한국컴퓨터정보학회 동계학술대회 논문집 제22권 제1호 (2014. 1)

안드로이드 리소스 도용 방지를 위한 난독화 도구의 설계 및 구현

박희완*, 김흥수°
°*한라대학교 정보통신방송공학부
e-mail:heewanpark@halla.ac.kr*, jaengsama@nate.com°

Design and Implementation of An Obfuscation Tool for Preventing the Theft of Android Resources

Heewan Park*, Heung-Soo Kim°
°*School of Info. & Comm., Broadcasting Engineering, Halla University.

● 요 약 ●

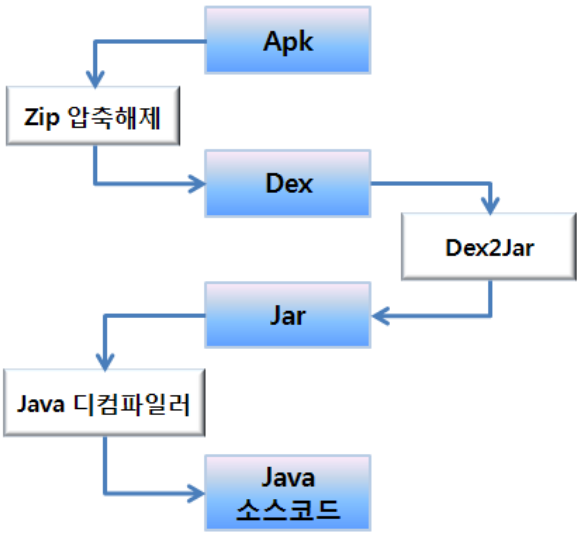
소프트웨어는 대부분 바이너리 형태로 배포되기 때문에 역공학 분석이 쉽지 않다. 그러나 안드로이드는 자바를 기반으로 한다. 자바는 클래스 파일의 형태로 배포되는데 클래스 파일은 자바 소스 프로그램의 정보를 대부분 유지하고 있기 때문에 역공학 기술을 적용하기가 타 언어에 비해 쉽다. 이 문제에 대처하기 위해서 프로그램의 기능을 그대로 유지하고, 프로그램을 분석하기 어려운 형태로 변환하는 다양한 난독화 방법이 제안되었다.
안드로이드 환경에서는 안드로이드 SDK와 함께 배포되는 난독화 도구인 프로가드(Proguard)가 가장 널리 사용된다. 그러나 프로가드는 문자열 난독화를 기능을 제공하지 않는다. 본 논문에서는 프로가드의 한계를 보완할 수 있는 문자열 난독화 기법을 제안하고 구현하였다. 본 논문에서 제안하는 문자열 난독화 기법을 먼저 소스 코드에 적용하고, 이후 프로가드의 난독화 기능을 추가로 사용한다면 안드로이드 프로그램을 역공학 분석으로부터 보호하는 효과적인 방법으로 사용될 수 있을 것이다.

키워드: 난독화(obfuscation), 디컴파일러(decompiler), 리소스(resource)

I. 서 론

자바프로그램은 일반적으로 클래스 파일 형태로 배포되고 이 클래스 파일은 하드웨어 독립적인 자바 가상기계(Java Virtual Machine : JVM)에서 실행되어야 하기 때문에 원본 자바 프로그램의 심볼릭 정보를 그대로 유지하고 있다. 그로 인해서 수많은 자바 클래스 분석 도구들과 역컴파일러가 개발되었고, 이러한 도구들을 이용하면 자바 클래스 파일에서 원본과 거의 동일한 코드를 생성할 수 있다. 즉, 자바 프로그램은 역공학 분석과 코드 도용의 위협을 받고 있다[1].

안드로이드는 자바를 기반으로 한 오픈된 모바일 플랫폼이다. 즉, 자바 언어로 프로그래밍 하여 생성된 클래스 파일을 안드로이드 플랫폼 도구인 dx를 사용하여 달빅(dalvik)코드로 변환한다. 그런데 달빅 코드는 Dex2jar[2]와 같은 도구를 사용하면 다시 자바 클래스 파일로 변환될 수 있기 때문에, 안드로이드 역시 자바의 취약점을 그대로 가지고 있고, 역공학에 의해서 쉽게 분석될 수 있다. 더욱이 안드로이드 리소스 파일은 별다른 도구 없이도 단순히 압축파일의 압축해제 만으로 쉽게 추출 해 낼 수 있다.



