Journal of The Korea Society of Computer and Information
Vol. 21 No. 2, pp. 61-69, February 2016

www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2016.21.2.061

Design and Implementation of the Authentication System for In-app Billing in Mobile Environments

Ho-Jun Seok *, Seog-Gyu Kim**

Abstract

In this paper, we propose the authentication server system that prevent hacking in In-app billing applications. And we also propose the methods to verify electronic receipt for the payment of internal app payments and to check the integrity of the applications. Then we designed the payment metabolic system that checks between products-offer list and paid subscription if payment system is hacked with new hacking technologies different from existing ones. And then we implemented proposed authentication system and experimented with about 10,000 average internal application payments per an hour. It shows that proposed system has defensive techniques that counter attacks against in-app billing but it takes more than 0.8916 seconds than no-certification system that is considered as relatively short time.

▸ Keyword : In-app billing, authentication system, mobile application, prevent hacking

I. Introduction

스마트폰의 보급에 따라 스마트 기기에 사용되는 앱 제작에 많은 개발사들이 뛰어들고 있으며 그에 따라 앱시장도 급격한 성장을 하고 있다. 뛰어난 아이디어로 제작된 앱 또는 모바일 게임이 수익 창출로 연결이 되는 것이 중요하지만 해커들이나 일반 사용자가 해킹툴을 이용하여 내부결제(인 앱 결제)를 무력화하는 등의 해킹 피해가 빈번하게 발생하고 있다.

스마트폰 콘텐츠의 과금 시스템은 초기에는 어플리케이션 자체를 판매하는 것에서 현재는 어플리케이션은 무료로 배포 후 어플리케이션 내에서 다양한 상품을 선택하고 구입하는 시스템으로 변화되었다. 이러한 시스템을 인앱결제 혹은 내부결제라 한다. 하지만 내부결제 시스템이 해킹의 위험에 노출 되어 있고 실제 해킹에 관한 피해사례도 많이 나타나고 있다. 그래서 내부결제 시스템의 구현에 있어 해킹 방지 방안 및 구현에 관해 논하고자 한다. 최근 애플 앱스토어와 구글 플레이 스토어의 앱들의 결제 방법을 분석한 자료에 따르면 애플 앱스토어의 앱 내 구매 기준 수익 비율은 77%에서 92%로 증가했고, 구글 플레이 마켓의 경우에는 89%에서 98%로 증가하였다. 결제 기술(앱 내 구매 기능이 있는 무료앱)이 분명하게 수익 면에서 선두를 차지하고 있고, 앱 내 구매 기능이 있는 유료 앱 및 앱 내 구매 기능이 없는 유료 앱과 결제 기술은 매우 작은 수익 점유율을 차지하고 있다. 유료 또는 무료 앱 콘텐츠에 대한 선호도는 카테고리 별로 다르게 나타난다. 가장 시장 규모가 큰 게임 분야에 있어서도 90%가 앱 내부 결제로 이뤄진다. 각 카테고리 별로 내부 결제, 유료 앱 등으로 구성되지만 대부분의 카테고리에서 내부결제가 큰 비중을 차지하고 있다. 모든 국가의 집계 데이터를 기준으로 한 것으로 최대 수익을 카테고리 순으로 분석한 자료에 의하면 내부결제 기능이 있는 앱이 압도적으로 늘어나고 있는 추세이다. 그러므로 모바일 기기를 위한 앱 시장에서 내부결제에 대한 보안이 중요한 화두로 떠오르고 있다.

본 논문은 스마트 기기 앱에서의 결제 시스템인 내부결제에 있어 해킹의 공격 방법을 살펴보고 그 공격 방법에 관한 방어 방법을 제안을 하고 그에 따른 내부결제 시스템의 구현에 관해 논하고자 한다. 이를 위하여 안드로이드 설치 파일인 APK 구조를 분석하여 안드로이드 APK의 보안 취약성을 알아보고 안드로이드 플랫폼과 마켓 내부결제 시스템의 구조 및 해킹 방법에 관해서 살펴본다[1]. 또한, 내부결제 해킹 방식인 역공학

• First Author: Ho-Jun Seok, Corresponding Author: Seog-Gyu Kim
*Ho-Jun Seok(rucrazy99@naver.com), Dept. of Information & Communication Engineering, Andong National University
**Seog-Gyu Kim(sgkion@andong.ac.kr) Dept. of Information & Communication Engineering, Andong National University
• Received: 2016. 01. 22, Revised: 2016. 01. 27, Accepted: 2016. 02. 19.
• This work was supported by a grant from 2012 Special Research Funds of Andong National University

공격과 DNS를 조작하여 전자 허위 영수증 발행 방법에 대하여 분석한 후 클라이언트 프로그램 위변조를 검사하여 무결성을 확인할 수 있는 시스템 및 전자 영수증 검증 방법과 상품 지급 내역과 결제 내역을 비교 할 수 있는 대사 시스템의 구현 및 설계 방법에 관해 제안 하고자 한다.

본 논문의 구성은 다음과 같다. 2장에는 안드로이드 설치 파일인 APK 구조, 안드로이드 APK의 역공학에 대한 취약성 및 안드로이드 플랫폼과 마켓 내부결제 시스템의 구조 및 해킹 방법에 관해서도 살펴본다. 3장에서는 내부결제 해킹을 방어하기 위한 6가지 시스템 설계 방법에 대하여 기술하고 4장에서는 시스템 구현과 성능비교에 관하여 서술하고 마지막으로 5장에서는 결론 및 향후 연구방향에 대해 논한다.

