Journal of the Korea Convergence Society
Vol. 11. No. 11, pp. 1-7, 2020

ISSN 2233-4890 / eISSN 2713-6353
https://doi.org/10.15207/JKCS.2020.11.11.001

레이아웃 파일 변환을 이용한 안드로이드 난독화 기법의 설계 및 구현

박희완
한라대학교 정보통신소프트웨어학과 부교수

Design and implementation of Android obfuscation technique using layout file transformation

Heewan Park
Associate Professor, Department of Information Communication and Software, Halla University

요 약 안드로이드 앱은 주로 apk 파일 형태로 배포되고, apk 파일의 압축 해제하면 앱 디자인과 관련된 xml 파일, 이미지, 사운드와 같은 리소스 파일을 추출할 수 있다. 만일 은행이나 금융과 관련된 앱의 리소스가 도용되어 가짜 앱이 배포된다면 개인정보가 유출되거나 금융사기를 당할 수 있다. 따라서 앱을 배포할 때 코드뿐만 아니라 디자인을 도용하기 어렵게 만드는 노력이 필요하다. 본 논문에서는 xml 파일을 자바 코드로 변환한 후 프로가드(Proguard)를 이용하여 난독화하는 도구를 구현하였고 실행 성능을 평가하였다. 본 논문에서 제안하는 레이아웃 난독화 기법을 사용하면 앱 구동 성능을 높일 수 있으며 화면 디자인 도용으로 인한 불법 복제 피해를 예방하는 효과도 있을 것으로 기대한다.

주제어 : 안드로이드, 난독화, 레이아웃 난독화, 리소스 난독화, 소프트웨어 보호

Abstract Android apps are mostly distributed as an apk files, and when the apk file is uncompressed, resource files such as xml files, images, and sounds related to app design can be extracted. If the resources of banking or finance-related apps are stolen and fake apps are distributed, personal information could be stolen or financial fraud may occur. Therefore, it is necessary to make it difficult to steal the design as well as the code when distributing the app. In this paper, we implemented a tool to convert the xml file into Java code and obfuscate using the Proguard, and evaluated the execution performance. If the layout obfuscation technique proposed in this paper is used, it is expected that the app operation performance can be improved and the illegal copying damage caused by the theft of the screen design can be prevented.

Key Words : Android, Obfuscation, Layout obfuscation, Resource obfuscation, Software protection

*This work was supported by 2020 Research Grant of Halla University.
(이 연구는 2020년도 한라대학교 교내연구비 지원에 의한 것임.)
*Corresponding Author : Heewan Park(heewanpark@halla.ac.kr)
Received August 31, 2020 Revised September 29, 2020
Accepted November 20, 2020 Published November 28, 2020

1. 서론

최근 네이버와 구글 같은 국내외 유명 포털 사이트를 사칭한 피싱(Phishing) 사이트가 빈번하게 포착되고 있다[1]. 피싱이란 포털사, 은행, 금융기관 등 유명 사이트를 사칭한 위장 사이트에 개인정보를 입력하도록 유도해 수집한 정보를 악용하는 공격 기법이며, 피싱 사이트는 이러한 행위를 하기 위해서 정식 사이트를 모방해서 제작된 가짜 사이트를 의미한다.

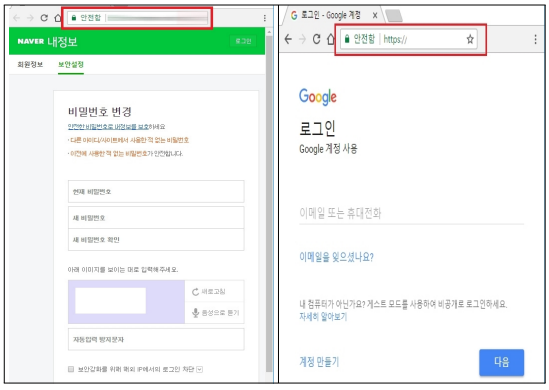


Fig. 1. Example of the web site phishing attack

Fig. 1은 홈페이지 화면 디자인을 도용한 피싱 공격에 대한 예시이다. 최근에 발견된 네이버와 구글 사칭 피싱 사이트의 경우에는 주소입력창에 안전한 사이트임을 표시하는 보안 열쇠 아이콘까지 표시하기 때문에 일반 이용자들은 정식 사이트와의 구별이 쉽지 않다[2].

