Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.1, Feb. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.1.105

Xposed를 이용한 안드로이드 악성코드 분석을 위한 API 추출 기법 설계 및 구현에 관한 연구*

강 성 은,† 윤 홍 선, 정 수 환‡
숭실대학교

Design and Implementation of API Extraction Method for Android Malicious Code Analysis Using Xposed*

Seongeun Kang,† Hongsun Yoon, Souhwan Jung‡
Soongsil University

요 약

최근 지능화된 안드로이드 악성코드는 정적 분석만으로는 악성행위에 대한 탐지가 어려워지고 있다. SO파일, 동적로딩을 이용한 코드 호출 및 문자열 난독화를 적용한 악성코드의 경우 분석을 위해 다양한 툴을 이용하여도 원본 코드에 대한 정보 추출이 어렵다. 이 문제를 해결하기 위해서 다양한 동적 분석기법이 있지만, 동적 분석은 루팅 환경이나 에뮬레이터 환경을 요구한다. 그러나 동적 분석의 경우 악성코드들이 루팅 및 에뮬레이터 탐지를 실시하여 분석 환경을 탐지 하고 있다. 본 논문은 이를 해결하고자 다양한 루팅 탐지 기법을 조사하여 실단말에서 루팅탐지 우회 환경을 구축하였다. 또한, Xposed를 이용하여 안드로이드 악성코드 분석을 위한 SDK 코드 후킹 모듈을 설계하였고, 코드 흐름을 위한 인텐트 추적, 동적 로딩 파일에 대한 정보, 다양한 API 정보 추출을 구현하였다. 이를 통해 악성코드의 난독화 된 정보 및 다양한 악성 행위 정보를 분석하고자 한다.

ABSTRACT

Recently, intelligent Android malicious codes have become difficult to detect malicious behavior by static analysis alone. Malicious code with SO file, dynamic loading, and string obfuscation are difficult to extract information about original code even with various tools for static analysis. There are many dynamic analysis methods to solve this problem, but dynamic analysis requires rooting or emulator environment. However, in the case of dynamic analysis, malicious code performs the rooting and the emulator detection to bypass the analysis environment. To solve this problem, this paper investigates a variety of root detection schemes and builds an environment for bypassing the rooting detection in real devices. In addition, SDK code hooking module for Android malicious code analysis is designed using Xposed, and intent tracking for code flow, dynamic loading file information, and various API information extraction are implemented. This work will contribute to the analysis of obfuscated information and behavior of Android Malware.

Keywords: Android, Malware, Analysis

Received(11. 08. 2018), Modified(01. 25. 2019), Accepted(01. 25. 2019)

* 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2016-0-00078, 맞춤형 보안서비스 제공을 위한 클라우드 기반 지능형 보안 기술 개발)

† 주저자, y7k90@naver.com
‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. 서 론

2014년 이후 세계 인구 당 스마트폰의 보급률은 73%로 PC의 보급률 24.5%를 넘어 점차적으로 증가하고 있다. 전 세계적으로 스마트 폰의 사용이 증가함에 따라 단순히 전화 통화뿐만 아니라 개인정보를 저장하여 은행 및 결제 서비스, 어플리케이션을 이용한 업무 서비스를 하는 활동이 증가하고 있다[1]. IDC 보고서에 따르면 2016년 3분기에 안드로이드의 점유율이 86%로 가장 높은 것으로 나타났다[2]. McAfee의 2017년 보고서에 따르면 이러한 안드로이드 시장을 노리는 악성코드가 2015년 4분기 이후부터 점차적으로 증가하여 2억개 이상의 안드로이드 악성코드가 발견되었고 이중 새로운 형태의 악성코드는 2017년 2분기에 대비 3분기에 60% 급등하였다[3]. 이러한 안드로이드 악성코드는 점점 지능화되어 단순한 분석을 통해서는 새로운 악성코드를 탐지하기 어렵다. 특히 지능화된 악성코드는 분석을 난해하게 만들기 위해서 많은 Anti-Analysis 기능을 접목하고 있다. 지능화된 악성코드는 정적 분석을 방지하기 위해서 문자열 및 에러 난독화와 동적 로딩 파일 및 SO파일을 적용하여 분석가들에게 어려움을 주고 있다. 또한, 정적 분석의 경우는 많은 시간과 인력이 필요하게 되어 새로운 악성코드가 생길 때마다 빠른 시간 내에 분석이 어려운 실정이다. 이러한 대응책으로 동적 분석을 적용하여 실행 중에 악성코드를 분석하여 API 호출 및 변수 추출이 필요하다. 하지만 지능화된 악성코드는 동적 분석을 탐지하기 위해서 루팅 및 에뮬레이터 검사를 하여 루팅이 되지 않은 실단말이 아닌 경우 악성행위를 수행하지 않고 있다. 이는 루팅 탐지 우회 환경이 구축된 실단말에서 동적 분석을 수행하여 악성 행위를 분석할 필요가 있다.

본 연구에서는 이러한 지능화된 악성코드들의 루팅 및 에뮬레이터 탐지를 대응하는 Xposed을 이용한 루팅 탐지 우회 분석환경 구축을 통해 빠른 시간 내에 동적 행위를 추출하고 정적 분석으로는 파악하기 어려운 인텐트의 호출 및 내포된 정보를 추출하고, 동적 로딩 파일과 자바 리플렉션 호출들을 추출한다.