II. Related Works

1. 안드로이드 어플리케이션

안드로이드 어플리케이션은 자바 형태의 언어로 개발되며, APK(Android Application Package) 파일 형태로 배포된다. 여기에 포함되는 코드는 바이트코드와 네이티브코드 이 두 종류이다. 안드로이드 어플리케이션 20 안드로이드 어플리케이션 역공학 보호기법은 바이트코드 기반으로 구성되며 필요에 따라서 빠른 연산이 요구되는 부분은 네이티브코드로 따로 작성된다.

1.1. APK 파일

APK 파일은 그림 1과 같이 크게 4가지 구성요소로 이루어진다[2]. 첫 번째 요소로 DEX 파일(.dex files) 은 어플리케이션의 모든 바이트코드를 포함한다. 보통은 한 개(classes.dex)로만 구성되지만 때에 따라서는 두 개 이상인 경우도 있다. 두 번째 요소로 컴파일된 리소스(resoures.arsc)는 윈도우 레이아웃, 문자열 상수 등과 같이 컴파일될 수 있는 XML 파일들을 포함한다. 세번째 요소로 컴파일 되지 않는 리소스는 이미지, 네이티브코드 등과 같은 것으로 원본 그대로 보관된다. 마지막 요소로 매니페스트 파일(AndroidManifest.xml)은 시작클래스, 퍼미션 등과 같은 정보를 포함한다. 이 구성요소들은 ZIP 알고리즘으로 묶여 확장자가 'apk'인 파일로 존재한다.

Fig. 1. APK file component

1.2. DEX 파일

DEX(Dalvik EXcutable format) 파일은 안드로이드 가상머신(Dalvik machine) 위에서 실행되도록 구성된 것으로, 어플리케이션 구동 시 생명 주기(life cycle)에 따라 가장 처음으로 호출되는 메소드 정보를 가지고 있다. 하지만, 극히 드문 경우로 시작 메소드가 네이티브코드에 있는 경우도 존재한다[3].

DEX 파일에서는 하나 이상의 클래스 정보가 저장되어 있으며, 문자열 식별자 리스트, 타입 식별자 리스트,메소드 프로토타입 식별자 리스트, 필드 식별자 리스트, 메소드 식별자 리스트, 클래스 정의 리스트, 데이터 영역으로 구성되어 있다. 이들은 문자열 식별자 리스트를 제외하고는 다른 리스트의 인덱스를 참조하는 식으로 구성되어 있다. 이렇게 구성된 DEX 파일은 클래스명, 메소드명, 필드명은 물론이고 디버그 정보에는 변수명, 파라미터명 등 코드 분석에 도움 되는 정보도 포함하고 있다.

1.3. ELF 파일

ELF(Executable and Linking Format) 파일은 리눅스에서의 실행, 오브젝트 파일, 공유 라이브러리, 또는 코어 덤프를 할 수 있게 하는 바이너리 파일이다. 안드로이드에서는 공유 라이브러리 형태로서, 외부 라이브러리 용도로 사용된다.

이 파일은 네이티브코드를 담고 있으며, 암호 모듈이나 그래픽 엔진과 같이 많은 연산을 요구하는 어플리케이션을 위해 존재한다. 이는 확장자가 'so'이며 APK 파일 /lib/ 하위 경로에 보관되지만 어플리케이션에 따라 없는 경우도 있다.

공유 라이브러리는 실행 시간에 동적으로 로드되며, 바이트코드 단에서 JNI(Java Native Interface)를 통해 사용된다. 이것은 네이티브코드로 구성되어 있기에 상대적으로 바이트코드보다 역공학에 강한 장점을 가지고 있다.

2. 기존 앱 마켓의 내부결제 프로세스와 공격점

2.1 구글 플레이 스토어 내부결제 프로세스

구글 Developer Console에서 인앱 상품(내부결제)에서는 세 가지 형태의 구입 형태를 제공하고 있다. 이 세 가지 형태로 구별 하는 것은 대부분의 마켓이 동일한 형태를 가진다. 그림 2와 같이 구글 내부 결제에서는 "관리되는 제품", "관리되지 않는 제품", "구독" 이렇게 세 가지 형태로 사용되고 있다[4]. "관리되는 제품"이란 소진이 되지 않고 사용자에게 영구적으로 귀속이 되는 상품을 판매할 경우에 사용된다. 다시 말해 똑같은 상품을 두 번 이상 구매할 수 없는 상품을 말하는 것이다. 이 "관리되는 제품"은 마켓에서 지속적으로 관리를 해주기 때문에 해킹의 위험이 높지가 않다. "구독"의 형태는 주기적으로 상품을 구입 하는 경우, 예를 들어 잡지 판매를 하는 경우 년/월 단위로 내부결제를 가능하게 된다. "구독"은 "관리되는 제품"과 같이 마켓에서 구매 내역을 관리하기 때문에 비교적 안정적인 결제 형태이다. 하지만 이 "관리되는 제품"과 "구독"은 일반 앱과 게임에서 과금 정책으로 거의 사용되지 않는 형태이다.

일반적으로 앱과 게임에서 가상 화폐를 구입하는 경우 상품을 구입 횟수 제한 없이 구매가 가능하도록 한다. 이것이 “관리되지 않는 제품”의 구매 형태인 것이다. 내부결제 비율의 대부분을 차지하고 있는 “관리되지 않는 제품”은 왜 안정성을 보장하지 못할 까? 라는 의문은 “관리되지 않은 제품” 말 그대로 마켓에서 관리를 하지 못하는 구조상의 문제가 있는 것이다. “관리되지 않는 제품”을 개발사에서 가장 많이 사용하는 이유는 소진되는 제품을 판매하여 수익의 극대화를 하기 위해서 이다.