유사 웹사이트를 이용한 피싱을 예방하기 위한 노력으로 HTML, Javascript, PHP 파일에 대한 난독화 노력이 연구되었다[3-5]. 웹 페이지 소스의 구성 요소인 HTML, Javascript, PHP 등을 난독화하여 쉽게 도용하기 어렵게 만들면 화면 디자인을 도용한 피싱 공격을 예방하는 효과를 거둘 수 있기 때문이다.

웹사이트뿐만 아니라 모바일 앱을 통한 피싱도 증가하고 있다. 금융 관련 앱에서 사용된 xml 파일이나 이미지가 그대로 도용되고 위·변조되어 가짜 앱이 다수 유포된다면 이러한 앱을 통해서 개인 정보가 유출되거나 금융사고가 발생할 수 있다[6].

Fig. 2. Example of the mobile application phishing attack

Fig. 2는 실제로 금융 앱의 화면을 그대로 복제한 가짜 앱을 통한 사기 사건에서 사례로 제시된 내용이다[6].

안드로이드 앱은 압축 파일인 apk 형태로 패키징되어 배포되기 때문에 apk 파일의 압축을 풀면 앱의 실행과 관련된 classes.dex 파일과 앱 디자인과 관련된 xml 파일, 그리고 이미지, 사운드와 같은 리소스 파일들을 쉽게 추출할 수 있다. classes.dex 파일은 안드로이드 스튜디오 개발 환경에 포함된 프로가드(Proguard)를 사용해서 난독화하여 보호할 수 있다. 그러나 xml 파일이나 이미지, 사운드 파일에 대한 난독화 방법은 개발 환경에서 별도로 제공하지 않는다[7,8].

일반적으로 프로그램에서 보호해야할 대상은 외형적인 디자인이 아니라 핵심 알고리즘이기 때문에 지금까지 난독화에 대한 연구는 주로 프로그램 소스 코드와 바이너리가 대상이었으며 이미지, 사운드와 같은 리소스 파일이나 xml 형식의 레이아웃 파일에 대해서는 보호에 대한 연구는 미흡했다[9-12]. 이러한 사건을 예방하기 위해서는 앱의 알고리즘뿐만 아니라 디자인과 같은 리소스를 쉽게 복제하지 못하게 만드는 노력도 필요하다.

본 논문에서는 안드로이드 앱에서 화면을 디자인하기 위해서 사용되는 xml 파일이 쉽게 도용되지 않도록 보호하기 위한 레이아웃 파일 난독화 기법을 제안한다. 레이아웃 난독화 기법을 사용하면 레이아웃 파일이 외부로 유출되는 것을 예방할 수 잇기 때문에 화면 디자인 도용을 예방할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 살펴보고, 3장에서 본 논문에서 제안하는 레이아웃 파일 난독화 기법에 대해서 소개하며, 4장에서 시스템 구현 및 평가를 하고, 5장에서 결론을 맺는다.

2. 관련 연구

안드로이드 환경에서 사용이 가능한 대표적인 난독화 도구는 프로가드(Proguard)이다[7,8]. 프로가드는 프로그램에서 사용된 변수, 메소드, 클래스 이름과 같은 식별자를 의미가 없는 간단한 문자로 변환하여 난독화시킨다. 이러한 난독화 기법은 프로그램의 성능 저하가 없기 때문에 성능이 중요한 모바일 환경에 적합하다. 하지만 프로가드는 아래와 같은 한계를 가지고 있다.

