Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.3, Jun. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.3.549

안드로이드 뱅킹 어플리케이션 내 중간언어 분석을 통한 보안 검사 지점 우회 취약점 연구*

이 우 진,† 이 경 호‡
고려대학교 정보보호대학원

A Study on the Vulnerability of Using Intermediate Language in Android: Bypassing Security Check Point in Android-Based Banking Applications*

Woojin Lee,† Kyungho Lee‡
Graduate School of Information Security, Korea University

요 약

최근 모바일뱅킹 사용의 일상화와 더불어 은행권에서 모바일뱅킹의 사용 비중이 더욱 커져감에 따라 보안 위협도 증가하고 있다. 이에 국내 금융권에서는 뱅킹 어플리케이션 내에 보안 솔루션들을 도입하고, 상시 실행되는지 여부를 확인하기 위하여 보안 검사 지점들을 설정하여 어플리케이션의 안정성을 보장하고 있다. 본 논문에서는 국내 주요 은행들의 안드로이드 뱅킹 어플리케이션 디컴파일(decompile)시 추출되는 중간언어를 정적 및 동적 분석하여 보안 검사 지점들을 우회하는 모바일 백신 프로그램 미작동의 취약점을 보인다. 또한, 결과를 통해 이를 악용할 수 있는 공격을 알아보고 대응 방안을 제시한다.

ABSTRACT

In recent years, as the proportion of mobile banking has become bigger with daily usage of mobile banking, security threats are also increasing according to the feeling. Accordingly, the domestic banking system introduces security solution programs in the banking application and sets security check points to ensure the stability of the application in order to check whether it is always executed. This study presents a vulnerability of inactivity bypassing mobile vaccine program operation checkpoints using the intermediate language statically and dynamically analysis when decompiling the android banking applications of major banks in Korea. Also, through the results, it identifies possible attacks that can be exploited and suggest countermeasures.

Keywords: Mobile banking, Bypassing vulnerability, Android mobile vaccine program

I. 서 론

최근 금융권의 대표적인 전자금융거래 서비스로는 스마트폰을 기반으로 한 모바일뱅킹을 꼽을 수 있다. 한국은행에 따르면, 2016년 8월말 국내 모바일뱅킹 서비스 이용자가 7000만 명을 돌파하는 등 큰 규모를 가지고 있기에 금융 보안의 중요성은 더욱더 대두되고 있다[1]. 특히 모바일뱅킹을 대상으로 하는 대부분의 공격은 안드로이드 기반 스마트폰을 대상으로 이루어지며[2], 국내 스마트폰에 대한 안드로이드 사용자 점유율은 85.8%를 차지하고 있어 그 심각성은 점점 높아지고 있다[3]. Kaspersky Lab의 보고서에 따르면, Fig. 1.과 같이 안드로이드 모바일 뱅킹 어플리케이션을 대상으로 하는 악성코드 수는 2015

Received(03. 14. 2017), Modified(05. 19. 2017), Accepted(05. 20. 2017)
* 본 연구는 2016년도 동계학술대회에 발표한 우수논문을 개선 및 확장한 것임
* 본 연구는 미래창조과학부 및 한국인터넷진흥원의 "고용계약형 정보보호 석사과정 지원사업"의 연구결과로 수행되었음 (과제번호 H2101-17-1001)
† 주저자, kongsclub@korea.ac.kr
‡ 교신저자, kevinlee@korea.ac.kr(Corresponding author)

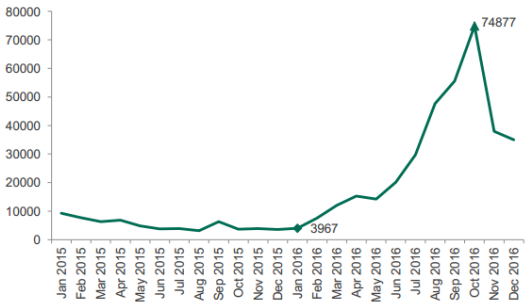

Fig. 1. The change in the number of users attacked with Android banking malware 2015-2016

년에 비하여 2016년에는 그 수가 폭발적으로 증가하고 있다[4]. 이에 금융사들은 이러한 공격들에 맞서 스마트폰 전자금융거래에 대한 안정성을 보장하기 위해 다양한 보안 솔루션 등을 도입하고 있다[5].

안드로이드 운영체제는 이식성이 높은 자바 언어로 구현되어 확장 및 응용에 있어 편리함이 있으나, 그만큼 역공학(Reverse Engineering)이 쉽다. 이를 이용하여 어플리케이션을 분석 및 위변조한 뒤, 악성코드를 삽입한다. 이는 자가 서명으로 재서명 및 빌드가 가능하기 때문에 리패키징(Repackaging) 공격이 쉽게 가능하다[6]. 또한, 어플리케이션 배포에 있어 개방적인 정책을 가진 구글 플레이스토어의 특성상, 통신사와 같은 제 3의 배포자가 어플리케이션을 배포할 수 있는 스토어가 활성화 되어 있기 때문에 위에서 언급한 정식 어플리케이션과 유사한 악성 어플리케이션의 배포가 쉽다[7].

본 논문에서는 안드로이드 뱅킹 어플리케이션에 적용된 모바일 백신 프로그램의 작동을 우회 할 수 있음을 보인다. 안드로이드 어플리케이션 디컴파일(decompile)시 추출되는 중간언어인 smali 코드 분석을 통해 모바일 백신 작동 지점들을 찾아 코드를 수정함으로써 작동을 우회한다. 이러한 취약점을 이용하여 뱅킹 어플리케이션 내 피싱(Phishing) UI 등을 통한 금융정보 탈취 악성 코드를 삽입하여 리패키징(Repackaging) 후 재배포한다면, 이는 심각한 피해가 될 수 있음을 알아본다. 또한, 본 논문에서의 분석 방법으로 모바일 백신 프로그램 뿐 만 아니라 루팅 폰 검사 등의 우회 가능함을 이용하여, 악용될 경우 모바일 뱅킹 보안에 큰 위협을 줄 수 있음을 알아본다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 연구를 분석한 뒤, 3장에서 안드로이드 취약점 분석을 위한 배경지식을 설명한다. 4장에서는 보안 검사 우회 취약점에 대한 실증적인 실험과 결과를 보인다. 5장에서는 이에 대한 대응방안에 대하여 알아보고, 6장에서는 결론을 통해 본 논문의 정리와 함께 향후 연구에 대하여 설명하고자 한다.

II. 관련연구

본 장에서는 안드로이드 뱅킹 어플리케이션의 보안과 관련된 선행 연구들을 알아본다. 스마트폰 뱅킹의 시장과 규모가 커져감에 따라 보안에 대한 중요성은 매우 중요하게 인식되어 왔고, 이에 관한 연구도 지속적으로 활발하게 진행되어 왔다.

안드로이드 운영체제를 대상으로 악성 행위를 막기 위한 백신 프로그램 관련 연구는 오랜 시간 진전되어왔다. 특정 악성코드에 대하여 백신 프로그램이 검출하는 성능을 비교하는 연구부터 악성코드를 머신러닝을 통해 분류하고 이를 적용하는 연구까지 다양한 방면으로 연구가 진행되고 있다[8][9].

