

# 안드로이드 환경에서의 핀테크 앱 보안 점검을 위한 분석 방법

김상후, 허혜지, 류재철  
충남대학교

## 요약

오늘날 스마트폰 앱을 통해 다양한 핀테크 서비스를 이용할 수 있게 되었고, 안전한 금융거래를 위하여 앱 보안은 반드시 고려되어야 하는 상황이다.

따라서 본 고에서는 안드로이드 환경에서 핀테크 서비스를 제공하는 앱의 보안을 점검하기 위하여 안드로이드 앱의 구조를 소개하고, 디컴파일 도구를 이용한 정적 분석 방법과 디버깅, 네트워크 패킷 분석과 같은 동적분석 방법을 제공하고자 한다.

## I. 서론

핀테크는 금융(Financial)과 기술(Technology)의 합성어로 금융과 ICT(Information and Communication Technology)의 결합을 통해 새로이 나타나 성장하고 있는 산업 및 서비스 분야 등을 의미한다.

이러한 핀테크는 스마트폰과 결합되어 빠르고 간편한 형태의 금융서비스를 제공하게 되었다. 기존에는 금융서비스를 이용하기 위하여 은행에 방문하거나, ATM 기기를 사용하거나, PC를 통한 인터넷뱅킹을 해왔다. 하지만 현재는 휴대성이 높고 다양한 기능을 제공하는 스마트폰이 많이 보급되어, 금융 앱을 설치하는 것만으로도 기존의 다양한 금융서비스를 대체할 수 있게 된 것이다. 더 나아가 스마트폰을 이용한 금융서비스는 비단 은행거래에 국한된 것뿐만 아니라 의류, 도서, 식품 등의 다양한 쇼핑이나 항공, 숙박 등의 예약 등 여러 결제가 필요한 서비스까지 모두 제공할 수 있게 되었다.

하지만 스마트폰을 이용한 핀테크의 시장 규모가 세계적으로 급증함에 따라 사이버범죄 또한 점점 증가하고 있는 상황이다. 따라서 스마트폰에서의 범죄는 다양한 형태로 발생할 수 있기 때문에 보안에 신경을 많이 써야 한다.

안드로이드 앱의 보안을 위해서는 한국인터넷진흥원에서 제공하는 “Android-Java 시큐어코딩 가이드”와 같이 시큐어코

딩과 관련한 문서를 참조하여 개발 할 때부터 보안을 강화할 수 있다[2]. 하지만 악의적인 목적을 가진 공격자들의 경우 역공학 기술을 통하여 앱의 취약점을 확보할 수 있기 때문에, 앱을 개발한 후에도 다양한 보안솔루션을 적용하여 역공학을 어렵게 하기도 한다. 그러나 이러한 보안솔루션들도 공격자 관점에서 앱을 분석하는 방법을 알아야, 앱을 보호하기 위한 기술을 효율적으로 개발하고 적용할 수 있는 것이다.

안드로이드 앱을 분석하는 방법은 크게 정적분석과 동적분석으로 나뉜다. 정적분석은 앱의 실행파일이나 라이브러리 등을 보고 분석하는 방법으로, 디컴파일러 등의 도구를 이용하여 앱으로부터 원본 코드를 추출하는 작업이 선행되어야 한다. 그러나 코드가 암호화되어 있거나 값을 동적으로 받아오는 등 코드를 보는 것만으로는 분석이 어려울 때가 있다. 이 경우 앱 코드를 직접 실행하며 데이터의 변화를 확인하거나 실행흐름을 제어하는 동적분석이 필요하다. 특히 핀테크 앱의 경우 금융서비스에서 발생하는 트랜잭션을 전송하기 위해 네트워크 통신이 반드시 필요하므로 서버와 주고받는 패킷도 분석 대상에 속한다. 또한 안드로이드 앱은 자바코드로 작성되는 어플리케이션 계층 외에도 네이티브 코드로 작성되는 라이브러리가 포함될 수 있기 때문에 분석 대상의 구조 파악이 중요하다. 이때 각 계층별 코드를 분석하는 방법은 서로 다르기 때문에 먼저 앱 구조를 살펴보고, 각 코드에 맞는 분석방법 및 도구를 선택해야 한다.

