모바일 어플리케이션 바이너리 보안에 관한 연구

민재원*, 정성민*, 정태명**
성균관대학교 전자전기컴퓨터공학과*
성균관대학교 정보통신공학부**
e-mail :{jwmin,smjung} @imtl.skku.ac.kr*, tmchung@ece.skku.ac.kr**

A Study on Securing Binaries of Mobile Applications

Jae-Won Min*, Sung-Min Jung*, Tai-Myoung Chung**
Dept. of Electrical and Computer Engineering, Sungkyunkwan Univ*
School of Information Communication Engineering, Sungkyunkwan Univ. **

요 약

스마트폰이 대중화되면서 모바일 시장의 규모가 급격하게 발전했다. 애플의 앱스토어에 등록된 어플리케이션의 숫자는 이미 50 만개를 돌파했고 안드로이드 마켓에 등록 숫자도 25 만개를 넘었다. 하지만 모바일 시장의 발전과 더불어 불법복제도 증가하기 시작했다. 애플의 아이폰의 경우, 탈옥을 하면 Cydia 라는 마켓을 통해서 공짜로 어플리케이션을 다운받을 수 있고 안드로이드는 불법 복제된 어플리케이션을 디바이스에 다운받아 실행하면 유료 어플리케이션을 공짜로 설치할 수 있다. 이러한 불법 복제를 막기 위해서 애플과 구글은 각각 DRM 시스템을 구축했다. 하지만 이들의 시스템의 문제점은 어플리케이션의 바이너리에 대한 보안이 올바르게 이루어지지 않았다는 점이다. 따라서 본 논문에서는 모바일 어플리케이션의 바이너리의 보안을 연구하여 기존 DRM 보다 더 나은 시스템을 제안한다. 새로운 시스템은 여러 보안 계층을 설계하여 의미 있는 바이너리의 추출을 막고 결과적으로 어플리케이션의 불법 복제의 숫자를 줄일 수 있다.

1. 서론

스마트폰의 판매 숫자가 빠른 속도로 증가하면서 모바일 시장이 역시 크게 성장했다. 스마트폰과 태블릿 PC 같은 모바일 디바이스들은 어플리케이션을 온라인 스토어에서 구입하여 다운로드 받는다. 현재 애플의 앱스토어의 등록 어플리케이션 개수는 50 만개를 넘었고 구글의 안드로이드 마켓 또한 25 개를 초과했다.

하지만, 모바일 시장이 발전하면서 반대로 불법 복제 또한 증가하기 시작했다. 애플의 아이폰의 경우, 루트 권한을 획득하는 탈옥과정을 거치면, Cydia 라는 어플리케이션 스토어를 사용할 수 있다. 이 곳에서는 정식 앱스토어에서 유료로 구입해야 하는 어플리케이션들을 무료로 다운받을 수도 있고, 정식 앱스토어에 등록되지 않은 어플리케이션들도 등록되어있다. 안드로이드에서는 루트 권한을 획득하는 것을 루팅이라고 하는데 안드로이드 기기는 루팅을 할 필요도 없이 인터넷에서 불법 복제된 어플리케이션 설치파일을 다운받아 파일을 옮겨서 실행을 하면 공짜로 사용할 수 있다.

이와 같은 불법 복제를 막기 위해서 모바일 어플리케이션 스토어 별로 DRM 시스템을 구축했지만, 바이너리 수준의 보안이 완벽하게 이루어지지 않았고, 불법 복제 어플리케이션은 여전히 인터넷을 통해서 유포되고 있다. 따라서 본 논문에서는 어플리케이션의 바이너리 레벨의 보안 시스템을 제안한다.

본 논문의 구성은 다음과 같다. 2 장에서는 바이너리 보안 기술과 현재 모바일 어플리케이션 스토어에서 사용하고 있는 DRM 시스템을 분석하고 3 장은 본 논문에서 제안하는 바이너리 보안 시스템의 계층 구조와 각 계층의 특징을 설명한다. 4 장 결론에서는 내용을 정리하고 향후 연구 방향을 설명한다.

