Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.3, Jun. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.3.519

분석 회피 기능을 갖는 안드로이드 악성코드 동적 분석 기능 향상 기법*

안 진 웅,† 윤 홍 선, 정 수 환‡
숭실대학교

An Enhancement Scheme of Dynamic Analysis for Evasive Android Malware*

Jinung Ahn,† Hongsun Yoon, Souhwan Jung‡
Soongsil University

요 약

지능화된 안드로이드 악성코드는 안티바이러스가 탐지하기 어렵도록 악성행위를 숨기기 위하여 다양한 분석 회피 기법을 적용하고 있다. 악성코드는 악성행위를 숨기기 위하여 백그라운드에서 동작하는 컴포넌트를 주로 활용하고, 자동화된 스크립트로 악성 앱을 실행할 수 없도록 activity-alias 기능으로 실행을 방해하고, 악성행위가 발견되는 것을 막기 위해 logcat의 로그를 삭제하는 등 지능화되어간다. 악성코드의 숨겨진 컴포넌트는 기존 정적 분석 도구로 추출하기 어려우며, 기존 동적 분석을 통한 연구는 컴포넌트를 일부만 실행하기 때문에 분석 결과를 충분히 제공하지 못한다는 문제점을 지닌다. 본 논문에서는 이러한 지능화된 악성코드의 동적 분석 성공률을 증가시키기 위한 시스템을 설계하고 구현하였다. 제안하는 분석 시스템은 악성코드에서 숨겨진 컴포넌트를 추출하고, 서비스와 같은 백그라운드 컴포넌트인 실행시키며, 앱의 모든 인텐트 이벤트를 브로드캐스트한다. 또한, 분석 시스템의 로그를 앱이 삭제할 수 없도록 logcat을 수정하고 이를 이용한 로깅 시스템을 구현하였다. 실험 결과 본 논문에서 제안한 시스템을 기존의 컨테이너 기반 동적 분석 플랫폼과 비교하였을 때, 악성코드 구동률이 70.9%에서 89.6%로 향상된 기능을 보였다.

ABSTRACT

Nowadays, intelligent Android malware applies anti-analysis techniques to hide malicious behaviors and make it difficult for anti-virus vendors to detect its presence. Malware can use background components to hide harmful operations, use activity-alias to get around with automation script, or wipe the logcat to avoid forensics. During our study, several static analysis tools can not extract these hidden components like main activity, and dynamic analysis tools also have problem with code coverage due to partial execution of android malware. In this paper, we design and implement a system to analyze intelligent malware that uses anti-analysis techniques to improve detection rate of evasive malware. It extracts the hidden components of malware, runs background components like service, and generates all the intent events defined in the app. We also implemented a real-time logging system that uses modified logcat to block deleting logs from malware. As a result, we improve detection rate from 70.9% to 89.6% comparing other container based dynamic analysis platform with proposed system.

Keywords: Android, Malware, Dynamic Analysis

Received(12. 28. 2018), Modified(04. 25. 2019), Accepted(05. 25. 2019)

* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2016-0-00078, 맞춤형 보안서비스 제공을 위한 클라우드 기반 지능형 보안 기술 개발)과 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2019-0-00477, 가상화된 신뢰실행환경을 이용한 안드로이드 보안 프레임워크 기술 개발)

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

I. 서 론

전 세계에 스마트폰과 태블릿 PC와 같은 모바일 디바이스의 보급률이 높아지고 있으며, 모바일 OS 중 안드로이드 OS가 가장 높은 점유율을 지닌다[1]. IDC는 2016년부터 안드로이드 OS가 85%로 높은 점유율을 유지하고 있으며, 2022년까지 85.5%의 높은 점유율을 유지할 것으로 전망한다[2]. 또한 안드로이드 디바이스 사용자는 꾸준히 증가하여 2019년에 25억 명에 이를 것으로 추정된다[3]. 이에 따라 이러한 모바일 시장을 목표로 하는 악성코드 또한 증가하는 추세이다. 2018년 9월 발간된 McAfee의 보고서에 따르면, 높은 점유율을 보유한 안드로이드 OS를 겨냥한 악성코드가 꾸준히 증가하여 새로운 악성코드가 지난 분기 대비 27% 증가하며 매 분기 증가하는 추세를 나타내고 있다[4]. 이러한 안드로이드 악성코드는 점차 지능화되며 분석을 난해하게 만들고 앱의 수명을 늘리기 위하여 분석을 회피하는 기법을 적용하고 있다. 지능화된 악성코드는 분석 환경에서 앱이 자동 실행하는 것을 막기 위하여 컴포넌트를 숨기는 기법을 적용하고, 악성코드가 실행되는 것을 사용자에게 숨기기 위해 백그라운드에서 실행되는 컴포넌트에서 주로 악성행위를 수행한다[5]. 또한 앱에서 무의미한 더미 로그를 무한정 생성하거나 분석 로그를 삭제하여 악성코드 분석을 방해하는 기법을 사용한다. 동적 분석을 제공하는 기존 연구에서는 이러한 분석 회피 기법을 적용한 안드로이드 악성코드를 분석하는데 어려움이 있으며, 지능화되는 악성코드를 위하여 분석 시스템 또한 고도화가 이루어져야한다.