본 고에서는 스마트폰 중 안드로이드 환경에서 동작하는 앱을 대상으로 보안 점검을 수행하기 위한 앱 분석 방법에 대해 정리한다. 2장에서 안드로이드 앱의 구조를 살펴보고, 3장에서 앱의 원본 소스코드를 복원하고 분석하는 정적분석 방법에 대해 살펴본다. 4장에서는 정적분석의 한계점을 극복하기 위해 앱을 직접 실행하며 분석하는 동적분석 방법과 네트워크 통신에 관한 분석에 대해 설명한다. 마지막으로 5장에서 결론을 맺는다.

## II. 안드로이드 앱 구조

안드로이드 앱은 설정파일, 소스코드 등의 패키지로 확장자는 apk 이다. 아래 <그림 1>은 패키지 파일 내 포함된 파일들이며, 패키지 파일의 확장자를 zip으로 변경한 뒤 압축해제를 하면 파일을 확인할 수 있다.





 Meta-INF	전자서명 관련 파일 저장
 res	그림, 문자열 등 리소스 파일 저장
 assets	Assets 파일 저장
 lib	Native library 저장
 AndroidManifest.xml	안드로이드 앱 설정 파일
 Classes.dex	메인 실행 코드 파일
 Resources.arsc	컴파일된 리소스 파일

그림 1. 안드로이드 앱 구조

안드로이드 앱의 여러 가지 파일 중 앱 분석을 위해 필요한 파일은 다음과 같다.

### 1. AndroidManifest.xml

안드로이드 앱의 퍼미션, API 레벨, 액티비티, 서비스 등 앱을 구성하는 컴포넌트 및 기능, 링크 할 라이브러리 등의 설정 정보를 저장하는 xml 파일로 인코딩 되어있다. 여러 가지 설정 정보 중 앱을 분석하기 위해 반드시 알아두어야 할 정보들은 메인 액티비티, 디버깅 옵션이 있다.

메인 액티비티는 앱을 실행했을 때 가장 먼저 호출되는 코드를 저장한 것으로 이를 이용하면 앱 실행 초기에 동적분석을 시작할 수 있다.

또한 안드로이드 앱을 동적분석 하기 위해선 Android Manifest.xml 파일 내 디버깅 옵션이 활성화 되어야 한다. 그러나 상용 안드로이드 앱의 디버깅 옵션은 기본적으로 비활성화 되어 있다. 따라서 안드로이드 앱을 동적분석하기 위해선 AndroidManifest.xml 파일을 디코딩 한 후 값을 수정하여 디버깅 옵션을 활성화 해 주어야 한다.

### 2. Classes.dex

안드로이드 앱은 자바언어로 개발되어, 안드로이드에서 DVM(Dalvik Virtual Machine)을 통해 동작한다. 일반적으로

자바언어는 JVM(Java Virtual Machine)에서 동작하기 위한 Bytecode 형태로 된다면, 안드로이드에서는 Dalvik Bytecode 로 한 번 더 변환된다. 이렇게 변환 과정을 거치면 확장자가 .dex(Dalvik EXecutable)인 파일이 된다. 대개의 안드로이드 응용 프로그램 패키지(Android Application Package, APK)는 압축 파일의 형태를 띠고, 그 안에 메인 실행코드인 dex 파일을 포함하고 있다. 이 파일은 디컴파일을 지원하는 다양한 도구를 통해 자바코드 등으로 복원하여 분석을 수행할 수 있다.

### 3. Native library

안드로이드의 구조를 살펴보면 어플리케이션 계층은 자바언어로 동작하며, 커널, 프레임워크, 라이브러리 등은 C, C++ 언어로 개발되어 동작한다. 따라서 안드로이드는 어플리케이션 계층과 네이티브 계층의 호환을 위하여 자바 네이티브 인터페이스(Java Native Interface, JNI)를 제공하고 있다. 즉, 안드로이드 앱도 개발자가 C, C++의 언어로 개발한 라이브러리를 포함시킬 수 있는데, 해당 라이브러리가 lib 폴더안에 .so 확장자로 저장되어 있다. 이 파일은 ARM 아키텍처에서 동작하는 바이너리로 IDA, GDB 등을 통해 분석을 수행할 수 있다.

## III. 정적분석

정적분석은 이 APK 파일 안에 있는 dex 파일로부터 원본에 가까운 자바코드로 복원하고, 분석하는 과정이 대부분이다. <표 1>은 앱의 정적분석을 위해 디컴파일을 도와주는 도구들을 정리한 것이다.