2. 관련 연구

2.1 바이너리 보안

안티 디버깅은 디버거로 바이너리를 분석하는 것을 방해하는 기술이다. 디버거가 현재 실행 중인지 확인하는 방법으로는 다음 표와 같이 나눌 수 있다[8].

API based	시스템 정보를 이용
Exception based	Exception 방해 여부
Process and Thread Block	프로세스/쓰레드 블록 수정 여부
Modified code	코드의 수정 여부
Hardware and register based	CPU 레지스터와 하드웨어 브레이크포인트 확인
Timing and latency	명령어 실행에 소요된 시간 확인

<표 1> 안티 디버깅 기술

코드 난독화는 분석가가 바이너리를 이해하기 어렵게 하는 기술이다. 보안의 목적으로 코드의 의미를 숨기거나 프로그램의 불법적인 수정을 막기 위해 주로 사용된다. 의미 없는 코드를 포함시키거나, 코드를 컴파일할 때 심볼 정보를 삭제하는 등 여러 가지 방법이 있다. 코드 난독화의 장점은 암호화된 것처럼 보이지만 그대로 실행이 된다는 점이고, 단점은, 시간을 지연시킬 뿐 결국 언젠가는 분석이 된다는 점이다[2].

패킹된 프로그램은 파일의 텍스트 섹션이 암호화되어 저장된다. 프로그램이 실행되면, 암호화되어 있던 텍스트 영역이 임시적으로 메모리에 복호화된다. 패킹은 바이너리 보안뿐만 아니라 악성코드가 자신이 분석되는 것을 막기 위해서도 사용된다. 패킹의 단점은 파일이 복호화되어 메모리에 저장된다는 점이지만, 패킹을 여러 번하거나 알려지지 않은 패킹 알고리즘을 사용한다면 보다 더 효과적으로 분석을 방해 할 수 있다.

2.2 모바일 DRM

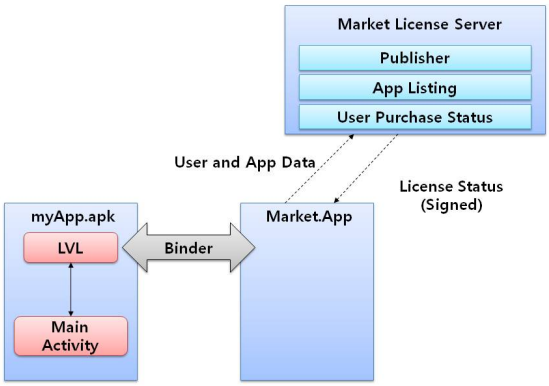
2.2.1 애플 Fairplay

애플은 Fairplay DRM 시스템을 사용하고 있다. 원래는 iTunes 의 음악에 적용을 했던 시스템이지만, 아이폰이 출시되고 앱스토어가 생기면서 일반 어플리케이션을 암호화하는 용도로 확장되어 사용하게 되었다.

앱스토어는 기본적으로 모든 어플리케이션을 암호화하여 배포한다. 사용자는 어플리케이션을 다운받고 실행을 하면, 애플의 모바일 운영체제인 iOS 내부의 어플리케이션 로더가 어플리케이션을 복호화해서 실행을 한다[3].

2.2.2 안드로이드 마켓 DRM

안드로이드 마켓은 애플의 Fairplay 와는 다른 방식으로 DRM 을 구축하였다. 애플은 어플리케이션을 암호화해서 배포하지만, 안드로이드 마켓은 어플리케이션을 암호화하지 않는다. 안드로이드 마켓의 인증과정은 (그림 1)과 같다.

