논문 2021-1-6 http://dx.doi.org/10.29056/jsav.2021.06.06

모바일 게임용 안드로이드 에뮬레이터 탐지 기법
(Nox와 LD Player 탐지 기법 중심으로)

김남수*, 김성호*, 박민수**, 조성제**†

Detecting Android Emulators for Mobile Games
(Focusing on Detecting Nox and LD Player)

Nam-su Kim*, Seong-ho Kim*, Min-su Pack**, Seong-je Cho**†

요 약

많은 게임 앱이나 금융 앱들의 경우, 동적 역공학 공격을 방어하기 위해 에뮬레이터 탐지 기능을 탑재하고 있다. 그러나 기존 안드로이드 에뮬레이터 탐지 방법들은, 실제 기기와 유사해진 최신 모바일 게임용 에뮬레이터를 탐지하는데 한계가 있다. 이에 본 논문에서는 Houdini 모듈과 라이브러리 문자열 기반으로 모바일 게임용 에뮬레이터를 효과적으로 탐지하는 기법을 제안한다. 구체적으로, bionic의 libc 라이브러리에 포함된 특정 문자열, Houdini 관련된 시스템 콜 수행과정 분석과 메모리 매핑을 통해, 잘 알려진 Nox와 LD Player 에뮬레이터를 탐지하는 기법을 제시한다.

Abstract

Many game and financial apps have emulator detection functionality to defend against dynamic reverse engineering attacks. However, existing Android emulator detection methods have limitations in detecting the latest mobile game emulators that are similar to actual devices. Therefore, in this paper, we propose a method to effectively detect Android emulators for mobile games based on Houdini module and strings of a library. The proposed method detects the two emulators, Nox and LD Player through specific strings included in libc.so of bionic, and an analysis of the system call execution process and memory mapping associated with the Houdini module.

한글키워드 : 모바일 게임, 안드로이드, 에뮬레이터 탐지, Houdini 모듈

keywords : mobile game, Android, emulator detection, Houdini module

1. 서 론

안드로이드 에뮬레이터(Android emulator)는 데스크톱 PC에 안드로이드 환경을 구축하여 모바일 앱의 실행을 가능하게 해 준다. 흥미로운 모바일 게임을 하고 싶은데 해당 모바일 기기를 갖지 못한 사용자들, PC의 큰 스크린과 높은 해상도에서 모바일 게임을 하고 싶은 사용자들, 그

* 단국대학교 컴퓨터학과
** 단국대학교 소프트웨어학과
† 교신저자: 조성제(email: sjcho@dankook.ac.kr)
접수일자: 2021.06.04. 심사완료: 2021.06.19.
게재확정: 2021.06.20.

리고 키보드와 정확한 마우스 제어를 통해 더 효과적으로 모바일 게임을 즐기려는 사용자들로 인해, 데스크 톱 컴퓨터(PC)에서 모바일 게임을 하려는 요구가 점차 증가하고 있다. 이에 모바일 게임을 PC에서 플레이할 수 있게 하는 Nox, LD Player, Bluestacks, Memu 등의 안드로이드 에뮬레이터들이 주목받으면서 확산되고 있다.

이러한 에뮬레이터들이 사용자에게 편의를 제공하는 반면, 범죄에 악용되는 사례도 발생하고 있다. 개발자는 안드로이드 앱 개발 시 런타임 오류를 정정하기 위해 디버깅 목적으로 에뮬레이터를 활용한다[1]. 안드로이드 악성코드(멀웨어, malware) 분석가는 정적으로 분석이 어려운 멀웨어를 실제 실행하면서 동적으로 분석하기 위해 에뮬레이터를 사용한다[2, 3].

한편, 앱 공격자는 사유 앱(proprietary app)을 역공학하여 비즈니스 로직(business logic)이나 핵심 알고리즘을 도용하기 위해서 또는 취약점을 찾아 앱을 해킹하기 위해서 에뮬레이터를 악용하기도 한다.

스크린샷을 통해 전자책(eBook) 복제 및 유포, GPS와 기기명 변경을 통한 알리바이 조작, 인스턴스 복제를 이용한 다계정 생성 및 운영, 에뮬레이터의 원격 코드 실행 취약점을 공격하거나 가상화폐 채굴 악성코드 설치 등의 사례가 발생하였다[4].