본 논문에서는 지능화된 악성코드를 분석하기 위하여 안드로이드 악성코드에서 분석을 회피하는 기법을 정리하고, 분석 회피 기법을 적용한 악성코드를 분석하기 위해 향상된 동적 분석 기능을 갖는 분석 시스템을 제안한다.

II. 관련 연구

2.1 안드로이드 악성코드의 분석 회피 기법

2.1.1 안드로이드 앱의 컴포넌트를 숨기는 기법

안드로이드 앱은 앱 아이콘을 클릭하는 것으로 앱을 실행할 수 있으며, 최초 실행 시 아이콘에 매핑된 컴포넌트인 메인 액티비티가 실행된다. 앱에서 사용되는 컴포넌트는 AndroidManifest.xml 파일에서 정의한다. Fig.1.과 같이 〈activity〉 요소에 LAUNCHER 카테고리와 Main 액션을 〈intent-filter〉요소를 통해 지정하는 것으로 매핑할 수 있다. LAUNCHER 카테고리와 MAIN 액션을 지정하면 앱 아이콘과 해당 액티비티를 연결하며, 앱의 시작 지점으로 정의하는 것을 의미한다. 앱 아이콘 클릭 시 아이콘에 매핑 된 액티비티가 실행되며, 이러한 액티비티를 메인 액티비티라 한다.

그러나 AndroidManifest.xml 파일 내에서 액티비티를 정의하는 방법에는 〈activity〉 요소 외에도 〈activity-alias〉 요소가 존재한다. 〈activity-alias〉 요소는 액티비티에 이름 외에 별칭을 정의하는 기능을 제공한다. Fig.2.와 같이 액티비티를 정의하기 위하여 〈activity-alias〉 요소를 사용하는 경우, 〈activity〉와 동일한 문법을 이용하기 때문에 LAUNCHER 카테고리와 MAIN 액션을 지정하여 메인 액티비티를 정의할 수 있다. 〈activity-alias〉 요소를 이용할 경우, 사용자가 앱 아이콘 클릭 시 name 속성에 매핑된 액티비티를 실행하도록 요청하지만, 실제로 실행되는 액티비티는 targetActivity 속성에 정의된 액티비티이다.

메인 액티비티를 정의하기 위하여 〈activity-alias〉 요소를 이용한 경우, 안드로이드 SDK(Standard Development Kit)의 빌드 도구인 AAPT(Android Asset Packaging Tool)[7]를 이용한 컴포넌트 추출 시 Fig.3.과 같이 메인 액

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Fig. 1. Main activity definition in AndroidManifest.xml[6]

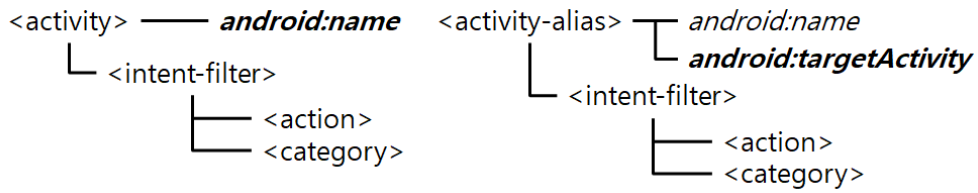


Fig. 2. Comparison of activity definition between 〈activity〉 and 〈activity-alias〉

```
root@S4URC-Set01-Host23:~# aapt d badging c2c2e5e83a45d25cc0a56c5f16bf091f.apk
package: name='com.net.bigger.flower.source.generally' versionCode='63' versionName='6.3'
sdkVersion:'10'
targetSdkVersion:'22'
uses-permission:'android.permission.INTERNET'
uses-permission:'android.permission.ACCESS_NETWORK_STATE'
uses-permission:'android.permission.WRITE_EXTERNAL_STORAGE'
application-label:'Music Player'
application-icon-160:'res/mipmap-xxhdpi-v4/ic_launcher_music.png'
application-icon-480:'res/mipmap-xxhdpi-v4/ic_launcher_music.png'
application: label='Music Player' icon='res/mipmap-xxhdpi-v4/ic_launcher_music.png'
uses-permission:'android.permission.READ_EXTERNAL_STORAGE'
```

Fig. 3. Sample of failure to extract main activity using AAPT

티비티를 추출하는 것이 불가능하다. 또한 안드로이드 애플리케이션을 역공학 하는 도구인 Androguard[8]는 〈activity-alias〉 요소로 정의된 액티비티를 추출할 수 있지만 액티비티의 name 속성으로 정의된 별칭만을 획득하며, 실제로 실행되는 메인 액티비티는 targetActivity 속성에 정의되어 있기 때문에 이를 획득할 수 없다. 이러한 기존 도구는 메인 액티비티가 〈activity-alias〉 요소를 이용하여 정의된 경우 이를 추출할 수 없다는 문제점을 지닌다. 이에 따라, 안드로이드 악성코드는 〈activity-alias〉 요소를 이용한 메인 액티비티를 정의하는 기법을 적용해 컴포넌트가 분석되는 것을 회피할 수 있다.