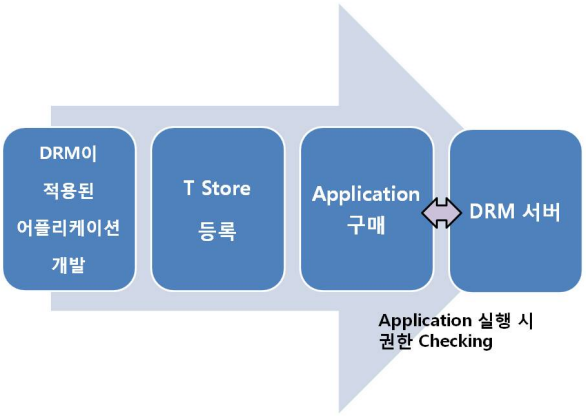


(그림 1) 안드로이드 마켓 DRM[5]

안드로이드 디바이스는 신뢰할 수 있는 라이선스 서버에 인증 요청을 보낸다. 서버는 사용자가 해당 어플리케이션의 구매 기록이 있는지 또는 어플리케이션이 무료인지를 확인을 하고 인증 결과를 사용자에게 전송한다. 라이센스 정보를 올바르게 확인하기 위해서는 사용자 정보와 어플리케이션 정보가 필요하다. 따라서 어플리케이션과 안드로이드 마켓 클라이언트는 정보를 취합해서 서버에 전송을 한다. 이때, 어플리케이션이 라이센스 서버에 직접적으로 요청을 보내지 않고 원격 IPC 로 안드로이드 마켓 클라이언트를 호출해서 전송한다. 안드로이드 마켓 클라이언트가 어플리케이션보다 높은 권한을 가지고 있기 때문에 사용자 정보와 디바이스 정보를 가져와 어플리케이션을 대신해 서버로 전송을 한다. 라이센스 서버와 사용자 사이에서 주고받는 데이터는 RSA 알고리즘으로 암호화가 되어있다[5].

2.2.3 T Store Application DRM

SK 의 T Store 는 안드로이드 마켓과 비슷한 구조의 DRM 시스템을 사용한다. T Store 는 개발자에게 각각의 플랫폼에 맞는 DRM 라이브러리를 개발자 사이트를 통해서 제공한다. 하지만 라이브러리를 사용하는 것은 의무가 아니다. T Store 가 제공하는 라이브러리의 적용 여부는 개발자가 판단한다. 개발자는 어플리케이션 ID 를 발급받고 DRM 라이브러리를 적용하고 자체 툴을 이용하여 검증을 한 후에 어플리케이션을 등록한다. 다음 그림은 T Store DRM 의 동작 방식을 설명한 것이다.



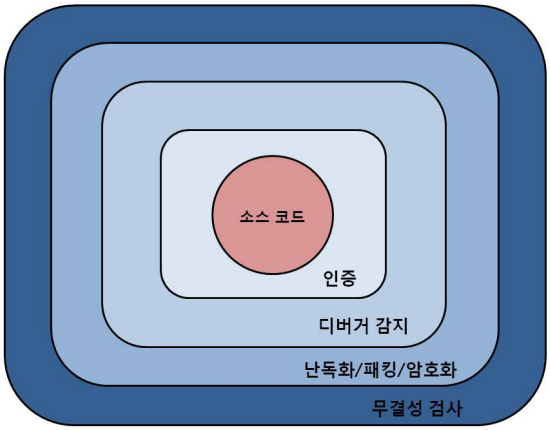
(그림 2) T Store 동작 방식

어플리케이션이 T Store 에 등록된 후, 사용자가 구매를 하면, 실행할 때 마다 DRM 서버와 통신을 해서 해당 어플리케이션의 불법 복제 여부를 확인한 후에 결과 값을 다시 클라이언트로 전송한다.

T Store 도 안드로이드 마켓과 마찬가지로 어플리케이션 고유 정보를 통해서 인증을 하지만, 실행 파일의 바이너리를 보호하지 않기 때문에 쉽게 불법 복제가 가능하다[4].