안드로이드 에뮬레이터를 사용하여 게임 앱을 디버깅하거나 게임 조작을 자동화하기 용이하다[5]. 애니팡 게임의 경우 에뮬레이터에서 조작을 자동화하여 고득점을 올리는 프로그램이 개발되었으며, 에뮬레이터 기반으로 작성한 모바일 게임 자동화 프로그램을 판매하여 금전적인 수익을 얻으려는 사례도 발생하고 있다[5].

또한 에뮬레이터를 이용하여 다중 인스턴스와 매크로를 사용하여 자동으로 반복하여 게임 재화를 획득하는 작업인 “gold farming”을 안내하는 사이트[6]도 나타났으며, LD player를 사용하여 게임 봇을 실행하기도 한다. 에뮬레이터를 사용하여 16,000대의 실 기기(real device)를 모방하여, 실제 새로운 실 기기에서 해당 계정을 접속하는 고객인 것처럼 보이게 하여 현금을 갈취하는 사건도 있다[7]. 또한 APK(Android Package, 안드로이드 앱의 배포버전)가 변조될 경우, 실제 기기에서는 보안 기능이 적용되어 정상적으로 설치가 되지 않으나, LD player에서는 설치가 가능하고 실행될 수 있어 위험하다.

이에 금융 앱, 헬스케어 앱, 게임 앱들의 경우 동적 역공학 공격을 방어하기 위해서, 에뮬레이터를 탐지할 필요가 있다. 즉, 중요한 자산이나 민감한 개인정보를 관리하는 앱들의 경우에는 공격자에 의한 동적 역공학 공격을 방어하는 한 방법으로, 실행 환경이 에뮬레이터인 경우 실행을 중지할 필요가 있다. 게임 앱들의 경우에는, 에뮬레이터를 악용한 범죄를 방지하기 위해서도 에뮬레이터 탐지 기법을 적용할 필요가 있다.

본 논문에서는 잘 알려진 모바일 게임용 안드로이드 에뮬레이터인 LD Player, Nox의 특징을 분석한다. 두 에뮬레이터의 경우, AOSP x86을 기반으로 되어 있으며 ARM ABI (Application Binary Interface) 기반의 앱들을 실행하기 위해 Houdini라는 모듈이 사용된다[8, 9]. Houdini 모듈 기반으로 이들을 효과적으로 탐지하는 기법을 제안한다. 또한, 라이브러리의 문자열 기반으로 이들 에뮬레이터를 탐지하는 방법을 제안한다.

논문의 구성은 다음과 같다. 2장에서 안드로이드 에뮬레이터와 관련된 배경 지식을 설명하고, 3장에서는 에뮬레이터 탐지에 관련된 기존 연구들을 조사한다. 4장에서는 시스템 특성(system properties) 기반의 기존 탐지 기법의 한계에 대해 설명하고, 5장에서 Houdini 모듈과 라이브러리 문자열 기반 에뮬레이터 탐지 기법을 제안한다. 6장에서는 Houdini 모듈 기반 에뮬레이터 탐

지 기법의 한계점을 서술하고 7장에서 결론 및 향후 연구에 대해 기술한다.

2. 배경 지식

2.1 안드로이드 에뮬레이터

이 절에서는 안드로이드 기본 에뮬레이터인 AVD (Android Virtual Device), 그리고 모바일 게임용 안드로이드 에뮬레이터인 Nox, LD Player에 대해 기술한다.

AVD (Android Virtual Device)는 스마트폰이나 태블릿, Wear OS, Android TV, Automotive OS기기의 특성을 정의한다. Android Studio에서 인터페이스로 AVD를 만들고 관리하며 실제 기기의 기능들을 제공하며 전화 및 SMS 수신, 여러 네트워크 속도, 회전 및 기타 하드웨어 센서를 시뮬레이션하고 기기의 위치를 지정하며 Google Play 스토어에 액세스하는 등 다양한 작업을 할 수 있다. 또한 AVD는 그래픽 가속 및 GPU를 소프트웨어에서 하드웨어로 선택하여 에뮬레이션하여 성능을 대폭 향상시킬 수 있다. 가장 기본적인 에뮬레이터며 개발자의 디버깅 용도로 사용된다[1]. 안드로이드 에뮬레이터를 탐지하는 여러 기존 연구들[2, 10, 11]에서는 AVD나 일반 에뮬레이터를 대상으로 하였고, 모바일 게임용 에뮬레이터를 대상으로 한 탐지 연구는 거의 수행된 적이 없다.