표 1. 안드로이드 앱 디컴파일 도구

도구명	디컴파일		변환 형태
	JAVA	Native	
dex2jar	O	X	dex → java
jd-gui	X	X	java view
apktool	O	X	dex → smali
JEB	O	X	Dex → Java, Smali
IDA	O	O	dex → smali so → Assembly

### 1. Dex2jar & jd-gui

dex2jar와 jd-gui 두 도구는 보통 한 쌍으로 같이 사용한다 [3][4]. 먼저 <그림 2>과 같이 dex2jar를 이용해서 apk파일이나 dex파일을 입력으로 넣으면, 디컴파일 결과로 jar 파일을 생

성한다. 입력 파일이 apk 파일인 경우 apk 파일 내의 classes, dex 파일을 자동으로 찾아내어 디컴파일을 수행한다.

```
D:\paper\dex2jar>d2j-dex2jar.bat -o test.jar test.apk
dex2jar test.apk -> test.jar
```

```
D:\paper\dex2jar>
```

그림 2. Dex2jar 실행방법

이때 출력되는 jar 파일은 <그림 3>과 같이 class 확장자를 가진 자바 코드를 압축한 파일 형태이다. 해당 파일은 하나씩 압축 해제할 필요 없이 jd-gui 도구를 이용해서 열면 <그림 4>와 같이 코드를 볼 수 있다.

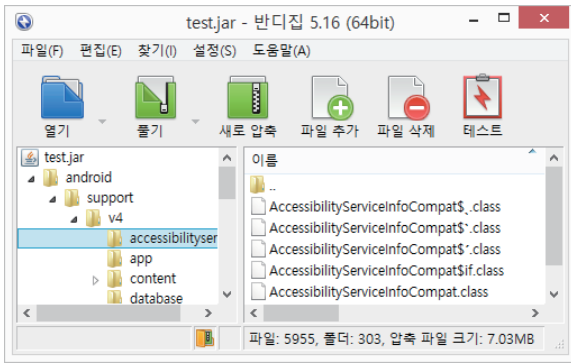


그림 3. Class 파일이 저장된 Jar 파일

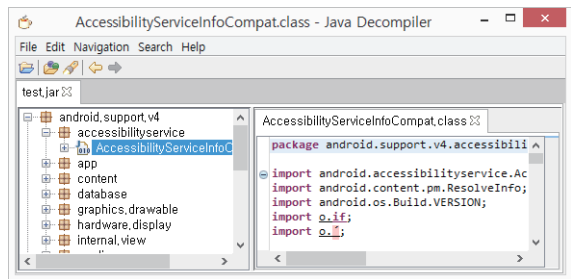


그림 4. jd-gui 실행 결과

이렇게 디컴파일 된 자바 코드는 변수 이름 정도를 제외하고, 원본 자바 코드와 거의 동일한 코드 구조 및 흐름을 갖는다. 따라서, 안드로이드에서 개발할 때 제공하는 ProGuard와 같은 난독화 기술을 적용하지 않는다면 무단 코드 도용과 같은 지적 재산권 침해 등을 초래할 수 있다.

## 2. Apktool

apktool 도구는 내부적으로 smali/baksmali 라는 오픈 소스 도구를 응용하는데, 결과적으로 추출해주는 파일은 dex2jar와

다르게 어셈블리어 수준의 smali 코드를 추출하게 된다[5]. 해당 도구는 리패키징 기능도 제공하여 디컴파일 된 코드를 수정한 후 다시 apk 파일로 만들 수 있게 해준다. 공격자의 경우 이 도구를 이용하여 앱을 리패키징 함으로써 위변조를 하기도 한다.

## 3. JEB

JEB는 유료 도구이긴 하지만 APK 파일 또는 dex 파일을 입력데이터로 넣으면 <그림 5>와 같이 dex2jar의 결과인 자바 코드와 apktool을 이용해 디컴파일한 결과인 smali 코드를 한 번에 볼 수 있다는 장점이 있다[6]. 또한 <그림 6>처럼 함수 혹은 변수의 호출 관계를 추적할 수 있도록 크로스 레퍼런스 기능을 제공하므로 더 빠르게 분석할 수 있다.