II. 관련연구

2.1 Xposed

Xposed는 실행 중에 동적으로 코드의 수정을 가능하게 하는 안드로이드 어플리케이션 후킹 도구이다. 예를 들어 File.exists 메소드를 후킹하여 기존에 존재하는 파일의 존재 유무에 상관없이 false로 리턴 값을 변경 할 수 있고, TelephonyManager를 후킹하여 전화번호를 수정할 수 있다. 이 외에도 안드로이드에서 이용되는 메소드를 후킹하여 파라메터 값 변조, 예외처리 등 다양한 방법을 통해 동적으로 값을 수정한다[4].

Fig.1.은 안드로이드 시스템의 부팅 및 xposed의 부팅 과정에 대한 내용이다. 일반적인 부팅(a)은 초기 부팅 시에 init_process에서 zygote_process를 생성한다. Zygote는 안드로이드 시스템의 핵심 요소이며 core 라이브러리인 core libs를 담고 있다. Zygote는 app_process를 통해서 필요한 클래스를 로드하고 모든 어플리케이션들은 zygote에 의해서 분기 되어 생성되어지며 이 때 zygote가 가지고 있는 core libs를 어플리케이션들은 가지게 된다[5].

반면 Xposed 부팅(b)은 app_process를 확장하여 XposedBridge라는 jar 파일을 클래스 경로의 추가하여 실행 중에 특정 지점에서 메소드를 호출하여 응용 프로그램 동작의 수정을 가능하게 한다. 따

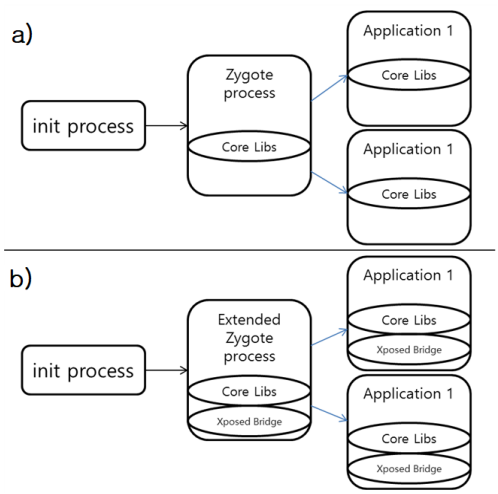

Fig. 1. Normal Runtime Architecture(a)
Xposed Runtime Architecture(b)

라서 확장된 zygote가 클래스 경로에 따른 Xposed Bridge를 로드하고, 이후 어플리케이션을 분기하게 된다.

2.2 루팅

루팅이란, 안드로이드 운영체제의 취약점을 이용하여 루트 권한을 획득하는 행위를 말한다. 안드로이드는 리눅스 기반 운영체제이기 때문에 일반 사용자가 루트 권한을 획득할 수 없다. 하지만 루팅을 통해 관리자 권한을 획득 할 수 있으며 단말을 임의적으로 조작하기 위해서 많이 사용되고 있다. 루팅 방법은 부팅 전 후로 Pre-boot 방법과 Post-boot 방법으로 나뉜다[6].

2.2.1 Pre-boot

Fastboot : Pre-boot의 첫 번째 방법은 Fastboot 모드(삼성 폰에서는 오딘모드)를 이용하는 방법이다. 안드로이드 장치에 부트로더는 루트의 권한을 가지고 있다. Fastboot모드는 완전한 부팅을 이루지 않고 부트로더까지 부팅을 하여 부트로더에서 얻은 루트권한을 이용하는 방법이다. 즉, 루트 권한을 가진 부트로더가 system.img와 boot.img를 접근하여 부트 파티션과 시스템 파티션을 구성한다. 만약 system.img 또는 boot.img가 SU가 설치 된 수정된 이미지라면 쉘 명령중 하나인 SU가 실행 가능해진다.

Custom Recovery : 두 번째 방법은 리커버리 모드를 이용하는 방법이다. 리커버리 모드란, 백업 OS 모드이다. 백업 OS 모드는 장치에 있는 OTA 패키지를 통해서 부팅을 한다. Fastboot를 이용하여 TWRP와 같은 수정된 리커버리 모드를 적용하면 쉽게 SU를 설치 할 수 있다.

2.2.2 Post-boot

Rooting apps/tools : Post-boot의 첫 번째 방법은 리눅스의 취약점을 이용하는 방법이다. 안드로이드는 리눅스 기반으로 만들어져 있기 때문에 기존 리눅스에 있는 취약점이 안드로이드에도 전해졌다고 볼 수 있다. 그렇기 때문에 리눅스의 취약점을 이용하는 King root 앱 같은 원터치 루팅 앱으로 권한을 획득할 수 있다. 하지만 이러한 원터치 루팅 앱들은 안드로이드버전에 따라 권한 획득 여부가 달라진다는 단점이 있다.

Privileged ADB : Post-boot의 두 번째 방법은 ADB의 권한을 이용한 방법이다. 안드로이드 장치는 PC와 연결할 경우 ADB를 수행할 수 있다. ADB는 자신의 고유 쉘이 존재한다. ADB 쉘이 루트 권한을 가진 쉘로 권한을 상승 시키면 ADB 쉘을 이용하여 SU를 설치 할 수 있다.

2.2.3 루팅 탐지

루팅은 분석가들에게 분석을 위한 환경을 제공하지만, 반대로 악성어플리케이션에서 실시하는 분석환경 탐지에 의해 악성행위가 수행되지 않을 수 있다. 이러한 이유로 루팅 탐지에 대한 우회 기술을 접목하여 분석 환경에서 악성 행위를 수행 시켜야 한다. 루팅을 탐지 하는 방식은 다음과 같다

BUILD Tag 체크 : 기본적인 수정되지 않은 안드로이드 이미지들은 Build.TAGS의 값이 release-key 이다. 하지만 수정된 이미지들을 사용할 경우 test-key 와 같은 Build.TAGS값이 설정되어 있다.