Nox는 중국의 MoreTech에서 개발된 게임용 에뮬레이터로 고성능과 높은 호환성에 중점을 두고 있다. 게임패드, 컨트롤러, 키보드 컨트롤과 같은 주요기능들과 복잡한 작업을 기록하기 위한 기본 매크로 레코더가 함께 제공된다. 다중 인스턴스를 제공하여 한 번에 여러 게임을 즐길 수 있으며 Oracle VirtualBox (VMDK)를 사용하며 x86과 AMD 구조와 호환된다[12].

LD Player는 1억 회 이상 다운로드된 게임용 인기 에뮬레이터로 중국의 Xuanzhi에서 개발하였다. 경량 에뮬레이터로 일반 게임들을 수행할 수 있게 해 준다. APK가 변조되어 서명값(META-INF)의 무결성이 손상된 경우에도, 해당 애플리케이션을 수행할 수 있게 해주는 특징을 갖고 있다[13].

2.2 AOSP x86을 지원하는 Houdini 모듈

다양한 안드로이드 기기는 다른 CPU를 사용하므로 서로 다른 명령 집합을 지원한다. CPU와 명령 집합의 각 조합에는 고유한 ABI (Application Binary Interface)가 있으며, ABI는 플랫폼에서 지원하는 네이티브 API(native API)를 참조할 수 있다. 지원되는 ABI는 armeabi-v7a, arm64-v8a, x86, x86_64 가 있다.

일반 실 기기는 주로 ARM ABI을 지원하고 있으며 대부분 x86 ABI를 지원하지 않고 있다. x86 ABI는 AOSP(Android Open Source Project)를 x86 환경으로 이식하여 개발된 프로젝트 AOSP x86에 지원되는 명령어 셋 이다[14].

많은 안드로이드 게임 앱들은 네이티브 라이브러리에 의존하고 있으며 x86 ABI와 ARM ABI를 선택적으로 지원할 수 있다. AOSP x86의 경우에는, x86 ABI로만 실행할 수 있는 환경을 제공하고 있어, ARM ABI만 지원하는 게임 앱의 경우에는 AOSP x86 환경에서 정상적으로 구동될 수가 없었다. Yang 등[15]에 따르면 연구 대상 게임 앱의 27.4%만이 x86 ABI를 지원하였고, 나머지 72.6%의 앱은 ARM ABI만 지원하였다. 이에 ARM ABI만 지원하는 앱들을 AOSP x86 환경에서 구동되게 할 필요가 있다.

이를 위해 Google에서 AOSP x86 5.1부터, Houdini 라이브러리 모듈을 탑재하여 ARM ABI 기반의 앱을 Android VM 라이브러리인 libart.so의 내부에서 Houdini 모듈을 통해 x86 ABI로 변

환하여 구동할 수 있는 환경을 제공하고 있다[8, 9, 16]. Houdini 모듈을 탑재한 x86 에뮬레이터는 ARM 기반의 에뮬레이터보다 빠른 실행환경을 제공할 수 있기 때문에, AOSP x86 기반의 여러 최신 모바일 게임용 안드로이드 에뮬레이터들도 Houdini 모듈 기반으로 동작하고 있다[15]. Nox와 LD Player도 Houdini 모듈 기반으로 동작한다.

3. 관련 연구

Petsas 등[10]은 정적 분석 정보, 동적 센서 정보, 가상 머신 관련 복잡도를 사용하여 에뮬레이터를 탐지하였고, Vidas 등[17]은 에뮬레이터 탐지 범위를, 동작, 성능, 하드웨어 및 소프트웨어 구성요소, 시스템 설계로 지정하였다.

Lin 등[18]은 에뮬레이터를 구성하는 OS, 하이퍼바이저(hypervisor), 하드웨어 계층으로 나누어, 각 계층의 특성을 분석하고 이 특성 기반으로 에뮬레이터를 탐지하였다. 또한, 에너지 효율을 측정하여 에뮬레이터를 탐지하는 방법을 제시하였다.

Gajrani 등[2]은 에뮬레이터를 이용하여 멀웨어를 분석하였다. 기존 에뮬레이터(AVD)의 바이너리를 변경하여 멀웨어를 효율적으로 동적 분석하는 방법을 제안하였다.

Jang 등[19]에 의하면, 광고를 시청하고 현금화할 수 있는 앱을 대상으로, 에뮬레이터를 사용하여 여러 대의 인스턴스(instance) 환경을 쉽게 구성하고 광고비용을 챙기는 Mobile ad fraud 또는 Mobile Click Fraud Attack(MCFA)을 시행할 수 있음을 보였다. 그리고 x86과 ARM 환경 등을 고려하지 않고 소프트웨어와 하드웨어적 차이를 중심으로 에뮬레이터를 탐지하는 기법을 제시하였다

Sahin 등[11]은 ARM 에뮬레이터(AVD)를 기준으로 실제 기기에서 수행되는 ARM 명령어 체계의 차이점을 분석하여 에뮬레이터 탐지 성능을 개선하였다.