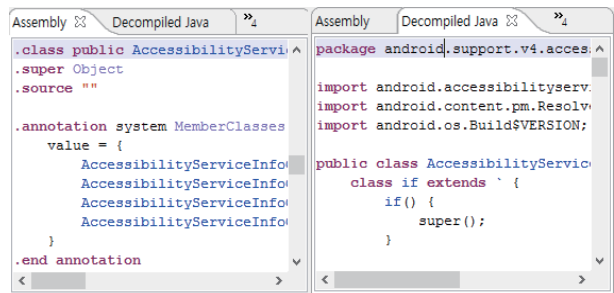


그림 5. JEB - 디컴파일 결과(Smali, Java)

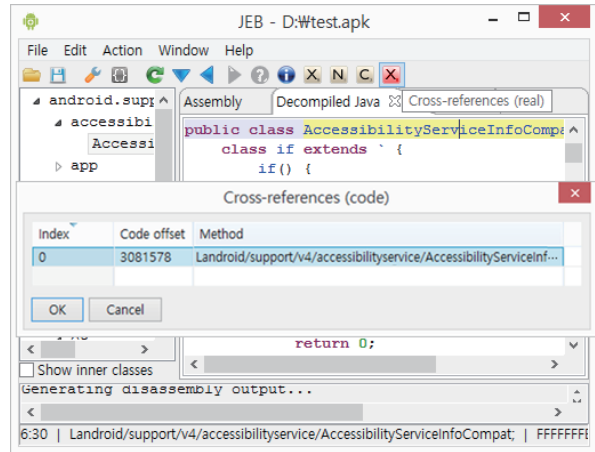


그림 6. JEB - 크로스 레퍼런스 기능

## 4. IDA

IDA는 다양한 플랫폼에서 동작하는 파일에 대해 디컴파일 및 디버깅을 지원하는 유료 도구이다[7]. IDA 실행 후 분석 대상 apk 파일을 열면 <그림 7>과 같이 apk 내 파일 목록이 출력된다. 이때 classes.dex 파일 또는 \*.so 파일을 선택하면





```
D:\paper>adb pull /data/app/kr.or.kftc.mobilewallet-1.apk .
[100%] /data/app/kr.or.kftc.mobilewallet-1.apk
```

```
D:\paper>
```

그림 11. ADB 명령어를 통한 앱 추출

## 2. 앱 디컴파일

앞서 소개한 안드로이드 앱 디컴파일 도구 중 apktool을 이용하여 <그림 12>와 같은 명령을 입력하면 dex 파일이 smali 코드 형태로 변환된 결과를 얻을 수 있다.

```
D:\paper>java -jar apktool_2.2.0.jar d kr.or.kftc.mobilewallet.apk -o .\target
I: Using Apktool 2.2.0 on kr.or.kftc.mobilewallet.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
```

그림 12. APKTOOL - 디컴파일

## 3. 디버깅 옵션 활성화

기본적으로 상용 앱은 디버깅 옵션이 비활성화된 상태로 배포된다. 따라서 동적분석을 위한 디버깅 활성화를 위해 아래의 2가지 작업을 수행해야 한다.

### 가. AndroidManifest.xml 파일 수정

Apktool을 이용하여 디컴파일 한 결과 폴더에는 디코딩된 AndroidManifest.xml 파일이 존재한다. 해당 파일 안에 debuggable 옵션이 없거나, false 상태로 설정되어 있다면 디버깅이 불가능하다. 따라서 해당 옵션이 없다면 만들어주되, 옵션을 true로 설정 해 준다.

### 나. 디버깅 코드 추가

디버깅이 활성화 되었다면 앱을 실행한 후 언제든지 디버거를 붙일 수 있다. 그런데 만약 특정 코드가 실행되기 전 디버거를 붙이고 싶다면 <그림 13>과 같은 코드를 이용할 수 있다. 이 코드는 디버거가 붙을 때까지 실행을 멈추고 대기하라는 코드로, 메인 액티비티의 초기에 추가 할 경우 앱을 실행하자마자 디버깅이 가능하다. 일반적으로 onCreate()가 앱 실행 시 가장 먼저 호출된다.

```
.method public onCreate(Landroid/os/Bundle;)V
    .locals 3
    .param p1, "arg0"    # Landroid/os/Bundle;

    .prologue
    .line 84
    invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
```

그림 13. 디버깅 코드 추가

메인 액티비티는 AndroidManifest.xml 파일에서 확인 가능하며, 액티비티 중 “android.intent.action.MAIN”속성을 가진 액티비티가 메인 액티비티에 해당한다. <그림 14>의 경우 “activity.main.IntroActivity” 이다.

```
<activity android:name="activity.main.IntroActivity" ...>
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        ...
    </intent-filter>
</activity>
```