2.1.2 백그라운드 컴포넌트를 이용한 악성 행위 수행

안드로이드 앱의 컴포넌트는 액티비티 뿐만 아니라 서비스와 브로드캐스트 리시버가 존재한다. 서비스는 백그라운드 컴포넌트이기 때문에 UI 없이 동작할 수 있으며, 악성 앱은 사용자에게서 악성 행위를 숨기기 위하여 주로 서비스에서 악성행위를 수행한다[5]. 또한, 사용자가 아이콘을 클릭하지 않아도 자동으로 서비스를 실행할 수 있도록 브로드캐스트 리시버를 활용한다. 브로드캐스트 리시버는 디바이스에서 이벤트를 방송하면 이를 수신하여 실행되는 컴포넌트이며, 등록된 이벤트가 발생하면 앱에서 이벤트를 수신하여 브로드캐스트 리시버가 실행된다. 브로드 캐스트 리시버는 앱 내의 서비스를 실행하여 백그라운드 컴포넌트만으로 악성 행위를 수행할 수 있다. 따라서 악성 앱은 사용자에게 알리지 않고 악성행위를 수행하기 위하여 백그라운드 컴포넌트인 서비스와 브로드캐스트 리시버를 활용할 수 있다. 이에 더해, 안드로이드 앱은 메인 액티비티가 존재하지 않더라도 서비스와 브로드캐스트 리시버만으로 구성할 수 있다. 이러한 특수한 앱을 분석하기 위해 동적 분석 시스템은 서비스를 실행시킬 수 있는 기능을 포함하여 메인 액티비티가 존재하지 않는 앱에 대응해야한다.

2.1.3 Logcat의 결점을 이용한 분석 로그 삭제 기법

안드로이드에서는 설치된 앱의 동작과 오류 메시지를 기록하기 위하여 logcat 기능을 사용할 수 있다. 악성코드 분석가들은 앱을 분석 할 때, 앱이 동작하는 과정에서 생성된 로그를 실시간으로 분석하거나 저장하여 악성 앱을 탐지할 수 있다. 반면, 디바이스가 루팅 된 경우, logcat 버퍼에 기록된 로그를 삭제하기 위하여 “logcat -c” 명령어를 사용할 수 있다[5]. logcat의 -c 옵션은 logcat 버퍼에 기록된 로그를 모두 삭제하는 명령이다. 앱에서 직접 Runtime.exec() 메소드를 호출하여 “logcat -c“

명령어를 실행한 경우, 모든 로그가 삭제되어 분석 기록 또한 삭제되는 문제점이 발생한다.

또한, logcat은 버퍼 크기의 최댓값이 존재한다. 따라서 안드로이드 앱을 동적 분석 시, 분석 대상인 앱이 더미 로그를 무수히 많이 생성하여 버퍼 크기를 초과하여 logcat에 기록된 분석 로그가 삭제되는 문제점을 지닌다.

2.2 기존 안드로이드 동적 분석 시스템

안드로이드 앱을 분석하기 위한 방법은 정적 분석과 동적 분석이 존재한다. 이 중 동적 분석은 분석 대상인 앱을 분석 환경에서 실제로 실행하여 앱이 실행되는 도중 수행하는 행위들을 수집하여 분석하는 방식을 의미한다. 안드로이드 악성코드는 이러한 동적 분석을 회피하기 위한 기법들을 적용하므로, 동적 분석 시스템들은 악성코드에 적용된 분석 회피 기법을 파악하고 이에 대응할 필요성이 존재한다. Table.1은 앞서 언급된 동적 분석 툴들을 종합적으로 비교한 결과이며, 세부내용은 다음과 같다.

Tracedroid[9]는 QEMU 기반 안드로이드 에뮬레이터로 동적 분석을 수행하며, 안드로이드 프레임워크를 수정 후 빌드 한 분석 플랫폼을 이용한다. 함수 호출을 추적하기 위하여 바이트코드 인터프리터를 후킹하고, 네이티브 코드에서 호출하는 시스템 콜을 분석하기 위해 strace를 이용한다. 앱의 액티비티와 서비스를 추출하기 위하여 Androguard를 이용하기 때문에 메인 액티비티를 〈activity-alias〉로 정의한 경우 메인 액티비티를 실행할 수 없다는 문제점을 지닌다. 또한, 앱에 등록된 시스템 이벤트에 모두 대응할 수 없고 SMS 수신 등 특정 시스템 이벤트에 대해 에뮬레이팅하는 단점을 지닌다.

CopperDroid[10]는 QEMU를 기반으로 수정된 안드로이드 에뮬레이터에서 앱을 동적 분석하고 시스템 콜을 추적하는 시스템이다. 시스템 콜을 커널에서 추적하기 때문에 자바와 네이티브 코드에서 발생하는 행위를 분석할 수 있다는 장점을 지닌다. 동적 분석 시 Monkeyrunner을 이용해 UI 이벤트를 랜덤으로 발생시켜 앱을 테스트한다. 그러나 앱의 서비스를 실행하지 않기 때문에 메인 액티비티가 존재하지 않는 앱의 분석이 불가능하다는 단점을 지닌다. 또한, 시스템 이벤트는 배터리 부족 상태, 재부팅, SMS, 전화 수신에 대해서만 발생시키기 때문에 앱에 등록된 모든 이벤트에 대응할 수 없다.

