논문 2014-1-3

실행코드 암호화 및 무결성 검증을 적용한 안드로이드 앱 보호 기법

심형준*, 조상욱*, 정윤식**, 이찬희**, 한상철***, 조성제**†

A Technique for Protecting Android Applications using Executable Code Encryption and Integrity Verification

HyungJoon Shim*, Sangwook Cho*, Younsik Jeong**, Chanhee Lee**, Sangchul Han***, Seong-je Cho**†

요 약

본 논문에서는 안드로이드 애플리케이션(앱)을 역공학 공격으로부터 방어하는 기법을 제안한다. 이 기법에서 서버는 안드로이드 패키지 파일인 APK 내에 있는 원본 실행코드(DEX)를 암호화하고, 실행 시 이를 복호화 할 수 있는 스텁(stub) 코드를 APK에 삽입하여 배포한다. 스텁 코드는 자신에 대한 공격을 탐지하기 위해 무결성 검증 코드를 포함한다. 사용자가 해당 APK를 설치・실행할 때, 스텁 코드는 자체의 무결성을 검증한 후, 암호화된 원본 실행코드를 복호화하고, 이를 동적 로딩(dynamic loading)하여 실행한다. 앱의 원본 실행코드는 암호화되어 배포되므로 지적재산권을 효과적으로 보호할 수 있다. 또한, 스텁 코드에 대해 무결성을 검증하므로, 제안 기법의 우회 가능성을 차단한다. 우리는 15개의 안드로이드 앱에 제안 기법을 적용하여 그 유효성을 평가하였다. 실험 결과, 13개의 앱이 정상적으로 동작함을 확인하였다.

키워드 : 안드로이드 애플리케이션, 실행 코드, 클래스 동적 로딩, 암호화

Abstract

In this paper, we propose a method for protecting Android applications against reverse engineering attacks. In this method, the server encrypts the original executable code (DEX) included in an APK file, inserts into the APK file a stub code that decrypts the encrypted DEX later at run-time, and distributes the modified APK file. The stub code includes an integrity validation code to detect attacks on itself. When a user installs and executes the APK file, the stub code verifies the integrity of itself, decrypts the encrypted DEX, and loads it dynamically to execute. Since the original DEX is distributed as an encrypted one, we can effectively protect the intellectual property. Further, by verifying the integrity of the stub code, we can prevent malicious users from bypassing our method. We applied the method to 15 Android apps, and evaluated its effectiveness. We confirmed that 13 out of them operates normally.

Keyword : Android application, Executable code, Dynamic class loading, Encryption

※ 이 논문은 2014년 한국소프트웨어감정평가학회 하계 학술발표대회에서 발표된 '암호화 및 동적로딩 기법을 활용한 안드로이드 애플리케이션 실행코드 보호' 논문을 확장한 것임.

* 단국대학교 소프트웨어보안
** 단국대학교 컴퓨터학과
*** 건국대학교 컴퓨터공학과
† 교신저자 : 조성제(Email: sjcho@dankook.ac.kr)

※ 본 연구는 미래창조과학부가 지원한 2014년 정보통신·방송(ICT) 연구개발사업과 문화체육관광부 및 한국저작권위원회의 2014년도 저작권 기술개발사업의 연구결과로 수행되었음

접수일자: 2014.6.20 수정완료: 2014.6.27

1. 서 론

최근, 스마트폰의 보급률이 크게 늘어남에 따라, 스마트폰에서 사용되는 애플리케이션(application, 앱) 또한 그 수가 크게 증가하고 있다.

안드로이드 앱의 경우, 비교적 쉽게 역공학이 가능한 자바(Java) 언어로 작성되어 있어 역분석으로 인한 소스코드 복원 및 이의 무단 도용이 손쉬울 뿐만 아니라, 위·변조 공격에도 취약하다.