Fig. 2. In-app Billing in Google Play market

하지만 “관리되지 않는 제품”의 판매형태에서의 보안 및 관리는 마켓이 아닌 개발사의 책임으로 되어 있고 개발사에서 구매 기록과 결제 검증을 진행 하여야 된다. Google In App Billing에서는 “구독”, “관리되는 제품”, “관리되지 않은 제품”으로 나누어 관리하며 그에 대한 기본 프로세스는 동일하다. isBillingSupported() 메소드 호출을 통해 앱 내부결제가 가능한지 여부를 확인하고 getPurchases()메소드를 통해 사용자가 소유하고 있는 제품의 정보를 받아 올 수 있다. getPurchases()메소드를 통해 영구 구매인 “관리되는 제품”의 보유를 알 수 있고 “관리되지 않는 제품”에서 소진 되지 않는 제품의 정보를 알 수 있다. getSkuDetails()로 판매 가능한 제품들의 상세 정보를 받아오는 메소드이며 getBuyIntent()로 구매를 진행한다[5]. “관리되지 않는 제품”에서의 구매를 소진시키는 메소드가 consumePurshase()이다. 이 절차에서의 소진은 예를 들어 게임의 화폐를 구매하게 되면 화폐로 변환할 수 있는 티켓을 앱에게 전달한다. 그 티켓을 화폐로 변화 되는 순간을 소진으로 보고 있다. 이 소진의 절차에 있어서 구조적 문제가 발생 하는데 앞서 말한 티켓의 유효성 검사 및 제품을 안전적으로 지급하는 부분을 본 논문에서 보안하고자 한다.

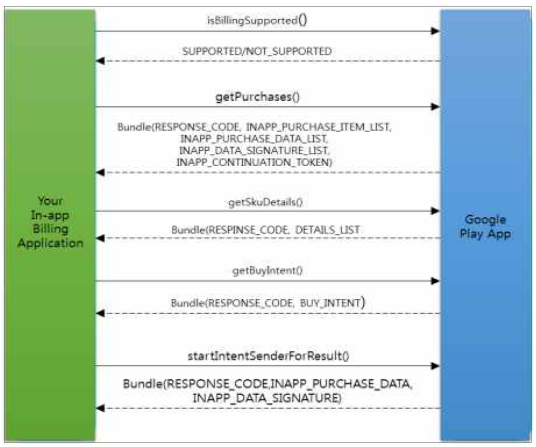

Fig. 3. In-app Billing sequence for a purchase request

Fig 4. In-app Billing sequence for a consumption request

구글 플레이 마켓을 예로 들어 설명하였지만 각 마켓에서 결제 형태는 비슷하게 진행된다. 구글 플레이 마켓이 타 마켓 보다 안정성과 신뢰성이 세계 최고의 수준의 서비스를 제공하고 있다. 하지만 어플리케이션에서 신뢰성 확보 및 결제 과정의 통신 보안, 결제 직후의 제품 전달과정의 보안은 개발사의 몫이다.

2.2 허위 전자 영수증 발행

아래 표은 내부결제 해킹에 가장 사용이 많이 되는 방법이다. DNS를 조작하여 결제 마켓으로 가야되는 결제 요청을 해킹 프록시 서버로 요청이 가게 되고 해킹서버에서는 정상적인 결제가 이뤄지지 않았지만 결제 승인을 보낸다[6].

Table 1. Hacking for In-app Billing

① 내부결제 적용 앱 검색, 루트권환 획득
② /etc/hosts 파일 변조
③ 앱 실행 및 결제 시도
④ Freedom Proxy Server로 결제 요청
⑤ FreeCard-xxx 생성 및 지불 요청
⑥ 마켓 결제 승인
⑦ Freedom Proxy Server에서 마켓 응답 처리 후 앱에 결과 전달

Fig. 5. Hacking process for In-app Billing

1) 내부결제 앱 검색 및 루트 권한 획득

스마트폰 운영체제 iOS와 Android는 운영체제 자체를 해킹이 가능하다[7]. iOS는 탈옥, Android는 루팅이라고 불리는 해킹 방식은 사용자 혹은 타 앱에서 관리자 권한을 취득하는 방식이다. 두 OS 모두 Linux 플랫폼 기반이기 때문에 시스템 파일의 접근 권한을 획득하는 방식으로 시스템 자체의 보안을 제거 할 수 있다.

2) /etc/hosts 파일 변조

그림 6과 같이 루팅 혹은 탈옥을 통해 얻어진 관리자 권한으로 /etc/hosts 파일을 변조한다. 이 파일은 host 파일을 관리하는 파일인데 이 파일을 변조하여 앱 마켓으로 가야되는 결제 요청을 해킹서버로 주소를 변경하게 된다.

Fig. 6. Falsified hosts file

3) 결제 시도 시 FreeCard 생성 및 지불 요청

정상적 결제 요청에는 마켓 서버로 결제 프로세스가 진행되지만 호스트 파일이 변조가 된다면 해킹 Proxy Server로 결제 요청이 진행되게 된다[8]. 다음 그림은 FreeCard가 생성되어 해킹서버에서 FreeCard로 허위 결제 수단을 생성을 하게 되고 그 결제 수단으로 결제가 이뤄지게 된다. 허위 결제 수단이기 때문에 결제는 이뤄지지 않는다.