이러한 안드로이드 운영체제 중에서 특히 뱅킹 어플리케이션을 대상으로 하는 연구가 시간이 지나며 점차 체계화되어 가고 있다. 이를 기술적인 측면에서 바라보면, 모바일 뱅킹 어플리케이션을 대상으로 보안 구간마다 위협을 도출하고 실험을 통해 체계적인 보안 검사를 진행한 연구가 진행되었다[10]. 또한, 관리적인 측면에서는 중요 정보 흐름의 관리를 위한 강화된 ISMS 모델을 스마트폰에 특화하여 적용하는 연구도 제시되었다[11].

한편 국내에서도 안드로이드 뱅킹 어플리케이션에 대한 연구가 계속되고 있다. 국내 대표적인 안드로이드 뱅킹 어플리케이션들의 무결성 검증 취약점 연구로, 쉽게 리패키징되어 악용될 수 있는 무결성 검증 우회에 대한 취약점을 보이고 대응방안을 제시함으로써 국내 안드로이드 뱅킹 보안 발전에 기여를 해왔다[12]. 최근에도 이러한 금융권 어플리케이션을 대상으로 다양하고 심층적인 연구는 계속 진행되고 있다[13].

III. 배경지식

본 장에서는 안드로이드 뱅킹 어플리케이션 내 모바일 백신 프로그램 우회 취약점을 분석하기 위한 기본적인 배경 지식을 알아본다.

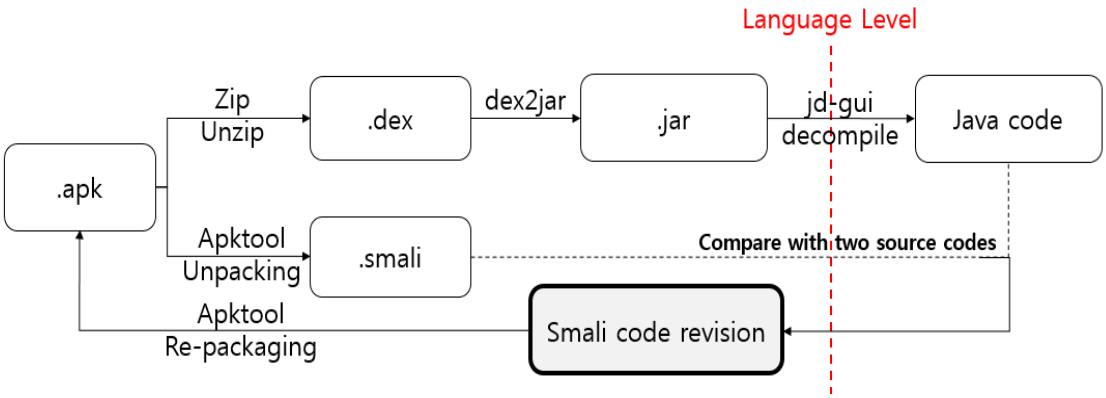

Fig. 2. Android application analysis flow diagram

3.1 안드로이드 분석 과정 및 도구

APK(Android Package)는 안드로이드 플랫폼에서 어플리케이션 설치를 위해 배포하는 패키지 파일이다. 패키징된 apk 파일은 자바 표준 jar 파일과 유사한 압축 Zip 파일 포맷 형태로 구성되어 있기 때문에 간단한 압축 프로그램으로도 언패킹(Unpacking)이 가능하다[14]. 스마트폰으로부터 추출한 apktool을 이용하여 apk 파일을 언패킹하면 Fig. 3.과 같이 xml 확장자를 가진 파일과 lib, res 등의 디렉터리들이 추출된다[15]. 어플리케이션의 이름, 버전, 엑세스 권한 등의 중요 요약 정보를 포함하는 AndroidManifest.xml 등 다양한 요소가 추출되나, 해당 절에서는 본 논문에서 필요로 하는 dex 파일과 smali 파일에 초점을 맞추어 흐름에 따라 다루어본다.

본 논문에서 실험을 진행할 전체적인 취약점 분석 절차는 Fig. 2.와 같다. 화살표의 윗부분은 분석에 사용하는 도구를, 아랫부분은 분석 과정을 의미하며, 붉은 점선은 코드 언어의 레벨이 다름을 나타낸다. 먼저 스마트폰으로부터 추출한 apk 파일의 압축을

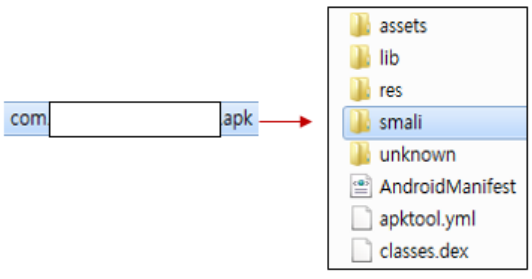

Fig. 3. the results after unpacking apk file

해제하면, dex 파일과 smali 파일들이 추출된다. 이 중 dex 파일은 dex2jar를 이용하여 jar 파일로 변환하고[16], 자바클래스들로 이루어진 jar 파일은 jd-gui와 같은 자바 디컴파일러 도구를 이용하여 디컴파일한다[17]. 본 논문에서는 앞서 추출된 smali 파일과 jar 파일의 자바코드(Java code)와 비교분석하여 우회를 위한 smali 코드 수정을 한다. 이후 이를 apktool로 리패키징(Repackaging)하여 다시 apk 파일로 변환하여 실행함으로써 이 전체적인 과정의 실험을 반복한다.

3.2 언어의 분류

Fig. 2.의 빨간 점선을 기준으로 .jar 파일을 한 번 더 디컴파일 과정을 거쳐야하는 우측의 자바코드는 좌측의 dex 파일 및 smali 파일과는 다른 레벨의 코드임을 표시하였다. 자세한 코드 언어의 의미는 이하 소단원부터 차례로 설명한다.

3.2.1 Dex 파일

Dex(Dalvik Executable) 파일은 apk 언팩 과정에서 안드로이드에서 디스어셈블러(disassembler)에 의해서 생성된다. 이는 응용프로그램들이 작동할 수 있게 해주는 달빅 가상 머신(Dalvik Virtual Machine)에 의해 컴파일 된 이루어진 자바 바이트 코드(Bytecode)이다. dex 파일은 바이너리(binary) 코드로 이루어져있어 사람이 읽기 힘든 형식의 파일이다. 이에 dex 파일을 jar 파일로 변환하여 비교적 이해하기 쉬운 자바코드로 디컴파일한 후 분

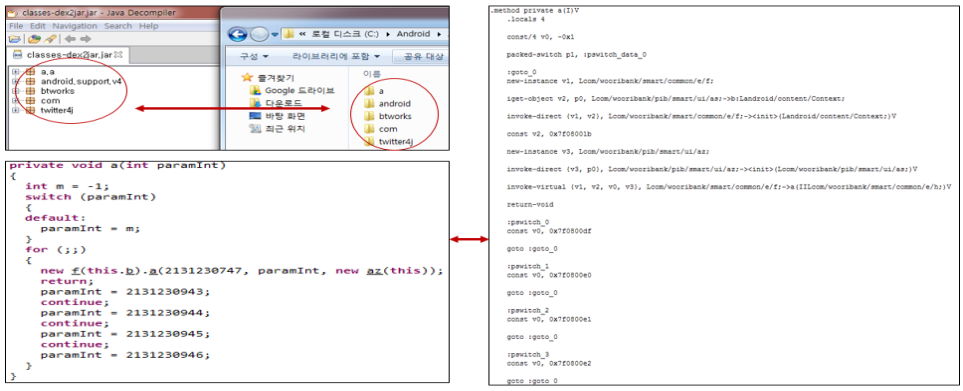

Fig. 4. Comparison between java code and smali code

석한다.

