http://dx.doi.org/10.5392/JKCA.2016.16.05.013

안드로이드 콘텐츠 저작권 침해 방지를 위한 서버 기반 리소스 난독화 기법의 설계 및 구현

Design and Implementation of Server-based Resource Obfuscation Techniques for Preventing Copyrights Infringement to Android Contents

박희완
한라대학교 정보통신방송공학부

Heewan Park(heewanpark@halla.ac.kr)

요약

소프트웨어는 대부분 바이너리 파일 포맷으로 배포되기 때문에 역공학 분석이 쉽지 않다. 그러나 안드로이드는 자바를 기반으로 하며 가상머신 위에서 동작한다. 따라서 안드로이드 역시 자바와 유사하게 역공학 도구에 의해서 쉽게 분석될 수 있다. 이 문제를 극복하기 위해서 다양한 난독화 기법이 제안되었다. 안드로이드 환경에서는 안드로이드 SDK에 포함되어 배포되는 난독화 도구인 프로가드(Proguard)가 가장 널리 사용된다. 프로가드는 자바 소스 코드를 역공학 분석으로부터 보호할 수 있다. 그러나 이미지, 사운드, 데이터베이스와 같은 리소스를 보호하는 기능은 가지고 있지 않다. 본 논문에서는 안드로이드 앱의 리소스를 보호할 수 있는 리소스 난독화 기법을 제안하고 구현하였다. 본 논문에서 제안하는 리소스 난독화 기법을 적용하면 효과적으로 리소스 도용을 예방할 수 있을 것으로 기대한다.

■ 중심어 : | 안드로이드 애플리케이션 | 리소스 보호 | 리소스 난독화 | 프로가드 |

Abstract

Most software is distributed as a binary file format, so reverse engineering is not easy. But Android is based on the Java and running on virtual machine. So, Android applications can be analyzed by reverse engineering tools. To overcome this problem, various obfuscation techniques are developed. In android environment, the Proguard is most widely used because it is included in the Android SDK distribution package. The Proguard can protect the Java source code from reverse engineering analysis. But it has no function to protect resources like images, sounds and databases. In this paper, we proposed and implemented resource obfuscation framework to protect resources of android application. We expect that this framework can protect android resources effectively.

■ keyword : | Android Application | Resource Protection | Resource Obfuscation | Proguard |

I. 서 론

자바 응용 프로그램은 일반적으로 확장자가 .class인 클래스 파일 형태로 배포되거나, 여러 개의 클래스 파일을 하나의 파일로 압축한 포맷인 .jar 형태로 배포된다. 그런데 클래스 포맷은 exe 파일과 같이 완전한 바이

접수일자 : 2015년 12월 23일
수정일자 : 2016년 02월 01일
심사완료일 : 2016년 02월 23일
교신저자 : 박희완, e-mail : heewanpark@halla.ac.kr

너리 형식이 아니라 바이트코드라는 중간언어 형식이다. 따라서 바이트코드에는 원본 자바 프로그램의 심볼릭 정보를 대부분 그대로 유지하고 있으며 분석이 매우 용이하다는 특징이 있다. 그로 인해서 수많은 분석 도구들과 역컴파일러가 개발되었고, 역공학 분석과 코드 도용의 위협을 받고 있다[1].

안드로이드는 자바를 기반으로 한 오픈 모바일 플랫폼이다. 즉, 자바 언어로 프로그래밍하여 생성된 클래스 파일을 안드로이드 달빅(Dalvik) 코드로 변환한다. 그런데 달빅 코드는 Dex2jar[2]와 같은 도구를 사용하면 자바 클래스 파일로 역변환될 수 있기 때문에, 안드로이드 역시 자바의 취약점을 그대로 가지고 있다. 더욱이 안드로이드 앱에 사용된 이미지, 사운드, 데이터베이스와 같은 리소스 파일들은 별다른 도구 없이도 단순히 압축 해제만으로 쉽게 추출 해 낼 수 있다.

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[3]. 특히, 이 과정에서 안드로이드 리소스 파일은 apk파일의 압축해제 단계에서 쉽게 추출해 낼 수 있다.