Fig, 7. FreeCard of App

4) 허위 영수증 발행

위 단계로 결제가 이뤄지게 되면 영수증이 발급이 된다. 이는 허위 결제 수단과 해킹 서버에서 만들어진 영수증이다. 이 영수증을 클라이언트에서만 구별하는 것은 불가능 하다. 허위 영수증에도 원본 영수증과 같이 모든 내용이 동일하게 작성되어 회신되기 때문에 추가적인 검증이 필요하게 된다.

III. Design Authentication System for In-app Billing

1. 앱 내부결제 보안 및 인증 시스템 설계

본 논문에서 제안하는 시스템의 순서도은 그림 8와 같다.

앱에서 사용자가 구매를 진행하게 되면 구매 요청을 IAPG 라이브러리에 구매 정보에 대한 요청이 보내지게 된다. 구매 요청을 받은 IAPG 라이브러리는 IAPG 서버로부터 구매 정보를 받아 오게 되는데 구매정보를 클라이언트에서 보관하지 않고 서버에서 보관하며 요청이 들어왔을 경우에만 정보를 클라이언트에게 전송하여 구매 정보에 관한 보안을 높이게 된다. 서버에서 받아온 구매정보에서 일치하는 마켓으로 결제를 요청하게 되고 그 결과를 라이브러리에서 1차적으로 검증하게 된다. 상품을 지급하는 시점에서 서버가 2차적으로 검증을 하게 되는 프로세스로 2중 확인 작업을 하게 된다. 결제를 진행하며 데이터 중 검증 되지 않는 데이터가 있게 되면 모든 구매 진행은 오류 처리가 되고 상품을 지급하지 않게 된다. 기존 마켓 라이브러리에서는 클라이언트에서 검증을 마치도록 설계가 되어 있지만 IAPG 시스템은 클라이언트와 서버에서 같이 결제 검증을 진행하기 때문에 보안을 높일 수 있다.

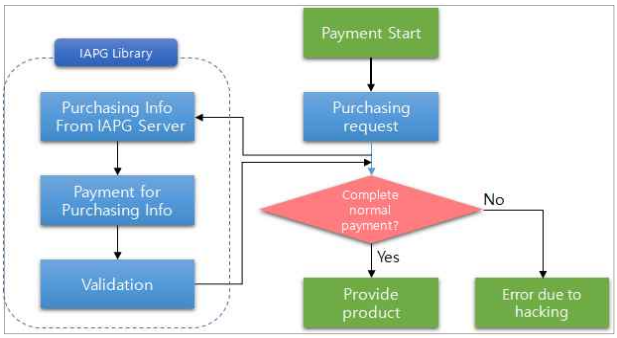

Fig. 8. Proposed In-app Billing Sequence

2. 보안 데이터 흐름

그림 9과 같이 내부 결제에 보안에는 IAPG 서버, IAPG 라이브러리, 앱, 게임서버 4개의 노드로 구성되어 있고 정보가 오가며 검증 절차가 이루어진다.

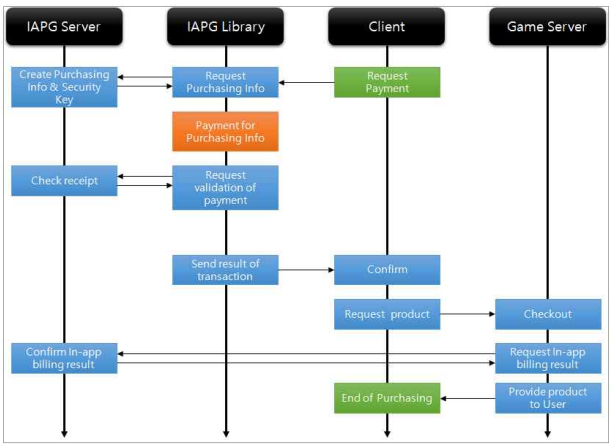

Fig. 9. Data-flow of Proposed In-app Billing

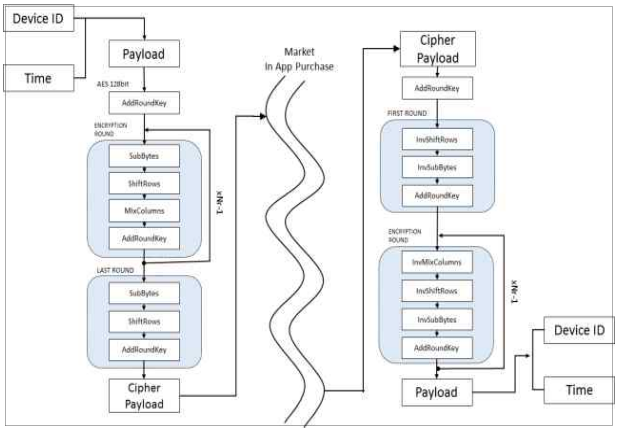

Fig. 10. Process of Payload Creation

IAPG 서버에서는 구매 정보 및 보안키를 발급하여 결제 보안에 인증 절차를 책임지는 역할을 하게 된다. 마켓에 결제 요청은 클라이언트 단에서만 보내야 되기 때문에 다중적으로 보안을 해야 된다. 라이브러리에서 마켓의 결제 결과를 통보를 받은 뒤 마켓에서 발급한 영수증이 위변조가 되지 않는 영수증인지 IAPG에서 검사를 하게 된다. 검사절차는 구매 요청시 생성한 보안키와 돌아온 영수증의 보안키가 일치하는지 검사하여 확인할 수 있다. 마켓에서 정상적으로 결제 성공을 했다면 IAPG 서버에서 게임(구매기록)서버로 상품 지급 요청을 하게 된다. 상품을 클라이언트에서 직접 지급하지 않고 게임서버를 통해 지급을 해야만 안정적으로 지급을 완료 할 수 있게 되는 것이다. 게임서버로 명명한 서버에서는 전자 상품 내역이 관리되는 서버로 게임의 화폐를 보관하는 금고 같은 역할을 한다.