Droidbox[11]는 안드로이드 에뮬레이터인 AVD (Android Virtual Device)에서 앱을 실행하여 파일 I/O와 SMS, 네트워크를 이용한 정보 유출의 발생을 모니터링하여 제공하는 오픈소스 도구이다. Androguard를 이용하여 동적 분석을 수행하기 위한 앱의 컴포넌트를 추출하며, TaintDroid[14]로 정보 유출이 발생하는지 확인한다. 하지만, Androguard는 〈activity-alias〉로 정의한 액티비티를 추출할 수 없기 때문에 Droidbox 또한 동일한 문제점이 존재한다. 또한, 동적 분석 수행 시 Monkeyrunner를 이용하여 UI 이벤트를 발생시키지만, 서비스를 실행하지 않고 시스템 이벤트는 전화 통화와 SMS 두 종류만 에뮬레이션하는 한계점을 가진다.

Mobile-Sandbox[13]는 QEMU를 기반으로 하는 안드로이드 에뮬레이터로 자동 분석을 수행하는 시스템이다. 자바뿐만 아니라 네이티브 코드에 대한 추적이 가능하며, Droidbox와 TaintDroid를 기반으로 데이터 유출이 발생하는지 모니터링한다. 동적 분석 시 Monkeyrunner를 이용하여 UI 이벤트를 발생시켜 앱을 테스트한다. 그러나 앱 컴포넌트를 추출하기 위하여 AAPT 도구를 사용하기 때문에 〈activity-alias〉로 정의된 메인 액티비티는 획득할 수 없어, 앱을 실행할 수 없다는 단점을 지닌다. 또한, 백그라운드 컴포넌트인 서비스를 실행하지 않으며, 시스템 이벤트를 발생시키지 않아 브로드캐스트

Table 1. Comparison of dynamic analysis tools

	〈activity-alias〉 activity	Run service	Broadcast event
Tracedroid[9]	No	Yes	Partial
CopperDroid[10]	Unknown	No	Partial
Droidbox[11]	No	No	Partial
Mobile-Sandbox[12]	No	No	No
C-Android[13]	No	No	No

```
<activity-alias android:enabled="true" android:icon="@mipmap/ic_launcher_settings" android:label="@string/app_alias"
android:name="com.dot.MainApplication$BackActivity" android:noHistory="true"
android:targetActivity="com.dot.MainApplication$MainActivity" android:theme="@android:style/Theme.NoDisplay">
▼<intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity-alias>
```

Fig. 4. Sample code of 〈activity-alias〉 in AndroidManifest

리시버를 실행할 수 없다는 문제점이 있다.

C-Android[14]는 LXC[15] 컨테이너 기반의 자동화된 안드로이드 동적 분석 플랫폼이다. 동적 분석을 수행하는 컨테이너 플랫폼은 호스트 OS 상에서 프로세스로 실행된다. 따라서 기존 ARM 에뮬레이터가 하이퍼바이저로 인해 성능이 저하되는 단점을 극복하였다. 그러나 C-Android는 동적 분석 시, 컴포넌트를 추출하기 위하여 AAPT 도구를 이용하기 때문에 〈activity-alias〉로 메인 액티비티를 정의한 앱은 분석이 불가능하다. 또한, 앱 분석을 위하여 메인 액티비티만 실행하기 때문에 서비스를 실행하지 않고 시스템 이벤트를 발생시키지 않는다.

대부분의 툴들은 〈activity-alias〉로 정의된 액티비티를 분석하지 못하였으며, 서비스 및 브로드캐스트리시버의 경우 또한 제한적인 분석을 보여준다. 따라서 본 논문에서는 C-Android가 제공하는 동적 분석 플랫폼을 기반으로, 지능화된 악성코드의 분석 회피 기능에 대응하기 위하여 앱의 컴포넌트를 추출하고 메인 액티비티 및 백그라운드 컴포넌트를 모두 실행하여 분석 성공률을 높이는 분석 시스템을 제안한다.

III. 제안 기법

3.1 앱 컴포넌트 추출 및 앱 분석 절차 설계

본 논문에서는 안드로이드 앱의 컴포넌트를 추출하고, C-Android의 컨테이너 기반 동적 분석 플랫폼을 이용하여 앱의 컴포넌트를 실행하여 구동률을 향상시키는 시스템을 설계 및 구현한다. 앞서 언급한 바와 같이, 안드로이드 악성 앱은 〈activity-alias〉 요소를 이용하여 앱의 메인 액티비티를 숨기는 회피 기법을 적용할 수 있다. 또한, 서비스와 브로드캐스트 리시버를 이용하여 악성 행위를 수행하여 사용자에게 보이지 않는 기법이 존재한다. 이러한 지능화된 악성 앱을 분석하기 위하여 정적 분석을 통해 앱의 컴포넌트를 추출하고, 추출한 컴포넌트를 실행하여 동적 분석을 수행하는 기법을 설계 및 구현한다.

3.1.1 분석 우회 기법이 적용된 메인 액티비티 추출

Androguard, AAPT 등 기존 도구가 추출할 수 없는 〈activity-alias〉 요소로 정의된 메인 액티비티를 추출하여 〈activity〉 요소 혹은 〈activity-alias〉 요소로 정의된 메인 액티비티의 이름을 추출한다. 〈activity〉 요소는 액티비티를 정의하기 위하여 name 속성을 이용하는 반면, 〈activity-alias〉 요소로 정의된 메인 액티비티는 Fig.4.과 같이 액티비티의 별칭을 name 속성으로 정의하고 실제 실행되는 액티비티는 targetActivity 속성으로 정의한다. 따라서 〈activity-alias〉 요소로 정의된 메인 액티비티를 추출하기 위해서는 name 속성이 아닌 targetActivity 속성에서 액티비티의 이름을 추출하여야한다. 메인 액티비티는 안드로이드 앱의 시작 지점이기 때문에 동적 분석 시, 메인 액티비티를 최우선적으로 실행하여 앱을 구동한다.