Jing 등[20]은 실제 기기와 에뮬레이터의 차이를 자동으로 추출하여 아티팩트(artifact)라고 하는 API, 파일, 하위 문자열 모음집을 생성하는 프레임워크를 제시하였고, Yoon 등[21]은 다양한 API와 x86 CPU 기반의 에뮬레이터 탐지기법을 제시하였다.

이러한 기존 연구들과 달리, 본 논문에서는 Houdini 모듈을 기반으로 새로운 방식의 모바일 게임용 안드로이드 에뮬레이터 탐지 기법을 제안한다.

4. API를 이용한 안드로이드 에뮬레이터 탐지 기법

이 절에서는 API (Application Programming Interface)를 이용한 에뮬레이터 탐지 기법의 한계에 대해 분석한다. 기존 논문들[5, 17, 22]의 경우, 안드로이드 OS.Build 또는 Telephony Manager 등의 안드로이드 API를 사용하여 에뮬레이터를 탐지하였다.

그러나 최신 모바일 게임용 안드로이드 에뮬레이터의 경우, 실 기기(예: 샤오미 폰, 갤럭시 폰)를 선정하고 이를 에뮬레이터 Build 정보로 재정의하여 기존 에뮬레이터 탐지 기법을 회피하고 있다. 이러한 방법은, 기존 AVD를 탐지하는데 사용되었던 에뮬레이터 문자열 특징 정보들(예로 Build 정보)을 편집하여, 정교한 멀웨어가 에뮬레이터를 탐지하지 못 하게 하는 기법[2]과 유사하다.

본 연구진은, API를 이용한 기존 에뮬레이터 탐지 기법들이 Nox, LD Player, BlueStacks를

탐지하는데 한계가 있음을 확인하였다[23]. 연구 결과가 표 1에 요약되어 있다. 표 1은 기존 연구[17]에서 에뮬레이터 탐지에 사용된 특징들(표의 두 번째 칸)과 현재 Nox와 LD Player의 특징 정보를 대조하였다. LD Player에서는 기존과 같이 에뮬레이터를 특정할 수 있는 정보가 일부 발견되었지만, Nox의 경우에는 해당 정보가 없었다.

표 1. 이전 에뮬레이터 탐지 정보와 현재 게이밍 에뮬레이터의 특징 정보 비교
Table 1. Comparison of previous emulator detection information and current gaming emulator information

System Properties	Detection feature	Nox (pixel2)	LD Player (pixel2)
BOARD	unknown	universal 8890	Google
MODEL	sdk	Google pixel 2	Google pixel 2
PRODUCT	sdk	Google pixel 2	Google pixel 2
HOST	android-test	SWHE7705	ubuntu
MANIFACTURE	unknown	Google	google
DEVICE	generic	x86	aosp
HARDWARE	goldfish	samsung exynos8890	Google
TAGS	test-key	release-key	release-key
user mode	ro.secure=1 ro.debuggable=0	ro.secure=1 ro.debuggable=0	
root mode	ro.secure=1 ro.debuggable=1	ro.secure=1 ro.debuggable=0	ro.secure=1 ro.debuggable=1
getNETWORKCountryIso	N/A	KR	KR
getNetworkOperatorName		SKTelecom	KT Freetel Co.Ltd.

또한 [5]에서는 게임용 에뮬레이터의 시스템 특성(System Properties)을 열거하고, 이 특성 기반으로 BlueStacks, GenyMotion, Andy 등을 탐지하였고, Nox와 LD Player를 포함하지 않았다. 또한, 게임용 에뮬레이터가 실 기기 정보를 반영하여 계속 진화하는 상황에 대해서는 다루지 않았다.

5. 에뮬레이터 탐지 기법

이 절에서는 잘 알려진 모바일 게임용 안드로이드 에뮬레이터인 LD player와 Nox를, 라이브러리 문자열 기반으로, 그리고 Houdini 모듈 기반으로 탐지하는 기법을 제안한다.

5.1 라이브러리 문자열 기반의 탐지

Bionic은 안드로이드를 위한 표준 C 라이브러리로 libc, libdl, libm, libpthread로 구성된다. 모바일 게임 에뮬레이터에서 사용되는 Bionic은 libc.so를 각 에뮬레이터마다 커스텀(custom)하여 서비스하고 있어, libc.so를 분석하여 특정 에뮬레이터를 식별할 수 있다.