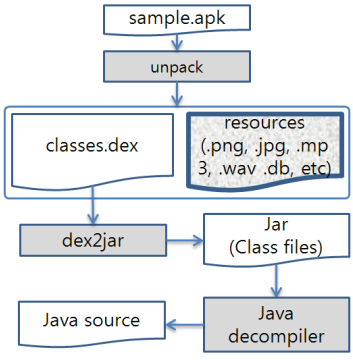


그림 1. 안드로이드 패키지로부터 자바 소스와 리소스를 추출하는 과정

이렇게 쉽게 안드로이드 어플리케이션의 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[4-7]. 기존의 난독화 기법들은 코드 난독화에 집중되어 있고 리소스에 대한 보호 기법은 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 어플리케이션의 취약점을 보완하기 위해 안드로이드 리소스 난독화 기법을 설계하고 구현하였고 성능에 대한 평가를 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 살펴보고, 3장과 4장에서 본 논문에서 제안하는 리소스 난독화 기법을 소개하며, 5장에서 실험 및 평가를 하고, 6장에서 결론과 향후 연구 과제에 대해서 논의한다.

II. 관련 연구

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

_id	word	mean
67	attach	붙이다, 압류하다, 부수하다, 소속되다
68	attain	달성하다
69	attract	끌어당기다, 유인하다, 얻다, 매력이 있다
70	available	이용할 수 있는, 채용 가치가 있는
71	awaken	깨우다
72	balance	균형, 평형을 잡는 것, 지배력, 평가
73	barely	간신히
74	barren	불모의, 열매를 맺지 않는, 초라한, 무력한
75	barrier	방벽, 개찰구, 출발구, 말뚝 울타리
76	beneficial	유익한, 수익권이 있는
77	bias	선입견, 경향, 편의의
78	blend	섞다, 뒤섞이다, 혼색, 블렌드
79	blossom	꽃, 청춘, 발달하여 되다
80	bold	대담한, 도전적인, 굵은

그림 2. 안드로이드 앱으로부터 추출한 데이터베이스

[그림 2]는 구글 스토어에 등록된 영어단어 암기장 어플리케이션에서 추출해낸 데이터베이스 파일이다. 프로그램에서 중요하게 다루어지는 데이터를 간단하게 apk 파일의 압축해제만으로 추출 가능하다. 또한, 개발자 본인의 서명만 필요한 셀프 사이닝(self signing)을 사용하기 때문에 안드로이드 공식 마켓 외의 경로를 통

해서 앱 배포가 가능하다. 따라서 안드로이드 앱은 난독화 기술이 절실하게 필요하게 되었고, 구글은 난독화를 위한 도구로써 프로가드[8]를 안드로이드 SDK에 포함시켜 배포하고 있다.

프로가드는 식별자 변환에 의해 클래스명과 변수명, 메소드명을 변경하며 적용 시 성능 저하가 없어 성능 측면에서 모바일 환경에 적합하다[9]. 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 수행하지 않는다. 제어흐름 난독화는 기존 프로그램에 없었던 분기문을 추가하는 것과 같이 제어 흐름을 원본 프로그램과 다르게 바꾸는 것을 의미한다. 셋째, 프로가드는 리소스 파일에 대한 보호를 전혀 고려하지 않는다. 즉, 안드로이드 어플리케이션은 기본적으로 리소스파일이 단순히 압축시킨 형태로 사용자들에게 배포된다. 이와 같은 한계로, 프로가드는 난독화 결과에 대해서 100% 완벽한 안전성을 보장하지는 않는다. 따라서 프로가드의 한계를 보완할 수 있는 별도의 난독화가 필요하다.

리소스 보호와 관련된 기존 연구[10][11]와 본 논문과의 차별화된 내용은 다음과 같이 요약할 수 있다.

첫째, 기존 연구는 난독화에 사용된 킷값이 앱 자체에 포함되어있어서 정적분석에 매우 취약했다. 본 논문에서는 서버 기반 난독화 기법을 제안하여 정적 분석에도 킷값이 노출되지 않도록 하였다.

둘째, 기존 논문에서는 제시하지 않았던 실행 성능 측정 및 난독화에 따르는 오버헤드를 구체적으로 평가하였다.