3.2.2 smali 파일

smali 파일 또한 dex와 같이 apk 언팩과정에서 생성으로써, 바이너리 코드로 구성된 dex 파일을 사람이 읽을 수 있도록 단순 변환한 코드 파일이지만, 자바코드와 달리 가독성이 떨어진다[18]. 하지만, Fig. 4.와 같이 자바소스파일과 동일한 패키지를 가지고 클래스 단위로 생성되기 때문에 자바코드와 비교해서 분석한다면, 기능 및 제어흐름은 충분히 이해할 수 있다.

3.2.3 smali 코드 수정의 이유

가독성이 좋은 자바 코드가 아닌 상대적으로 어려운 smali 코드를 분석 및 수정하여 실험하는 이유는 다음과 같다.

- 자바코드를 수정 할 경우 다시 컴파일을 해야 하나, 보통 금융권 어플리케이션에는 자바소스코드 난독화가 기본적으로 다 되어 있어 다시 컴파일이 힘든 경우가 존재한다.

- 본 논문에서 다루는 인증 우회 루틴은 많은 소스코드를 변경하는 것이 아닌, 기존 루틴의 방향만을 바꿔주는 소량의 코드 수정만을 필요로 하기 때문에 굳이 한 단계 더 디컴파일 및 다시 컴파일을 할 필요가 없다.

smali 코드 수정은 텍스트 편집기에서 쉽게 가능하며, apktool의 리패키징 과정에서 dex 형식의 바이너리 코드로 자동 변환되기에 분석 실험 시간 또한 크게 줄일 수 있다.

3.3 국내 모바일 뱅킹 백신 프로그램

현재 국내에서 전자금융거래를 하고 있는 안드로이드 뱅킹 어플리케이션을 임의적으로 10개를 선택하여 조사한 결과, 모든 어플리케이션에서 대표적으로 세 가지 종류의 모바일 백신 프로그램을 적용하고 있다.

안드로이드 뱅킹 어플리케이션에 대한 공격은 구동 시 외부에서 스니핑 분석, 재전송 공격, 브라우저 익스플로잇이 있으며, 내부에 은닉해있는 악성코드를 이용한 메모리 누수 등 정보 유출의 가능성이 있다[10]. 이에 금융권에서는 이러한 공격에 대비하여 자체적으로 암호화 통신 등의 대응을 하지만, 최신 악성코드에 대한 대응은 전문 보안 업체의 백신 프로그램에 대비하여 상대적으로 대응기술이 부족한 면이

Table 1. Current status of banking applications applying the vaccine

Mobile vaccine manufacturer	The number of domestic banking applications applying the vaccine
A Vaccine	8
B Vaccine	1
C Vaccine	1
Sum	10

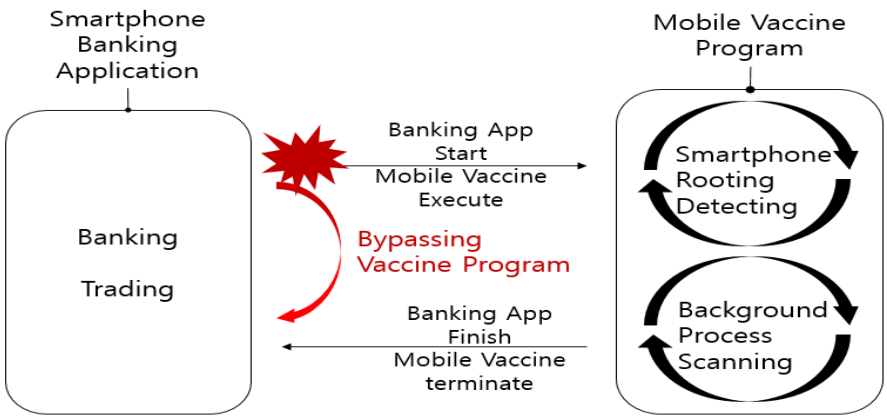

Fig. 5. Process diagram about how to bypassing mobile vaccine program while banking application is running

존재한다. 이에 국내 대다수의 금융기관은 안드로이드 뱅킹 어플리케이션에 전문 보안 업체에서 제작한 모바일 백신 프로그램 사용을 의무화하고 있다.

국내 금융권에 적용된 모바일 백신 프로그램의 주요 기능은 첫 번째로 기기에 악성 어플리케이션이 설치되어 있는지 실시간으로 탐지하여 안전한 금융거래를 보장하도록 보호한다. 두 번째로는 실행 전부터 기기 내에 악성 어플리케이션이 존재하는 경우 서비스를 중단하고 악성 어플리케이션을 삭제한다. 마지막으로는 금융감독원 지침 준수를 위한 스마트폰 루팅(Rooting) 검진 모듈을 별도 지원한다. 이러한 모바일 백신 프로그램은 Fig. 5.와 같이 뱅킹 어플리케이션 실행 시 자동으로 실행이 되며, 해당 어플리케이션의 서비스가 종료될 때 함께 종료됨으로써 뱅킹 어플리케이션을 보호한다[19][20].

다음 장부터 진행될 실험에서는 Fig. 5.와 같이 모바일 백신 프로그램이 시작되는 부분을 포함하여 보안 검사하는 지점들을 파악한 뒤, 중간언어 수정을 통해 우회하여 모바일 백신 프로그램을 미작동하도록 한다. 우회된 구간에서는 모바일 백신 프로그램이 작동하지 않으므로, 실시간 감시가 불가능한 구간을 통한 공격의 가능성이 있음을 알아본다.

IV. 모바일 백신 프로그램 우회 실험 및 결과

본 장에서는 뱅킹 어플리케이션의 모바일 백신 프로그램 우회를 위한 방법을 살펴본다. 실험에 앞서 가장 중요한 것은 앞 장에서 살펴본 바와 같이 모바일 백신 프로그램을 검사하는 지점을 찾는 것이 중요하다. 전체적인 실험의 흐름은 Fig. 6.과 같다.

모바일 백신 프로그램 없이 뱅킹 어플리케이션을 실행할 경우 어플리케이션은 다양한 방법으로 설치를 유도하고 있다. 방법은 크게 두 가지로 분류하며 아래와 같다.

- Fig. 7.과 같이 보안 검사 알림을 통해 설치 요구 알림을 띄워주고 유저가 확인을 하게 되면 어플리케이션이 종료된다.

- 위와 같은 알림이 없이 어플리케이션이 바로 종료되거나, 바로 구글 플레이 스토어의 모바일 백신 다운로드 페이지로 이동한다.