3.1.2 서비스와 인텐트 이벤트 추출

또한, 앱의 서비스와, 브로드캐스트 리시버에 등록된 모든 이벤트를 추출한다. 악성 앱은 사용자에게 악성 행위를 숨기기 위해 백그라운드 컴포넌트인 서비스에서 주로 악성 행위를 수행하기 때문에, 서비스를 추출하여 악성 행위를 수행하는 백그라운드 컴포넌트를 실행할 수 있고 동적 분석 시 코드 커버리지를 향상시킬 수 있다. 앱은 Fig.5.와 같이 서비스와 브로드캐스트 리시버만 존재하고 메인 액티비티가 존재하지 않을 수 있다. 이러한 앱은 메인 액티비티를 실행하는 방법으로는 분석이 불가능하다. 이 경우 서비스를 실행하면 해당 앱의 프로세스가 시작되며, 메인 액티비티가 존재하지 않더라도 앱의 동적 분석을 수행할 수 있다.

브로드캐스트 리시버는 리시버에 등록된 이벤트를

```
<application android:label="com.android.providers.downloadsmanager">
    <receiver android:name=".DownloadCompleteReceiver" android:enabled="true">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </receiver>
    <service android:name=".DownloadManageService" />
</application>
```

Fig. 5. Malware sample with no main activity

방송하면 이를 수신하여 실행되는 컴포넌트이며, 해당 인텐트 이벤트를 발생하는 것으로 브로드캐스트 리시버를 실행할 수 있다. 따라서 동적 분석 시, AndroidManifest.xml를 파싱하여 브로드캐스트 리시버에 등록된 이벤트를 모두 발생시켜 리시버를 실행할 수 있으며, 이에 따라 동적 분석의 코드 커버리지를 향상시킬 수 있다.

3.2 안드로이드 로깅 방해 대응 기법 설계

3.2.1 logcat 버퍼 크기를 이용한 로그 삭제 대응 기법 설계

logcat은 안드로이드에서 제공하는 기본 로깅 도구이다. logcat은 버퍼 크기의 최댓값이 존재하며, 기록된 로그의 양이 버퍼 크기를 초과하면 logcat에 기록된 로그가 삭제되는 문제점이 발생한다. 따라서 앱이 logcat에 기록된 로그를 삭제하기 위하여 무수히 많은 더미 로그를 logcat에 기록하는 기법을 시도할 수 있다. logcat이 가지는 이러한 버퍼 크기의 한계점을 악용한 분석 회피 기법에 대응하기 위하여, 본 논문에서는 실시간으로 logcat의 로그를 파일로 기록하는 시스템을 제안한다. 로그의 양이 버퍼 크기를 초과하면 logcat에 기록된 로그가 삭제되지만 파일에 저장한 로그는 삭제되지 않으므로 동적 분석 시 분석 기록을 실시간으로 파일에 저장하여 logcat의 버퍼 크기가 초과되는 문제점을 극복하고 로그를 저장하는 것이 가능하다.

3.2.2 명령어를 이용한 로그 삭제 대응 기법 설계

안드로이드 앱을 동적 분석하는 환경이 루팅 되었을 경우, 악성 앱은 Runtime.exec() 메소드로 "logcat -c" 명령을 실행할 수 있다. 이를 통해 앱이 logcat에 기록된 로그를 모두 삭제하여 분석 기록을 삭제하는 기법을 사용할 수 있다. 이러한 회피 기법에 대응하기 위하여, AOSP(Android Open Source Project)의 logcat 소스코드를 수정할 수 있다. logcat 명령어 옵션 중 로그를 삭제하는 -c 옵션에 해당하는 소스코드를 수정하여 로그 삭제를 수행하지 않도록 한다. 이후 수정된 logcat 소스코드를 빌드하여 logcat의 삭제를 수행하지 않으며 다른 모든 기능은 정상적으로 수행하는 logcat 바이너리를 이용할 수 있다. 수정된 logcat 바이너리는 logcat에 기록된 로그를 삭제하지 못하지만 다른 logcat 명령은 정상적으로 수행하여 앱의 동작을 방해하지 않는 시스템을 구현할 수 있다.

```
for i in self.xml:
    for item in self.xml[i].getElementsByTagName("activity-alias"):
        for sitem in item.getElementsByTagName("action"):
            val = sitem.getAttributeNS(NS_ANDROID_URI, "name")
            if val == "android.intent.action.MAIN":
                x.add(item.getAttributeNS(NS_ANDROID_URI, "targetActivity"))

        for sitem in item.getElementsByTagName("category"):
            val = sitem.getAttributeNS(NS_ANDROID_URI, "name")
            if val == "android.intent.category.LAUNCHER":
                y.add(item.getAttributeNS(NS_ANDROID_URI, "targetActivity"))

z = x.intersection(y)
```