III. 리소스 암호화 알고리즘

3.1 리소스 보호를 위한 기본 설계 원칙

안드로이드 리소스를 보호하기 위해서는 리소스 암호화가 반드시 필요하다. 리소스를 암호화 알고리즘에 대한 선택은 다양할 수 있지만, 암호화에 사용되는 킷값이 모바일 앱 내부에 저장될 경우에는 정적 분석에 의해서 킷값이 노출될 수 있고, 리소스를 더 이상 보호할 수 없다.

따라서 본 논문에서는 리소스 보호를 위한 기본 원칙을 다음과 같이 두 가지로 정했다. 첫째 암호화 및 복호화에 사용되는 킷값은 앱 내부에 저장하지 않고 서버로부터 전송 받는다. 둘째 암호화에 사용되는 알고리즘은 리소스의 중요도 및 앱의 실행 성능을 고려하여 다양한 선택을 할 수 있다.

3.2 암호화 알고리즘의 선택

현대 암호 알고리즘 중 대표적인 대칭 암호화 알고리즘은 DES(Data Encryption Standard) 알고리즘과 AES(Advanced Encryption Standard)알고리즘이 있다.

DES 알고리즘[12]은 대칭키 기반의 암호화 알고리즘으로 블록 암호화 방식을 사용하여 전송 정보의 해당 64비트 평문 블록에 적용되어 동일한 길이의 암호문을 생성한다. DES의 단점은 전수공격에 취약하다는 점이다. 실제로 컴퓨터의 성능발달과 해독알고리즘의 개량으로, 전수조사에 의한 DES암호 공격이 실행되어 1999년 1월 18일 DES Challenge III에서 1만 여대의 컴퓨터와 전용칩을 이용하여 DES암호를 22시간 15분만에 해독을 하였다[13].

AES 알고리즘[14]은 기존의 암호 표준인 DES의 안정성에 대한 논란이 대두되자 미국 국립 표준 기술 연구소(NIST)가 DES를 대체할 알고리즘으로 발표하여 미국 정보 표준으로 지정된 블록암호 알고리즘이다. AES는 현재 전 세계적으로 가장 널리 사용되고 있는 대표 블록 암호 알고리즘이며 알고리즘에서 사용되는 키 크기에 따라서 AES-128, AES-192, AES-256으로 구별된다.

리소스 암호화를 위해서 다양한 알고리즘이 선택될 수 있으나 본 논문에선 암호화의 계산 복잡도가 낮지만 동작 속도가 매우 빠른 XOR 알고리즘을 사용하였다. 그 이유는 모바일 환경에서 동작하는 앱의 성능을 고려했기 때문이다. XOR 알고리즘의 경우에는 이미 리소스 헤더 정보가 알려진 경우에 킷값을 역으로 유추해낼 수 있는 가능성이 존재한다. 따라서, 암,복호화에 소요되는

성능 저하보다도 리소스의 보호가 더 우선될 경우에는 DES 또는 AES와 같이 검증된 알고리즘을 사용하는 것을 반드시 고려해야만 한다.

3.3 XOR 연산을 이용한 리소스 난독화

본 논문에서 제시하는 안드로이드 리소스 난독화 방법은 리소스를 난독화하기 위해서 비트연산인 XOR을 사용한다. 암/복호화의 기본 원리는 다음과 같다.

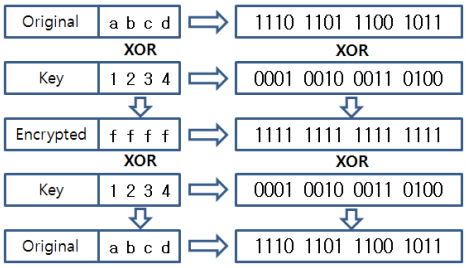


그림 3. XOR 연산을 통한 암호화와 복호화 프로세스

[그림 3]과 같이 원문 'abcd'를 킷값 '1234'와 XOR 연산을 하여 나온 암호 'ffff'를 다시 킷값으로 XOR 연산을 거치면 원문이 나오게 된다. 이러한 XOR의 특성을 이용하면 원본 리소스 파일의 비트스트림을 암호화하고, 암호화된 비트스트림을 다시 원래의 비트스트림으로 복구하는 것을 동일한 알고리즘으로 처리할 수 있다.

