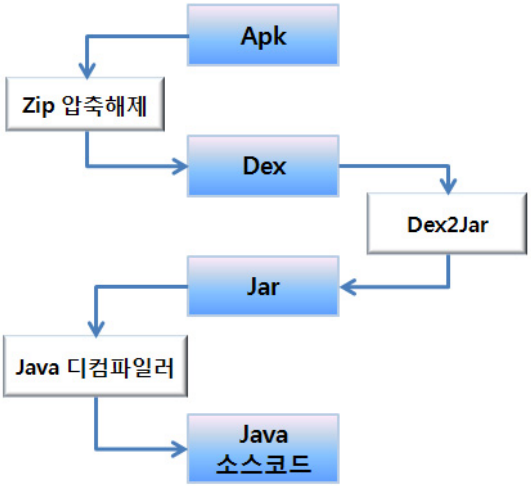
소프트웨어는 대부분 바이너리 형태로 배포되기 때문에 역공학 분석이 쉽지 않다. 그러나 안드로이드는 자바를 기반으로 한다. 즉, 자바 언어로 프로그래밍하고 생성된 클래스 파일을 dx라는 도구를 사용하여 안드로이드용 달빅(Dalvik) 코드로 변환한다. 따라서 안드로이드 역시 자바의 취약점을 가지고 있고, 자바용으로 개발된 역공학 도구에 의해서 쉽게 분석될 수 있다. 한편으로 자바 프로그램의 저작권을 보호하고 핵심 알고리즘이 노출되지 않도록 다양한 난독화 도구들이 개발되었다. 그 중에서 안드로이드 SDK에 포함되어 함께 배포되고 있기 때문에 널리 사용되고 있는 프로가드(Proguard)에 대해서 대표적인 기능 및 사용법, 프로가드로 난독화된 코드가 원본과 비교하여 어떻게 변경되었는지 평가한다. 그리고 프로가드가 가지고 있는 한계를 알아보고, 이것을 극복할 수 있는 방법을 모색한다.

1. 서론

최근, 소프트웨어 제작 및 분석 기술이 눈부신 발전을 이루었고 이 과정에서 소프트웨어 산업에 유용한 개발 기법과 도구들이 많이 개발되었다. 그러나 이러한 분석 기법들은 경우에 따라서 특정 소프트웨어의 취약성을 발견해 내어 불법적인 조작을 가하거나 지적 재산권을 침해하는 목적으로도 악용될 수도 있다.

소프트웨어는 대부분 바이너리 형태로 배포되기 때문에 역공학 분석이 쉽지 않다. 하지만, 자바 프로그램의 경우 대부분 바이트 코드로 배포되고, 바이트 코드는 하드웨어에 의존하지 않는 이식성 높은 코드로서 분석이 용이하기 때문에 역공학에 취약하다는 단점이 있다. 따라서 소스가 공개되지 않은 자바 프로그램이라도 역공학 도구를 사용하기만 하면 원본 소스코드를 간단하게 추출할 수 있는 경우가 많다.

안드로이드는 자바를 기반으로 한다. 즉, 자바 언어로 프로그래밍하여 생성된 클래스 파일을 안드로이드 SDK 플랫폼에 포함되어 있는 도구인 dx를 사용하여 안드로이드용 달빅 코드로 변환한다. 따라서 안드로이드 역시 자바의 취약점을 그대로 가지고 있고, 역공학에 의해서 쉽게 분석될 수 있다.



(그림 1) 안드로이드 Apk 파일로부터 Java 소스 코드를 추출하는 방법

예를 들어 (그림 1)과 같이 분석하고자 하는 안드로이드 apk 파일이 있을 때 apk의 압축을 zip으로 해제하여 dex를 추출하고 Dex2Jar 도구를 사용하여 Jar로 변환한다. 그리고 Jar 디컴파일러를 이용하면 Java 소스코드를 얻는 것이 가능하다.