File 체크 : File.exists 메소드는 File에 설정된 경로의 해당되는 파일이 존재하는지 여부를 판단하는 메소드이다. 이 메소드를 이용하여 SU, Supersu, 루팅앱들을 체크한다.

Shell commend 체크 : 자바에서 RunTime.exec 메소드나 Process Builder 메소드를 이용하면 쉘 커맨드를 실행 할 수 있다. 쉘 커맨드에서 pm, ls, su 등 프로세스나 파일을 체크하여 루팅을 확인한다.

설치된 어플리케이션 체크 : ActivityManager와 ApplicationPackageManager는 어플리케이션에 대한 다양한 정보를 가지고 있다. 예를 들어 ApplicationPackageManager. getInstalledPackages 메소드는 현재 설치된 패키지에 대한 정보를 제공한다. 이 외에도 현재 실행중인 어플리케이션 정보, 현재 구동중인 서비스 정보, 패키지의 존재 유

무를 판단하는 메소드 등이 있다. 어플리케이션, 패키지 네임을 바탕으로 ActivityManager, ApplicationPackage Manager 메소드를 사용하여 루팅을 체크 할 수 있다.

2.3 안드로이드 인텐트

안드로이드에서 인텐트는 컴포넌트 간에 실행 및 정보 전달을 목적으로 사용된다. 이러한 인텐트의 유형에는 명시적 인텐트와 암시적 인텐트가 있다. 악성 어플리케이션은 이러한 인텐트를 활용하여 다른 컴포넌트에서 수행된 값을 얻어 오거나 악성 서비스를 실행하는 목적으로 사용하여 정적 분석을 어렵게 하고 있다[7].

2.4 자바 리플렉션

자바 리플렉션은 자바 언어의 한 기능으로서 자바 가상 머신에서 실행 중인 어플리케이션의 런타임 동작을 검사하거나 변경을 위해서 사용된다. 자바 리플렉션은 런타임 동작 중에 원래 코드의 동작 방향과 다른 방향으로 변경 시켜 정상적인 사용을 불가능하게 만들 수 있는 기능이다. 자바 리플렉션을 활용하여 실행 시간 동안에 동적으로 클래스 정보를 얻거나 클래스를 로딩 할 수 있다[8]. 현재 악성코드들은 이러한 리플렉션을 활용하여 난독화를 진행하고 있다[9].

Fig.2.는 FakeInstaller 악성코드에서 리플렉션을 이용하여 난독화 된 코드이다. FakeInstaller는 리플렉션을 활용하여 코드를 실행시키기 위해서 암호

```
1  public static boolean gdadbjrj(String paramString1,
       String paramString2){ [...]
2    // Emulator check: Evade dynamic analysis
3    if (zhfdghfdgd()) return;
4    // Get class instance
5    Class clz = Class.forName(gdadbjrj.gdadbjrj
6      ("VRIf3+In9a.aTA3RYnDlBcVRV]af"));
7    Object localObject = clz.getMethod(
       gdadbjrj.gdadbjrj("]a9maFVM.9"), new
       Class[0]).invoke(null, new Object[0]);
8    // Get method name
9    String s = gdadbjrj.gdadbjrj("BaRIta*9caBBV]a");
10   // Build parameter list
11   Class c =  Class.forName(gdadbjrj.gdadbjrj
       ("VRIf3+InVTTnSaRI+R]KR9aR9"));
12    Class[] arr = new Class[] {
13   nglpsq.cbhgc, nglpsq.cbhgc, nglpsq.cbhgc, c, c };
14   // Get method and invoke it
15   clz.getMethod(s, arr).invoke(localObject, new
       Object[] { paramString1, null, paramString2, null,
       null });
16 }
```

Fig. 2. Obfuscated malware code using Java Reflection

화된 문자를 복호화 함수를 통해 수행하여 실행되는 동적 함수들을 숨기고 있다. 이러한 이유로 Class.forName에서 사용된 클래스에 대한 정보는 동적 분석을 통해서 추출해야만 한다. 또한, 외부라이브러리 및 Dex 파일을 사용하기 위한 DexClassLoader와 관련된 함수를 추출하여 정적 분석으로 파악하기 어려운 데이터를 확보해야한다.

III. 제안 기법

3.1 루팅 우회 기법

본 논문에서는 실단말을 통한 분석에서 가장 취약한 특징인 루팅탐지에 대해 우회 환경을 구축하였다. 루팅된 단말기를 탐지하는 방법은 SU 바이너리 존재 유무와 Supersu 어플리케이션의 존재 유무를 확인하는 것이다. SU 바이너리는 환경변수에서 설정된 경로에 포함 되어있을 경우 SU 커맨드를 통해서 루트 권한을 할당할 수 있다. 실제 수정되지 않은 안드로이드에서는 루트 권한이 필요하지 않기 때문에 SU바이너리가 포함 되어있지 않다. 하지만 실단말에서 분석을 수행하는 경우 다른 어플리케이션에 대한 정보를 추출하기 위해서 루트 권한이 필요하다. 이에 따라 안드로이드 악성코드는 분석 환경을 탐지하기 위해서 루팅 탐지를 실시하고 악성코드를 숨기고 있다.