Fig. 6. Extracting main activity from 〈activity-alias〉 element

```
def get_events(self):
    eventList = list()
    for i in self.xml:
        for item in self.xml[i].getElementsByTagName("receiver"):
            for actionItem in item.getElementsByTagName("action"):
                IntentEvent = actionItem.getAttributeNS(NS_ANDROID_URI,"name")
                eventList.append(IntentEvent)

    return eventList
```

Fig. 7. Event extraction from broadcast receiver

```
root@S4URC-Set01-Host23:~/apk_parse# python parsing_manifest.py c2c2e5e83a45d25cc0a56c5f16bf091f.apk
packageName com.net.bigger.flower.source.generally
main_activity com.dot.MainApplication$MainActivity
get_acrivities ['com.dot.MainApplication$MainActivity']
services ['com.dot.MainApplication$MainService']
receivers ['com.dot.MainApplication$MainReceiver']
events [u'android.intent.action.REBOOT', u'android.intent.action.ACTION_SHUTDOWN', u'android.intent.acti
on.QUICKBOOT_POWEROFF', u'android.intent.action.BOOT_COMPLETED', u'android.net.wifi.STATE_CHANGE', u'and
roid.net.wifi.WIFI_STATE_CHANGED', u'android.net.wifi.supplicant.CONNECTION_CHANGE', u'android.net.conn.
CONNECTIVITY_CHANGE']
```

Fig. 8. Components extraction result from malware sample

IV. 구 현

4.1 앱 컴포넌트 추출 및 앱 분석 절차

안드로이드 앱 내부에서 사용되는 컴포넌트는 AndroidManifest.xml 파일에서 정의하고 있으며, XML 형식을 따르고 있으므로 이를 파싱하여 앱의 컴포넌트를 추출할 수 있다. 〈activity〉 요소가 액티비티명을 정의하기 위하여 name 속성을 이용하는 것과 달리 〈activity-alias〉 요소는 실제 액티비티명을 targetActivity 속성으로 정의한다. 따라서 Fig.6.와 같이 〈activity-alias〉 요소의 targetActivity 속성으로부터 액티비티명을 획득하는 코드를 작성하여 앱에서 실제 실행되는 메인 액티비티의 이름을 추출할 수 있다.

앱의 서비스는 〈service〉 요소의 name 속성에서 서비스명을 추출할 수 있으며, 브로드캐스트 리시버에서 수신하는 이벤트는 Fig.7.과 같이 〈receiver〉 요소의 하위 요소인 〈action〉 요소에서 name 속성으로 정의된 이벤트명을 추출할 수 있다.

Fig.8.은 안드로이드 악성 앱에서 컴포넌트를 추출한 결과이다. 〈activity-alias〉로 메인 액티비티를 정의하는 기법을 적용하였으나 정상적으로 메인 액티비티를 추출할 수 있으며, 앱 내의 서비스와 브로드캐스트 리시버 및 브로드캐스트 리시버에서 수신하는 인텐트 이벤트의 목록을 추출할 수 있다. 이를 바탕으로 동적 분석 시, 앱의 컴포넌트를 실행하여 앱의 실행율을 증가시키고 코드 커버리지를 향상시킬 수 있다.

4.2 안드로이드 로깅 방해 기법 대응

logcat 의 소스코드 원본은 AOSP 소스코드에 포함되어 있다. 이 중 logcat.cpp 소스코드에서 logcat에 기록된 로그를 삭제하는 코드를 제거하면 삭제 기능만을 제공하지 않고 다른 기능은 정상적으로 수행하는 logcat을 사용할 수 있다. 이후 AOSP를 빌드하여 수정된 logcat 바이너리를 획득 할 수 있다. 안드로이드의 logcat은 /system/bin 디렉토리 내부 logcat 경로에 존재하며, 이를 수정된 logcat으로 대체할 수 있다. 본 논문의 분석 시스템은 동적 분석 플랫폼으로 C-Android에서 제공하는 안드로이드 컨테이너를 이용하기 때문에 logcat을 대체하는 것이 가능하다. 수정된 logcat 실행파일은 로그 삭제를 제외한 모든 기능을 정상적으로 수행할 수 있기 때문에 앱이 logcat을 사용하더라도 에러가 발생하지 않고 정상적으로 동작할 수 있다.

Fig.9.은 실제 악성 앱이 동적 분석 중 logcat의 로그를 삭제하기 위해 "logcat -c" 명령어를 수행했음을 나타낸다. 이와 같이 악성 앱이 로그 삭제를 시도할 수 있으나, 본 논문에서 제안한 시스템은 로그 삭제 기능을 제공하지 않는 수정한 logcat을 이용하

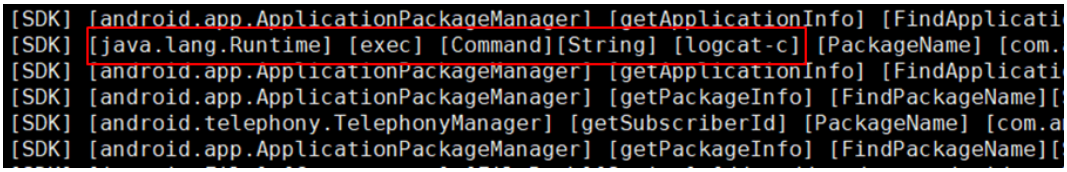


Fig. 9. Malware sample with deleting logcat buffer

였기 때문에 분석 로그는 삭제되지 않으면서 앱 또한 정상적으로 동작하여 분석을 수행할 수 있다는 장점을 지닌다.