2. 관련 연구

자바 디컴파일러를 이용하면 대부분의 자바 클래스로부터 자바 소스 파일을 얻을 수 있다. 따라서 자바 기반으로 프로그램을 만들 경우에는 프로그램의 저작권을 보호하고 핵심 기능이 유출되는 것을 막기 위한 노력이 필요하다. 자바 프로그래머들의 이러한 요구 사항에 부응하여 자바 클래스의 분석을 어렵게 하는 목적으로 다양한 난독화 도구들이 개발되었다.

<표 1> 대표적인 자바 난독화 도구의 기능 비교

저자/회사	프로그램	축약	최적화	난독화	선검증	라이선스
Eric Lafortune	ProGuard[1]	✔	✔	✔	✔	Free (GPL)
Jochen Hoenicke	Jode[5]	✔	✔	✔		Free (GPL)
Hidetoshi Ohuchi	Jarg[6]	✔		✔		Free (BSD)
yWorks	yGuard[7]	✔		✔		Free
Sable	Soot[8]		✔			Free (LGPL)
Sable	JBCO[9]			✔		Free (LGPL)
PreEmptive	DashOPro[10]	✔	✔	✔		상용
Zelix	Klass Master[11]	✔	✔	✔		상용
Lee Software	Smokescreen Obfuscator[12]	✔		✔		상용
U. of Arizona	SandMark[13]		✔	✔		상용
Innaworks	mBooster[14]	✔	✔		✔	상용

<표 1>은 대표적인 자바 난독화 도구[4]들의 특징을 요약한 것이다. 여기서 축약(Shrinking)은 프로그램 실행에서 불필요한 정보를 줄여주는 것을 의미한다. 즉, 코드에는 포함되어 있으나 사용되지 않는 클래스, 필드, 메소드를 삭제해주는 기능을 한다.

최적화(Optimization)는 정적 분석을 통한 최적화를 의미한다. 제어 흐름 분석(control flow analysis)이나, 자료 흐름 분석(data flow analysis)을 통해서 얻게 된 정보를 기반으로 상수식을 미리 계산하거나 사용되지 않는 코드(dead code)를 제거하거나 인라인 기법을 사용하기도 한다.

난독화(Obfuscation)는 디버깅 정보를 없애고, 클래스와 메소드, 변수 이름을 임의로 변경하여 이해하기 어려운 형태로 바꾸는 역할을 한다.

선검증(Preverification)은 클래스 검증 작업을 미리 진행하는 것을 의미한다. 자바 ME(Micro Edition)과 자바6 버전은 분리된 검증 기능을 제공하고, 선검증 정보를 클래스 파일에 추가할 수 있다. 이것은 클래스 로더가 검증하는 단계를 단순화시켜서 빠르게 클래스가 로드될 수 있다.

3. 프로가드 난독화 도구

3.1 프로가드의 최적화 성능

프로가드[1]는 자바를 위한 난독화 도구로 개발이 되었으나 안드로이드 SDK[2]에 기본으로 탑재되고, 안드로이드 2.3 버전부터는 빌드 과정에서 자동으로 프로가드를 실행시켜주는 기능이 포함되면서 대표적인 안드로이드용 난독화 도구가 되었다.

프로가드는 사용되지 않는 코드를 지우는 최적화 작업을 수행하고, 클래스와 필드 이름을 짧게 줄여주는 변환을 수행한다. 따라서 클래스 이름과 필드 이름이 텍스트 형태로 클래스 파일에 그대로 저장되는 바이트코드의 특성에 의해서 짧게 줄어든 이름 변경 작업만으로도 클래스 크기를 줄여주는 효과가 있고, 이름이 짧은 형태로 변경된 클래스와 필드는 이름만으로 그 의미를 이해하기 어려워지기 때문에 난독화의 효과도 거둘 수 있다.