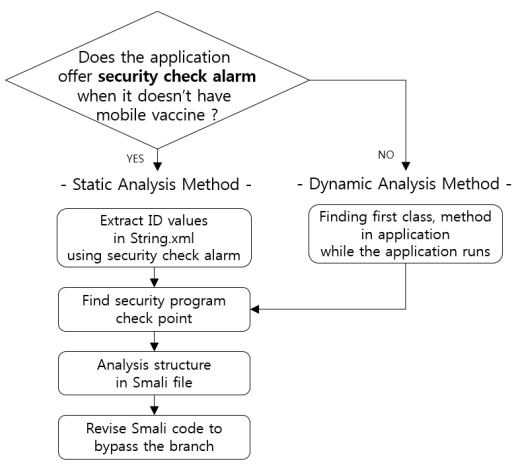

Fig. 6. bypassing test process

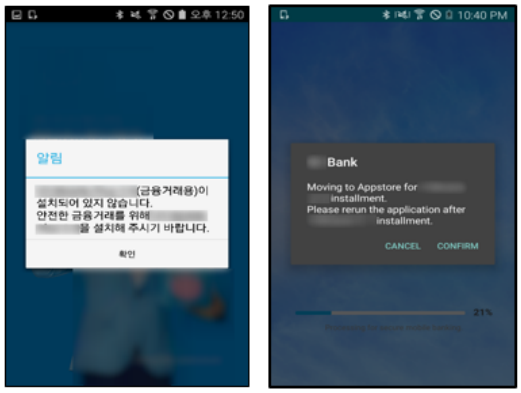

Fig. 7. mobile vaccine checking alarm

본 논문에서는 위 두 가지의 차이점에 따라 보안 검사 지점 찾는 방법을 두 가지로 분류하여 진행한다. 첫 번째 경우는 보안 알림 내의 문구를 이용하여 string.xml 내 해당 ID 값을 찾아 이를 근거로 보안 검사 지점을 찾아 들어가는 정적 분석 방법을 이용한다. 한편, 두 번째 경우는 직접 실행을 하여 그 지점에 대한 로그를 작성하여 나오는 결과를 보고 보안 검사 지점을 파악하는 동적 분석 방법을 이용한다. 그 이후에 해당 위치의 루틴을 파악하여 흐름을 바꿔줌으로써 보안 우회를 시도한다. 즉, 전체적인 흐름은 어플리케이션의 코드 수정 후 리패키징을 통해 실험하는 정적 분석 실험이지만, 보안 검사 지점을 파악하는 부분에 있어서는 일부 정적 분석 방법을 이용한다.

실험대상으로는 국내 주요 은행의 뱅킹 어플리케이션을 무작위로 선정하여 진행하였으며 실험환경은 Table 2. 와 같다. 디컴파일을 하기 위한 apktool 은 경우에 따라 코드를 수정하여 작업을 진행하였다.

4.1 정적분석 방법

본 절에서는 보안 검사 지점을 찾기 위한 정적분석을 통해 우회 방법에 대하여 살펴본다. 정적분석이란 어플리케이션을 실행시키지 않고 디컴파일된 소스코드를 분석하는 것을 의미한다. 정적분석은 다음 절에서 다룰 동적 분석에 비하여 비교적 적은 노력으로 빠른 분석이 가능하다. 자세한 실험의 절차는 이하 소단원부터 차례로 설명한다.

Table 2. Test environment

	Name	Version
Device	Samsug Galaxy S4 LTE-A (SHV-E330S)	Android Lollipop v5.0.1
Tools	adb	v1.0.36
	apktool	Self-editing tool using v2.2.1, v2.1.0,v2.0.3, v1.4.1
	dex2jar	v2.0
	jd-gui	v1.4.0
	ddms	v25.2.5

4.1.1 String id 값 추출

모바일 백신 프로그램이 실행되지 않은 상태에서 뱅킹 어플리케이션을 실행할 경우 보안 알림과 함께 어플리케이션이 강제적으로 종료되는 경우로 실험을 전개한다. 이러한 보안 알림을 이용하여 모바일 백신 프로그램이 작동되는 위치를 파악한다. 기본적으로 안드로이드 어플리케이션에서 발생하는 알림은 String 형식으로 res/values-ko/strings.xml 파일에 저장된다. 실험 대상 어플리케이션에서는 Fig. 8.와 같이 해당 위치에서 알림에 대한 문구를 찾을 수 있으며, 이 String의 이름인 alert_v3_not_installed_msg을 찾을 수 있다. 이 이름을 이용하여 동일 파일에서 String에 대한 id값인 0x7f0800df를 찾을 수 있다.

4.1.2 Smali 구조 분석

위의 절에서 찾은 id값을 이용하여 디컴파일한 smali 폴더 내에서 해당되는 smali 파일을 찾는다. 실험 대상 어플리케이션에서는 Fig. 9.의 68번 라인과 같이 0x7f0800df값의 위치를 찾을 수 있으며, 동시에 모바일 백신 프로그램의 설치 및 실행 여부를

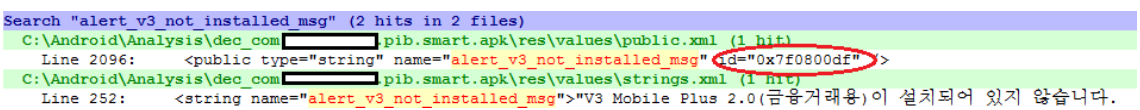

Fig. 8. Extract Id values in String.xml using security alarm

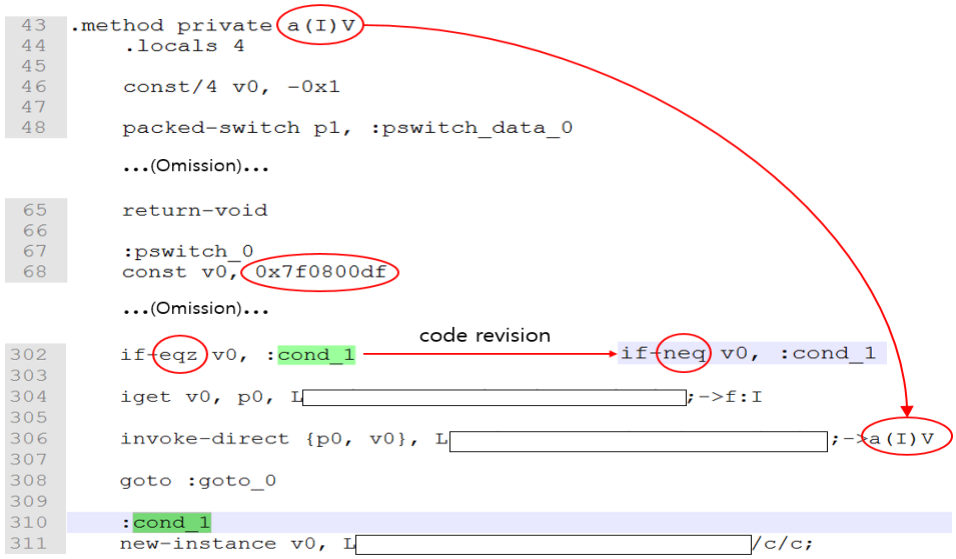

Fig. 9. Revise smali code to bypass the branch

검사하는 부분임을 알 수 있다. 해당 위치가 실행되지 않도록 우회하기 위하여 id값보다 위쪽의 코드 중 if, goto문과 같은 조건문을 통해 루틴이 실행되지 않도록 할 수 있는 요소가 있는지 파악한다. 만약 존재하지 않는다면, Fig. 9.와 같이 이것이 포함된 a(I)V 자체를 우회할 수 있는 방법을 찾는다.