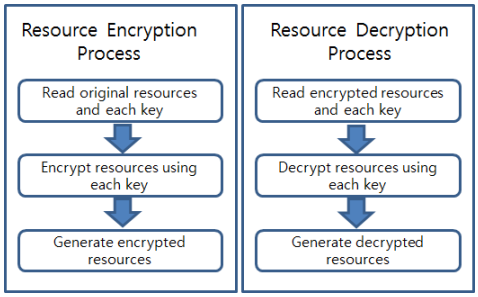
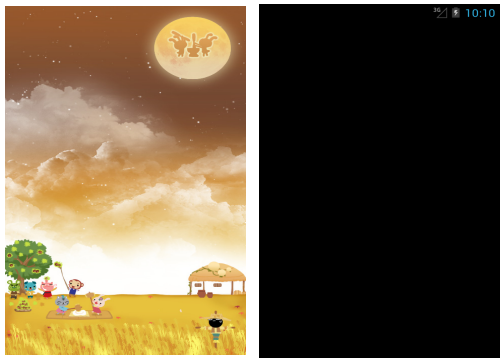


그림 4. 리소스 암호화와 복호화 프로세스

본 논문에서 제안하는 리소스 난독화 도구의 암/복호화는 [그림 4]와 같이 동작한다. 안드로이드 앱 개발 과정에서 보호하고자 하는 리소스를 선별하고 리소스 암호화 도구를 사용해서 변환한다. 즉, 리소스 난독화가 적용된 앱은 원본 리소스 대신 암호화된 리소스를 포함한 형태로 배포된다.

리소스를 난독화 하더라도 원본 앱과 동일하게 실행되어야 한다. 따라서 리소스에 접근하는 기존 소스 코드를 변환하여 복호화 과정을 거치도록 수정하여야 한다. 복호화 루틴은 암호화에 사용한 킷값을 이용해서 앱이 실행될 때 복호화를 수행한다. 이 방법은 이미지, 사운드, 데이터베이스와 같이 리소스 종류와 상관없이 적용 가능하다는 특징이 있다.



(a) 난독화 전에 정상 이미지 파일로 인식함 (b) 난독화 후에 이미지 파일로 인식하지 못함

그림 5. 애니팡 배경 이미지 파일의 난독화 전과 후 비교

[그림 5]의 (a)는 SUNDAYTOZ 사의 게임 애니팡 1.2.6 버전의 apk 파일의 압축을 해제하여 추출한 background.png 이미지이다. [그림 5]의 (b)는 32바이트의 문자열 "f8a9af151c601ea8aa47381efa61a8c5"을 사용하여 난독화한 결과이다.

```
89 50 4E 47 0D 0A 1A 0A  00 00 00 0D 49 48 44 52
00 00 01 E0 00 00 03 54  08 02 00 00 00 E4 72 69
88 00 00 00 19 74 45 58  74 53 6F 66 74 77 61 72
                .PNG........IHDR
                .......T......ri
                .....tEXtSoftwar
```

(a) 난독화 전 애니팡 배경 이미지 파일의 헤더부분

```
EF 68 2F 7E 6C 6C 2B 3F  31 63 36 3D 78 2D 25 6A
61 61 35 D7 33 38 32 31  6E 63 36 31 61 DC 11 5C
EE 38 61 39 78 12 74 6D  45 30 59 56 45 12 00 4A
                .h/~ll+?1c6=x-%j
                aa5.3821nc61a..\
                .8a9x.tmE0YVE..J
```

(b) 난독화 후 애니팡 배경 이미지 파일의 헤더부분

그림 6. 애니팡 배경 이미지 파일의 난독화 전과 후의 헤더 부분 비교

[그림 6]의 (a)는 암호화 전 이미지 파일의 헤더(header) 정보이다. 헤더 부분에서 PNG라는 키워드를 찾을 수 있기 때문에 PNG 포맷의 이미지 파일로 인식된다. [그림 6]의 (b)는 암호화 후 이미지 파일의 헤더이다. 헤더부분에서 PNG라는 키워드가 사라지고 내용도 암호화되었기 때문에 정상적인 이미지 파일로 인식되지 못한다. 이 경우에 복호화 킷값을 알아야만 원본 파일로 복호화가 가능하다.