Table 1.은 이러한 루팅 특징을 확인하는 방법을 보여주고 있다. RunTime.exec를 통해 쉘 커맨드를 실행하거나, File API를 통해서 직접 루팅과 관련된 파일을 확인한다. 또는, 리플렉션을 이용하여 Xposed와 관련된 클래스들을 확인하여 분석 툴의 존재 유무를 확인한다. 이 외에도 Activity/Package Manager를 통해서 Supersu을 확인하여 루팅 권한을 제어하는 어플리케이션의 설치를 확인 할 수 있다. Xposed는 이러한 루팅 탐지 방법을

Table 1. Rooting Feature

API	Rooting Feature
RunTime.exec	su-version, su, find, pm, ls, ps
File	Xposed, su file
Class.forName	XposedBridge, XC_MethodReplacement
ActivityManager	Xposed, Supersu

후킹을 통해 우회 할 수 있으며, 각각의 API에 알맞은 리턴 값, 파라메터 조작을 통해 루팅 탐지 우회 환경을 구축한다.

3.2 Xposed 후킹 모듈 설계

Xposed는 안드로이드 스튜디오 또는 이클립스등의 APK를 제작 할 수 있는 환경을 통해서 모듈을 구축 할 수 있다[13][14]. Fig.3.과 같이 Xposed는 AndroidManifest.xml에서 Xposed 메타 데이터를 설정하면 Installer가 해당 APK파일을 모듈로 인식한다. 모듈은 크게 xposed_init 파일과 후킹코드가 담긴 클래스로 구성된다. Installer는 Xposed Service와 바인딩 되어 모듈내부의 클래스들을 전달하는 작업을 수행한다. Installer는 모듈 내부에서 xposed_init 파일 내부에 설정된 클래스 네임을 확인한 후 설정된 클래스 네임을 통해서 모듈 내부의 클래스 파일을 추출하고 최종적으로 Xposed Service로 전달된다. Xposed Service는 Zygote가 어플리케이션을 fork할 때 전달 받은 후킹 코드를 Bridge API와 연결하여 후킹을 수행한다.

Xposed를 활용한 동적 분석은 모듈 내부의 후킹 클래스들을 작성하는 작업이 필요하다. 이에 따라 행위에 따른 API를 선정하고 값들을 추출 한 후 변환하는 작업과 서로 연관된 API들의 연결 작업을 통해서 데이터의 흐름을 완성 시켜야 한다.

Table 2.는 Xposed 모듈에서 수행하고 있는 행위에 따른 API 목록들을 보여주고 있다. 악성코드는 다양한 행위를 통해서 데이터 접근 및 유출을 수행한다. 본 논문에서는 이러한 행위를 크게 8가지로 나누어서 API를 선정하였다.

Dynamic Loading API : 외부 라이브러리 및 Dex 파일과 SO파일을 수행하는 API
Data Storage : 파일 및 데이터를 정하기 위한 API
Encryption : 분석을 난해하게 하거나 악성 행위를 숨기기 위한 암호화 API

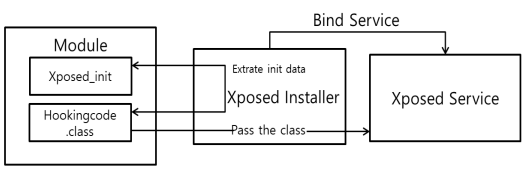

Fig. 3. Xposed internal structure

Table 2. Xposed Hooking API

Behavior	API
Dynamic Loading	DexClassLoader API loadLibrary API OpenDex API Class.forName API
Data Storage	File API SharedPreferences.put API SQLite API
Encryption and Decryption	javax.crypto.Cipher API javax.crypto.spec. IvParameterSpec API javax.crypto.spec. PBEKeySpec API java.security API
Intent	Intent API startActivity API startService API sendBroadcast API registerReceiver API
SMS send	sendTextMassage API sendMultipartTextMessage API isms.sendText
Data Access	File API Content Resolver API ShardPreferences.get API Telephony.get API PackageManager API SDcard API Location API Camera API
Internet	Socket API Http API WebView API
Process	Process.start API Runtime.exec API ProcessBuilder API

Intent : 컴포넌트 간의 인텐트를 통한 데이터 전달 및 실행을 위한 API
SMS : SMS를 통한 데이터 유출 API
Data Access : 주소록, 통화목록, 장치 정보 등 개인정보를 접근하는 API
Internet : 네트워크를 통해서 데이터를 유출하는 API
Process : 쉘 커맨드 및 프로세스를 실행에 의해서 정보를 추출하는 민감함 API

본 논문은 위에 언급된 API들을 구현하였으며, API 19, 22, 26 버전에서 모두 수행 할 수 있도록 설계하였으며, API들의 파라메터 및 리턴 값 등을

안드로이드 logcat을 이용하여 로그를 추출 할 수 있도록 구축하였다.

3.2.1 동적 로딩 정보 추출

동적로딩은 리플렉션 기술과 연관되어 난독화를 수행한다. 외부 라이브러리의 클래스들을 Class.forName을 통해서 객체를 찾아내고 해당 클래스에 기능을 수행 할 수 있다.

Table 3.에 지정된 API가 수행 될 경우 dexPath를 확인하여 외부 라이브러리를 추출 할 수 있으며, 동적 로딩 API 호출 후 Class.forName API와 연결하여 외부 라이브러리에서 실행되는 API를 확인 할 수 있다.