4.1.3 우회 코드 수정

위의 절에서 살펴본 a(I)V가 실행되는 위치는 Fig. 9.의 302번 라인과 같다. a(I)V가 실행되지 않도록 하기 위해서 윗부분의 if-eqz v0, : cond_1 문을 코드 수정을 한다. 이 if문의 의미는 v0이 0이면 아래쪽의 cond_1로 넘어간다는 의미로, v0의 정확한 값은 알 수 없더라도 v0이 0이 아니기 때문에 cond_1로 가지 않고, 순차적으로 코드를 수행하여 a(I)V로 진입한다는 기존의 흐름을 알 수 있다. 즉, if문을 Table 3.을 참고하여 기존과 반대 의미인 nez로 코드 수정을 하게 되면 if문과 cond_1 사이에 있는 a(I)V를 우회할 수 있다.

Table 3. Comparison if branch in smali

if-eqz v0	↔	if-nez v0
if-eq v0, v1		if-ne v0, v1
if-ge v0, v1		if-le v0, v1

코드 수정한 파일을 apktool로 리패키징하여 생성한 변조된 어플리케이션을 스마트폰에 설치 및 실행하면, 변조된 어플리케이션에서는 모바일 백신 프로그램이 작동하지 않음을 확인할 수 있다. 또한, 해당 어플리케이션에는 이와 같은 모바일 백신 검사 지점이 총 4군데 존재함을 확인하였으며, 이 지점들 마다 본 실험 방법의 코드 수정을 통한 우회가 가능하다.

4.2 동적분석 방법

본 절에서는 보안 검사 지점을 찾기 위한 동적 분석을 통해 우회 방법을 살펴본다. 동적 분석 방법이란 어플리케이션을 직접 실행시키면서 직접 입력이나 로그를 발생시키고 이에 대한 결과를 확인하면서 분석하는 방법이다. 본 실험의 전체적인 흐름은 정적인 실험이지만, 보안 검사 알림이 발생하지 않는 경우가 존재하여 동적 분석을 진행하여 보안 검사 지점을 찾도록 한다. 실시간으로 실행되는 루틴을 확인하며 따라가기 때문에 정적분석에 비하여 정확할 수 있으나, 금융권은 동적 디버깅을 방지하기 위한 입력값 등에 대한 메모리 해킹 방지 기술이 적용되어 있으며, 소스코드 위치 파악의 경우는 코드 난독화가 되어있기 때문에 원하는 지점을 찾는데 많은 시간이 소모된다.

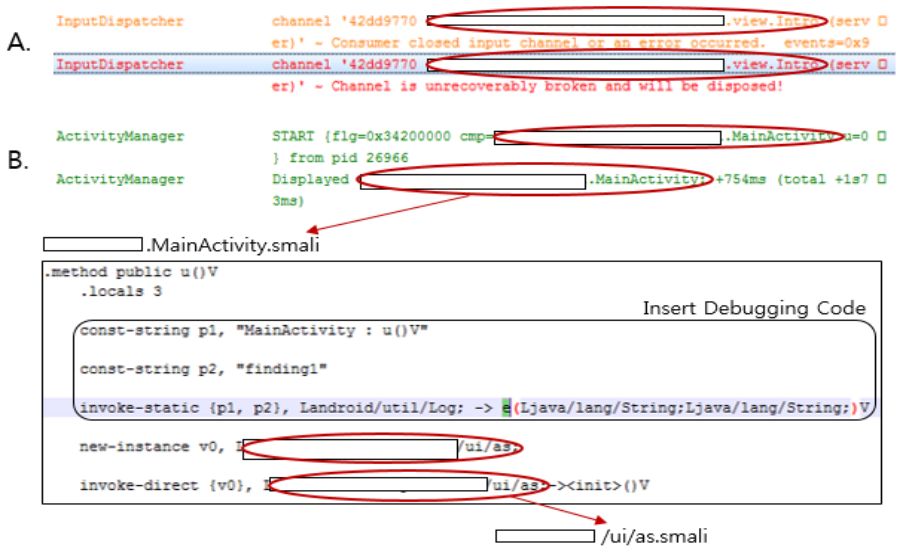

Fig. 10. Finding classes and methods including security check point using DDMS's massages

4.2.1 보안 검사 예상 지점 클래스 추출

모바일 백신 프로그램이 실행되지 않은 상태에서 뱅킹 어플리케이션을 실행 할 경우 보안 알림 없이 검은 화면과 함께 강제적으로 종료되거나, 구글 플레이스토어 모바일 백신 다운로드 페이지로 강제 이동되는 경우는 실험으로 전개한다. 디컴파일된 소스 코드에만 의존하여 해당 위치를 찾기에는 최소 1000개 이상의 클래스들이 a,bz,ci 등으로 난독화가 되어있어 시작지점과 클래스의 역할 등을 파악하여 흐름을 찾는 것이 어렵다. 즉, 해당 반응이 나온 지점을 디컴파일된 코드 내에서 찾기 위해서는 해당 지점에 대한 동적인 메시지를 살펴본다. 이를 확인하기 위하여 ddms(Dalvik Debug Monitor Service)라는 안드로이드 내 자바 가상 머신인 Dalvik을 모니터링, 디버깅 해주는 시스템을 사용한다. 해당 시점의 어플리케이션의 pid를 설정하고 발생되는 메세지를 살펴보면, Fig. 10.의 A은행과 B은행들과 같이 클래스 정보를 포함한 메세지가 발생한다. 이 때, 다량의 메시지가 타 위치의 정보도 포함한 많은 클래스의 정보를 담아 짧은 시간 내에 발생되기 때문에 위와 같이 정확한 메시지를 찾기 위해서는 분석에 대한 많은 시간과 노력이 필요하다. 이후, 발견한 위치에 있는 클래스를 디컴파일된 smali 폴더에서 찾아 확인하면, Fig. 10.의 아래와 같은 코드를 발견할 수 있다.

4.2.2 해당 클래스 내 메소드 분석

해당 클래스에서 보안 검사 지점의 분기점을 찾기 위해서는 직접 로그를 발생시키는 실험을 통해 확인한다. 실시간으로 디버깅을 확인 할 수 있는 netbeans와 같은 도구에 연결하면 비교적 수월하나, 금융권 어플리케이션에는 동적 디버깅에 대한 Attach가 불가능하여 완벽한 동적인 분석은 어렵다. 이에 분기가 예상되는 지점 앞에 위의 디버깅 코드를 삽입하여 원하는 내용을 입력하여 리패키징 후 변조된 어플리케이션을 다시 실행할 때 ddms 내에서 로그 정보를 보면서 흐름을 파악한다. Fig. 10.의 Insert Debugging Code와 같은 코드를 많은 구간에 서로 다른 로그 설정을 한 뒤, 결과에 대한 변화를 분석하면서 단서를 찾아나간다. Fig. 10.에서 /ui.as.smali에 해당되는 메소드가 존재함을 알 수 있다. 이와 같은 방법을 as.smali에서 반복한 뒤 보안 검사 지점에 해당되는 method를 찾으면, 정적 분석 방법에서 다룬 우회 방법을 통해 method의 분기를 바꿔줌으로써 우회를 한다.