본 논문에서 제안한 XOR 암/복호화는 앱 자신이 해독키를 이용해서 리소스를 해독하는 모듈이 앱에 포함되어야만하기 때문에 정적 분석에 취약하다는 단점이 있다. 즉, 디컴파일러를 이용해서 안드로이드 소스 코드를 생성한 후, 소스코드를 분석하여 리소스 복호화가 이루어지는 루틴을 찾고, 해당 소스코드로부터 복호화 키값을 찾는 것이 정적 분석을 통해서 가능하다. 이 문제의 근본 원인은 복호화 킷값이 앱에 포함되어 있다는 것이다. 본 논문에서는 이 문제를 해결하기 위해서 복호화 킷값을 앱 자체에 포함시키지 않고 키 관리 서버에 저장시키는 방법을 제안하고, 앱이 실행될 때 서버에 접속하여 복호화 키를 요청하고 서버로부터 수신된 키를 사용해서 복호화를 수행하는 서버 기반 리소스 난독화 시스템을 구현하였다.

IV. 키 관리 서버를 통한 리소스 복호화

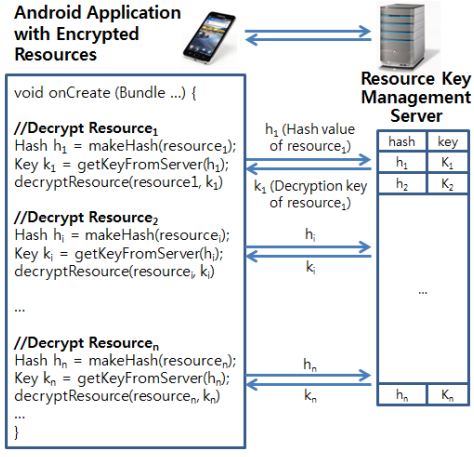


그림 7. 서버 기반 리소스 복호화 프로세스

[그림 7]은 키 관리 서버 기반 리소스 복호화 과정을 보여준다. 안드로이드 앱은 리소스 키 관리 서버에게 리소스 해시값을 전달하고, 리소스 키 관리 서버는 해당하는 리소스에 대한 복호화 킷값을 안드로이드 앱에게 돌려주는 역할을 한다. 구체적인 리소스 복호화 절차는 다음과 같다.

- Step 1. 안드로이드 앱이 실행되면 onCreate 메소드가 호출되고 이곳에서 암호화된 리소스에 대한 복호화 작업을 시작한다. 먼저 각 리소스에 대한 해시값을 계산해서 서버에 전달한다. 리소스 이름이 아니라 해시값을 전달하는 이유는 네트워크 패킷 분석을 통해서 리소스 이름이 노출될 수 있기 때문이다. 해시값을 사용하면 리소스 이름을 노출시키지 않고 안전하게 서버로부터 복호화 킷값을 받을 수 있다.
- Step 2. 리소스 서버는 안드로이드 앱으로부터 전달받은 해시값을 검색하여 해당 리소스의 복호화 킷값을 찾아서 킷값을 요청한 안드로이드 앱에게 되돌려준다.
- Step 3. 안드로이드 앱은 서버로부터 전달받은 킷값을 사용하여 암호화된 리소스를 복호화하고 메모리에 리소스를 로드한다. 이후 리소스를 사용해야 하는 상황에서는 더 이상의 복호화 작업이 불필요하며 이미 메모리에 로드된 리소스를 재사용 가능하다.

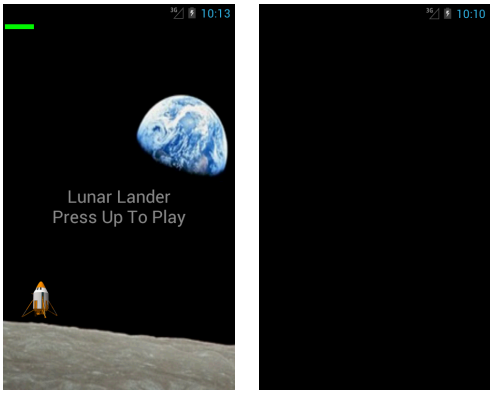
키 관리 서버 기반 리소스 난독화 및 복호화 방법은 복호화 킷값을 앱 자체에 포함시키지 않고 키 관리 서버에 저장하고, 앱이 실행될 때 서버에 접속하여 킷값을 요청하여 복호화를 수행한다. 따라서 정적분석에 의해서 킷값이 노출되는 것을 막을 수 있다.