<표 2> 프로가드의 난독화에 의한 코드 크기 감소 효과

입력 프로그램	원본 크기	난독화 크기	최적화 결과
Worm, midlet from Oracles'JME	10.3K	8.5K	18%
Javadocking library	290K	201K	30%
Proguard itself	648K	348K	46%
JDepend, a Java quality metric tool	57K	28K	51%
Java6 Runtime Class	53K	18M	66%
Tomcat, Apache servlet	1.1M	295K	74%
JavaNCSS, Java source metric tool	632K	152K	75%
Ant, Apache build tool	2.4M	242K	90%

<표 2>는 프로가드 사이트에 공개된 프로가드의 난독화에 의한 최적화 결과[3]이다. 프로가드에 의해서 자바 클래스 파일은 원본과 비교하여 18% ~ 90%에 이르는 코드 사이즈 감소의 효과를 얻을 수 있음을 보여준다.

3.2 프로가드 실행 방법

프로가드를 이클립스 통합 환경에서 실행하려면 먼저 ADT(Android Development Tool)를 최신 버전으로 업데이트해야 한다. 이후, 새 프로젝트를 생성하면 프로젝트 폴더의 최상단에 proguard.cfg 라는 파일이 자동으로 생성된다. proguard.cfg 파일에는 난독화에 사용되는 기본적인 옵션값으로 초기화가 되어 있다. 목적에 맞게 이 파일을 수정하여 사용할 수 있다.

난독화를 수행하려면 프로젝트 최상단 폴더의 properties 파일을 열어서 “proguard.config=proguard.cfg”와 같이 설정값을 추가한다. 그리고 안드로이드 프로젝트를 서명된 버전으로 엑스포트하면 proguard.cfg 설정값에 의해서 난독화된 apk파일이 생성된다.

프로가드가 정상적으로 실행이 되었다면, 난독화된 apk 파일뿐만 아니라 proguard 라는 폴더를 생성하고 다음과

같은 4가지 파일을 추가로 생성한다.

1) **dump.txt** : 프로그램 내부에서 사용 중인 클래스, 메소드, 필드에 대한 정보를 가지고 있다.

2) **mapping.txt** : 난독화 이전의 클래스와 메소드 이름이 난독화 이후에 어떤 이름으로 변경되었는지에 대한 정보를 가지고 있다.

3) **seeds.txt** : 난독화되지 않고 원래의 이름으로 남아있는 클래스와 메소드에 대한 정보를 가지고 있다.

4) **usage.txt** : 프로그램 내에서 사용되지 않아서 클래스에서 제거된 메소드, 필드에 대한 정보를 가지고 있다.

프로가드는 난독화 결과에 대해서 100% 완벽한 안정성을 보장하지는 않는다. 예를 들어, 난독화 과정에서 이름이 바뀌면 안 되는 클래스 이름이 변경될 수도 있고, 동적으로 사용되는 클래스나 메소드, 필드의 경우에는 전혀 사용하지 않는 것으로 잘못 판단하여 코드에서 제거시키는 문제를 발생시킬 수도 있다. 따라서 프로가드에 의해서 난독화된 프로그램은 엄밀한 테스트를 거쳐서 실행에 오류가 없음을 확인할 필요가 있다.

만일 반드시 본래의 이름을 유지해야 할 클래스나 메소드가 있다면 proguard.cfg 파일에 이러한 정보를 추가하여 오류를 사전에 예방할 수 있다.[14]

3.3 프로가드 적용 결과

프로가드는 다양한 난독화 및 최적화를 수행하여 코드를 변경시킨다. 본 논문에서는 간단한 예제를 가지고 프로가드가 원본 코드를 어떻게 난독화된 형태로 바꾸는지를 확인하였다. 본 논문에서는 디컴파일 도구로서, jad 자바 디컴파일러[16]를 선택하여 실험하였다.