위와 같은 동적 분석 방법에는 예상 분기에 로그를 설정하고 체크하면서 진행을 해야 하기 때문에, 정적분석과 비교하여 더 많은 시간과 어려움이 따른다. 특히, 금융권 어플리케이션의 경우는 모두 난독화가 되어 있으므로, 흐름을 따라가면서 실제 코드상의 어떤 역할을 하는 클래스인지를 유추하면서 진행

Table 4. Bypassing mobile vaccine programs in banking applications test results

Banking apps	Security alarm exist in apps	Decompile	Repackaging	Security alarm based analysis	Dynamic Analysis	bypass the apps
A	○	○	○	○	-	○
B	○	○	○	○	-	○
C	○	○	○	○	-	○
D	○	○	○	○	-	○
E	○	○	○	○	-	○
F	○	○	X	X	X	X
G	○	X	X	X	X	X
H	X	○	○	X	○	○
I	X	○	○	X	○	○
J	X	X	X	X	X	X

해야하므로 일반 어플리케이션과 비교하여 더 많은 노력과 시간이 소모된다. 하지만, 불가능한 요소만은 아니기에 이러한 보안 취약점에 대한 대책은 반드시 필요하다.

4.3 실험 결과

국내 주요 은행들의 안드로이드 뱅킹 어플리케이션 10개를 선택하여 모바일 백신 프로그램 검사 지점 우회 실험을 진행하였다. 먼저, 어플리케이션 내 보안 알림의 존재 여부, 디컴파일/리패키징 가능 여부, 보안 검사 알림 기반 우회, 그렇지 않은 경우 동적 우회들을 조사하여 Fig. 11.과 같이 결과를 확인한 결과는 Table 4.와 같다.

Table 4.와 같이 10개 중 절반 이상인 7개의 어플리케이션에서는 우회가 가능하였다. 어플리케이션 내 모바일 백신 프로그램이 존재하지 않거나 실행되지 않는 경우 보안 알림이 오는 어플리케이션은 10개 중 7개가 존재하였다. 이들 중 디컴파일이나

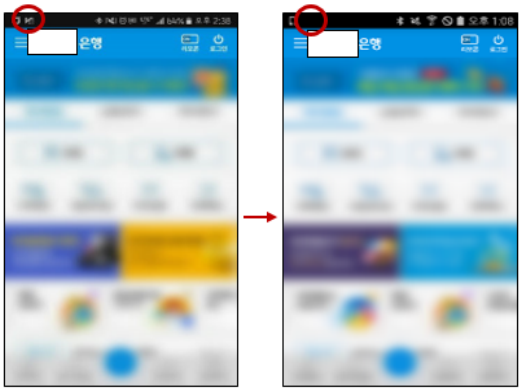

Fig. 11. Test result screen

리패키징이 불가능한 경우는 본 연구의 실험과 같이 반복을 못하므로 해당 되는 2개의 어플리케이션은 분석이 불가능하였다. 이에 반해, 모바일 백신 프로그램이 없는 경우 검은 화면과 함께 종료되는 경우와 바로 플레이 스토어로 넘어가는 경우들이 3개의 어플리케이션에서 존재하였다. 이 중 1개의 어플리케이션은 디컴파일시 파일들이 제대로 추출되지 않아 분석이 불가능하였고, 나머지 2개의 어플리케이션은 동적 분석으로 보안 검사 지점을 찾아 코드를 수정하여 리패키징 한 뒤 우회에 성공함을 확인할 수 있었다.

본 실험에서는 apktool만을 사용하여 디컴파일/리패키징을 하였기 때문에 3개의 어플리케이션에 대하여 정상적인 실험을 하지 못하였다. 달빅 바이트코드 단위에서 난독화를 하여 디컴파일을 방지하거나[21], Dex 파일 형식에서 자바 클래스명을 255자 이상으로 조작하는 방법으로 역공학을 방지하고 있어 디컴파일이 어려운 경우가 존재하였다[22]. 하지만, 위와 같은 실험 방법으로 루팅 폰 환경 등의 변경된 OS에서의 실행 우회, 알 수 없는 출처 제한 등을 모두 우회한 상황이므로 이러한 보안 검사 우회 방법에 대한 보안 강화는 반드시 필요하다.

모바일 백신 프로그램의 목적은 금융권 어플리케이션이 실행됨과 함께 작동이 되고, 종료될 때 까지 실시간 악성코드 감지와 루팅 상황에 대한 위협에 대하여 안전을 보장하는 것이다. 하지만, 모바일 백신이 우회로 인한 미작동 되는 부분이 존재함으로써, 실행 시 악성 행위를 당하더라도 차단되지 않기 때문에 이를 이용한 위험성은 더욱 커질 수 있다.

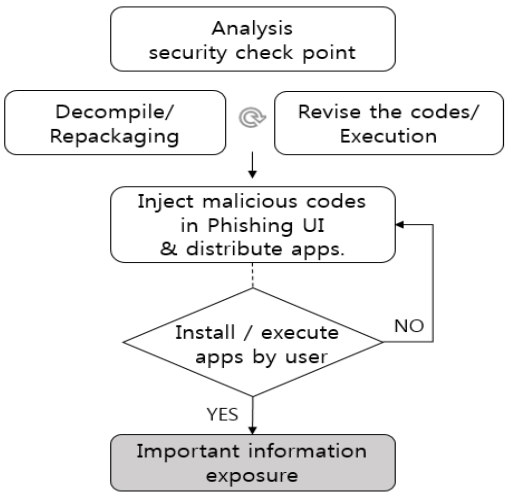

Fig. 12. Possible attack scenario

4.4 취약점을 이용한 공격

본 절에서는 취약점 분석 과정으로 실험한 결과에 대한 분석 및 이에 대한 공격 시나리오를 알아본다. 본 논문의 취약점 분석 방법을 응용한 공격 가능 시나리오는 Fig. 12.와 같다.

먼저 목표 어플리케이션에 대하여 본 논문과 같이 분석 및 실행을 반복하여 보안 검사 지점들을 파악한 뒤, 위변조 검증 지점 등의 우회 가능 여부를 조사한다. 이후, 원하는 지점까지 우회가 가능했다면 이를 이용하여, 공격자가 원하는 코드를 삽입한 뒤 실제 금융 어플리케이션과 동일한 UI의 변조된 어플리케이션을 배포한다. 실제 사례로 정상 어플리케이션에 악성코드를 심은 뒤, 변조된 어플리케이션으로 재배포하여 시스템정보를 탈취하는 악성 어플리케이션이 수많은 개인정보를 탈취하였다[23]. 이러한 변조된 악성 어플리케이션을 통해 안드로이드의 접근성 권한에 대하여 사용자에게 권한 요구로 유도한다면, 이를 이용하여 더 큰 유효한 공격이 가능하다[24]. 또한, 만약 사용자가 유사한 악성 뱅킹 어플리케이션을 설치 및 실행한다면, 사용자의 계정부터 비밀번호, 공인인증서 파일, 공인인증서 비밀번호까지 모든 입력정보가 유출 될 수 있으므로 이에 대한 심각성과 파급효과는 더 커질 가능성이 있다.

V. 대응방안

5.1 기술적 대응방안

5.1.1 리소스 암호화

본 논문에서 밝히는 취약점 분석 방법의 가장 큰 핵심은 보안 알림 메시지에 있다. 최초 모바일 백신 프로그램 미설치 혹은 미작동시, 이에 관한 보안 알림을 토대로 String name 값과 Id 값을 이용하여 작동 여부를 검사하는 위치를 찾아나간다. 즉, 최초 보안 알림 문구가 어플리케이션 내부에 존재하지 않도록 한다면, 1차적인 정적인 분석을 막는데 충분히 효과적이다.

res/values/에 있는 string.xml이나 array.xml 등은 루팅된 단말에서 추출된 apk 파일을 디컴파일하면 데이터 확인이 가능하다. 이러한 데이터 확인이 불가능하도록 string.xml의 내용은 암호화하거나 숨기는 것으로 이러한 취약점 분석 방법을 막을 수 있다.