V. 실험 및 평가

실험환경은 다음과 같다. 리소스 키 관리 서버는 2.0GHz Intel Core i7-3667U CPU, 4GB RAM, Windows 7 운영체제 상에서 Java 언어로 구현하였으며, 서버에 접속하는 안드로이드 기기로 LG Optimus-

EX, 버전은 ICS(4.0.4)를 사용하였다. 네트워크 환경은 EFM 네트웍스 ipTIME 공유기를 통해서 IEEE 802.11g(최대속도 54Mbps) Wi-Fi를 사용하였다.

실험 대상 앱은 안드로이드 SDK에 포함된 샘플 프로젝트의 하나인 'LunarLander'를 사용하였다. 그리고 복호화 키를 앱 자체에 포함시키는 로컬 난독화(Local Obfuscation) 방법과 복호화 키를 서버에 저장시키는 서버 난독화(Server Obfuscation) 방법을 모두 테스트하였다.



(a) 복호화 성공한 결과 (b) 복호화 실패한 결과

그림 8. 'Lanar Lander' 앱에 대한 복호화 실험

[그림 8(a)]는 정상적으로 복호화가 수행되어 실행된 결과이고, [그림 8(b)]는 올바른 복호화 키를 얻지 못한 경우 이미지가 정상적으로 표시되지 못한 결과를 보여준다.

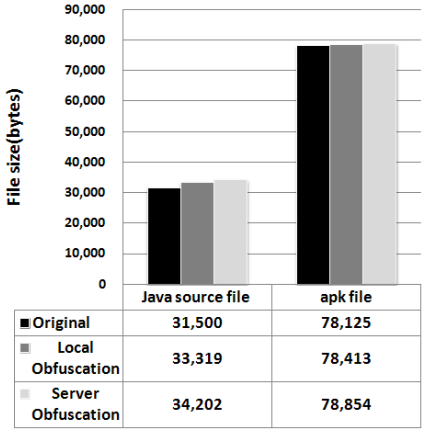


그림 9. 원본, 로컬 난독화, 서버 난독화를 적용한 파일의 크기 비교 결과

[그림 9]는 원본 앱과 로컬 난독화, 서버 난독화에 대해서 자바 소스 파일의 크기 증가 및 apk 파일의 크기 증가를 보여준다. 난독화가 적용될 경우, 원본 앱에 리소스 복호화 루틴이 포함되기 때문에 소스 및 apk 파일이 증가한다. 서버 난독화의 경우, 로컬 난독화 방식을 기반으로 서버 접속 및 복호화 키 전송 루틴이 추가된 형태이다.

난독화 전과 후의 앱의 크기를 비교했을 때, 로컬 난독화에 의해서 소스 코드는 1819byte 증가하였고, 서버 난독화에 의해서 2702kbyte 증가하였다. apk 파일은 컴파일 단계를 거치면 불필요한 정보들이 대부분 제거되어 파일 크기 증가 폭이 감소하였다. 지역 난독화는 원본 대비 288byte 증가하였고, 서버 난독화는 729byte 증가하였고, 원본 앱 대비 1%미만이었다.

[표 1]은 원본 앱에서 사용된 4개의 이미지를 복호화하는 시간과 서버로부터 키를 전송받는 시간을 정리한 표이다. 실행할 때마다 오차가 존재하기 때문에 100번을 실행한 후 평균값을 구했다. 이미지 복호화 시간은 파일 크기에 비례하여 10kbyte 미만의 작은 이미지들은 2ms 정도 소요되었고, 36kbyte의 이미지는 7.74ms 소요되었다.