일례로 작년 국내 인기 모바일 게임이 그대로 복제된 사건이 발생했는데, 이는 원작 게임의 실행코드를 역분석하여 개발된 것이다. 그 외에도 결제모듈 부분을 해킹해 캐쉬를 불법적으로 취득한 사례도 발생하고 있다[1]. 2014년 1월 16일자 디지털데일리에 따르면, 앱 비즈니스가 확대되면서 앱 위변조 방지 기술 개발이 시급하다고 한다.

스마트폰 앱에 대한 역공학 공격이나 위변조 문제를 해결하기 위하여 소스코드 난독화(obfuscation)나, 바이너리 난독화 등의 제품들이 있다. 구글은 난독화 도구로 ProGuard[2]를 제공한다. 그러나 코드 난독화라는 것은 역공학을 어렵게 만드는 것이며, 역공학을 원천적으로 차단하는 것은 아니다[3].

최근 안드로이드 앱을 대상으로 한 역공학 방지 기술이나 위변조 방지 기술에 대한 관심이 확산되면서, 스트링 암호화, 전체 클래스 암호화 등에 대한 기술개발들이 활발히 진행되고 있다. 관련 해외 기술로는 ProGuard, DexGuard[4] 등이 있다. 2013년 10월 16일자 및 2014년 3월 13일자 보안뉴스에 의하면, “바이너리 코드 암호화”는 실행파일로부터 코드 분석을 불가능하게 하는 강력한 방어 기술로, 안드로이드 앱의 경우, 실행코드(DEX)와 String, Asset 등의 앱 자산들을 모두 암호화하는 가장 진화한 방법이며, 앱 개발 후에도 적용 가능하다.

이에 본 논문에서는 안드로이드 앱에 대한 지적재산권을 보호하고 위변조를 방지하기 위해, 앱 실행코드 암호화 기법을 제안한다. 제안 기법에서는, 개발된 앱의 실행코드를 역공학 공격으로부터 보호하기 위하여 APK 내에 존재하는 원본 실행코드(DEX)를 암호화하고, 실행 시 이를 복호화 할 수 있는 스텁(stub) 코드를 포함하도록 APK를 조작하여 배포한다. 이후 스텁 코드에서 정상 실행을 위해 스텁 코드의 무결성을 확인한 후, 암호화된 원본 실행코드를 복호화하여 동적 로딩(dynamic loading)한다. 제안 기법은 난독화 기술 또는 리소스 암호화 보다 강력한 방어 기술이다.

2. 관련 연구

2.1 ProGuard

ProGuard[2]는 구글에서 권장하는 안드로이드 앱 소스코드 난독화 도구로 최근에는 Android SDK(Software Development Kit)에 내장되어 배포되고 있다. 기본적으로 앱이 릴리즈 모드로 빌드 할 때에 적용이 되며, 파일 최적화 과정에서 의미 없는 코드를 제거해 주고, 앱 소스코드를 난독화 해 주는 기능이 있다[5].

그러나 난독화 도구의 특성상 앱 역공학에 소요되는 시간을 대폭 증가시키지만, 역공학 후 소스코드의 분석을 근본적으로 막지는 못한다. 따라서, 앱의 역공학을 근본적으로 막을 수 있는 기법에 대한 연구가 필요하다.

2.2 DexGuard

Saikoa에서 제작한 DexGuard[4]는 ProGuard

의 상용 버전이다. DexGuard는 소스코드 난독화 뿐만 아니라 문자열, 클래스명 및 assets 파일 암호화, 위변조 감지 코드 등의 추가적인 보안기능도 제공한다. DexGuard의 암호화 기법에서는 복호화 루틴(decryption routine)이 애플리케이션에 포함되기 때문에 APK 파일을 역공학해서 얻은 복호화 루틴을 분석하면 원본 앱의 소스코드가 유출될 가능성이 있다[6].