Fig. 13.의 A.는 기존의 apk 파일 디컴파일시 res/values/string.xml에서 찾을 수 있는 보안 알림 문구이다. 여기서 string name을 통해 id 값을 추출함으로써 본 실험의 보안 검사 지점을 찾을 수 있었다. 이에 반해 B.는 string.xml의 문구를 암호화한 작업의 예시이다[25]. 이 경우 문구가 암호화

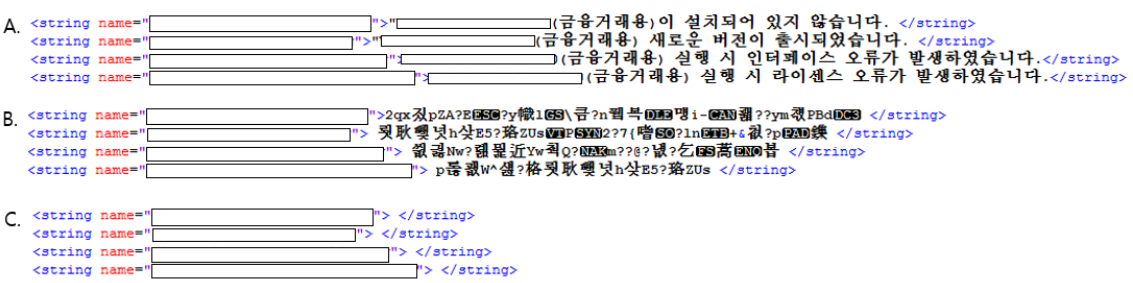

Fig. 13. res/string.xml encryption

되어 있기 때문에 해당된 String name과 id 값을 찾기 힘들기 때문에, 본 실험과 같은 공격을 막는데 효과적이다. 단, 서버와 어플리케이션의 암호화키 관리 및 암호화/복호화 처리에 있어서 어플리케이션 성능에 관한 고려는 필수적으로 고려해야한다. 한편, C.는 string.xml의 데이터 값들을 모두 숨긴 경우의 예시이다. string.xml의 내용들을 네이티브 라이브러리(natvie library)를 이용하여 Java가 아닌 C/C++로 작성한 apk 내 다른 위치에 숨겨 저장하거나, 별도의 서버를 두어 관리하며 해당되는 결과만을 전송한다. 즉, 정적인 코드 분석을 방지하기 위해서 실행시점에 메모리의 실행코드 영역에 실제코드를 생성하는 방법을 적용한다[26]. 이는 소스코드의 정적 분석 시에는 실제 코드를 숨기고 실행 시에 실제 코드로 치환하여 처리되기 때문에 실행 코드를 보호할 수 있는 방법이다. 위와 비교하여 비교적 강력한 보안이지만, 구현하여 도입하는 비용과 개발자 및 운영자의 어플리케이션의 유지/보수 관점에서 보면, 성능 및 효율적 측면에서 역효과를 일으킬 가능성이 있다.

5.1.2 제어흐름 난독화 강화

소스코드 난독화란 디컴파일시 추출되는 코드 분석 및 공격에 대하여 보호하기 위한 기술이다. 본 연구에서 확인한 현재 금융권 어플리케이션의 자바 코드의 패키지명, 클래스명, 메소드명, 변수명 등을 의미 없는 문자로 치환하는 문자열 암호화는 대부분 적용되어 있었다. 이와 더불어 약 3개월 단위로 치환된 문자열을 바꿔주는 결과를 확인 할 수 있어, 이전에 분석한 결과가 무의미하게 되는 경우도 존재하여 실험 과정에서 분석시간에 많은 영향을 주었다. 이렇듯 난독화 기술은 완벽한 보호는 아니지만, 분석을 지연시킴으로써 공격을 지체시키고 공격을 어렵게 만들 수 있다. 본 연구에서는 어플리케이션의 흐름을 바꿔주는 것만으로 우회를 성공하고 있어 이에 대한 난독화 강화의 필요성이 있다. 따라서 제어흐름 변환과 같이 코드의 흐름을 먼저 이해하고 이 흐름을 바꿔주거나, dummy를 추가함으로써 로직 파악을 어렵게 하는 방법을 추가할 수 있다[27].

5.1.3 무결성 검증 지점 확대

국내 금융권은 모바일을 이용한 전자금융거래의 안정성을 보장하기 위한 연구와 함께 무결성 검증 기술을 발전시켜왔다[12]. 현재 주요 무결성 검증 기술로는 대표적으로 메모리 검증이 있다[28]. 실행시점에서의 메모리에 존재하는 Dex 바이트 코드 데이터 값 등을 서버에 전송하여 서버에 보관중인 원본 바이트코드의 정보와 비교하여 검증된 결과만을 받아 검사한다. 이 과정은 물론 암호화 통신을 할 뿐만 아니라, 메모리 정보는 실제 실행중인 바이트코드 값과 동시에 서버에서 보관 중인 값을 구하여 일치하도록 전송할 수 가 없기에 어려운 점이 있다. 또한, 분석을 위해서 서버와의 통신을 이용한 메시지를 분석해야 하지만, 금융권 서버는 실시간으로 이를 탐지하기 때문에 ip 차원에서 차단되기 때문에 지속적인 분석에는 제약이 있다. 이러한 무결성 검증 구간을 어플리케이션 실행 부분에 포함시킨다면, 임의로 변조된 어플리케이션을 통해 반복 실험을 하는 본 실험의 취약점을 이용한 공격 가능성을 더욱 줄일 수 있을 것이다.

5.2 관리적 대응방안

5.2.1 금융분야 취약점 분석, 평가 기준 항목 점검

현재 스마트폰 뱅킹 부문의 모바일 어플리케이션 진단 항목은 금융위원회에서 지정한 전자금융감독규정 금융분야 취약점 분석, 평가 기준 15개 항목이 있다. 이 중 악성코드 및 프로그램 위변조 대응 분야의 악성코드 방지, OS 변조 탐지 진단 항목에 대한 구체적인 평가 방법이 제시되어야할 필요성이 있다.

현재 악성코드 방지 항목의 경우는 모바일 백신이 작동 되지 않을 경우 보안 알림 혹은 강제 종료를 통해 어플리케이션을 실행하지 못하게 한다면 진단 결과는 양호로 판단한다. 그러나 본 연구의 방법으로 우회가 가능한 부분들이 존재한다. 이를 방지하기 위하여 리소스 암호화 혹은 이러한 우회 방지 검사와 같은 깊이 있는 판단을 통한 양호 판단이 요구된다. OS 변조 탐지 항목 또한, 위와 같이 보안 알림을 통해 루팅 폰 등에서 실행되지 않도록 할 경우, 양호 판단을 한다. 본 연구의 방법으로 물론 OS 변조 탐지도 가능함에 따라 비교적 깊이 있는 진단이 가이드로 나온다면 앞으로 일어날 수 있는 보안사고의 가능성을 크게 줄일 수 있을 것이다.