(그림 146) 안드로이드 Apk 파일로부터 Java 소스 코드를 추출하는 방법

예를 들어 그림1과 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[7]. 특히, 이 과정에서 안드로이드 리소스 파일은 Apk파일의 Zip압축해제 단계에서 쉽게 추출해 낼 수 있다. 이렇게 쉽게 안드로이드 애플리케이션의 이미지, 사운드, 데이터베이스 등 중요한 콘텐츠파일을 추출해 낼 수 있다.

이 문제에 대처하기 위해서 프로그램의 기능을 유지하면서 코드를 다른 형태로 변환해주는 난독화 기법이 계속해서 제안되고 있다[3,4,5,6]. 기존의 난독화 기법들은 코드 난독화에 집중되어 있고 리소스에 대한 보호기법은 관심 밖이었다. 난독화를 통해 완전한 프로그램의 보호는 불가능 하지만 프로그램이 불법 복제나 역공학에 의해서 공격당하는 것을 어렵게 만들어 지적 재산권 침해를 예방할 수 있다.

본 논문에서는 이러한 안드로이드 애플리케이션의 취약점을 보완하기 위해 안드로이드 리소스 난독화 기법을 설계하고 구현하였다.

II. 관련 연구

자바를 기반으로 한 안드로이드는 역공학 도구들로 분석되기 쉬워 코드, 리소스 도용이나 불법복제에 쉽게 노출되어 있다.

	_id	word	meaning
1	1	advertising copy	광고문구
2	2	precede	선행하다, 앞서다
3	3	secretary	비서, 장관
4	4	ingredient	(요리의)재료
5	5	awful	끔찍한, 지독한
6	6	secondhand	중고의, 간접적인
7	7	finalize	~의 결말을 짓다
8	8	overhead compa	머리 위 짐칸
9	9	quantity	양
10	10	officer	사무원, 직원
11	11	steady growth	꾸준한 증가(성장)
12	12	perform at local f	지역 축제에서 공연
13	13	comparable	비교할 만한
14	14	figure out	이해하다
15	15	pass out	내놓다, 죽다
16	16	contact	연락하다

(그림 2) 안드로이드 Apk 에서의 데이터베이스 추출

그림2는 실제 구글 스토어에 등록된 영어단어 암기장 애플리케이션에서 추출해낸 데이터베이스 파일이다. 프로그램의 중요한 데이터를 이렇게 쉽게 Apk파일의 압축해제만으로 추출 가능하다. 또한, 개발자 본인의 서명만 필요한 셀프 사이닝(self signing)을 사용하기 때문에 안드로이드 공식 마켓 외의 경로로도 앱 배포가 가능하다.

따라서 안드로이드 앱은 난독화 기술이 절실하게 필요하게 되었고, 구글은 난독화를 위한 도구로써 프로가드[9]를 Android SDK에 포함하여 배포하고 있다.

프로가드는 식별자 변환에 의해 클래스명과 변수명, 메소드명을 변경하며 적용 시 성능 저하가 없어 성능 측면에서 모바일 환경에 적합하다[8]. 하지만 프로가드는 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.

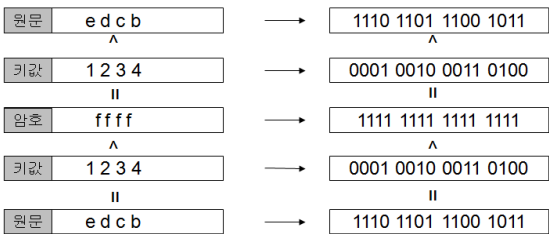
둘째, 프로가드는 제어 흐름 난독화를 의도적으로 수행하지 않는다. 제어흐름 난독화는 기존 프로그램에 없었던 분기문을 추가하는 것과 같이 제어 흐름을 원본 프로그램과 다르게 바꾸는 것을 의미한다.

셋째, 프로가드는 리소스파일에 대한 보호를 전혀 고려하지 않는다. 즉, 안드로이드 애플리케이션은 리소스파일이 공개된 상태로 사용자들에게 배포된다.

이와 같은 한계로, 프로가드는 난독화 결과에 대해서 100% 완벽한 안정성을 보장하지는 않는다. 따라서 프로가드의 한계를 보완할 수 있는 별도의 난독화가 필요하다.