그림 14. AndroidManifest.xml 파일에 선언된 메인 액티비티

## 4. 리패키징

디버깅 옵션 활성화 및 smali 코드 수정을 완료하였으면 다시 APK 파일 형태로 만들어주어야 한다. Apktool은 리패키징 옵션도 제공해주기 때문에 <그림 15>와 같은 명령을 통해 손쉽게 APK 파일로 재생성 할 수 있다.

```
D:\paper>java -jar apktool_2.2.0.jar b target -o target_resigned.apk
I: Using Apktool 2.2.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
```

그림 15. APKTOOL - 리패키징

```
D:\paper>keytool -lnk -genkey -v -keystore D:\paper\key.jks -alias testkey -keyalg RSA -keysize 2048 -validity 10000
키 저장소 비밀번호 입력:
새 비밀번호 다시 입력:
이름과 성을 입력하십시오.
[Unknown]:
조직 단위 이름을 입력하십시오.
[Unknown]:
조직 이름을 입력하십시오.
[Unknown]:
구/군/시 이름을 입력하십시오?
[Unknown]:
시/도 이름을 입력하십시오.
[Unknown]:
이 조직의 두 자리 국가 코드를 입력하십시오.
[Unknown]:
CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown이<가> 맞습니까?
[아니오]: y
```

```
다음에 대해 유효 기간이 10,000일인 2,048비트 RSA 키 쌍 및 자체 서명된 인증서(SHA256withRSA)를 생성하는 중
: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
<testkey>에 대한 키 비밀번호를 입력하십시오.
<키 저장소 비밀번호와 동일한 경우 Enter 키를 누름:>
```

```
ID:\paper\key.jks을<를> 저장하는 중J
```

```
D:\paper>
```

그림 16. 앱 서명용 키 생성

단, 이렇게 생성된 APK 파일은 앱 서명이 되지 않은 상태이기 때문에 안드로이드에 설치를 할 수 없다. 따라서, 앱 서명을 위해 keytool과 jarsigner 2개의 명령을 이용하여 앱 서명 작

업을 추가적으로 수행해야 한다. Keytool과 jarsigner는 둘 다 JDK(Java Develop Kit)에 포함된 도구로 앱 서명에 필요한 키 생성 및 사인을 기능을 제공한다. 먼저 <그림 16>처럼 명령을 입력하여 키를 생성하는데, 이 때 암호를 설정하게 된다. 이 암호는 앱 서명을 위해 생성한 키를 이용할 때 다시 한번 사용해야 하는데, <그림 17>과 같이 중간에 “asdf1234”로 입력한 부분에 실제 설정한 암호를 입력하면 된다.

```
D:\paper>jarsigner.lnk -sigalg SHA1withRSA -digestalg SHA1 -storepass asdf1234 -keypass asdf1234 -tsa http://timestamp.digicert.com -keystore D:\paper\key.jks D:\paper\target_resigned.apk testkey jar signed.
```

D:\paper>

그림 17. 앱 서명

## 5. 앱 설치

서명을 마친 앱을 스마트폰에 설치하기 위해 스마트폰 저장소에 apk 파일을 복사한 후 설치하거나, <그림 18>과 같이 ADB 명령어를 이용하여 설치할 수 있다.

```
D:\paper>adb install target_resigned.apk
[100%] /data/local/tmp/target_resigned.apk
pkg: /data/local/tmp/target_resigned.apk
Success
D:\paper>
```

그림 18. ADB 명령어를 통한 앱 설치

## 6. 디버깅

과거에는 NetBeans 프로그램이 많이 사용되었으나 현재에는 Android Studio라는 공식 안드로이드 IDE를 이용하여 좀 더 간편하게 디버깅을 수행할 수 있다. 이어서 apktool을 이용하여 smali 코드로 변환작업을 할 때, “-d” 옵션을 추가하여