또한, logcat의 버퍼 크기를 초과하도록 더미 로그를 무수히 많이 생성하여 로그를 삭제하는 앱에 대응하기 위하여 실시간 로깅 시스템을 구현한다. 동적 분석을 시작하기 전, 분석을 수행하는 프로세스와 함께 별도의 하위 프로세스를 생성하고, 분석 도중 logcat 에 기록되는 로그를 실시간으로 수집하여 파일에 기록하는 로깅 시스템을 구현한다. logcat의 버퍼 크기가 초과되어 logcat 내의 로그가 삭제되더라도 파일에서 기록된 분석 로그는 삭제되지 않는다는 장점을 지닌다. 이를 통하여 더미 코드를 무수히 많이 생성하는 악성 앱 또한 성공적으로 분석할 수 있는 로깅 시스템을 구현한다.

4.3 안드로이드 앱 동적 분석 시스템 설계

본 논문에서는 분석을 회피하기 위한 기법이 적용된 안드로이드 악성 앱의 분석률을 높이기 위한 동적 분석 기능을 향상시키는 기법을 제안한다.

C-Android에서 제공하는 안드로이드 컨테이너 플랫폼에 앞서 서술한 기법들을 적용하여 Fig.10.과 같이 분석을 수행하는 시스템을 설계 및 구현하였다.

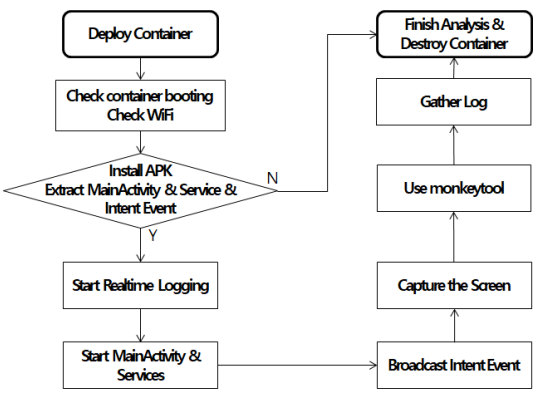


Fig. 10. Proposed system architecture

앱의 AndroidManifest.xml 파일에서 메인 액티비티, 서비스와 브로드캐스트 리시버 및 브로드캐스트 리시버에서 수신하는 이벤트를 추출한다. 앱을 실행하기 전 실시간으로 분석 로그를 기록하는 프로세스를 생성하며, 로그를 삭제하지 못하도록 수정된 logcat을 이용하여 분석 로그를 기록한다. 이후 앱에서 추출한 메인 액티비티를 실행한다. 메인 액티비티를 실행하면 앱의 프로세스가 생성되며, 앱의 메인 액티비티가 존재하지 않을 경우 서비스가 최초 실행될 때 앱 프로세스가 생성된다. 메인 액티비티를 실행한 후 코드 커버리지를 향상시키기 위해 앱에 포함된 모든 서비스들을 실행하며, 브로드캐스트 리시버에서 수신하는 모든 인텐트 이벤트를 실행한다. 이후 Monkey tool로 UI 이벤트를 랜덤으로 발생시켜 앱을 테스트한다. 동적 분석 과정이 모두 완료되면 기록된 동적 분석 로그를 수집하고 컨테이너를 중지 후 분석을 종료한다.

4.4 성능 분석

본 논문에서 제안한 분석 기능의 성능 향상을 확인하기 위해 악성 앱을 이용한 분석 실험을 진행하였다. 실험을 위하여 안드로이드 동적 분석 플랫폼인 AMAaaS[16]에서 제공한 안드로이드 악성 앱 1000개를 분석 샘플로 사용하였다. 실험 환경으로 C-Android[14]에서 제공하는 컨테이너를 동적 분석 플랫폼으로 이용하였다. 비교 대상은 제안 기법을 적용하지 않은 분석 결과이며, 동일한 분석 환경에서 〈activity〉 요소를 이용하여 정의된 메인 액티비티만 실행하여 분석을 수행하였다.

Table 2.과 같이 제안 기법을 적용하기 않았을 경우 악성 앱에 대한 분석 성공률은 70.9%인 반면, 제안 기법을 적용하였을 경우 분석 성공률은 89.6%로 19.3%만큼 향상되었다. 이는 메인 액티비티를 〈activity-alias〉 요소로 정의한 앱과, 앱 내에 메인 액티비티가 존재하지 않지만 서비스와 브로드캐스트 리시버로 구성된 앱에 대한 분석을 수행할 수 있

Table 2. Success rate for analyzing 1000 malware samples

	Total sample	Original C-Android	Applying our proposal
Activity-alias	171	16	171
No main activity	16	0	16
Can not install	104	16	0
No anti-analysis	709	709	709
Total	1000	709	896

기 때문에 성공률이 향상되었음을 나타낸다.

반면, 악성 앱 샘플 중에서 설치가 불가능한 앱은 AndroidManifest.xml의 형식이 올바르지 않은 경우, 앱 서명이 올바르게 서명되지 않은 경우, APK 파일 형식이 맞지 않는 경우 등 앱 자체가 가지는 문제점으로 인해 설치가 불가능한 경우이다. 이러한 앱은 실제 단말인 Nexus 5로 실험한 결과, 실제 단말에서도 마찬가지로 설치가 불가능함을 확인하였다.