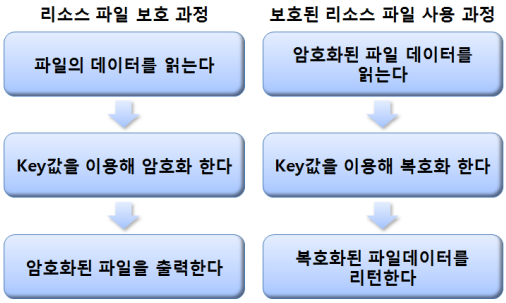
III. XOR 연산을 이용한 리소스 난독화 방법

본 논문에서 제시한 난독화 방법은 안드로이드 애플리케이션의 리소스를 난독화하기 위해서 비트연산자 XOR을 사용한다. 비트연산자 XOR을 이용한 암/복호화 원리는 다음과 같다.



(그림 3) XOR을 이용한 암/복호화

그림3과 같이 원문 'edcb'를 킷값 '1234'와 XOR 연산을 하여 나온 암호 'ffff'를 다시 킷값으로 XOR 연산을 거치면 평문이 나오게 된다. 이러한 XOR의 특성을 이용해 리소스 파일의 비트스트림을 암호화하고 암호화된 비트스트림을 다시 원래의 비트스트림으로 복구하는 것을 동일한 알고리즘을 통해서 가능하게 한다.



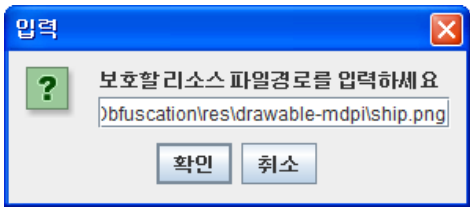
(그림 4) XOR을 이용한 리소스 파일 보호 및 보호된 리소스파일 사용 단계

제안한 리소스 난독화 도구는 그림4와 같이 안드로이드 프로그램 실행 시 원래의 리소스 파일이 원본 프로그램의 실행결과와 같은 결과가 나올 수 있게 프로그램 코드에 파일데이터 복호화 엔진 코드를 추가한다. 그리고 암호화된 각각의 리소스를 추가된 복호화 엔진 메소드의 파라미터로 보낸다. 이와 같은 방법으로 본 논문에서 설계하고 구현한 도구는 리소스파일을 암/복호화 하여 원래프로그램의 기능은 그대로 유지하면서 안드로이드 애플리케이션의 Apk파일로부터 콘텐츠를 추출하기 어렵게 만드는 역할을 한다.

IV. XOR연산을 이용한 리소스 난독화 도구 구현

안드로이드 리소스파일 난독화를 구글 스토어에 등록되어 있는 애플리케이션의 리소스에 적용하여 결과를 살펴보고자 한다. 실험에 사용한 리소스파일은 영어사전의 데이터베이스 파일과 게임 이미지, 사운드파일 등으로 하였다.

실험은 안드로이드 Apk 파일의 압출을 풀어 나온 리소스 파일을 본 논문에서 제안한 도구를 거친 결과, 암호화된 파일을 복호화 하여 나온 결과파일을 확인하여 도구를 거친 후 리소스 보호가 잘 되었는지를 확인하고, 복호화 과정을 거쳤을 때 원래의 파일로 돌아오는지를 확인하는 방식으로 진행하였다.



(그림 5) 안드로이드 리소스 난독화 도구

그림5는 안드로이드 리소스 난독화 도구의 실행 화면이다. 리소스 파일을 읽어 들여 사용자가 지정한 킷값과 XOR 연산하여 암호화 한다. 암호화된 리소스는 안드로이드에서 구현 시 별도로 추가된 엔진을 거쳐 암호화에 사용된 킷값과 XOR 연산하여 복호화 되어 사용된다.

이미지 파일 난독화 실험에 사용된 파일은 SUNDAYTOZ의 게임 '애니팡(v1.1.12)' Apk 파일에서 추출해낸 이미지를 사용하였다. 이미지 확장자는 안드로이드 애플리케이션에서 가장 많이 사용되는 PNG 파일이다.