Nox의 libc.so에 'Nox'라는 문자열과, 'Vphone', 'houdini' 등의 문자열이 포함되어 있어 이를 기반으로 Nox를 탐지할 수 있다 (그림 1 참조).

```
aComEpicgames   db 'com.epicgames.',0   ; DATA XREF: access+62↑o
aLibhoudiniSo   db 'libhoudini.so',0    ; DATA XREF: access+78↑o
aComBignox      db 'com.bignox',0       ; DATA XREF: access:loc_2631E↑o
aComVphone      db 'com.vphone',0       ; DATA XREF: access+A8↑o
aDataDataComBig db '/data/data/com.bignox',0
                                        ; DATA XREF: access+BE↑o
aDataDataComVph db '/data/data/com.vphone',0
                                        ; DATA XREF: access+D4↑o
aSdcardAndroidD db '/sdcard/Android/data/com.bignox',0
                                        ; DATA XREF: access+EA↑o
aSdcardBignox   db '/sdcard/BigNox',0   ; DATA XREF: access:loc_2639A↑o
aNox            db '/Nox',0             ; DATA XREF: access:loc_263B9↑o
aComRacoondigiJ db 'com.racoondigi.jqdj',0
                                        ; DATA XREF: access+13F↑o
aAppRdx         db '/app_rdx',0         ; DATA XREF: access+173↑o
aDevLogMain     db '/dev/log/main',0    ; DATA XREF: __openat:loc_264BE↑o
aLibwebviewchro db 'libwebviewchromium.so',0
                                        ; DATA XREF: __openat+87↑o
aSystemAppWebvi db '/system/app/webview/webview.apk',0
                                        ; DATA XREF: __openat:loc_26526↑o
aProcVersion    db '/proc/version',0    ; DATA XREF: __openat+EF↑o
aSystemEtcVersi db '/system/etc/.version',0
                                        ; DATA XREF: __openat+108↑o
aSystemEtc      db '/system/etc/.',0    ; DATA XREF:  openat+187↑o
```

그림 1. Nox의 system/lib/libc.so
Fig 1. system/lib/libc.so in Nox

LD Player의 libc.so에는 'tencent'와 'ld', 'vbox' 등의 문자열을 확인할 수 있으며, 루팅(rooting)에 사용되고 있는 su 파일 등을 확인할 수 있다(그림 2 참조). 이를 기반으로 LD Player를 탐지할 수 있다.

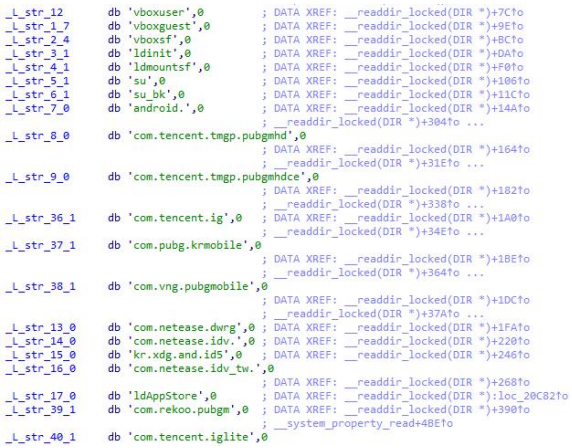

그림 2. LDplyaer의 system/lib/libc.so
Fig 2. system/lib/libc.so in LDplayer

5.2 시스템 콜 기반의 탐지

x86 상에 구축된 에뮬레이터들은 성능을 향상시키기 위해 시스템 API 함수들을 x86 명령들로 구성한다. 한편, ARM ABI 기반의 앱이 x86 에뮬레이터(Nox, LD Player)에서 동작할 경우 x86 명령을 직접 호출하지 못 하고, 시스템 콜(system call)을 통해 함수를 호출한다. 즉, 에뮬레이터에서 dlsym, dlopen과 같은 API 일부는 시스템 콜을 통해 접근된다. 이 경우 특정 API의 내부 명령이 시스템 콜인 경우 에뮬레이터로 판단할 수 있다.