표 1. 'Lunar Lander' 앱의 모든 이미지 파일에 대한 복호화 소요 시간과 서버 통신 소요 시간

Resources (image file)	Size (byte)	Decryption time(ms)	Server time(ms)
earthrise.png	36,623	7.74	37.14
land_crashed.png	8,221	1.91	37.24
lander_firing.png	8,526	2.13	38.21
lander_plain.png	7,548	1.88	40.72
total		13.66	153.31

서버 난독화의 경우 서버로부터 키를 받아오기 위해 필요한 시간이 필요하다. 이 시간은 네트워크 상황에 따라서 변동될 수 있으며 본 실험 환경에서는 37.14~40.72ms 소요되었다. 서버로부터 이미지 데이터를 직접 다운받는 것이 아니라 복호화에 필요한 킷값 32바이트만 받아오기 때문에 이미지 크기와는 상관없다. 그러나

리소스 개수가 많아질 경우 개수에 비례하여 시간이 소요될 수 있다는 단점이 있다. 따라서 실행 성능을 높이기 위해서는 모든 리소스 파일을 난독화하는 것 보다는 중요한 리소스를 선별하여 난독화하는 방법을 선택할 수 있다.

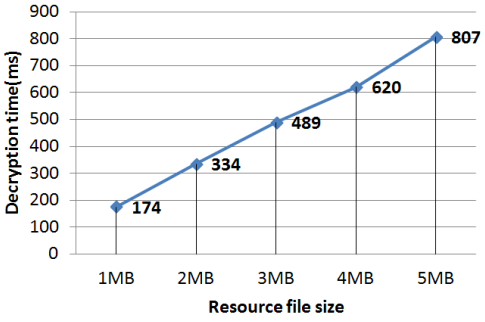


그림 10. 리소스 파일 크기의 변화에 따른 복호화 시간 증가

[그림 10]은 이미지 파일 크기에 대한 복호화 시간을 보여준다. 이미지 복호화는 파일 크기에 비례하여 증가하며 5MB의 이미지 파일의 경우 0.8초 정도 소요되는 것을 확인하였다. 모바일 게임의 경우, 게임 용량과 로딩 속도를 고려해야 하기 때문에 배경 그림의 경우에도 500KB를 넘지 않았다. 예를 들어, 실제로 애니팡 게임에서 사용된 모든 이미지 파일의 크기를 합산한 결과 1.8MB였다. 이 경우 복호화로 인한 오버헤드는 300ms 정도가 된다.

리소스의 복호화 작업은 안드로이드 앱이 처음 실행될 때 한번만 수행하면 리소스는 복호화된 형태로 메모리에 유지되고 리소스가 필요한 시점에서 언제든지 재사용할 수 있기 때문에 복호화로 인한 실행시간 오버헤드는 앱 실행 한번으로 충분하다.

5장의 실험을 통해서 확인된 리소스 난독화에 대한 시사점을 요약하면 다음과 같다.

첫째, 리소스 난독화에 의한 파일 크기 증가를 고려해보면, 난독화가 적용되어도 컴파일된 apk 파일의 증가는 원본 앱 대비 1% 미만이므로 파일 크기에 대한 부담은 무시할 수 있다.

둘째, 리소스 난독화에 소요되는 시간은 복호화 시간과 서버 통신 시간으로 구분할 수 있으며, 리소스 개수가 많아질 경우 개수에 비례하여 서버와의 통신 시간이 증가한다. 따라서 실행 성능을 높이기 위해서는 모든 리소스 파일을 난독화하는 것 보다는 중요한 리소스를 선별하여 난독화하는 방법을 선택할 수 있다.

셋째, 리소스 자체 파일이 사이가 클 경우 복호화 시간이 비례하여 증가한다. 따라서 앱의 실행 속도를 높이려면 앱 초기 구동시 자주 사용되는 리소스를 미리 복호화해서 메모리에 유지하면 실행시간 오버헤드를 줄일 수 있다.

VI. 결 론

본 논문에서는 안드로이드 리소스를 보호하기 위한 방법으로 XOR 비트 연산을 사용한 서버 기반 리소스 난독화 방법을 제안 및 구현하였고, 실험을 통해서 성능을 평가하였다.

본 논문에서 제안한 난독화의 특징은 XOR 연산을 사용하기 때문에 난독화 전과 후의 리소스 파일 크기가 동일하고, 복호화 알고리즘이 단순하여 빠르게 복호화가 수행된다는 특징이 있다.