3. 본론

어떤 정보를 보호하려고 할 때 여러 계층의 보안을 사용하는 것은 한 가지의 보안 계층을 사용하는 것보다 더 많은 장점을 가진다. 여러 계층을 사용하면, 공격자가 한 계층을 우회해도 다른 계층은 동작하고 있기 때문에 상대적으로 더 안전하다. 아래 그림은 바이너리 보안 시스템의 계층 구조이다.



(그림 3) 계층 구조

3.1 인증

어플리케이션이 구매자의 디바이스 외에 곳에서 설치되는 것을 막기 위해서는 인증 절차가 필요하다. 인증 절차가 시작되면, 어플리케이션의 고유 정보, 디바이스 정보와 사용자 정보를 취합해서 암호화하여 DRM 서버에 전송한다. 인증 요청을 받은 DRM 서버는 실제로 인증을 요청한 사용자가 해당 어플리케이션을 구매했는지 여부를 구매목록에서 확인하고, 인증 결과 값을 다시 전송한다. 개발자는 인증 결과값에 따라 어플리케이션의 실행을 제한한다. 인증 절차는 어플리케이션 안에 코드로 구현되기 때문에 다른 보안 계층에서 코드를 수정하기 전에 선행되어야 한다.

3.2 디버거 탐지

GDB 는 널리 사용되는 디버거이다. 크래커들은 GDB 와 같은 디버거를 통해서 어플리케이션을 실행시킨 다음, 바이너리를 분석하거나 복호화된 코드를 메모리로부터 파일로 덤프할 수 있다. 덤프 파일을 얻게 되면, 리패키징을 해서 토렌트 사이트 등에 배포를 할 수 있다. 따라서 디버거를 통해서 어플리케이션을 실행하는 것을 막는 것은 매우 중요하다. Mac OS X 의 경우 디버깅에 사용되는 ptrace 시스템콜의 사용을 제한할 수 있는 기능이 있지만, 애플의 모바일 운영체제인 iOS 는 sys/ptrace.h 파일이 없기 때문에 기능을 구현해야 한다. 반면에 안드로이드의 경우 비교적 쉽게 디버깅을 막을 수 있다. Manifest file 에 있는 android:debuggable 속성을 false 로 설정하면 된다. 그 외에도 디버거의 특성을 이용한 탐지 방법, SIGTRAP 시그널을 사용하는 방법 등 여러 가지 방법들이 있다 [7]. 플랫폼에 독립적인 시스템을 만들기 위해서 특정 플랫폼의 API 를 사용하지 않고 흔히 사용하는 디버거들이 전송하는 특정 시그널이나 삽입하는 명령어 등을 추출하여 이들을 탐지한다. 디버거 탐지 코드는 어플리케이션의 도입부에 삽입을 하고, 탐지 결과에 따라서 어플리케이션의 실행을 제한한다.

3.3 소스 코드 보안

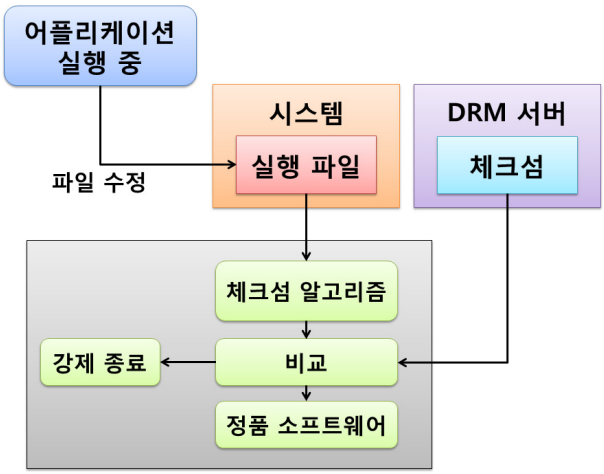
모바일 어플리케이션 보안에 있어서 소스 코드가 차지하는 비중은 크다. 만약, 크래커가 어플리케이션의 소스 코드를 자유롭게 패치 할 수 있다면, 다른 보안책은 무용지물이 될 수 있다. 예를 들어 인증 코드를 패치하여 항상 참으로 실행 흐름을 만들어버리면 어떤 디바이스에서도 실행이 될 수 있다. 또한 디버거 탐지 결과를 항상 거짓으로 하여 디버거를 탐지하지 못하게 변조할 수도 있다. 따라서 코드의 분석을 방해해야 한다. 우선 인증과 디버거 탐지가 구현된 어플리케이션 코드를 난독화한다. 난독화는 다양한 방법으로 할 수 있는데, 대표적으로 소스코드 레벨 또는 어셈블리어 레벨 난독화가 있다[7]. 두 가지 방법 모두 장단점이 있지만 본 시스템에서는 공격자가 어셈블리어를 분석하는 상황을 가정하기 때문에 어셈블리어 레벨의 난독화를 한다. 난독화를 한 이후에 실행파일을 패킹한다. 만약 파일이 언패킹 되더라도 난독화된 코드이기 때문에 최소한 평문 코드는 노출되지 않는다.