3. 결제의 안정성을 추구할 수 있는 일회용 Payload 생성

보안을 강화하기 위해 그림 10과 같이 앱과 게임에서 사용되는 내부결제에 Payload 항목이 존재한다. Payload의 용도는 결제가 이뤄질 때 시작할 때의 Payload와 결제 승인 후 Payload가 동일한지를 검사를 하여 결제 중간에 위변조가 되었는지를 확인할 수 있는 기능이다. 이 Payload를 결제가 이뤄질 때 마다 같은 값을 사용하여도 되지만 본 과제에서는 결제의 보안을 높이기 위해 일회용 Payload를 생성하여 사용하게 된다. 일회용 Payload를 사용함의 장점은 과거의 승인결과의 영수증의 재사용을 방지하고 금번의 결제를 요청한 사용자인지를 판별할 수 있어 일회용 Payload를 개발하고자 한다. 일회용 Payload는 결제를 요청하는 장치의 Device ID와 현재 시간을 가지고 생성을 하게 된다. 생성한 값을 AES 128bit를 통해 암호화를 하게 되고 암호화된 Payload를 해당 마켓의 결제 요청을 할 때 사용하게 된다. 결제가 끝난 후 결제 승인 영수증에 Payload를 복호화 하여 Device ID와 현재 시간을 대조하는 것으로 프로세스가 종료되게 된다.

NDK 영역에서 암호화와 복호화를 진행하게 되고 Device ID와 Time의 대조하는 프로세스도 NDK 영역에서 진행함으로 안정성을 확보할 수 있다. 플로우 차트의 SubBytes 연상 과정은 암호문이 비선형성을 갖도록 하기 위해 바이트 단위로 역 변환이 가능한 S-Box를 적용하는 것이다. 그 방법은 각 바이트를 GF(2^8) 위의 다항식으로 표현하여 mod 8차 기약다항식 상에서의 역수를 구하는 것으로 확장 유클리드 호제법을 이용하여 수행하게 된다. ShiftRows연산은 state의 각 행 단위로 정해진 수만큼 순환 시프트를 수행하는 것이다. 이 때 state의 0번째 행은 그대로 두고, 1번째 행은 1번,2번째 행은 2번, 3번째 행은 3번 왼쪽으로 순환 시프트를 수행하게 된다. 각 라운드 끝에는 AddRoundKey 연산을 하게 되는데 이 연산은 라운드 키와 현재 state를 비트 단위로 XOR를 수행하는 과정이다. 그러므로, 암호화 과정의 state와 라운드 키는 동일한 크기를 가지며, 1라운드를 수행하기 전에 초기 평문과 라운드 키를 XOR하는 과정이 필요하므로, AddRoundKey 연산은 전체 암호화 과정에서 Nr+1번을 수행하게 된다. 그리고 복호화 과정은 암호화의 역으로 진행되게 된다. 이런 Payload를 암호화 하는 것은 결제를 요청하는 과정에서 각 플랫폼에 데이터가 올라가는 시점과 통신 시점에서 해커에게 Payload값이 유출될 수 있기 때문에 결제 인증키인 Payload를 암호화하는 것은 필수 사항이다.

4. 전자 영수증 검증

아래 그림은 마켓에서 발행하는 전자 영수증이다. 마켓에서 결제가 완료 후 발급되는 전자 영수증은 Order ID에 판매자의 전자지갑 번호가 포함이 되어 있다. 전자지갑 번호가 개발사의 전자지갑 번호인지 확인하며 전자 지갑번호 뒤에는 영수증 번호가 기입되어 있다. IAPG서버에서 영수증 고유번호를 기존에 처리된 이력이 있는지를 확인하고 있다면 영수증 재사용으로 판단하여 부정결제 결과를 반송한다. 사용 이력이 없다면 신규 거래로 확인하고 IAPG 서버의 전자 영수증 보관함에 추가 한다. Package Name으로 개발사의 앱에서 발행된 영수증인지를 확인하고 다르다면 부정결제로 판별한다. Developer Payload의 필드를 IAPG만의 보안 일회용 Payload로 사용한다.

Developer Payload에 결제 시작에 발급된 일회용 Payload가 위변조 없이 회신되었는지를 확인하게 된다. 회신된 Payload를 복호화 후 시간과 기기 고유번호 까지 확인하여 정상 결제인지를 검증한다. 최종적으로 개발사에서 IAPG 서버에 입력한 Public Key로 전자 영수증 전체를 서명하고 서명된 값과 전자 영수증 마지막에 첨부된 Signature와 일치하는 지를 검사하게 된다. 위 과정을 모두 통과를 하게 되면 정상결제로 확인하고 상품지급을 허용하게 된다.

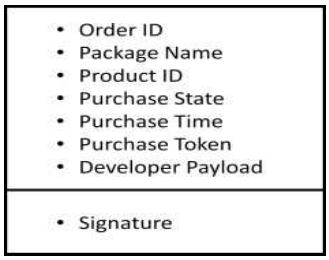

Fig. 11. Electronic Receipt

5. 앱(클라이언트) 위변조 검사

애플리케이션 위변조 검증 방법은 애플리케이션 실행 시점에 애플리케이션의 위변조 여부를 탐지하는 방법이다. 그림 12은 그 프로세스를 보여주는 그림이다.