로컬 난독화는 서버가 필요없이 빠르게 동작한다는 장점이 있다. 다만 정적분석을 통해서 복호화 킷값이 노출될 수 있는 위험이 있다. 서버 난독화는 복호화 킷값을 앱 실행시 서버로부터 받아오기 때문에 정적분석만으로는 킷값을 알 수 없다. 다만, 서버로부터 복호화 키를 받아오는 시간이 리소스 개수에 비례하므로 리소스 그룹별로 동일한 킷값을 사용하여 서버 통신을 줄이거나, 난독화 대상 리소스를 선별하는 방법을 택할 수 있다.

향후 연구 과제는 다음과 같다. 먼저 리소스 복호화의 오버헤드를 줄이는 방법이 필요하다. 만일 작은 이미지 파일이 다수 사용되는 앱의 경우에 모든 이미지 파일들을 서로 다른 킷값으로 복호화할때 서버와의 통신이 부담될 수 있기 때문이다. 이 경우에는 하나의 킷값으로 여러 개의 리소스를 그룹화하여 난독화하거나 복호화하는 방법이 사용될 수 있을 것이다.

추가로 리소스 난독화를 초보자도 쉽게 수행할 수 있

도록 도와줄 수 있는 사용자 인터페이스의 개발이 필요하다. 리소스 난독화 인터페이스에는 사용자가 원하는 리소스를 쉽게 선택할 수 있는 기능과 로컬 및 서버 난독화 기법을 선택할 수 있는 기능이 필요하다.

참 고 문 헌

[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] H. Park, H. Park, K. Ko, K. Choi, and J. Youn, "An Evaluation of the Proguard, Obfuscation Tool for Android," Proc. of the 37th KIPS conference, Vol.19, No.1, pp.730-733, 2012(4).

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

[5] C. Collberg, C. Thomborson, and D. Low, "Breaking Abstractions and Unstructuring Data Structures," Proc. of the International Conference on Computer Languages, ICCL98, pp.28-38, 1998(5).

[6] H. Park, S. Choi, and T. Han, "Advanced Operation Obfuscating Techniques using Bit-Operation," Transactions on Programming Languages, Vol.17, No.3, pp.8-20, 2003(11).

[7] C. Collberg, C. Thomborson, and D. Low, "Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs," Proc. of the Principles of Programming Languages, POPL98, pp.184-196, 1998(1).

[8] ProGuard, http://developer.android.com/tools/help/proguard.html

[9] P. Yuxue, J. Jung, and J. Lee, "The Technological Trend of the Mobile Obfuscation," Information & Communications Magazine, Vol.29, No.8, pp.65-71, 2012(7).

[10] H. Kim, K. Chae, and H. Park, "Design and Implementation of String Obfuscation Tool for Android Source Codes," Proc. of the Conference on Information Security and Cryptology - Winter 2012, Vol.22, No.2, pp.195-198, 2012(12).

[11] H. Kim and H. Park, "Design and Implementation of An Obfuscation Tool for Preventing the Infringement of Intellectual Property Rights of Android Contents," Proc. of the 2014 Spring Conference of the KIPS, Vol.21, No.1, pp.483-486, 2014(4).

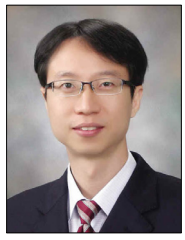
[12] National Bureau of Standards, *Data Encryption Standard*, FIPS PUB 46, 1977(1).

[13] RSA Laboratories, DES CHALLENGE III, http://www.emc.com/emc-plus/rsa-labs/historical/des-challenge-iii.htm

[14] NIST, *Advanced Encryption Standard (AES)*, FIPS PUB 197, 2001(11).

저 자 소 개

박 희 완(Heewan Park) 정회원



- 1997년 2월 : 동국대학교 컴퓨터공학과(공학사)
- 1999년 2월 : KAIST 전산학과(공학석사)
- 2010년 1월 : KAIST 전산학과(공학박사)
- 2004년 3월 ~ 2007년 2월 : 삼성전자 무선사업부 책임연구원
- 2010년 2월 ~ 2011년 8월 : ETRI 부설연구소 선임연구원
- 2011년 9월 ~ 현재 : 한라대학교 정보통신방송공학부 조교수

<관심분야> : 프로그램 난독화, 역공학, 악성코드 분석, 소프트웨어 워터마킹, 정적 및 동적 분석