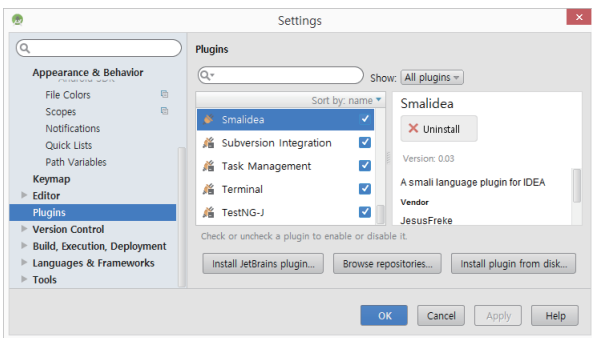


그림 19. Android Studio – 플러그인 등록

smali 코드 내 디버깅 심볼이 자동으로 삽입될 수 있도록 작업한다. 그러나 apktool 2.0.3 버전부터는 apktool에서 디버깅 옵션이 제외되었기 때문에, 대신하여 “smalidea”라는 Android Studio 용 플러그인을 <그림 19>와 같이 추가하고 디버깅을 수행할 수 있도록 한다[10].

안드로이드 앱 디버깅을 위해선 프로젝트를 생성해야 하며 이를 위해 앞서 apktool을 이용하여 디컴파일한 결과로 생성된 smali가 필요하다.

PC환경에서는 디컴파일된 smali 폴더를 Android Studio에 프로젝트로 가져와서 디버깅을 수행할 준비를 하고, 스마트폰 환경에서는 디버깅 옵션을 활성화한 리패키징 된 앱을 설치 및 실행한다.

앱을 실행시키게 되면, <그림 20>과 같이 Android Studio에서 PC와 연결된 스마트폰과 그 스마트폰 내에서 디버깅이 가능한 앱 목록이 뜨게 된다. 이 앱을 선택하게 되면 Android Studio에서 열고 있는 smali 코드를 통해 디버깅을 수행할 수 있다.

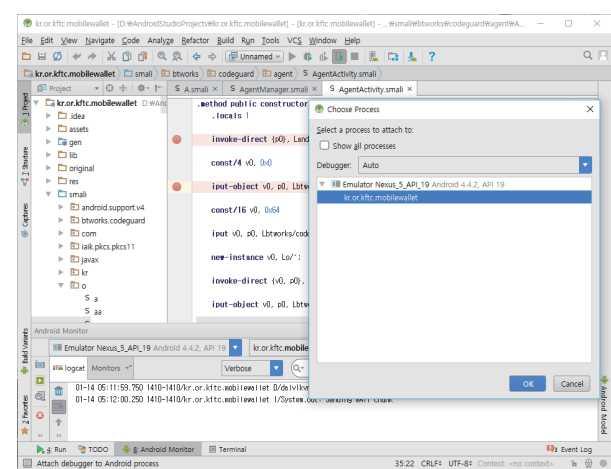


그림 20. Android Studio를 이용한 디버깅

## 7. 네이티브 라이브러리 디버깅

네이티브 라이브러리를 분석하기 위해서 스마트폰안에 디버거 혹은 디버깅 서버를 위치시키고 루트 권한으로 실행시킨다.

안드로이드 플랫폼은 리눅스를 기반으로 하며, 따라서 네이티브 라이브러리 디버깅도 리눅스 파일과 동일하게 gdb를 이용한다[11]. 그러나 안드로이드 스마트폰은 gdb가 기본으로 설치되어 있지 않으므로, 안드로이드용 gdb를 따로 크로스 컴파일해야 한다.

또 다른 디버깅 방법으로 IDA를 이용하는 것이 있다. IDA를 통해 디버깅을 하기 위해서는 스마트폰 내 디버깅 서버가 실

행되고 있어야 한다. 이를 위해 IDA에서 제공하는 “android\_server” 파일을 스마트폰에 복사하고 <그림 21>과 같이 실행한다.

```
shell@lentis:~/data/local/tmp $ ./android_server
IDA Android 32-bit remote debug server(ST) v1.17. Hex-Rays (c) 2004-2014
Listening on port #23946...
```

그림 21. android\_server 실행

그 후 PC에서 <그림 22>와 같이 adb 명령어를 입력하여 스마트폰 내부의 디버깅 서버와 PC의 IDA가 통신을 할 수 있도록 포트포워딩을 해준다.

```
D:\wpaper>adb forward tcp:23946 tcp:23946
D:\wpaper>
```

그림 22. 포트포워딩(23946 포트)

마지막으로 <그림 23>과 같이 IDA를 실행하여 remote debugging으로 안드로이드를 선택한 후 분석 대상 프로세스를 선택하여 디버깅을 수행한다.