3.4 무결성 검사

소스 코드를 난독화하고 패킹을 해도 완벽하게 모바일 어플리케이션 바이너리가 불법 복제로부터 안전한 것은 아니다. 만약 디버거 탐지를 실패할 경우 앞에서 설명했듯이 메모리를 덤프하거나, 실행파일을 수정할 수 있다. 실제로 애플의 DRM 인 Fairplay 는 복호화된 코드를 덤프해서 원래 실행파일에 덮어 쓰는 방식으로 크랙할 수가 있다. 또한 정적 분석 도구를 사용하면 디버거 탐지 코드가 포함되어 있어도 역어셈블된 코드를 읽을 수 있고 패치를 할 수도 있다. 따라서 디버거 탐지가 우회가 되어 실행 파일이 변조되는 것을 막는 보안 계층이 필요하다.

실행 파일이 패치되는 것을 방지하기 위해서는 체크섬을 확인해야 한다. 체크섬을 확인할 때는 확인 시점이 중요하다. 예를 들어 실행 파일을 변조하는 도중에 체크섬을 검사하면 크래킹 과정을 탐지할 수가 있지만, 반대로 체크섬 검사 시점이 노출되어 체크섬 검사가 실행되기 전이나 후에 크랙을 하고 시스템 설정을 변경하여 체크섬 검사를 무효화시킨다면 막을 방법이 없다. 따라서, 체크섬 확인 시점을 랜덤하게 하면, 크래커는 그 시점을 예상할 수가 없어 파일 변조를 탐지할 가능성이 높아진다. 체크섬 검사 루틴이 어플리케이션 내부에서 구현되면 역시 검사

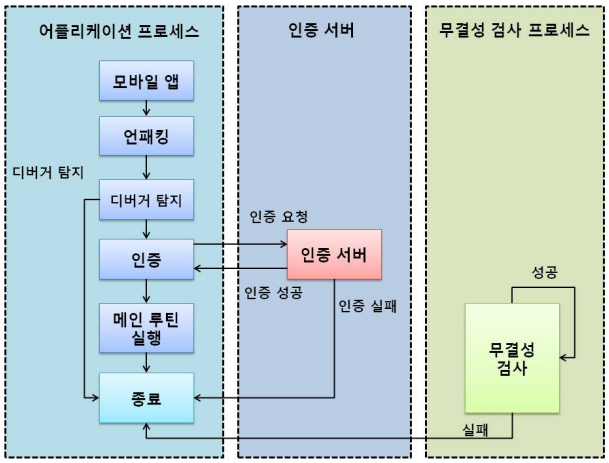
루틴이 수정될 가능성이 있기 때문에 시스템에서 독립적인 프로세스를 실행시켜 랜덤한 시간에 어플리케이션을 검사해야 한다. 어플리케이션의 체크섬은 실시간으로 DRM 서버로부터 다운로드를 받아 사용하고, 시스템 내부에 저장되지 않는다. 무결성 검사 과정은 다음 그림과 같다.