V. 결 론

지능화된 안드로이드 악성 앱은 앱의 수명을 늘리기 위해 동적 분석을 회피하기 위한 다양한 기법을 사용하고 있다. 백그라운드 컴포넌트를 활용하여 사용자로부터 악성 행위를 숨기거나 앱에 포함된 컴포넌트 자체를 숨기는 기법, logcat에 기록된 로그를 삭제해 동적 분석을 방해하는 기법을 적용하고 있다. 분석을 회피하기 위한 기법이 적용된 앱은 기존 도구로 메인 액티비티와 같은 컴포넌트를 추출하지 못하며, 자동화된 스크립트로 앱을 실행하지 못하도록 방해하며, logcat의 로그를 삭제하여 동적 분석이 어렵다는 문제점을 가진다.

본 논문에서는 이러한 분석을 회피하기 위한 기법을 사용하는 안드로이드 악성 앱을 분석하기 위하여, 향상된 분석 기능을 갖는 동적 분석 시스템을 설계 및 구현하였다. AAPT와 같은 기존 도구로 제대로 추출할 수 없는 컴포넌트를 포함하여 앱에 정의된 컴포넌트를 모두 추출하며, 이를 활용하여 실제 동적 분석 시 앱의 분석 성공률과 코드 커버리지를 향상시켰다. 또한, 무수히 많은 더미 로그를 남겨 로그를 삭제하는 분석 회피 기법에 대응하여 실시간 로깅 시스템을 구현하였으며, 앱이 logcat의 로그를 삭제하지 못하도록 방지하는 기능을 통해 악성 행위를 숨기는 앱 또한 분석 로그를 정상적으로 수집할 수 있도록 구현하였다.

하지만 악성 앱이 특정 조건이 충족되어야만 악성 행위를 수행하는 로직밤 기법을 적용하여 동적 분석을 회피할 수 있으며, 본 논문에서 제안하는 기법은 이러한 로직밤에 대응할 수 없다는 한계점을 가진다. 따라서 동적 분석 기능을 보다 향상시키기 위해 로직밤에 대한 추가적인 대응 기법이 필요하다. 이를 위하여 앱에 대한 정적 분석을 통해 앱이 악성행위를 수행하기 위한 로직밤의 조건을 추출하고, 해당 조건을 충족할 수 있도록 분석 시스템 혹은 분석 환경을 고도화하는 연구가 필요할 것이다.

References

[1] StateCounter, “mobile market share” Mobile operating system market share worldwide, 2018. http://gs.statcounter. com/os-market-share/mobile/worldwide, 2018-12-05.

[2] IDC, “mobile os” Smartphone OS Market Share, https://www.idc.com/promo/smartphone-market-share/os, 2018-12-07.

[3] Tiwari, Suman R, “A survey of Android malware Detection Technique”, Journal of Network Communications and Emerging Technologies vol.8, no.4, 2018.

[4] McAfee, “mobile threat” McAfee Labs Threats Report, 2018-12-07.

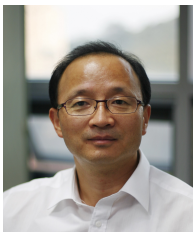
[5] Shan, Zhiyong, Iulian Neamtiu, and Raina Samuel, “Self-hiding behavior in Android apps: detection and characterization,” Proceedings of the 40th International Conference on Software

Engineering, ACM, 2018.
〔6〕 AndroidDevelopers, "android activity" https://developer.android.com/training/basics/activity-lifecycle/starting, 2018-12-12.
〔7〕 AAPT, "android parsing tool" https://developer.android.com/studio/command-line/aapt2, 2018-12-12.
〔8〕 Androguard, "android obfuscation" https://github.com/androguard/androguard, 2018-12-16.
〔9〕 Van Der Veen, Victor, Herbert Bos, and Christian Rossow, "Dynamic analysis of android malware," Internet & Web Technology Master thesis, VU University Amsterdam, August, 2013.
〔10〕 Kimberly Tam, Salahuddin J. Khan, Aristide Fattori, Lorenzo Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," NDSS, February, 2015.
〔11〕 Lantz, Patrik, "An android application sandbox for dynamic analysis," Master, lectrical and Information Technology, Lund university, Lund, Sweden, November, 2011.
〔12〕 Michael Spreitzenbarth, Thomas Schreck, Florian Echtler, Daniel Arp, Johannes Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," International Journal of Information Security, vol. 14, no.2, pp.141-153, April, 2015.
〔13〕 Chau, Ngoc-Tu, Souhwan Jung, "Dynamic analysis with Android container: Challenges and opportunities," Digital Investigation, 27, pp. 38-46, March, 2018.
〔14〕 William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," ACM Transactions on Computer Systems (TOCS), vol.32, no.2, June, 2014.
〔15〕 Linux Containers, "linux container" https://linuxcontainers.org/, 2019-05-18
〔16〕 AMAaaS, "amaaas.com" https://amaaas.com, 2019-05-19

〈저자소개〉

안 진 웅 (Jinung Ahn) 학생회원
2017년 2월: 숭실대학교 정보통신전자공학부
2017년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈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 8월: 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