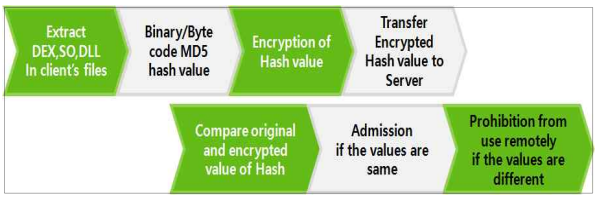

Fig. 12. Detection process for Forgery prevention

애플리케이션에서 실행 시점에 애플리케이션 실행 파일인 DEX,SO,DLL파일등을 추출하여 바이너리/바이트 코드를 해쉬값을 생성을 한다. 생성한 해쉬 값을 서버에서 알려준 공개키로 암호화를 하고 서버에 전송을 하게 된다. 서버에서는 전송 받은 암호화된 해쉬값을 복호화하여 서버에서 보관한 올바른 해쉬값과 비교하여 위변조 여부를 판단하게 된다. 일치 한다면 애플리케이션이 위변조가 되지 않았다고 판단하고 불일치 한다면 애플리케이션이 위변조가 되었다고 판단하여 원격으로 애플리케이션 사용을 차단하게 된다. 이는 자사가 보유한 특허의 내용이기도 하다. 본 과제의 시스템에서는 해쉬 함수를 MD5로 생성을 하게된다. 학회에서는 MD5가 안정성이 떨어져 SHA 해쉬 알고리즘을 권장하지만 본 시스템에서는 대용량의 데이터인 실행파일의 해쉬값을 추출하여야 되기 때문에 안정성이 떨어지지만 속도에서 뛰어난 MD5를 사용하여 설계를 하였다. 앱의 실행 파일의 구성요소인 DEX파일에 class파일들로 구성되어 있는데 이것들이 앱 실행 파일들이다. 안드로이드 플랫폼에서는 /data/app, /system/app, /dalvik-cache 아래 존재하고 되는데 이 파일 접근은 owner나 system group에게만 접근이 허용된다. 보통 실행 파일의 위변조는 설치된 파일을 변조하는 것이 아니라 APK를 풀어 DEX를 추출하고 dex에서 class파일 추출하게된다. 그리고 class를 디컴파일 하면 java파일을 손쉽게 구할 수가 있다. 획득한 java파일을 수정 후 역순으로 파파일을 하여 class을 만들고 dex메이커로 dex로 만든후 압축을 하면 APK로 변환 되게 된다. 하지만 java파일을 수정, 추가, 삭제를 하게 되면 dex의 해쉬 값이 다르게 생성이 되게 된다. 그래서 본 시스템에서 해쉬 값으로 무결점성을 확인하는 것이다.

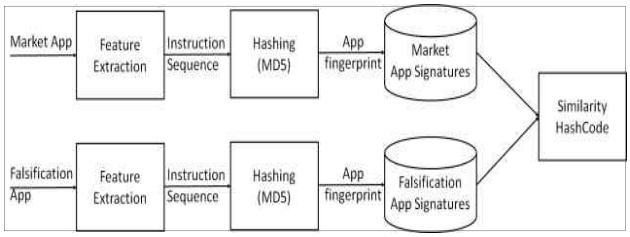

Fig. 13. Comparison of Hash value

앞서 말한것과 같이 안드로이드 플랫폼 마켓은 다양하게 있다. 그 중 몇몇 마켓에서는 개발사에서 업로드한 APK를 마켓에서 다시 패키징을 하는 경우가 종종 있다. 그래서 처음 개발사에서 만든 APK가 변경이 되는 경우가 있는데 그림 13과 같이 본 시스템의 해쉬값 추출로 위변조를 판단하는 방법에서는 이를 또한 위변조로 판단하고 차단하게 된다. 그래서 본 시스템에서는 APK 버전 따라 생성한 최초의 해쉬 값을 저장하게 되며 다수의 해쉬값의 애플리케이션이 원본 해쉬값으로 판단하게 된다. 만약 변조된 앱에서 해쉬값을 등록을 하더라도 비교 후 변조로 판명된 해쉬값을 가진 앱들은 모두 원격 사용 차단을 시키게 된다.

6. 판매내역관리 및 마켓판매내역 대조 툴

내부결제로 판매된 상품의 종류 및 수량 등을 앱 사용자 ID와 연결하여 관리한다. 개발사에서 사용하는 사용자 ID와 영수정보를 연결하여 원하는 정보를 쉽게 조회 가능한 어드민 툴을 제공하고자 한다. 기존 마켓에서 판매내역을 제공하지만 판매내역과 앱에서 사용하는 사용자 ID가 연관 지여지지 않기 때문에 개발사에서 판매관리에 어려움을 격고 있다. 그림 14처럼 클라이언트에서 IAPG SDK로 결제를 요청할 때 사용자 ID를 같이 받고 내부결제 프로세스를 진행 후 정상적으로 수행 되면 사용자 ID와 결제 결과를 함께 보관한다. 그래서 사용자가 개발사로부터 환불을 요청할 때 사용자 ID로 결제내역을 쉽게 조회하여 결제를 환불 해줄 수 있게 된다. IAPG에서 보안을 완벽하게 하더라도 혹시 고려되지 않은 새로운 해킹 수법이 발생하여 내부결제 보안의 결점이 생기더라도 마켓의 실제 결제 내역과 상품지급 내역을 대조하는 기능으로 부정 결제 사용자를 쉽게 색출할 수 있게 된다. 해킹 사고의 사후 관리 기능이다.