2.3 공개키 기반 앱 보호 프레임워크

안드로이드 프레임워크를 수정하여 암호화 된 DEX 파일을 복호화하는 연구가 있다[7]. 이 연구의 목표는 키 관리 방식과 앱 불법 복제 방지에 초점을 두고 있으나, 본 논문과 마찬가지로 앱의 실행코드인 classes.dex의 암·복호화를 한다는 점은 동일하다.

하지만, 이 방식은 암호화된 DEX 파일을 복호화하기 위해 안드로이드 플랫폼 소스코드를 수정하여 다시 빌드해야 한다는 제약이 있다.

3. 제안 기법

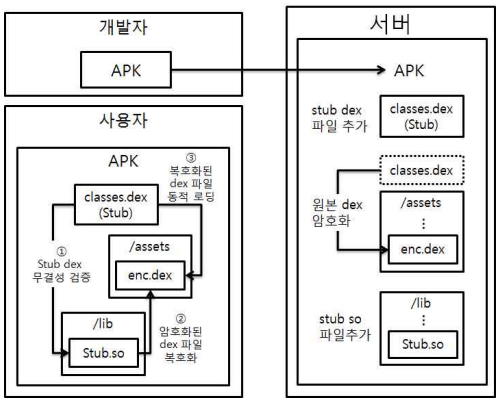


그림 1 제안 기법 개요도

제안 기법은 크게 실행코드 암호화, 앱 실행흐름 제어, 무결성 검증, DEX 파일 동적 로딩의 네 부분으로 구성된다.

서버에서는 원본 DEX 파일을 추출 및 암호화(enc.dex)하여 'assets' 디렉토리에 삽입하고, 스텁 코드 파일을 'classes.dex' 이름으로 삽입한다. 그 다음, 무결성 검증 기능과 암·복호화 기능을 수행하는 동적 라이브러리(so파일)를 삽입하고, AndroidManifest.xml에 스텁의 컴포넌트(Stub Component)를 선언한다. 기법 적용이 완료된 앱은 개발자로 전송되어 개발자의 사인(Sign)을 한 뒤 마켓에 업로드 된다.

사용자가 기법이 적용된 앱을 설치·실행할 때 먼저 스텁 코드의 무결성을 검증한다. 스텁 코드의 무결성이 검증되면, 암호화 된 DEX 파일이 복호화되고, 동적 로딩을 수행하여 원본 DEX로 실행 흐름을 변경한다.

3.1 실행코드 암호화

DEX 파일은 달빅 가상 머신(Dalvik Virtual Machine)상에서 수행되는 실행파일로 컴파일된 클래스들이 포함되어 있다.

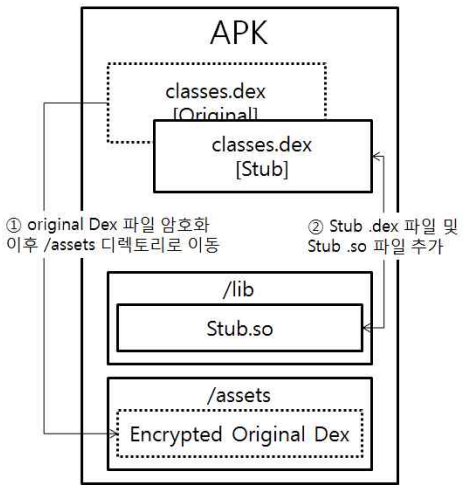


그림 2 제안 기법이 적용된 APK 파일

앱 개발자가 자바로 작성한 소스코드 대부분이 DEX 파일에 포함되고, 앱 실행에 직접적으로 영향을 준다. 따라서, 본 논문에서는 앱의 구성요소 중, 앱 실행에 직접적으로 영향을 주는 DEX 파일을 보호하기 위해 앱 실행파일을 AES 대칭키 암호화 방식으로 암호화를 수행한다.