부연 설명하면, ARM ABI 기반의 앱은 Houdini 모듈을 통해 system/lib/arm/nb 디렉토리에 있는 라이브러리를 사용하게 된다. libdl.so에 export되는 시스템 API(예: dlopen, dlsym, dlcose, dlerror) 사용될 경우 성능 상에 이유로 SVC(supervisor call)을 통해 x86 환경으로 전환하게 되며, 이 때 SVC 명령들(ARM 시스템 콜)은 0xef로 시작하게 된다(그림 3 참조). 그러므로 이러한 명령들을 기반으로 에뮬레이터를 탐지할 수 있다.

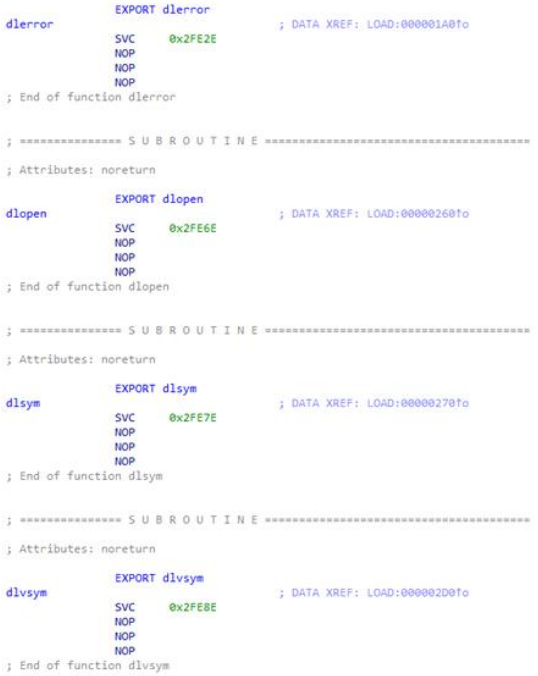

그림 3. system/lib/arm/nb/libdl.so의 시스템 API 함수
Fig 3. System API functions in system/lib/arm/nb/libdl.so

5.3 Houdini 관련 메모리 매핑 검사

앱 구동 시, proc/self/maps를 확인하면 메모리에 맵핑된 메모리 주소 공간을 확인할 수 있다. 이 때 Houdini 모듈이 실행될 경우, libhoudini.so가 매핑된 것을 확인할 수 있다. 그러므로 이러한 문자열을 확인하여, 에뮬레이터를 탐지할 수 있다(그림 4 참조).

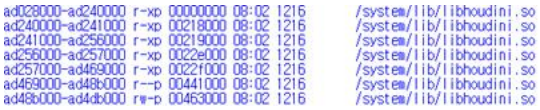

그림 4. LD Player 4의 /proc/self/maps 파일에서 libhoudini.so 확인

Fig 4. Checking libhoudini.so in /proc/self/maps of LD Player 4

그 외에도 메모리에 맵핑된 라이브러리들의 경우에는 경로가 나타나 있으므로 해당 라이브러리의 ELF Header에서 CPU의 아키텍처가 ARM인지 x86인지 확인할 수 있다(그림 6 참조). libc.so가 메모리에 맵핑되어 있을 때, system/lib/arm/nb/libc.so의 경우에는 ABI가 ARM이며, /system/lib/libc.so의 경우에는 ABI가 x86이다(그림 5와 그림 6 참조).

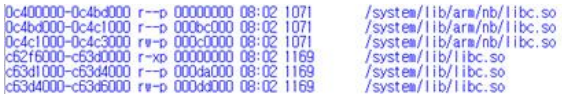

그림 5. LDPlyaer 4의 /proc/self/maps에서 .so 파일의 경로 확인

Fig 5. Checking the path of .so files in /proc/self/maps of LDPlyaer 4

그림 6. 로드된 ABI 정보 확인

Fig 6. Checking the loaded ABI information

6. 논 의

6.1 Houdini 모듈 기반 탐지의 한계

Houdini 모듈 기반 에뮬레이터 탐지는 앱에서 x86 기반의 ABI를 지원하지 않는다는 전제하에 유효하다. 즉, 제안 기법은 x86 기반의 실제 안드로이드 기기에는 적용할 수 없다. 일반적이지 않지만 과거에는 x86 기반의 실제 안드로이드 기기가 있어 houdini 모듈을 포함하였다. 하지만 이 기기들의 경우, 안드로이드 OS Lollipop (5.1) 이후로 공식적으로 출시되지 않고 있다. 그러므로 본 논문의 제안 기법을 안전하게 적용하기 위해, OS 버전을 5.1보다 높은 API 레벨을 min SDK로 지정하는 것을 권한다. 2021년 03월 08일 기준으로 실 기기에 테스트할 수 있는 아마존 서비스인 deviceFarm의 경우에, x86 기반의 디바이스는 Dell Venue 8 7840만 존재하고 있으며, OS 5.1 버전 까지만 존재하는 것으로 확인되었다 [21].