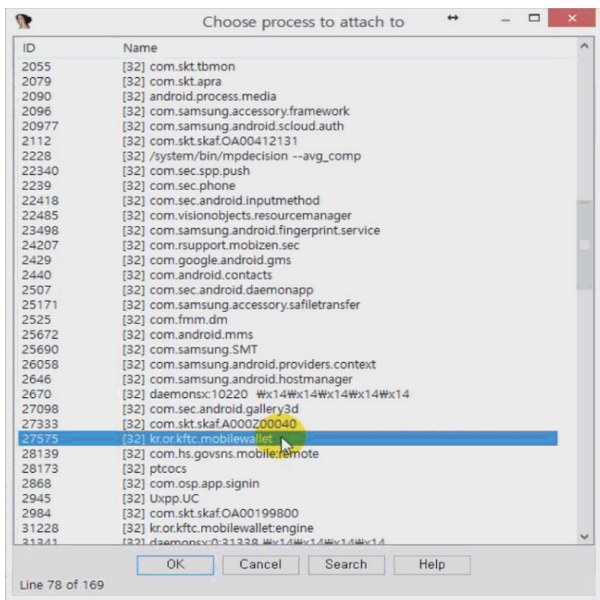


그림 23. IDA - 디버깅 대상 프로세스 선택

동적분석에서의 핵심은 앱을 디버깅이 가능하게 리패키징을 하고, 디버거를 통하여 분석을 수행하는 것이다. 이러한 분석 작업을 통해서 앱이 사용자의 어떤 데이터를 이용하고, 어디에 저장하고, 어디에 사용하는지 파악할 수 있게 된다. 따라서, 이

러한 정보의 유출과 같은 문제를 방지하기 위하여 앱의 무결성을 검증하고, 위변조 되지 않은 경우에만 정상 동작하도록 해야 한다.

## 8. 네트워크 패킷 분석

스마트폰 내부 데이터베이스에 데이터를 저장하는 일반 앱과는 달리 금융거래를 하는 은행 또는 핀테크 앱 들은 데이터를 필요로 할 때마다 서버로부터 가져오게 된다. 따라서 핀테크 앱들이 안전한 통신을 하는지 점검하기 위해선 서버와 주고받는 데이터의 암호화 여부 등을 점검할 필요가 있다.

서버와 주고 받는 패킷은 wireshark와 같은 도구를 이용하여 캡쳐해 볼 수 있다[12]. 스마트폰의 경우 무선 네트워크를 이용하므로 네트워크 패킷 캡처를 위한 환경 구축이 필요하다.

스마트폰 패킷을 캡처하기 위한 방법은 여러 가지가 있지만 그중 간편한 방법은 <그림 24>와 같이 Soft AP를 이용하는 방법이다. Soft AP란 소프트웨어를 통해 무선 AP(Access Point)를 구현하는 것으로 컴퓨터에 무선랜 카드를 장착하여 구성한다.



그림 24. 무선 네트워크 환경 구축

관리자 권한 명령 프롬프트에서 <그림 25>처럼 명령어를 입력하면 AP를 생성할 수 있고, 스마트폰은 컴퓨터가 생성한 AP에 접속하여 서버와 통신을 하게 되므로 중간에 위치한 컴퓨터를 통하여 인터넷을 통해 주고받는 패킷을 모두 캡쳐하여 확인할 수 있게 된다.

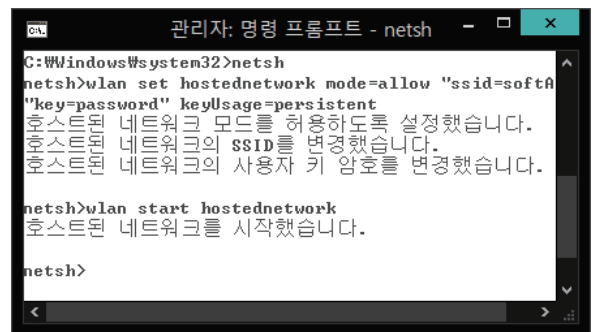


그림 25. softAP를 생성하는 명령어

즉, 이러한 앱의 네트워크 패킷 분석을 통하여 암호화되지 않은 데이터는 고스란히 노출될 수 있으므로, 정상적으로 암호화 송수신을 하는지 점검해야 한다.