첫째, 프로가드는 앱에서 사용된 문자열을 난독화하지 않는다. 만일 문자열 형식으로 중요한 사용자 정보가 저장되었을 경우에 이것은 역공학 분석으로 노출될 위험이 있다. 둘째, 프로가드는 프로그램의 제어 흐름에 대한 난독화를 수행하지 않는다. 제어 흐름의 난독화는 분기문이나 반복문을 변환하여 원본과 다른 제어 흐름을 생성하는 것을 의미한다. 셋째, 프로가드는 앱에서 사용된 아이콘, 사운드 파일, 데이터베이스 등 다양한 리소스와 화면 구성을 위해서 사용된 xml 파일을 난독화하지 않는다. 이와 같은 한계 때문에 프로가드를 보완할 수 있는 별도의 난독화에 대한 연구가 필요하다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    orientation="horizontal"
    background="@color/bg_white"
    clickable="true"
    layout_width="fill_parent"
    layout_height="fill_parent"
    xmlns:"http://schemas.android.com/apk/res/android">
    <RelativeLayout
        orientation="vertical"
        background="#ffdddddd"
        paddingTop="@dimen/dp_13"
        layout_width="64.0dip"
        layout_height="fill_parent">
        .....
```

Fig. 3. Layout file extracted from bank application

Fig. 3은 스마트폰에 은행 앱을 설치한 후 apk 파일을 백업하고, apktool 프로그램[13]을 사용하여 은행 앱으로부터 추출한 main_menu.xml 파일의 일부 내용이다. 즉, 기존 은행 앱으로부터 메인 화면을 구성하는 xml 파일과 이미지 파일을 도용하여 가짜 은행 앱을 만들어서 배포할 수 있기 때문에 이를 예방하기 위해서는 코드뿐만 아니라 리소스 파일에 대한 난독화가 필요하다.

리소스 난독화에 관련하여 기존에 진행된 연구[14,15]에서 이미지, 사운드, 데이터베이스와 같은 리소스 파일을 난독화하는 방법이 제안되었으나 앱의 화면을 구성하는 xml 파일에 대해서는 고려하지 않았다. 기존 연구[16]에서 xml 난독화에 대한 아이디어가 제안되었으나 난독화로 인한 효과가 입증되지 않았으며 난독화에 의한 성능평가 또한 이루어지지 않았다. 본 논문에서는 기존 연구를 확장하여 난독화의 효과를 검토하고 성능평가를 추가로 수행하였다.

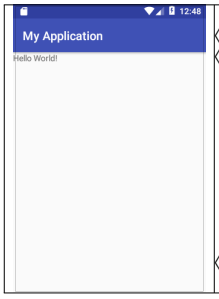
3. 안드로이드 레이아웃 난독화 기법

본 논문에서는 안드로이드 앱의 화면 구성을 위한 레이아웃 파일을 반드시 xml 형식으로 제작할 필요가 없고 자바 소스 코드를 사용할 수 있다는 점에 주목하여 다음과 같은 방법을 제안한다.

먼저, 화면을 디자인을 위한 xml 파일을 동일한 동작을 하는 Java 파일로 변경해서 기존 Java 소스의 화면 초기화 부분에 추가한다. 즉, xml 파일을 사용하지 않고 순수한 자바 Java 코드를 이용해서 화면을 구성하도록 수정한다. 그리고 Java 코드를 난독화할 수 있는 프로가드를 사용하여 전체 Java 코드를 난독화한다. 즉, 프로가드가 난독화를 지원하지 않아서 쉽게 노출될 수 있는 xml 파일을 Java 코드로 변경함으로써 난독화가 가능하고 결과적으로 원본 코드가 노출되기 어렵게 만드는 것이다.

안드로이드 xml 파일을 Java 코드로 변경하는 도구는 대표적으로 AX2J와 XmlToJava가 있다. AX2J는 오픈소스 프로젝트로 공개되어 있으며 XmlToJava는 소스코드가 비공개이고 정식 홈페이지를 방문한 후 웹 버전으로만 사용할 수 있다. 이러한 변환 도구들이 개발된 이유는 정적으로 레이아웃을 구성하는 xml 파일은 프로그램 실행 전에 이미 화면 구성이 확정되기 때문에 프로그램 동작 중에 화면 레이아웃의 변화를 주기 어렵지만 화면 구성 레이아웃을 자바 코드로 변경하면 런타임에 동적으로 화면 디자인을 바꾸는 등 레이아웃의 변화를 줄 수 있기 때문이다.

본 논문에서는 xml 파일을 Java 코드로 바꾸는 기존 도구들과는 지향하는 목적이 다르다. 즉, 난독화되지 않은 xml 파일을 자바 코드로 변경한 후 자바 코드를 난독화하는 방법을 사용함으로써 결과적으로 xml 파일을 난독화하고 화면 구성 파일이 도용되지 않도록 하는 효과를 얻고자 한다.




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.
  android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
 <TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hello World!"/>
</LinearLayout>
```