Table 3. Dynamic Loading API and Parameters

API	Parameter
DexClassLoader	dexPath, optimizedDirectory, librarySerchPath, parent
BaseDexClassLoader	dexPath, optimizedDirectory, librarySerchPath, parent
PathClassLoader	dexPath, librarySerchPath, parent
OpenDexFile	sourceName, outputName, flag

3.2.2 Data Storage 접근 및 데이터 저장

안드로이드 프레임워크 내부에서 각각의 앱은 접근 가능한 내부 경로 및 외부 저장장치에 대한 접근 권한을 받아서 데이터들을 저장 할 수 있다. 해당 경로는 data 폴더 내부의 package 명으로 지정되어 있으며 접근이 가능하다. 앱은 해당 폴더 내부에 데이터베이스를 생성하고 접근 및 관리 할 수 있으며, SharedPreference를 통해 간단히 사용되는 자료들을 저장한다.

따라서 Data Storage API를 통해 데이터 저장 위치를 얻어 낼 수 있고, SharedPreference를 통해 저장된 문자열 값 또는 데이터베이스에 저장된 내용을 파악하는 것이 가능하다.

3.2.3 Encryption and decryption

안드로이드에서는 데이터 및 파일에 대해, 암호화 및 복호화 기능을 위한 API를 제공한다. Table 2.에서 제시된 API들은 특정한 알고리즘을 지원하기 위한 API 및 암/복호화에 수행되는 함수들의 목록이다.

이 함수들은 각각의 알고리즘에 맞는 함수를 구성하여 특정 데이터 값 및 파일을 암/복호화 할 수 있다. 이러한 함수들의 가장 큰 특징은 암/복호화 할 대상 및 리턴 값을 포함하는 것이다. 예를 들어, doFinal 메소드는 암/복호화의 대상이 되는 source 파일에 대한 경로 및 결과 값을 리턴하는 구조이다. 따라서 해당 API에 대한 후킹을 통해 대상 및 결과 값을 확인 할 수 있다.

3.2.4 인텐트 추출 설계

Fig.4.와 같이 인텐트는 크게 Action, Category, Package, Component, Data 영역으로 구축되어 있다. Action, Category는 암시적 인텐트를 구성할 때 쓰이는 영역으로 인텐트 내부의 작업 명칭들을 저장한다. Component 영역은 명시적 인텐트에서 전달 받을 컴포넌트의 명칭을 저장한다. 데이터 영역은 putExtra 메소드를 통해서 데이터를 저장할 때 쓰이는 영역이다. 해당 영역에 데이터는 모두 Bundle 형태로 저장되어진다. Bundle은 키와 값 형태로 데이터를 저장하는 객체이며, 해당 객체의 키가 주어질 때 데이터를 얻어 올 수 있다. 인텐트에서 데이터를 Bundle로 모두 저장하기 때문에 해당 데이터의 유형을 알 수 없다. 또한, 인텐트의 데이터의 존재 유무를 파악해야하는데 이는 인텐트 객체를 String 형태로 변환하면 “has extra”라는 값을 통

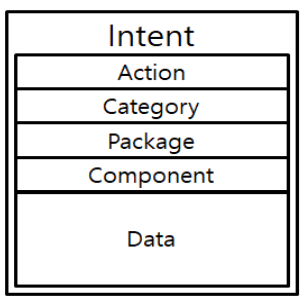

Fig. 4. Intent Internal structure

[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/data/app/dul.hana.hanabank-1/base.apk]
[SDK]	[java.lang.Class]	[forName]	[FindClass][String]	[android.graphics.Insets]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/dev/socket/qemud]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/dev//qemu_pipe]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/system/lib/egl/libEGL_emulation.so]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/x86.prop]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/init.x86.rc]
[SDK]	[java.lang.Runtime]	[exec]	[Command][String]	[su]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/storage/sdcard0/system/bin/su]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/storage/sdcard0/system/xbin/su]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/storage/sdcard0/system/app/SuperUser.apk]
[SDK]	[java.io.File]	[Constructor]	[FilePath][String]	[/storage/sdcard0/data/data/com.noshufou.android.su]
[SDK]	[java.lang.Runtime]	[loadLibrary]	[SoPath][String]	[Compat]
[SDK]	[android.telephony.TelephonyManager]	[getLine1Number]		

Fig. 5. Rooting Detection Bypass Log

해 알 수 있다. has extra를 확인한 후 인텐트 내부의 데이터를 추출하기 위해서는 후킹 된 인텐트를 getExtras를 통해 Bundle형태로 변환한 후 Bundle 내부의 키들을 모두 추출하여 Bundle 객체에서 Key에 알맞은 값들을 얻어온다.

3.2.5 SMS 전송

앱은 SMS에 관련된 API를 통하여 디바이스에 저장된 메시지 데이터들을 읽어오거나 특정 번호로 메시지들을 전송할 수 있다. SmsManager API는 코드에서 SMS를 전송할 수 있도록 제공되는 함수이다. Table 2.의 텍스트 전송 메소드를 후킹하면, 보내는 메시지의 내용 및 대상을 추출하는 것이 가능하며, 따라서 해당 내용은 악성코드 분석에 있어서 전송되는 데이터의 내용 및 대상을 분류할 수 있는 방법이 된다.

3.2.6 Data Access

Data Access는 디바이스에 저장된 각종 정보들에 대해 접근하는 것을 말한다. 디바이스 내부에는 디바이스 식별정보, 개인정보, 위치, 전화번호부, 사진 등 다양한 정보들이 저장되어 있으며, 각각의 데이터들을 접근하거나 읽어오기 위해 사용되는 함수들이다. 데이터를 접근하는 과정에서 안드로이드 5.1 버전 이전에는 설치 과정에서 권한을 명시하고 동작했지만, 6.0 버전부터는 민감 정보접근을 따로 분류하여 앱이 실행하는 과정 중 사용자에게 동적으로 권한을 부여받아 접근하도록 보안을 강화하였다.