## V. 결론

다양한 핀테크 서비스를 제공하는 스마트폰 중에서도 과반수 이상의 점유율을 차지하는 안드로이드 환경에서의 보안은 매우 중요하다. 특히 서비스를 제공하기 위해 배포되는 앱의 보안 위협은 사용자의 재산상 피해를 야기할 수 있기 때문에 더욱 고려해야 하는 부분이다.

따라서 본 고에서는 안드로이드 환경 속에서 핀테크 서비스를 제공하는 앱들의 보안 점검을 위해 공격자 관점에서의 분석 기술 및 방법을 정리해 보았다. 이를 통하여 개발자 외의 악의적인 목적을 가진 공격자가 앱의 정보를 어디서 획득하고, 앱을 어떻게 위변조하고, 어떤 부분에서 공격을 시도해 볼 수 있을지가 정해 봄으로써 사전에 차단할 방안을 마련할 수 있다.

## 참고 문헌

- [1] Android-Java 시큐어코딩 가이드, 한국인터넷진흥원, 2016.03.18
- [2] 국내외 핀테크 관련 기술 및 정책 동향 분석을 통한 연구분야 발굴, 한국인터넷 진흥원, 2016.04.25
- [3] dex2jar 2.0, <https://sourceforge.net/projects/dex2jar/>, 2016.10.11
- [4] jd-gui 1.0.0, <http://jd.benow.ca/>, 2015.03.25
- [5] apktool 2.2.1, <https://ibotpeaches.github.io/Apktool/>, 2016.10.18
- [6] JEB 2.2.10, <https://www.pnfsoftware.com/>, 2016.10.10
- [7] IDA, <https://www.hex-rays.com/products/ida/>
- [8] Android Studio <https://developer.android.com/studio/index.html>
- [9] Android Debug Bridge, <https://developer.android.com/studio/command-line/adb.html>
- [10] smalidea 0.03, <https://github.com/JesusFreke/smali/wiki/smalidea>, 2016.02.27
- [11] Wireshark 2.2.3, <https://www.wireshark.org>
- [12] gdb, <https://www.sourceware.org/gdb/>
- [13] 조준성, 최현재, 김소영, 김형식, "모바일 핀테크 애플리케이션의 보안 위협 및 대응책 연구", 정보과학회지, 34(4), pp34-41, 2016.04
- [14] 하동수, 이강효, 오희국, "안드로이드 어플리케이션 역공학 보호기법", 정보보호학회지, 25(3), pp19-28, 2015.06
- [15] 장준혁, 한승환, 조유근, 최우진, 홍지만, "안드로이드 환경의 보안 위협과 보호 기법 연구 동향", 보안공학연구논문지, 11(1), 2014.02
- [16] 최지선, 김태희, 민상식, 성재모, "스마트폰 뱅킹 앱 무결성 검증을 위한 보호기술 동향", 정보보호학회지, 23(1), pp54-60, 2013.02
- [17] 박정국, "핀테크(Fintech)와 정보보안", 정보과학회지, 33(5), pp23-32, 2015.05
- [18] 김순일, 김성훈, 이동훈, "안드로이드 스마트폰 뱅킹 앱 무결성 검증 기능의 취약점 연구", 정보보호학회논문지, 23(4), pp743-755, 2013.08
- [19] "안드로이드 기반의 악성코드 역공학", SANS Institute, 2012.07
- [20] 이찬희, 정윤식, 조성제, "안드로이드 애플리케이션에 대한 역공학 방지 기법", 보안공학연구논문지, 10(1), pp41-50, 2013.02
- [21] 신동휘, "FinTech 관련 국내외 보안사고 사례 현황", 정보과학회지, 34(4), pp25-28, 2016.04



## 약 력



김 상 후

2014년 충남대학교 컴퓨터공학과 졸업  
2016년 충남대학교 컴퓨터공학과 석사  
2016년~현재 충남대학교 컴퓨터공학과 박사과정  
관심분야: 시스템 보안, 모바일 보안, 취약점 분석



허 헤 지

2015년 충남대학교 컴퓨터공학과 졸업  
2017년 충남대학교 컴퓨터공학과 석사  
관심분야: 모바일 보안, 네트워크 보안, 디지털 포렌식



류 재 철

1985년 한양대학교 산업공학과 졸업  
1988년 Iowa State University 전산학 석사  
1990년 Northwestern University 전산학 박사  
1991년~현재 충남대학교 컴퓨터공학과 교수  
관심분야: 인터넷 보안, 금융 보안