Fig. 4. Activity screen and layout xml file

Fig. 4는 안드로이드의 대표적인 레이아웃인 선형 레이아웃(LinearLayout)에 'Hello World'라는 텍스트를 출력하는 단순한 xml 파일의 예시이다.

```
public LinearLayout initLayout(Context context) {
    LinearLayout root = new LinearLayout(context);
    ViewGroup.LayoutParams root_LayoutParams =
      new ViewGroup.LayoutParams (
      ViewGroup.LayoutParams.WRAP_CONTENT,
      ViewGroup.LayoutParams.WRAP_CONTENT);
    root_LayoutParams.width =
      ViewGroup.LayoutParams.MATCH_PARENT;
    root_LayoutParams.height =
      ViewGroup.LayoutParams.MATCH_PARENT;
    root.setOrientation(LinearLayout.VERTICAL);
    root.setLayoutParams(root_LayoutParams);

    TextView textView = new TextView(context);
    LinearLayout.LayoutParams textView_LayoutParams =
      new LinearLayout.LayoutParams (
      LinearLayout.LayoutParams.WRAP_CONTENT,
      LinearLayout.LayoutParams.WRAP_CONTENT);
    textView.setText("Hello World!");
    root.addView(textView);
    textView.setLayoutParams(textView_LayoutParams);
    return root;
}
```

Fig. 5. Translated Java code by AX2J

Fig. 5는 AX2J 프로그램[17]을 사용하여 xml 파일을 Java 코드로 변경한 결과이다. AX2J는 레이아웃 xml 파일을 입력으로 받아서 동일한 화면을 구성하는 initLayout 메소드를 생성해준다. 원본 Java 소스의 onCreate 메소드에서 initLayout 메소드를 호출해주면 xml 파일을 사용한 것과 동일한 실행 결과를 얻을 수 있다. 또한 오픈 소스 프로젝트로 공개되었기 때문에 상황에 맞게 수정하여 배포 가능하다는 장점이 있다.

Fig. 6은 XmlToJava 프로그램[18]을 사용한 결과이다. AX2J와는 달리 메소드가 아닌 코드를 직접 생성해주기 때문에 생성된 소스 코드를 onCreate 메소드에 추가한 후 앱을 실행하면 화면 디자인을 xml 과 동일하게 만들어준다. XmlToJava 프로그램의 단점은 오픈 소스로 공개되어있지 않고 공식 홈페이지 웹 버전만 제공되기

```
LinearLayout linearLayout_875 = new LinearLayout(this);
  linearLayout_875.setOrientation(VERTICAL);
  LayoutParams layout_169 = new LayoutParams();
  layout_169.width = LayoutParams.MATCH_PARENT;
  layout_169.height = LayoutParams.MATCH_PARENT;
  linearLayout_875.setLayoutParams(layout_169);

TextView textView_547 = new TextView(this);
  textView_547.setText("Hello World!");
  LayoutParams layout_234 = new LayoutParams();
  layout_234.width = LayoutParams.WRAP_CONTENT;
  layout_234.height = LayoutParams.WRAP_CONTENT;
  textView_547.setLayoutParams(layout_234);
  linearLayout_875.addView(textView_547);
```

Fig. 6. Translated Java code by XmlToJava

때문에 문제가 있을 경우에 직접 수정이 불가하고 변환 기법을 다른 프로젝트에 활용하기 어렵다.