PNG 파일 헤더

```
00 00 00 0D 49 48 44 52  .PNG........IHDR
08 06 00 00 00 41 C3 7B  ...-...j.....A.{
42 02 40 C0 7D C5 00 00  .....sRGB.@.}...
```

(그림 6) 암호화 이전의 PNG파일

암호화 된 PNG 파일 헤더

```
0B 0B 0F 03 68 64 73 1C  .[AI,&-D....hds.
03 0D 0F 0E 21 6D F4 35  ...#!,4$....!m.5
49 09 4F CE 5C E9 37 4E  .... _e.I.O.₩.7N
```

(그림 7) 암호화 된 PNG파일

그림6을 보면 안드로이드 애플리케이션 '애니팡'에서 추출한 PNG 파일의 헤더가 제대로 나와 있다. 반면 본 논문에서 제안한 난독화 도구로 암호화된 파일의 헤더는 그림 7과 같이 알 수 없는 형태로 되었다.

데이터베이스 파일 난독화 실험에 사용된 파일은 WaterBear Soft의 영어사전 애플리케이션 'Unforget (v1.1.4)' Apk 파일에서 추출한 데이터베이스 파일로 하였다. 이 파일은 사전 애플리케이션의 핵심 데이터인 영어단어의 정보가 저장된 데이터베이스 파일 중 하나인 'db_sample1.sqlite'로 하였다.

DB 파일 헤더

```
6F 72 6D 61 74 20 33 00  SQLite format 3.
00 00 00 6A 00 00 00 0B  .....@  ...j....
00 00 00 06 00 00 00 04  ................
```

(그림 8) 암호화 이전의 DB파일

암호화 된 DB 파일 헤더

```
64 79 62 6F 55 0C 04 4E  XZCgUI.(dyboU..N
0B 0B 0F 64 21 2C 37 45  ....!1.n...d!,7E
0B 0B 0F 08 21 2C 37 4A  ....!,7N....!,7J
```

(그림 9) 암호화 된 DB파일

암호화 되지 않은 DB파일을 보면 그림8과 같이 DB파일의 헤더가 'SQLite format 3'로 정확히 나온다. 반면, 암호화된 DB파일의 헤더부분은 그림 9와 같이 알 수 없는 형태로 되었다.

암호화된 파일은 이미지, 텍스트, 사운드, 데이터베이스 파일 등 전부 알 수 없는 형태로 되어 실행 불가능한 상태가 되었다. 암호화된 파일을 복호화 하였을 때에는 데이터스트림이 원본의 결과와 같음을 알 수 있고, 파일 실행 결과 또한 원본파일의 실행 결과와 똑같음을 알 수 있었다.

다음 실험으로 난독화도구로 보호된 파일을 실제 안드로이드 애플리케이션에 사용하여 보호되기 전과 같은 결과가 나오는지 확인하는 방식으로 진행하였다. 실험에 사용된 리소스는 안드로이드 SDK에서 기본으로 제공해주는 예제프로그램 'JetBoy'를 이용하였다.

파일명	ship	obf_ship
헤더정보	.png	알 수 없음
view 가능여부	O	X
실행결과		

(그림 10) 보호된 리소스의 안드로이드 실행결과

그림 10을 보면 보호 이전의 리소스(ship)와 보호된 리소스(obf_ship)의 실행결과가 같음을 볼 수 있다. 또한, 위 애플리케이션을 디컴파일하여 ship파일과 obf_ship파일을 추출하여 분석결과 ship파일은 헤더정보와 내용을 확인할 수 있었고, obf_ship 파일은 불가능하였다. 이와 같은 실험으로 본 논문에서 제시한 난독화 도구로 안드로이드 애플리케이션의 리소스를 보호 할 수 있음을 확인하였다.

V. 결 론

안드로이드는 자바를 기반으로 한다. 즉, 자바 언어로 프로그래밍하여 생성된 클래스 파일을 안드로이드 플랫폼에 포함되어 있는 도구인 dx를 사용하여 안드로이드용 달빅 코드로 변환한다. 따라서 안드로이드 역시 자바의 취약점을 그대로 가지고 있고, 자바를 기반으로 한 안드로이드는 역공학 도구들로 분석되기 쉬워 코드, 리소스 도용이나 불법복제에 쉽게 노출되어 있다. 안드로이드 애플리케이션의 리소스는 이미지, 사운드, 데이터베이스 파일 등으로 프로그램의 중요한 콘텐츠들을 담고 있다. 이렇게 프로그램의 중요한 데이터를 쉽게 Apk파일의 압축해제만으로 추출 가능하다. 또한, 개발자 본인의 서명만 필요한 셀프 사이닝(self signing)을 사용하기 때문에 안드로이드 공식 마켓 외의 경로로도 앱 배포가 가능하다.