(그림 4) 무결성 검사

3.5 시스템

다음 그림은 전체적인 보안 시스템의 프로세스이다.



(그림 5) 시스템 프로세스

모바일 개발자는 자신의 어플리케이션을 스토어에 올린다. 어플리케이션에 문제가 없다고 판단되면, 서버와 인증하는 코드, 그리고 디버거를 통해서 어플리케이션을 실행하는 것을 방해하는 코드를 개발자의 코드에 포함시켜서 규정에 맞는 알고리즘으로 코드를 난독화한다. 그 다음 단계에는 어플리케이션 파일을 패킹하고 어플리케이션 스토어를 통해서 어플리케이션과 체크섬을 배포한다. 해당 어플리케이션을 다운받은 사용자는 자신의 기기에 설치를 하고 실행을 한다. 모바일 디바이스에서는 체크섬 프로세스가 독립적으로 백그라운드에서 실행이 되면서 랜덤한 시간에 실행 중인 어플리케이션 중 하나를 선택해서 검사를 한다.

크래커가 어플리케이션을 불법복제를 하려면 (그림 5) 와 같이 여러 가지 계층의 보안 정책을 우회해야 하기 때문에 크랙을 하기가 어렵다. 어플리케이션이 실행되면, 바이너리가 언패킹되어 실행된다. 만약 디버거로 파일이 실행되었다면, 디버거 탐지 코드가 이를 발견하고 어플리케이션을 종료시킨다. 디버거가 탐지되지 않았다면 인증 서버를 통해서 디바이스 인증을 받고 어플리케이션의 메인 코드를 실행한다. 무결성 검사는 랜덤한 시간에 실행되며 파일이 변조되었으면 바로 어플리케이션을 종료한다.

4. 결론 및 향후 연구

모바일 시장이 성장하면서 반대로 모바일 어플리케이션의 불법 복제 또한 증가했다. 대표적인 스마트폰인 아이폰의 경우, 탈옥을 거쳐서 Cydia 같은 어플리케이션 스토어를 통해서 유로 어플리케이션을 무료로 설치할 수 있고, 안드로이드폰의 경우, 인터넷에 공유되고 있는 크랙된 어플리케이션을 설치할 수 있다. 기존의 DRM 시스템은 바이너리에 대한 보안을 하지 않아 쉽게 불법 복제를 할 수 있었다.

따라서 본 논문에서는 어플리케이션 바이너리를 불법 복제 및 수정으로 보호할 수 있는 이론적 모델을 제안하였다. 향후 연구에서는 실제로 시스템을 구현하여 검증을 하고, 상용화 수준으로 시간 오버헤드를 줄이는 방법을 연구해야 한다.

참고문헌

[1] “Bugging Debuggers”, http://theiphonewiki.com/wiki/index.php?title=Bugging_Debuggers, Aug.2011

[2] “Obfuscated Code”, http://en.wikipedia.org/wiki/Obfuscated_code, Aug.2011

[3] Ramya Venkataramu, ”Analysis and Enhancement of Apple’s Fairplay Digital Rights Management”, San Jose State University, May.2007

[4] “T store 개발자센터” http://dev.tstore.co.kr/devpoc/guide/guideAd.omp, Aug.2011

[5] ”Application Licensing” http://developer.android.com/guide/publishing/licensing.html, Aug.2011

[6] “How the Crack Works” http://thwart-ipa-cracks.blogspot.com/2008/10/how-crack-works.html, Aug.2011

[7]Andrew Griffiths,”Binary Protection Schemes”, Code Breaker’s Journal 2005

[8]Min-Jae Kim, Jin-Young Lee, Hye-Young Chang, SeongJe Cho, Yongsu Park, Minkyu Park, Philip A. Wilsey “Design and Performance Evaluation of Binary Code Packing for Protecting Embedded Software against Reverse Engineering”, 13th IEEE ISORC 2010