4. 시스템 구현 및 평가

4.1 시스템 구현

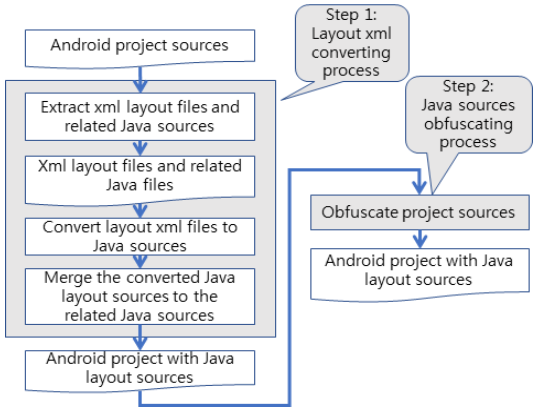


Fig. 7. Structure of layout obfuscation system

본 논문에서 제안하고 구현한 안드로이드 레이아웃 난독화 시스템의 구조도는 Fig. 7과 같다. 먼저 안드로이드 프로젝트 소스를 입력받아서 xml 레이아웃 파일과 Java 소스 파일을 추출한다. 그리고 xml 파일을 Java 소스로 변환한다. xml 파일을 Java 소스로 변환하는 엔진은 오픈 소스로 공개된 AX2J[17]를 사용하여 구현하였다. 변환된 소스 파일은 기존의 Java 소스와 통합되어 새로운 프로젝트 파일이 생성된다. 이후 프로가드 난독화를 통해서 전체 자바 소스 코드를 난독화하고, 프로젝트를 빌드하여 난독화된 apk 파일을 생성한다. 레이아웃 파일 난독화 시스템의 구현을 위해서 Java 언어를 사용하였고 Eclipse IDE 통합 개발 환경을 사용하여 Windows 10 시스템에서 개발하였다.

4.2 실험 및 평가

레이아웃 파일 난독화 시스템의 성능을 평가하기 위해서 대표적인 안드로이드 컨트롤인 TextView, Button, EditText를 사용하여 샘플 프로젝트를 만들었다. 일반적으로 xml 기반 화면 디자인 방식은 Java 기반 화면 디자인 방식보다 실행 성능이 느리다. 그 이유는 xml 파일을 파싱하여 구성 요소를 추출하고 화면 디자인을 생성해야 하기 때문이다. Java 기반 방식은 xml 파싱 작업이 필요 없으며 이미 컴파일된 결과를 실행하기 때문에 성능이 빠르다.

화면 디자인이 단순한 경우에는 xml 기반 방식과 Java 기반 방식의 실행 성능 차이를 체감할 수 없기 때문에 본 실험에서는 화면 디자인 레이아웃을 구성하기 위한 컨트롤의 개수에 따른 실행 성능을 비교하기 위해서 TextVew, Button, EditText를 각각 100개, 200개, 300개를 배치한 레이아웃 화면을 구성하여 실험하였다.

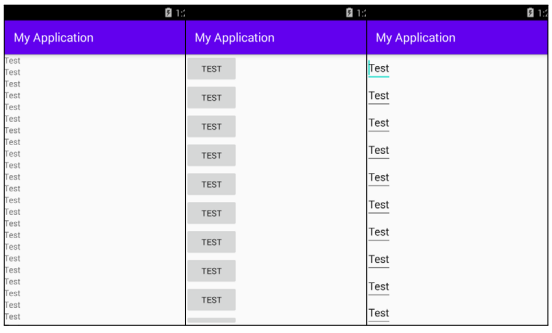
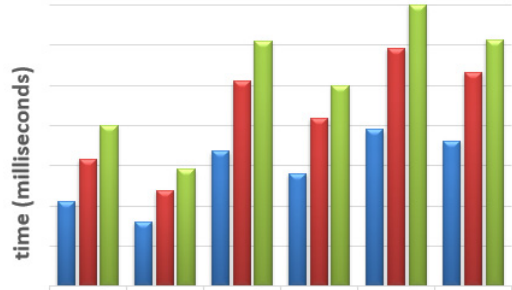


Fig. 8. Sample android applications for runtime performance evaluation

Fig. 8은 레이아웃 난독화 시스템의 성능 평가를 위한 TextView, Button, EditText 컨트롤을 사용한 안드로이드 샘플 애플리케이션의 실행 모습을 보여준다.