VI. 결 론

본 논문에서는 디컴파일시 추출되는 중간언어인 smali 코드의 분석을 통해 안드로이드 뱅킹 어플리케이션 내 보안 검사 우회 방법을 제시하였다. 또한, 실제 서비스를 하고 있는 국내 주요 10개 은행의 안드로이드 뱅킹 어플리케이션을 대상으로 분석 및 실험을 통해, 대부분의 어플리케이션의 모바일 백신 프로그램이 우회됨을 보임으로써 취약점을 가지고 있음을 보였다. 이 분석방법을 악용하여 정상적인 뱅킹 어플리케이션을 악성 뱅킹 어플리케이션으로 위변조하여 재배포한다면, 모바일뱅킹 이용자의 중요개인정보가 유출 될 수 있는 치명적인 보안 위협이 될 수 있다. 또한, 이러한 연구가 가능하게 된 원인을 분석하여 이에 대한 리소스 암호화와 같은 대응방안을 제안하였다.

본 논문의 보안 검사 지점 우회를 이용한 취약점 분석 방법은 스마트폰을 이용한 전자금융거래에 직접적으로 치명적인 영향을 주지 않는 한계가 있다. 하지만, 뱅킹 어플리케이션은 예상할 수 있는 최악의 상황을 고려하여 보안에 더욱 신경을 써야 할 필요가 있는 분야이다. 충분히 유효한 공격이 발생할 가능성이 있기 때문에 이러한 취약점에 대하여 대응해야하는 방안은 반드시 필요하다.

향후에는 하드웨어 레벨에서의 더욱 심층적인 분석방법을 이용하여 중요 정보 탈취 공격과 대응방법에 대하여 더욱 깊이 있는 연구를 할 계획이다.

References

[1] "Domestic Internet banking service in Q2 / 2016", The Bank of Korea, pp. 4-7, Aug. 2016

[2] Hyunho Cho , "E-finance and Financial Security", Financial Security Institute, vol 4, pp.5-32, Feb. 2016

[3] "Announcement of Survey on Internet Usage Conditions in Korea", Korea Internet & Security Agency, pp. 6-7, Feb. 2015

[4] Roman Unuchek, "Mobile malware evolution 2016", Kaspersky lab, pp. 10, Feb. 2017

[5] Jisun Choi, Taehee Kim, Sangshik Min and Jaemo Seung, "Protection technology trend for smartphone banking application integrity verification", Journal of Information Security, 23(1), pp. 54-60, Feb. 2013

[6] Jin-Hyuk Jung, Ju Young Kim, Hyeong-Chan Lee and Jeong Hyun Yi, "Repackaging Attack on Android Banking Applications and Its Countermeasures," Wireless Personal Communications, vol 73, no. 4. pp. 1421-1437, Dec. 2013

[7] Wu Zhou, Yajin Zhou, Xuxian Jiang and Peng Ning, "Detecting repackaged smartphone applications in third-party android marketplaces", CODASPY '12 Proceedings of the second ACM conference on Data and Application Security and Privacy, pp. 317-326, Feb. 2012

[8] Seungyong Yoon, Jeongnyeo Kim and Yongsung Jeon, "Analyzing Security Threats of Android-based Mobile Malware", Advanced Science and Technology Letters(SecTech 2016), Vol. 139, pp.310-315, Nov. 2016

[9] Yajin Zhou and Xuxian Jiang, "Dissecting Android Malware: Characterization and Evolution", 2012 IEEE Symposium on Security and Privacy, pp. 95-109, May. 2015

[10] Sriramulu Bojjagani and V.N. Sastry, "STAMBA: Security Testing for Android Mobile Banking Apps", Advances in Signal Processing and Intelligent Recognition Systems, pp. 671-683, Dec. 2015

[11] Jong Hyuk Park, Ki Jung Yi and Young-Sik Jeong, "An enhanced smartphone security model based on information security management system (ISMS)", Electronic Commerce Research, Vol. 4, no 3, pp. 321-348, Nov. 2014

[12] Soonil Kim, Sunghoon Kim and Dong

Hoon Lee, "A study on the vulnerability of integrity verification functions of android-based smartphone banking applications", Journal of the Korea Institute of Information Security and Cryptology, 23(4), pp.743-755, 2013

〔13〕 Nguyen, Thanh, McDonald, Jeffrey Todd and Glisson, William Bradley, "Exploitation and Detection of a Malicious Mobile Application", Proceedings of the 50th Hawaii International Conference on System Sciences, Jan. 2017

〔14〕 A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google Android: A Comprehensive Security Assessment," IEEE Security and Privacy, vol. 8, no. 2, pp. 35 - 44, Mar. 2010.

〔15〕 https://ibotpeaches.github.io/Apktool/

〔16〕 https://sourceforge.net/projects/dex2jar/

〔17〕 http://jd.benow.ca/

〔18〕 https://github.com/JesusFreke/smali/wiki

〔19〕 Ahnlab, http://www.ahnlab.com/kr/site/product/productView.do?prodSeq=67

〔20〕 NSHC, http://www.nshc.net/wp/portfolio-item/droid-x/T. Strazzere,

〔21〕 Parvez Faruki, Ammar Bharmal, Vijay Laxmi, M. S. Gaur, Mauro Conti, and Muttukrishnan Rajarajan, "Evaluation of Android Anti Malware Techniques against Dalvik Bytecode Obfuscation", 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 414-421,Sep. 2014

〔22〕 "Dex Education: Practicing Safe Dex", Blackhat USA 2012, Jul. 2012, http://www.strazzere.com/papers/Dex

〔23〕 Proofpoint Staff, "DroidJack Uses Side-Load…It's Super Effective! Backdoored Pokemon GO Android App Found", Jul. 2016, https://www.proofpoint.com/us/threat-insight/post/droidjack-uses-side-load-backdoored-pokemon-go-android-app

〔24〕 Jung-Woong Lee, In-Seok Kim, "A Study on the Vulnerability of Security Keypads in Android Mobile Using Accessibility Features", Journal of The Korea Institute of Information Security & Cryptology, 26(1), pp. 177-185, Feb. 2016

〔25〕 Vaibhav Rastogi, Yan Chen and Xuxian Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks", IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, vol. 9, no. 1,pp. 99-108, Jan. 2014

〔26〕 H. Cai, Z. Shao and A. Vaynberg, "Certified Self-Modifying Code," Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation, vol. 42, no.6, pp. 66-77, Jun. 2007.

〔27〕 Marius Popa, "Techniques of Program Code Obfuscation for Secure Software", Journal of Mobile, Embedded and Distributed Systems, vol.3, no.4, pp. 205-219, 2011

〔28〕 Namheun Son, Yunho Lee, Dohyun Kim, Joshua I. James, Sangjin Lee and Kyungho Lee,"A study of user data integrity during acquisition of Android devices", 13th Annual Digital Forensics Research Conference, vol. 10, Supplement, pp. S3 - S11, Aug, 2013

〈저 자 소 개〉

이 우 진 (Woojin Lee) 학생회원
2016년 2월: 연세대학교 컴퓨터공학과 학사
2016년 3월~현재: 고려대학교 정보보호대학원 금융보안학과 석사과정
〈관심분야〉 모바일보안, 전자금융보안, 정보보호 컨설팅

이 경 호 (Kyung-Ho Lee) 종신회원
1989년 8월: 서강대학교 수학과 학사
1997년 8월: 서강대학교 정보통신대학원 석사
2009년 8월: 고려대학교 정보보호대학원 박사
2011년 9월~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 위험관리, 정보보호 컨설팅, 정보보호 및 개인정보보호정책