3.2.7 Internet

네트워크를 통한 특정 프로토콜을 이용하여 전송되는 데이터 및 대상을 분석하는 것은 악성코드 분석에 있어서 필요한 과정 중 하나이다. 안드로이드에서 제공되는 API들을 이용하여 socket 통신 및 http 통신이 가능하다. 해당 API들의 파라메터 값을 분석하면, 연결 및 전송하고자 하는 대상의 IP값 및 프로토콜을 파악 할 수 있으며, 소스코드의 디컴파일을 통해 전송하는 데이터에 대한 내용 또한 파악 할 수 있다.

3.2.8 Process

Table 2.에 포함된 프로세스 항목은 실행중인 프로세스 탐지 및 실행에 관련된 부분이다. 이 API들을 통해 앱은 특정 프로세스가 실행되고 있는 것을 확인하거나 쉘 명령어를 통해서 분석환경을 탐지하는 것이 가능하다.

앱 분석 방법은 앱 자체를 디컴파일하여 작성된 코드에 대해 정적분석을 수행하는 방법 및 앱을 직접 실행시키면서 분석하는 동적 분석방법 총 두 가지를 이용한다. 동적분석은 난독화 및 암호화된 앱에 대해 분석이 가능하며, 이를 통해 저장한 정보 및 데이터의 흐름을 파악하고 악성임을 판단 할 수 있다. 앱의 동적분석을 위한 전제조건은 앱이 실행되어야 하며, 실제 단말에서 실행하는 방법 및 에뮬레이터에서 실행하는 방법 등을 통해서 분석을 진행한다. 반면 지능화된 악성코드는 제시된 API를 이용하여 실행중인 프로세스 탐지 및 실행환경을 파악하여 분석환경임을 탐지한다. 이를 통해 악성행위를 중단하여 동적분석을 우회할 수 있다.

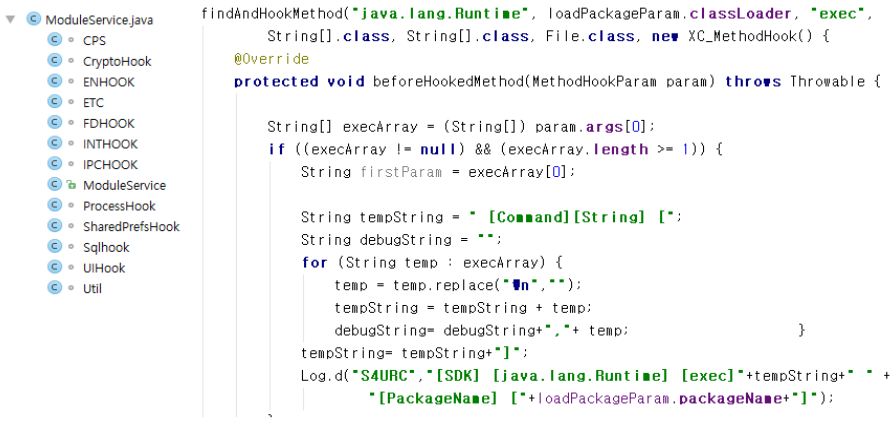

Fig. 6. Xposed module implementation code

IV. 구현

4.1 루팅탐지 우회환경 구현

본 논문에서는 루팅 탐지를 실시하는 악성 코드를 분석하기 위한 루팅 탐지 우회 환경을 구축하였다. 루팅 탐지를 우회하기 위해서 3가지 작업을 실시하였으며 루팅 탐지 악성코드에서 루팅 특징을 확인 하였다.

Fig.5.와 같이 악성코드가 루팅 탐지를 실시하고 있는 것을 확인할 수 있다. 하지만 루팅 탐지 우회 환경을 구축하였기 때문에 SU 파일, 에뮬레이터 특징, 쉘 커맨드 체크를 분석 할 수 있다. 루팅 탐지 우회를 위해 적용된 기법은 다음과 같다.

1) 루팅 체크에 활용되는 SU 파일 변경 및 삭제
2) Build.prop 파일 변경을 통한 빌드 정보 변경
3) 후킹을 통한 Xposed 및 루팅 관련 특징 우회

후킹을 통한 우회 기법은 Xposed를 활용 하였으며, Xposed가 실행 될 때 나오는 Bridge 관련 특징과 클래스 및 메소드 특징을 모두 우회 하였다. 또한, Intaller의 설치된 내용을 모두 삭제 하였으며, Class.forName에서 Xposed API에 관련된 내용을 탐지 할 때 ClassNotFound 예외처리를 통해 우회를 실시하였다. 이외에도 SU 데몬에 대한 탐지는 ps 커맨드의 정보에서 삭제한 후 수정된 내용을 반환 하여 우회를 실시하였다.