<표 3> 클래스 생성자 메소드 대한 난독화 결과

생성자	원본 자바 소스코드	`public HelloView(Context context) {`
		`  super(context);`
		`  setBackgroundColor(Color.WHITE);`

	dex 변환후 디컴파일	`public HelloView(Context context) {`
		`  super(context);`
		`  setBackgroundColor(-1);`
		`}`

	프로가드 난독화후 디컴파일	`public a(Context context) {`
		`  super(context);`
		`  setBackgroundColor(-1);`

<표 3>은 클래스 생성자 메소드에 대한 난독화 결과이다. 원본 자바 코드는 dex로 변경되어 다시 디컴파일될 경우 상수 변수가 상수 값으로 치환되었고, 클래스 이름은 변경되지 않았다.

프로가드로 난독화한 후 디컴파일한 결과를 확인해보면 사용자 정의 클래스인 HelloView가 a라는 이름으로 짧게 줄어든 형태로 변경된 것을 확인할 수 있다. 안드로이드에서 제공하는 기본 클래스 이름인 Context, 그리고 메소드 이름인 setBackgroundColor는 변경되지 않았다.

<표 4> if 문장의 난독화 결과

Max 함수	원본 자바 소스코드	`int Max(int x, int y) {`
		`  int max;`
		`  if ( x > y )`
		`  max = x;`
		`  else`
		`  max = y;`
		`  return max;`
		`}`

	dex 변환후 디컴파일	`int Max(int i, int j) {`
		`  int k;`
		`  if(i > j)`
		`  k = i;`
		`  else`
		`  k = j;`
		`  return k;`
		`}`

	프로가드 난독화후 디컴파일	`private static int a(int i, int j){`
		`  if(i > j)`
		`  j = i;`
		`  return j;`
		`}`

<표 4>는 조건문 if에 대한 난독화 결과이다. 원본 자바 코드는 dex로 변경되어 다시 디컴파일 되었을 때, 메소드 인자로 사용된 변수인 x, y, 그리고 지역 변수인 max가 각각 i, j, k 로 이름이 변경되었다. 그러나 메소드 이름 Max는 변경되지 않았고, if 문장 구조도 원본 자바 소스코드와 동일하게 유지되었다.

프로가드로 난독화한 후 디컴파일한 결과를 확인해보면 Max 함수명이 a로 변경되었으며 불필요한 지역 변수인 max가 제거되었고, 메소드 인자로 사용된 i, j만으로 메소드가 동작하도록 최적화되었다. 즉, 프로가드 난독화에 의해서 이름이 변경되었을 뿐만 아니라 최적화도 수행된 것을 확인할 수 있다.

<표 5>는 반복문 for에 대한 난독화 결과이다. 원본 자바 코드는 dex로 변경되어 다시 디컴파일 되었을 때, 메소드 이름인 Sum은 유지되었으며, 메소드 인자로 사용된 변수인 start와 end, i 가 각각 i, j, l로 변경되었다. 그러나 함수 이름 Sum은 변경되지 않았다. 추가로 for 문장 구조는 do와 while 구조로 변경되었다.

프로가드로 난독화한 후 디컴파일한 결과를 확인해보면 Sum 함수명이 b로 변경되었으며 반복문의 인덱스로 사용된 불필요한 지역 변수인 l이 제거되었다. 즉, i, j, k만으로 메소드가 동작하도록 최적화된 결과를 보여준다.

<표 5> for 문장의 난독화 결과

Sum 함수	원본 자바 소스코드	`int Sum(int start, int end) { int sum = 0; for( int i = start; i <= end; i++ ) sum += i; return sum; }`
	dex 변환후 디컴파일	`int Sum(int i, int j) { int k = 0; int l = i; do { if(l > j) return k; k += l; l++; } while(true); }`
	프로가드 난독화후 디컴파일	`private static int b(int i, int j) { int k = 0; do { if(i > j) return k; k += i; i++; } while(true); }`

3.4 프로가드의 한계

프로가드는 안드로이드 환경에서 손쉽게 적용할 수 있는 난독화 도구이다. 그러나 다음과 같은 한계를 가지고 있다.