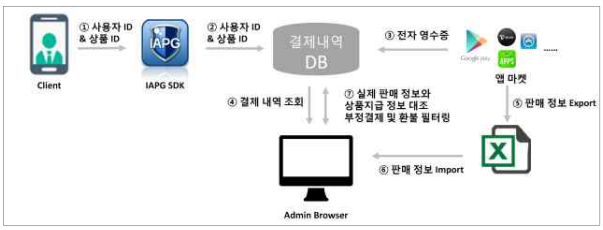

Fig. 14. Tool for check user ID and Sale-list

IV. Performance Comparison

1. 시험 장비 및 소프트웨어 구성

본 논문에서 제안하는 인증 및 보안 시스템의 구성과 구동 방식에 관한 실험을 표 2와 같은 환경에서 진행하였다. 앱 마켓은 앱 마켓 중 가장 진보한 내부결제 API를 제공하는 구글 플레이 스토어 마켓을 사용하였으며 시험 장비는 스마트폰 갤럭시 시리즈 S3, S6과 서버 2008과 MySQL, PHP, Apache를 사용하였다. 두 테스트 단말장비는 하드웨어 사양도 다르지만 안드로이드 버전 또한 다르다.

2. 해킹 시도 및 방어 실험

2.1 해킹 툴 Lucky Patcher로 일반 앱 해킹

실험 도구로 해킹툴 중 가장 최신 버전이며 진보적인 해킹 방식을 가지고 있는 Lucky Patcher를 사용하였다[9]. Lucky Patcher은 설치 된 앱을 다시 APK를 추출하여 변조 후 다시 패키징을 하게 된다. 그래서 광고 삭제와 인앱 결제 무력화, 퍼미션 변경까지 가능하다. Lucky Patcher와 같이 GUI로 제공되는 해킹툴을 사용한다면 일반 사용자들도 손쉽게 APK를 변조할 수 있게 되는 것이다.

Table 2. Experiment Environment

App market	Play Store(Google) In App Billing Version 3 API	
Test Device1	Device	Samsung Galaxy S6 Edge G925
	OS	Android Lolipop 5.1.1
Test Device2	Device	Samsung Galaxy S3
	OS	Android Kitkat4.4.4
Server Spec	OS	Windows Server 2008 R2 Standard
	DB	MySQL 5.6.20 CommunityServer
	Web language	PHP 5.6.0
	Web server	Apache HTTPD Server 2.4.10

그림 15과 같이 정상적으로 구동되는 'Prize claw 2' 게임 앱을 그림 16과 같이 실험 도구인 Lucky Patcher로 해킹을 시도해 보았다. 이와 같이 정상적인 게임의 상태를 Lucky Patcher을 이용하여 내부결제를 무력화 할 수 있게 된다.

Fig.15. 'Prize claw2'

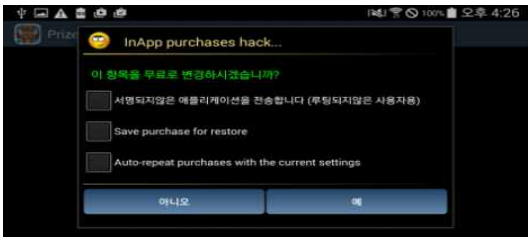

Fig. 16. Trial of Hacking with 'Lucky Patcher'

위 그림은 Lucky Patcher로 해킹을 진행하는 화면이다. 앱에 서명을 제거함으로 루팅되지 않은 기기에서도 내부결제를 무력화시킬 수 있게 할 수 있다. 이러한 방법으로 해킹을 진행하게 되면 리패키징된 APK를 설치하게 된다. 다음 그림에서와 같이 해킹이 완료가 되면 모든 내부 결제 항목이 지불 없이 구입이 가능해 지게 되는 것을 확인할 수 있다.

Fig. 17. Success of Hacking with 'Lucky Patcher'

2.2 해킹방지적용 앱을 해킹툴로 해킹 시도

본 시스템인 앱 내부결제 보안 시스템이 적용된 시험 앱을 제작 하였다. '보석 구매 결제창'은 일회성 내부결제 상품이며, '프리미언 구매 결제창', 영구 내부결제 상품을 나타내고 있다.

Fig. 18. Trial of Hacking with 'Lucky Patcher' on proposed system

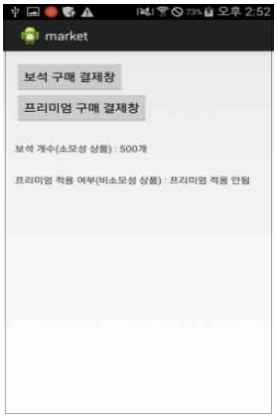

Fig 19. Fail of Hacking with 'Lucky Patcher' on proposed system

그림 18과 그림 19에서 확인한 것처럼 해킹 시도를 하였음에도 결제가 이뤄지지 않은 것을 확인 할 수 있다.

2.3 결제 인증 서버 데이터베이스 레코드

본 실험에서 약 10,000건의 결제를 시도하여 결제 상태 여부와, 일회용 페로드, 결제 의사 표현 시간, 결제가 실제 시작 시간, 결제 프로세스 완료 시간 등을 기록 하였다. 약 10,000건의 시도를 (1)결제가 정상적으로 진행된 경우, (2)결제를 시도하였지만 중간 취소 및 (3)부정 결제 시도 이렇게 3가지 형태의 결제 유형으로 구분하였다.