실행 성능 측정을 위해서 사용된 안드로이드 스마트폰은 LG X400이고 플랫폼 버전은 Oreo(8.1.0)이다.

Fig. 9는 xml 기반 화면 구성과 xml을 Java로 변환하여 순수하게 Java로 화면을 구성한 경우의 실행 성능 비교 결과를 보여준다. 순수하게 화면 구성을 위해서 소요된 시간 측정이 필요하다. 따라서 xml 방식에서는 안드로이드의 onCreate 메소드에서 setContentView (R.layout.activity_main); 명령어의 실행 전과 후에 시간을 측정하였으며 Java방식에서는 Java로 변환된 화면 디자인 코드의 실행 시작과 끝에서 각각 시간을 측정하였다. 그리고 실행 환경에 따른 오차를 고려하여 10회 반



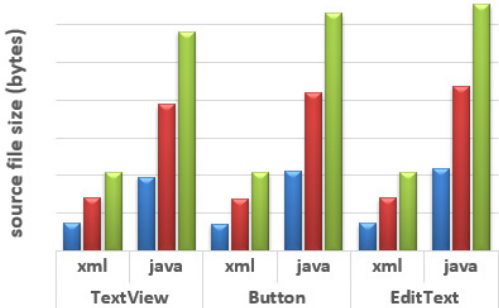
	xml	java	xml	java	xml	java
	TextView		Button		EditText	
100	1053.1	803.7	1680.2	1401.1	1954.2	1805.8
200	1585.6	1194.0	2553.4	2093.5	2968.0	2665.6
300	1998.6	1457.6	3054.8	2501.2	3509.5	3072.8

Fig. 9. Layout initialization performance comparison between xml and Java

복 실행하고 평균값을 계산하였다.

그 결과 TextView, Button, EditText 순서로 Java 방식의 성능이 xml 방식보다 우수했으며 크게는 37% (TextView 300개의 경우)부터 적게는 8% (EditText 100개의 경우) 실행 시간이 절약되었다.

본 논문에서 의도하는 바는 xml 파일이 노출되는 것을 방지하기 위한 방법으로 Java파일로의 변환을 시도했으나 부가적으로 실행 성능의 향상까지 얻을 수 있었다. 그러나 xml 파일을 Java 파일로 변환하는 경우에는 비록 xml 파일의 크기가 줄어들지만 Java 파일의 소스 코드 크기가 xml 파일의 감소보다 큰 폭으로 증가하는 단점이 존재한다.



	xml	java	xml	java	xml	java
	TextView		Button		EditText	
100	14,517	39,094	14,317	42,364	14,517	43,966
200	28,217	77,594	27,817	84,064	28,217	87,266
300	41,917	116,094	41,317	125,764	41,917	130,566

Fig. 10. The source file size comparison between xml and Java

Fig. 10은 xml 파일을 Java 파일로 변경하였을 때 소

스 파일 크기의 변화를 측정한 결과이다. TextView, Button, EditText 3가지 경우에 모두 Java 방식의 파일 크기가 작게는 32% (EditText 300개의 경우)부터 크게는 37% (TextView 100개의 경우)까지 증가한 결과를 보여주었다. xml 파일을 Java 파일로 변경함으로써 결과적으로 소스 파일 크기가 증가한 것은 단점이 될 수 있다. 그러나 앱이 배포될 때 소스 파일 형태로 배포되지 않고 컴파일된 apk 파일 형태로 배포되기 때문에 apk 파일의 크기 비교가 필요하다.