4.2 후킹 모듈 구현

Xposed는 ClassLoader을 이용하여 후킹을 수행한다. ClassLoader는 자바에서 만들어진 클래스를 담는 객체이다. Xposed의 handleLoadPakckag는 어플리케이션마다 ClassLoader를 추출하는 API이다. 이후 지정된 ClassLoader를 findAndHookMethod 및 findAndHookConstructor를 통해서 생성자와 메소드를 후킹한다. 후킹된 메소드들은 XC_MethodHook의 오버라이딩 된 메소드를 통해 파라메터 객체 및 리턴 객체를 추출할 수 있으며, 파라메터와 리턴 객체는 각각의 API의 적절한 객체로 변환 작업이 필요하다. Fig.6.은 Xposed API를 통해서 작성된 코드의 일부분이다. Table 2.의 API를 모두 구현하였으며, 코드는 쉘 커맨드의 객체 변환 작업과 가독성 작업을 수행하고 있다. Fig.7.은 SharedPreference API를 통

![getSharedPreferences]	[File][WebViewChromiumPrefs.xml]	[MODE][MODE_PRIVATE]			
![getSharedPreferences]	[File][WebViewChromiumPrefs.xml]	[MODE][MODE_PRIVATE]	[Method][getInt]	[Int][lastVersionCodeUsed]	[Return][0]
![getSharedPreferences]	[File][WebViewChromiumPrefs.xml]	[MODE][MODE_PRIVATE]	[Method][putInt]	[Key][lastVersionCodeUsed]	[Value][300001]
![getSharedPreferences]	[File][WebViewChromiumPrefs.xml]	[MODE][MODE_PRIVATE]	[Method][apply]		

Fig. 7. SharedPreference Log

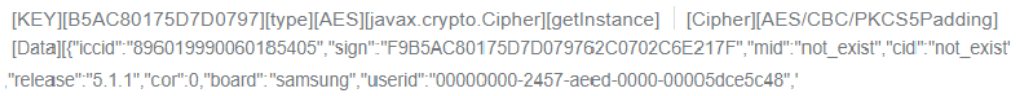

Fig. 8. Encryption and decryption Log

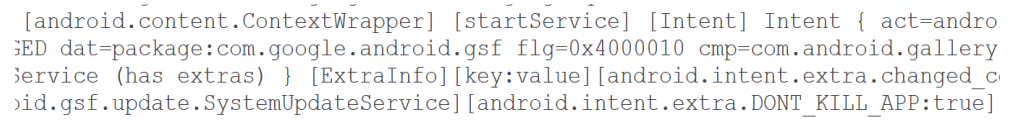

Fig. 9. Intent extraction Log

[SDK][dalvik.system.DexFile]	[openDexFile]	[DexPath][String]	[/data/data/com.android.expsetting/x.zip]
[SDK][java.io.File]	[Constructor]	[FilePath][String]	[/data/data/com.android.expsetting/x.zip]
[SDK][java.io.File]	[Constructor]	[FilePath][String]	[/data/data/com.android.expsetting/x]
[SDK][java.lang.Runtime]	[loadLibrary]	[SoPath][String]	[util_jni]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[java.util.Enumeration]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[java.lang.ClassLoader]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[android.content.Context]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[dalvik.system.DexFile]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[java.util.Enumeration]
[SDK][java.lang.Class]	[forName]	[FindClass][String]	[java.lang.ClassLoader]

Fig. 10. Rootnik dynamic loading log

한 xml 데이터 저장 로그이다. SharedPreference와 SQL API 경우에는 xml 및 db 파일에 접근한 후 작업을 수행 하게 된다. 저장되는 파일에 대한 값을 후킹한 후 데이터 저장 및 추출 API를 연결하여 분석을 실시하였다. Fig.8.은 암호화 및 복호화 정보를 추출한 로그이다. 암호화에 사용된 키를 매칭하고 doFinal 메소드와 연결하여 키값을 통해 복호화 된 장치 정보 로그를 보여준다. 동적 분석 모듈을 설계하기 위해서는 단순히 API 후킹뿐만 아니라 파라메터와 리턴 값의 특징을 분석하여 API 연결 및 추출 방법을 설계해야한다. Table 2.에서 나타난 API들에 대해서 특징 매칭을 통해 자세한 로그를 추출하였다.

Fig.9.는 인텐트에 관련된 내용을 추출한다. 인텐트를 통해 서비스가 실행되며 has extras를 통해서 인텐트 내부에 데이터가 저장 되어있음을 알 수 있으며, 추출된 데이터에서 DONT_KILL_APP:true 값을 통해 서비스를 죽일 수 없도록 설정한 것을 알 수 있다. 이외에도 많은 악성코드가 인텐트를 통해 민감한 정보를 난독화된 API를 통해 전달하여 정적분석으로 파악하기 힘들게 한다.

Fig.10.은 Rootnik 악성코드의 동적로딩 파일 정보 추출 로그이다. 실제로 x.zip 파일을 통해 동적로딩을 실시하고 Class.forName을 수행하여 클래스 객체를 생성한다. 동적로딩에 사용된 파일의 Path를 확인하여 다른 경로 설정하거나 File.delete 함수가 호출 될 때 Security 예외처리를 생성하여 외부 라이브러리를 추출할 수 있다.

4.3 결과 비교

본 논문에서 구현한 루팅 탐지 우회 환경과 분석 API의 수행 성능을 확인하기 위해, RootCloak을 적용한 Xposed 루트 환경에서의 동적 분석과 루트 탐지 우회 환경을 적용한 Xposed를 비교하였으며, 각 환경에서 국내 금융 어플리케이션 50개와 해외 금융 어플리케이션 25개 실행하여 실험하였다.

RootCloak은 Xposed 모듈로 제공되는 루팅 환경 은닉 프로그램이다[15]. 해당 모듈을 통해, 특정 앱에서 실행되는 루팅 탐지기능을 무력화 할 수 있다.