6.2 ARM ABI 앱의 동적 분석

최신 에뮬레이터는 앱을 동적으로 분석할 수 있는 환경을 제공한다. 에뮬레이터는 일반 모드(normal mode)에서 루트 모드(root mode)로 전환이 가능하고, 동적 인스트루멘테이션(dynamic instrumentation) 도구인 Frida를 원활하게 사용할 수 있게 지원한다. x86 ABI를 지원하지 않고 ARM ABI만을 지원하는 앱의 경우에 Houdini 모듈이 동작한다. Nox와 LD Player에서는 x86용 Frida 서버만 실행 가능하므로 ARM 명령을 정상적으로 hooking할 수 없어, x86 Frida 서버를 실행하는 에뮬레이터에서는 ARM ABI에 대한 동적 분석이 불가능하다. 따라서 ARM ABI만을 지원하는 앱이 Nox나 LD Player에서 수행될 경우, 해당 앱에 대한 동적 분석 공격을 어느 정도 완화할 수 있다.

7. 결론 및 향후 연구

기존의 시스템 특성 기반 안드로이드 에뮬레이터 탐지 기법들은 최신 모바일 게임용 안드로이드 에뮬레이터를 탐지하지 못 한다. 이에 본 논문에서는 Houdini 모듈과 라이브러리 문자열

기반으로 최신 모바일 게임용 안드로이드 에뮬레이터를 탐지하는 방법을 제시하였다. 즉, libc 라이브러리에 포함된 특정 문자열 정보, Houdini 모듈과 관련된 특정 API의 내부 명령이 시스템 콜인지의 여부, 메모리 매핑된 라이브러리 정보 등을 활용하여 Nox와 LD Player를 탐지하는 방법을 제안하였다.

향후, 모바일 게임용 안드로이드 에뮬레이터를 효과적으로 탐지할 수 있는 추가적인 특징정보에 대해 조사하고 적용할 계획이다. 또한, Nox와 LD Player 외의 다른 에뮬레이터들도 탐지하는 연구를 진행할 예정이다.

Acknowledgement

이 연구는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(no. 2021R1A2C2012574). 또한 과학기술정보통신부 및 정보통신기획평가원의 2021년도 SW중심대학사업의 결과로 수행되었음(2017-0-00091)

참 고 문 헌

[1] android developer (AVD User guide) https://developer.android.com/studio/run/managing-avds

[2] Jyoti Gajrani, Jitendra Sarswat, Meenakshi Tripathi, Vijay Laxmi, M. S. Gaur, and Mauro Conti, “A robust dynamic analysis system preventing SandBox detection by Android malware”, Proceedings of the 8th International Conference on Security of Information and Networks, 2015. DOI: https://doi.org/10.1145/2799979.2800004

[3] Android Developers Blog (Combating Potentially Harmful Applications with Machine Learning at Google: Datasets and Models) https://android-developers.googleblog.com/2018/11/combating-potentially-harmful.html

[4] Woohwan Nam, “Android Emulators Forensic Analysis Technique”, Journal of Digital Forensics, vol. 13, no. 4, pp. 303-316, 2019. DOI: 10.22798/kdfs.2019.13.4.303

[5] Jongseong Yoon, and Sangjin Lee, “A Study on android emulator detection for mobile game security”, Journal of the Korea Institute of Information Security & Cryptology 25(5), pp. 1067-1075, Oct. 2015. DOI: https://doi.org/10.13089/JKIISC.2015.25.5.1067

[6] gamebots.run (Mobile Game cheats : Game Bot News) https://gamebots.run/news/Use-Summoners-War-Auto-Farm-Bot-on-PC-1794

[7] arstchnica (“Evil mobile emulator farms” used to steal millions from US and EU banks) https://arstechnica.com/information-technology/2020/12/evil-mobile-emulator-farms-used-to-steal-millions-from-us-and-eu-banks/

[8] Ye Roger, “Android System Programming: Porting, customizing, and debugging Android HAL”, ISBN-13: 978-1787125360, ISBN-10: 178712536X Packt Publishing.