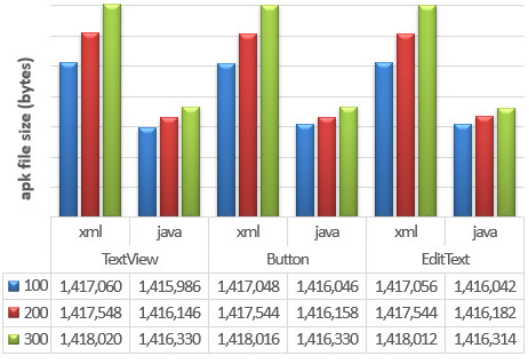


Fig. 11. The apk file size comparison between xml and Java

Fig. 11은 xml 파일을 Java 파일로 변경하고 최종 빌드 과정을 거쳐서 생성된 apk 파일 크기의 변화를 측정한 결과이다. TextView, Button, EditText 3가지 경우에 모두 Java 방식의 apk 파일 크기가 xml 방식보다 감소한 결과를 보여주었다. Fig. 10.에서 xml 방식보다 Java 방식의 소스 파일의 크기가 컸음에도 불구하고 컴파일된 apk 파일의 크기가 오히려 작은 것은 다음과 같은 2가지 이유로 설명할 수 있다.

첫째 xml 파일과는 달리 Java 파일의 경우에는 프로가드의 난독화 과정을 거친다. 즉, 난독화를 통해서 Java 소스 파일에 존재하던 변수나 메소드 이름과 같은 식별자의 길이가 짧게 줄어드는 효과가 있다. 둘째, Java 파일은 아스키 코드 형태이지만 컴파일 과정을 거치면서 바이너리 파일인 바이트 코드로 변경될 때 파일의 크기가 감소한다. 반면 xml 파일은 난독화 과정을 거치지 않으며 컴파일되지 않는다. 따라서 위 두 가지 원인으로 인해서 xml 파일을 Java 파일로 변경함으로써 소스 파일 크기가 증가하더라도 결과적으로 앱 사용자들에게 배포되는 apk 파일의 크기는 감소한다.

전체 실험 결과를 요약하면 다음과 같다. xml 파일을 Java 파일로 변환하여 화면 디자인을 구성할 경우 실행 성능이 더 좋아진다. 대신 Java 파일의 소스 파일의 크기는 xml 파일의 크기 감소분보다 더 증가하여 소스 파일 크기가 커지는 부담이 있다. 그러나 결과적으로 Java 파일은 xml 파일과 다르게 난독화 및 컴파일 과정을 거치기 때문에 앱 사용자들에게 배포되는 apk 파일의 크기는 감소한다.

그러나 xml 방식보다 Java 방식의 화면 디자인이 모든 면에서 우수한 것은 아니다. xml 방식으로 화면을 디자인하여 앱을 만들 때는 안드로이드 스튜디오에서 제공하는 화면 디자인 미리보기(preview) 기능을 사용할 수 있기 때문에 앱 실행 전에 화면 디자인을 미리 예상할 수 있고 유지 보수가 용이하다. 반면 Java 방식으로 화면 디자인을 할 경우에는 안드로이드 스튜이오에서 화면 디자인 미리보기가 불가능하다.

즉, xml 방식과 Java 방식 두 가지의 장점을 모두 활용하기 위해서는 개발 과정에서는 미리보기가 가능한 xml 방식으로 앱을 개발하고, 앱 배포 직전에 xml 파일을 Java 방식으로 변환하여 난독화 및 컴파일하여 배포하는 방법을 사용할 수 있다.

본 논문의 기여점은 다음과 같이 요약할 수 있다. 기존 난독화 도구가 지원하지 않는 xml 파일에 대한 난독화를 위해서 먼저 Java 파일로 변환을 하고 난독화하는 방법을 제안하고 도구를 구현하였다. Java 파일로 변환된 앱은 기존 xml 기반 앱보다 실행 성능이 더 향상되었으며 배포되는 apk 파일의 크기도 축소됨을 실험을 통해서 확인하였다.

5. 결론

지금까지 소프트웨어 위·변조 또는 도용을 방지하기 위한 방법으로써 다양한 프로그램 난독화 방법이 제안되었다. 그러나 난독화의 대상은 주로 프로그램 소스 코드 또는 바이너리 파일로 제한되었으며 그림 파일, 사운드 파일, 화면 레이아웃 파일과 같은 리소스 파일에 대한 중요성은 간과되었다. 그러나 은행 앱이나 금융 관련 앱에서 사용된 리소스 파일이 도용되어 가짜 앱이 시중에 유포된다면 그로 인해서 민감한 개인 정보가 유출되거나 금전적인 피해를 입을 수 있다.