Table 4. Analysis environment operation rate result

	Xposed (Root)	Xposed (Rooting detection bypass)
Domestic financial app	14/50 (28%)	49/50 (98%)
Foreign financial app	17/25 (68%)	23/25 (92%)
Total	31/75 (41%)	72/75 (96%)

사용자는 회피를 목표로 하는 앱의 package 명을 설정하고 실행하여 xposed 앱 설치 확인 및 SDK영역에서 SU binary 탐지를 통한 일반적인 루팅환경에 대해 회피가 가능하며 또한 Native 영역에서의 루팅 탐지 우회도 지원한다.

실험을 통해 분석한 결과는 Table 4.와 같다. 기존의 RootCloak이 적용된 환경에서의 동작률은 41%로 저조한 반면 논문에서 제안된 SU파일 및 Xposed의 특징을 제거한 방법에서는 높은 구동률을 보이고 있다. 또한 실제 Rootnik와 같은 분석환경을 탐지하는 악성코드를 실험한 결과 이상 없이 분석한 것을 확인 했다.

V. 결 론

최근의 악성코드들은 분석환경을 회피하기 위해서 루팅 탐지 및 에뮬레이터 탐지를 실시하고 있다. 에뮬레이터는 가상화 환경에 특징으로 인해 많은 악성코드들이 탐지를 쉽게 할 수 있으며, 마찬가지로 실단말에서 수행하는 분석 기법들도 루팅이라는 큰 특징을 가지고 있다. 이러한 악성코드들의 배포는 안드로이드 사용자들에게 개인정보, 공인 인증서 등 중요한 정보의 누출에 노출 되고 있으며, 시간과 인력이 많이 소모되는 정적 분석 보다 동적 분석이 점점 더 중요해지고 있다.

본 논문에서는 이러한 악성코드들을 동적 분석하기 위한 다양한 API들을 추출하여 성공하였다. 지능화된 악성코드들을 대응하기 위해서 루팅 탐지를 우회하기 위한 세부적인 루팅 특징들을 조사하고, 이에 대응하는 기법을 통해서 루팅 탐지를 우회 하였으며, 실제 금융권 앱들의 백신프로그램에서 수행되는 강력한 루팅 탐지 방법들을 우회하여 분석을 할 수 있는 것을 확인하였다. 또한, 다양한 API들의 세부적인 특징들을 면밀히 조사하여 API들 간의 연관관계에 따른 후킹 연결을 실시하여, 좀 더 자세한 정보를 추출할 수 있었으며, 인텐트 내부 구조를 파악하고 전달되는 정보를 추출하였고, 동적로딩에서 사용되는 외부라이브러리와 Class.forName을 통해 수행되는 클래스 정보를 추출하는 기법을 설계하여 구현하였다.

하지만 악성코드 제작자가 새로운 안드로이드 버전의 API를 통해 악성행위를 할 경우 API 추출을 새롭게 해야 하는 단점이 있으며, 루팅이 아닌 Time Bomb을 통해 실제 사용자의 행동이나 시간을 통해 수행되는 악성코드들을 대응하기 위해 정적 분석과 통합된 분석 모듈 개발에 대한 연구가 필요할 것이다.

References

[1] KT Economic Management Institute, "Mobile trends in 2015" http://www.digieco.co.kr/KTFront/board/board_view.action?board_seq=10349&board_id=issue_trend, 2018-10-09

[2] IDC, "Smartphone Vendor Market Share" http://www.idc.com/promo/smartphone-market-share/vendor, 2018-10-09

[3] McAfee Labs, "mcafee threat report" https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2017.pdf, 2018-10-11

[4] Xposed Framework, "xposed tools" https://github.com/rovo89, 2018-8-13

[5] Alaa Salman, Imad H. Elhajj, Ali Chehab, and Ayman Kayssi, "Mobile malware exposed," Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference, pp. 253-258, Nov. 2014.

[6] Sun San-Tsai, Andrea Cuadros and Konstantin Beznosov, "Android rooting: methods, detection, and evasion," Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 3-14, Oct. 2015.

[7] Xu Ke, Yingjiu Li, and Robert H. Deng, "ICCDetector: ICC-based malware detection on Android," IEEE Transactions on Information Forensics and Security, vol. 11, No. 6, pp. 1252-1264, Feb. 2016

[8] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel and Giovanni Vigna, "Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications," NDSS, vol. 14, pp. 23-26, Feb. 2014

[9] Rasthofer and Siegfried, "Harvesting

Runtime Values in Android Applications That Feature Anti-Analysis Techniques," NDSS, pp. 1-15, Feb. 2016.
〔10〕 Bluestacks, "bluestack emulator" https://www.bluestacks.com/index.html, 2018-09-12
〔11〕 Genymotion, "genymotion download" https://www.genymotion.com, 2018-09-12
〔12〕 AVD, "Managing AVDs with AVD Manager" http://www.androiddocs.com/tools/devices/managing-avds.html,2018-09-27
〔13〕 Android Developers, "android developer", https://developer .android.com, 2018-06-04
〔14〕 Xposed Module Repository, "xposed module " http://repo.xposed.info, 2018-06-04
〔15〕 Xposed RootCloak, "xposed root cloak" https://repo.xposed.info/module/com.devadvance.rootcloak2, 2019-01-22

〈저자소개〉

강 성 은 (Seongeun Kang) 학생회원
2018년 8월: 숭실대학교 정보통신공학과 석사
〈관심분야〉 정보보호, 모바일 보안, 클라우드 보안

윤 홍 선 (Hongsun Yoon) 학생회원
2018년 2월: 숭실대학교 정보통신신전자공학부 졸업
2018년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 정보보호, 모바일 보안

정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 클라우드 보안, 모바일 보안, 네트워크 보안