[9] github (Houdini module) https://github.com/Rprop/libhoudini

[10] Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis, “Rage against the virtual machine: Hindering dynamic analysis of android malware”, In Proceedings of the Seventh European Workshop on System Security, pp. 5:1 - 5:6, Apr. 2014. ACM. DOI : https://doi.org/10.1145/2592791.2592796

[11] Onur Sahin, Ayse K. Coskun, and Manuel Egele, “Proteus: Detecting Android

Emulators from Instruction-Level Profiles", Research in Attacks, Intrusions, and Defenses, pp. 3-24, Sept. 2018. DOI : https://doi.org/10.1007/978-3-030-00470-5_1

[12] Nox App Player Download (2016). Nox App Player Download for Windows PC, Mac, Laptop Retrieved May. 15, 2016. from http://noxappplayer.com/

[13] A highly-customizable free emulator (LDPlayer), https://ld-player.en.softonic.com/

[14] android-x86 (android-x86 download) https://www.android-x86.org/

[15] Qifan Yang, Zhenhua Li, Yunhao Liu, Hai Long, Yuanchao Huang, Jiaming He, Tianyin Xu, and Ennan Zhai, "Mobile Gaming on Personal Computers with Direct Android Emulation", Proceedings of The 25th Annual International Conference on Mobile Computing and Networking, pp.1-15 Aug. 2019. DOI : https://doi.org/10.1145/3300061.3300122

[16] Min Choi, Seung Ho Lim, "x86 Android performance improvement for x86 smart mobile devices", Concurrency and Computation: Practice and Experience, 28.10, pp.2770-2780, Jul. 2016. DOI : https://doi.org/10.1002/cpe.3189

[17] Timothy Vidas, Nicolas Christin, "Evading android runtime analysis via sandbox detection", Proceedings of the 9th ACM symposium on Information, computer and communications security, pp.1-6, Jun. 2014. DOI: https://doi.org/10.1145/2590296.2590325

[18] Jie Lin, Chuanyi Liu, Binxing Fang. "Out-of-Domain Characteristic Based Hierarchical Emulator Detection for Mobile", Proceedings of the 2nd International Conference on Information Technologies and Electrical Engineering, pp.1-5, Dec. 2019. DOI : https://doi.org/10.1145/3386415.3387091

[19] Daehee Jang, Yunjong Jeong, Sungman Lee, Minjoon Park, Kuenhwan Kwak, Donguk Kim, Brent Byunghoon Kang, "Rethinking anti-emulation techniques for large-scale software deployment", computers & security 83, pp.182-200, Jun. 2019. DOI: https://doi.org/10.1016/j.cose.2019.02.005

[20] Yiming Jing, Ziming Zhao, Gail-Joon Ahn, Hongxin Hu, "Morpheus: automatically generating heuristics to detect android emulators", Proceedings of the 30th Annual Computer Security Applications Conference, pp. 216-225, Dec. 2014. DOI : https://doi.org/10.1145/2664243.2664250

[21] Amazon Web Services (AWS Device Farm device list) https://awsdevicefarm.info/

[22] Sora Lee, Hyoungshick Kim, "Android Emulator Detection for Evading Dynamic Analysis", Proceedings of the Korean Information Science Society Conference, pp.846-848, Dec. 2015.

[23] Namsu Kim, Hanseul Choi, and Seong-je Cho, "Detecting Android Emulators based on API calls: Overview and Research Trends", Proceedings of the Spring Annual Conference of Korea Institute of Next Generation Computing, Aug. 2020.

저 자 소 개

김남수(Nam-Su Kim)

2020.8 단국대학교 소프트웨어학과 졸업
2020.9-현재 : 단국대학교 컴퓨터학과 석사 과정
<주관심분야> 시스템 보안, 안드로이드 보안

박민수(Min-su Pack)

2020.03-현재 : 단국대학교 소프트웨어학과 학사 과정
<주관심분야> 시스템 보안, 안드로이드 보안

김성호(Seong-ho Kim)

2010.2 명지대학교 통신공학과 졸업
2013.08 한양대학교 전자컴퓨터통신 공학석사
2020.09-현재 : 단국대학교 컴퓨터학과 박사 과정
<주관심분야> 바이너리 코드 분석, 악성 코드 분석

조성제(Seongl-Je Cho)

1989.2 서울대학교 컴퓨터공학과 졸업
1991.2 서울대학교 컴퓨터공학과 공학석사
1996.8 서울대학교 컴퓨터공학과 공학박사
2009.2-2010.2 Univ. of Cincinnati 방문교수
1997.3-현재 : 단국대학교 컴퓨터학과/소프트웨어학과 교수
<주관심분야> 소프트웨어 지적 재산 보로, 시스템 및 모바일 보안, 악성코드 분석, 시스템소프트웨어