본 논문에서는 안드로이드 앱의 리소스 도용 방지를 위해서 화면 디자인을 구성하는 xml 레이아웃 파일에 대한 난독화 기법을 제안하고 시스템을 구현하였다. 본 논문에서 제안한 레이아웃 파일 난독화 기법을 사용하면

앱의 화면 디자인이 기존 xml 파일에서 Java 파일로 이동하기 때문에 실행 성능이 향상되는 장점을 얻을 수 있으며 디자인을 도용하기 어렵기 때문에 위·변조된 앱으로부터 발생할 수 있는 피해를 사전에 예방할 수 있을 것으로 기대한다.

REFERENCES

[1] A. Almomani, B. B. Gupta, S. Atawneh, A. Meulenberg & E. Almomani. (2013). A Survey of Phishing Email Filtering Techniques. *IEEE Communications Surveys & Tutorials, 15(4),* 2070-2090.

[2] Security News. (2018). *Naver phishing site.* https://www.boannews.com/media/view.asp?idx=68740.

[3] S. Han, M. Ryu, J. Cha & B. U. Choi. (2014). HOTDOL: HTML Obfuscation with Text Distribution to Overlapping Layers. *IEEE International Conference on Computer and Information Technology,* 399-404.

[4] M. Maskur, Z. Sari & A. S. Miftakh. (2018). Implementation of Obfuscation Technique on PHP Source Code. *International Conference on Electrical Engineering, Computer Science and Informatics,* 738-742.

[5] Z. Y. Wang & W. M. Wu. (2014). Technique of Javascript Code Obfuscation Based on Control Flow Transformations. *Applied Mechanics and Materials,* 391-394.

[6] Financial consumer news. (2017). *Financial fraud surges through fake banking apps.* http://www.newsfc.co.kr/news/articleView.html?idxno=30477.

[7] Proguard. (2020). *Free Java class file shrinker, optimizer, obfuscator, and preverifier.* http://developer.android.com/tools/help/proguard.html.

[8] H. Park, H. Park, K. Ko, K. Choi & J. Youn. (2012). An Evaluation of the Proguard, Obfuscation Tool for Android. *Proc. of the 37th KIPS conference, 19(1),* 730-733.

[9] S. A. Sebastian et al. (2016). A study & review on code obfuscation. *Proc. of the World Conference on Futuristic Trends in Research and Innovation for Social Welfare,* 1-6.

[10] S. Dong et al. (2018). Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild. *Security and Privacy in Communication Networks,* 172-192.

[11] H. S. Park & T. Han. Choi. (2003). Advanced Operation Obfuscating Techniques using Bit-Operation. *Transactions on Programming Languages, 17(3),* 8-20.

[12] P. Yuxue, J. Jung & J. Lee. (2012). The Technological Trend of the Mobile Obfuscation. *Information & Communications Magazine, 29(8),* 65-71.

[13] Apktool (2020). *A tool for reverse engineering Android apk files*, https://ibotpeaches.github.io/ Apktool/.

[14] H. Park & H. Kim. (2014). Design and Implementation of An Obfuscation Tool for Preventing the Theft of Android Resources. *Proc. of the Korean Society of Computer Information Conference, 22(1),* 93-97.

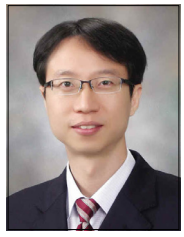
[15] H. Park. (2016). Design and Implementation of Server-based Resource Obfuscation Techniques for Preventing Copyrights Infringement to Android Contents. *Journal of the Korea Contents Association, 16(5),* 13-20.

[16] H. Park. (2019). Layout File Obfuscation Technique to Prevent Android App Theft, *Proc. of 13th KIISE and KBS Joint Symp,* 71-73.

[17] AX2J. (2020). *A tool converting your Android XML resource to native Java code.* http:// ax2j.sickworm.com

[18] XMLtoJava. (2020). S*imple android XML to Java code converter.* http://www.xmltojava.com/.

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 정보통신소프트웨어학과 부교수
- 관심분야 : 소프트웨어 난독화, 정적 및 동적 분석
- E-Mail : heewanpark@halla.ac.kr