이를 방지하기 위해 안드로이드 SDK에 포함되어 배포되는 난독화 도구 프로가드가 있다. 하지만 프로가드는 리소스 보호를 하지 않고, 제어 흐름 난독화를 의도적으로 수행하지 않는 한계가 있다.

이러한 프로가드의 한계를 보완할 수 있는 난독화 도구로서 본 논문에서는 안드로이드 애플리케이션에 사용되는 리소스를 난독화하는 도구를 설계 및 구현하였다. 제안한 난독화 기법을 사용하면 안드로이드 리소스 파일들은 XOR 연산을 사용해 암호화되고 암호화된 파일들은 안드로이드 프로그램이 실행될 때 자동으로 복호화되면서 실행된다. 따라서 본 논문에서 제안한 난독화 도구는 원래 프로그램의 기능을 그대로 유지하면서 안드로이드 프로그램에 대한 악의적인 공격들의 분석 비용을 증가시켜 프로그램의 지적 재산권을 보호하는데 기여할 수 있다.

VI. 향후 연구 과제

본 논문에서 설계한 리소스 난독화 방법의 향후 연구 과제를 요약하였다.

첫째, 제안한 리소스 난독화 도구를 이용해 암호화된 리소스를 사용할 경우 파일 호출 시간이 길어진다는 문제점이 있다. 하지만 원본 프로그램과 암호화된 리소스를 사용하는 프로그램의 성능에는 큰 차이가 없으며 차후 자료구조, 알고리즘 수정으로 성능 향상이 가능한 부분이다.

둘째, 본 논문에서 제안한 난독화 도구를 사용 시 안드로이드 애플리케이션 코드에 리소스를 복호화 하기위한 메소드가 추가된다. 또, 이 메소드를 사용하기위해 리소스 사용 시 리소스를 복호화 메소드의 파라미터로 넘겨주어야 한다. 하지만 이 부분이 자동화 되어 있지 않아 개발자가 직접 입력해야하는 불편함이 있다. 차후 리소스 난독화 도구를 업그레이드 하여 난독화의 자동화가 이루어진다면 더욱 편리하게 리소스를 보호 할 수 있을 것이다.

셋째, 리소스파일을 복호화 하기 위해 프로그램 코드에 추가된 엔진을 분석하여 리소스를 원래의 상태로 복원 가능하다. 즉, 본 논문에서 제안한 리소스 난독화 도구는 분석비용을 증가시키지만 분석을 원천적으로 차단하지는 못한다. 하지만 안드로이드 SDK와 함께 배포되는 난독화 도구인 프로가드를 이용해 프로그램 코드에 추가된 엔진을 난독화 해주어 함께 사용하면, 안드로이드 애플리케이션의 콘텐츠도용을 어렵게 하여 지적재산권 침해 방지에 기여할 수 있다.

덧붙여, 리소스 난독화뿐만 아니라 프로그램에 포함된 알고리즘 구조 자체를 변경시키는 다양한 난독화 알고리즘을 추가하는 등 연산 난독화도 고려해야 할 사항이다.

참고문헌

[1] P. Kouznetsov, "Jad - the fast JAva Decompiler," http://kpdus.tripod.com/jad.html.

[2] dex2jar, Tools to work with android .dex and java .class files, http://code.google.com/p/dex2jar/.

[3] C. Collberg, C. Thomborson, D. Low, "A Taxonomy of Obfuscating Transformation," Technical Report #148, Department of Computer Science, The University of Auckland, 1997.

[4] C. Collberg, C. Thomborson, D. Low, "Breaking Abstractions and Unstructuring Data Structures," Proc. of the International Conference on Computer Languages, ICCL98, pp.28-38. 1998.

[5] H. Park, S. Choi, T. Han, "Advanced Operation Obfuscating Techniques using Bit-Operation," Transactions on

Programming Languages, vol.17, no.3, 2003.

[6] C. Collberg, C. Thomborson, D. Low, "Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs," Proc. of the Principles of Programming Languages, POPL98, pp.184-196, Jan. 1998.

[7] H. Park, H. Park, K. Ko, K. Choi, J. Youn, "An Evaluation of the Proguard, Obfuscation Tool for Android,", Proc. of the 37th KIPS conference, April 2012.

[8] P. Yuxue, J. Jung, J. Lee, "The Technological Trend of the Mobile Obfuscation," Information & Communications Magazine, vol.29, no.8, pp.65-71, Jul. 2012.

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