첫째, 프로가드는 문자열을 난독화하지 않는다. 따라서 문자열 형태로 중요한 정보가 저장되었을 경우에 이 정보는 역공학 분석에 의해서 쉽게 노출될 수 있다. 이 문제를 해결하기 위해서는 문자열을 작은 조각으로 분할하여 저장하고 있다가 재조합해서 사용하거나, 암호화 및 복호화 함수를 사용할 수도 있다. 다만 이러한 암호화 및 복호화는 실행 시간에 영향을 끼칠 수 있다.

둘째, 프로가드는 제어 흐름 난독화를 의도적으로 수행하지는 않는다. 제어 흐름 난독화는 기존 프로그램에 없었던 분기문을 추가하는 것과 같이 제어 흐름을 원본 프로그램과 다르게 바꾸는 것을 의미한다. 비록 프로가드가 제어 흐름 난독화를 의도적으로 수행하지 않더라도 프로가드의 최적화 기능이 제어 흐름을 바꾸는 경우가 발생하여 디컴파일러의 분석을 어렵게 만드는 효과를 거두기도 한다. 제어 흐름 난독화를 의도적으로 수행하는 것은 프로그램의 실행 성능을 떨어뜨릴 수 있고, 프로그램의 사이즈도 커지는 문제가 생길 수 있다.

4. 결론 및 향후 과제

안드로이드는 자바를 기반으로 하기 때문에 자바와 마찬가지로 역공학 도구에 의해서 쉽게 분석될 수 있다는 문제점이 있다.

본 논문에서는 역공학 분석을 어렵게 하기 위한 난독화 도구 중에서 안드로이드 SDK에 포함되어 널리 사용되고 있는 프로가드에 대해서 대표적인 기능 및 사용법, 프로가드로 난독화된 코드가 원본과 비교하여 어떻게 다른지 예제를 통해서 알아보았다. 그 결과, 프로가드의 난독화에 의해서 코드 이해를 어렵게 하는 난독화 효과와 더불어 코드 크기를 줄이는 최적화 효과까지 얻을 수 있음을 확인하였다.

그러나 현재 프로가드는 문자열을 난독화하지 않고 제어 흐름을 의도적으로 난독화하지 않는다는 한계가 있다.

이러한 한계를 극복하고자 문자열 난독화 알고리즘과 제어 흐름 알고리즘을 구현한다면 난독화 강도는 지금보다 높아질 수 있으나 프로그램 실행 속도가 늦어질 수 있으며 부가적인 난독화 코드에 의해서 프로그램 크기도 커질 수 있다는 문제를 고려해야만 한다.

참고문헌

[1] 프로가드 공식 홈페이지 http://proguard.sourceforge.net/

[2] Android developer Web site, "http://developer.android.com/guide/developing/tools/proguard.html"

[3] 프로가드 난독화 결과 http://proguard.sourceforge.net/#results.html

[4] 자바 난독화 도구 리스트, http://proguard.sourceforge.net/#alternatives.html

[5] jode, http://jode.sourceforge.net/

[6] Jarg, http://jarg.sourceforge.net/

[7] yGuard, http://www.yworks.com/en/products_yguard_about.htm

[8] Soot, http://www.sable.mcgill.ca/soot/

[9] JBCO, http://www.sable.mcgill.ca/JBCO/

[10] DashOPro, http://www.preemptive.com/products/dasho/overview

[11] KlassMaster, http://www.zelix.com/klassmaster/index.html

[12] smokescreen, http://www.leesw.com/smokescreen/index.html

[13] SandMark, http://sandmark.cs.arizona.edu/

[14] mBooster, http://www.innaworks.com/mBooster

[15] 프로가드 매뉴얼, http://proguard.sourceforge.net/index.html#manual/usage.html

[16] jad decompiler, http://www.varaneckas.com/jad