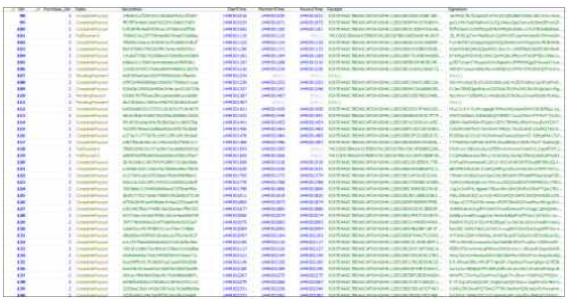

Fig. 20. DB record for In-app Billing

그림 21에 나타난 바와 같이 결제 승인 추가 소요시간은 평균 0.8916초로 나타났다. 이 시간은 보안 및 인증 시스템을 적용하지 않은 시스템에서는 이 시간이 소비 되지 않지만 보안 및 인증 프로세스를 진행하면서 소비 되는 시간이다.

결제 승인 추가 소요시간(초)	결제대기	지급대기	지급완료	부정결제시도	부정결제 시도 비율
0.8916	1334	314	7324	801	10.9366%

Fig. 21. Extraction Data for In-app Biling

이 프로세스 완료 시간은 통신 환경에 따라 달라지며 최소 0.7531초에 최대 2.8515초가 걸리는 것을 확인 하였다. 약 10,000건 중 부정 결제 시도는 801건으로 10.9366% 비율을 차지하였고 모든 부정 결제를 차단하였다.

V. Conclusions

본 논문에서는 스마트폰 앱 구조인 APK에서 앱 내부결제 해킹 방법들을 분석하여 그러한 해킹에 대한 방어를 할 수 있게 하는 시스템을 제안하였다.

스마트폰 애플리케이션에서 주 수익 형태가 초기에는 유료 앱 판매였으나 수익의 증가와 편의 사항의 이유로 내부결제 방식으로 결제 방식이 변형되었다. 이와 같이 결제 방식이 변경됨에 따라 보안의 문제도 발생하였다. 이를 본 논문에서는 새로운 보안 시스템을 제안하였으며 그 결과 앱 개발에 있어 결제 수단을 안정적으로 구현할 수 있다.

안드로이드 플랫폼의 특성인 가상머신 구동 방식은 여러 장점들이 있지만 역공학에 취약하다는 단점을 가지고 있다. 그래서 해커들은 이 역공학 방식으로 앱을 공격해 내부결제를 무력화를 시켰다. 하지만 본 논문에서는 앱의 변조를 검사하는 프로세스를 제안함으로 이 역공학 방법의 해킹을 방어 할 수 있게 하였다. 또한 내부결제 서버의 DNS를 조작한 해킹 방식도 클라이언트에서 결제를 진행하는 것이 아니라 서버에서 추가적인 인증을 통해 DNS를 조작하는 외부 해킹 방법 또한 방어 할 수 있게 된 것이다. 내부 결제의 해킹을 무력화 하는 방법으로 내부와 외부 공격 방식을 방어하는 시스템을 제안함으로 결제 수단인 내부 결제 프로세스를 안정적으로 제공할 수 있게 되었다. 해킹툴을 이용해 해킹을 시도한 실험에서 본 논문의 시스템이 적용되지 않은 앱은 해킹의 위험에 노출이 되었지만 제안한 인증 시스템을 적용한 앱은 해킹이 되지 않음을 확인 할 수 있었다. 이러한 방법에서 추가적인 인증 프로세스가 실행되기 때문에 프로세스 진행 시간이 증가되는 것은 필연적이다. 하지만 증가되는 시간은 약 10,000건을 시도하여 평균 추가시간을 구했을 때 건당 0.8916초로 비교적 짧은 시간이 증가되었음을 확인 할 수 있었다.

REFERENCES

[1] Ho-Jun Seok, Sung-Min Hwang, Seog-Gyu Kim. "An Authentication Sever System for In-App Purchase". Proceedings of KSCI Conference 2015, July 2015.

[2] Wonnam Lee. "A Study on the Android Vulnerability of Rooting/App Integrity Verification Module". Dec. 2014.

[3] Yuxue Piao, Jin-hyuk Jung, Jeong Hyun Yi. "Structural and Functional Analyses of ProGuard Obfuscation Tool". The Journal of Korea Information and Communication Society, Vol.38B No.08, August 2013.

[4] Google. "In-app Billing Security and Design".

http://developer.android.com/google/play/billing/billing_best_ practices.html

[5] Google. “In-app billing Version3 API”. http:// developer.android.com/google/play/billing/api.html

[6] Jordan Kahn. "Apple’s in-app purchasing process circumvented by Russian hacker". http://9to5mac.com/2012/07/13/apples-in-app-purchasing-process-circumvented-by-russian-hacker/

[7] Mulliner Collin, William Robertson, and Engin Kirda. "Virtual Swindle:an automated attack against in-app billingon android", Proceedings of the 9th ACM symposium on Information computer and communications security, June 2014.

[8] Jinsun Hong. “ Mobile Game Hacking”. http://www.inven.co.kr/webzine/ news/?news=128022

[9] Crossdotcom. “Lucky Patcher”. http://www.kgezzang.tk/173

Authors

Ho-Jun Seok received the B.S., M.S.. degrees Department of Information & Communication Engineering, Andong National University in 2011 and 2016. He has been working as reseacher in Nibens Company, Korea, since 2010.

His research interests include mobile computing, network security and system security.

Seog-Gyu Kim received the B.S., M.S.. and Ph.D. degrees in Electronic Engineering from Yonsei University, Korea, in 1990, 1992 and 1997, respectively

Dr. Kim worked as senior researcher in SK Telecom from 1997 to 2004. He joined the faculty of the Department of Information & Communication Engineering, Andong National University in 2006. He is currently a Professor in the Dept. of Information & Communication Engineering, Andong National University. He is interested in next-generation network and mobile computing, and IoT.