앱 실행코드를 암호화하기 위해 우선적으로 배포 파일 형태인 APK 파일에서 DEX 파일을 추출해야 한다. 추출된 원본 DEX 파일을 암호화하여 그림 2와 같이 'assets' 디렉토리에 삽입하고, 스텁 코드의 무결성을 검증하는 부분과 원본 DEX 파일의 복호화를 수행하는 부분, 동적 로딩을 수행하는 부분으로 구성된 스텁 코드를 'classes.dex' 이름으로 앱 루트 디렉토리에 삽입한다.

3.2 앱 실행흐름 제어

안드로이드 앱은 여러 구성요소가 포함 된 패키지 형태(Android Package)로 배포된다. 주요 구성요소로는 앱의 컴포넌트 설정들이 저장되는 AndroidManifest.xml 파일, 실행코드를 포함하는 DEX(Dalvik Executable) 파일, 앱의 인증서를 포함하는 META-INF 디렉토리, 실행에 필요한 리소스를 포함하는 res 디렉토리, 주로 바이너리 리소스가 저장되는 assets 디렉토리, 컴파일된 리소스를 저장하는 resource.arsc 파일 등이 있다 [8].

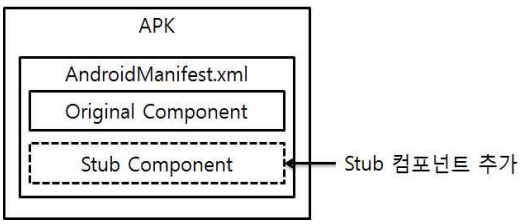


그림 3 스텁 컴포넌트 추가

원본 DEX 파일을 암호화하고, 스텁 코드 파일을 삽입하였기 때문에, 앱 실행 흐름을 스텁 코드 파일로 변경해 주어야 한다.

실행 흐름을 스텁 코드로 변경하는 방법 중 하나로 AndroidManifest.xml 파일에 먼저 실행되어야 할 컴포넌트 혹은 클래스를 선언해주는 방법이 있다. 본 논문에서는 Application 클래스를 상속하여 스텁 코드 파일이 우선으로 실행되도록 하였다. 이때 수행되는 스텁 코드 파일에서 무결성 검증과 복호화를 수행하는 Stub.so 파일이 실행된다.

3.3 스텁 코드에 대한 무결성 검사

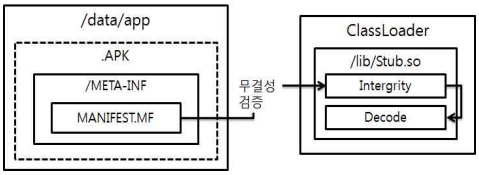


그림 4 스텁 코드 파일의 무결설 검증 방식

3.2 절에서 설명된 방법으로 앱의 실행 흐름이 스텁 코드로 변경되면, 스텁 코드는 Stub.so 내부의 함수를 호출한다.

스텁 코드에는 암·복호화를 수행하는 동적 라이브러리를 호출하는 부분, 복호화된 DEX파일을 동적로딩 하는 부분이 포함되어야 한다. 하지만, 암·복호화 코드가 자바(Java) 언어로 개발되면 역공학 여부에 따라 제안기법이 우회될 가능성이 있기 때문에 암·복호화 부분을 네이티브 수준에서 구현하고, 스텁 코드의 무결성을 보장해 주어야 한다.

본 논문에서는 Stub.so 라이브러리에서 스텁 코드의 해시 값을 APK 파일의 MANIFEST.MF 파일 내에 존재하는 DEX 해시 값과 비교하여 스텁 코드 파일의 무결성을 확인한 후, 암호화된 원본 DEX 파일을 복호화 한다. 동적으로 스텁

코드의 해시 값을 생성하지 않은 이유는 역공학 후에 수행하는 패키징 과정에서도 MANIFEST.MF 파일의 해시 값은 새로 생성이 되어 저장되기 때문이다. 즉, 앱을 최초 빌드 및 패키징 하는 과정과 역공학 후에 패키징 하는 과정 모두 패키징 도구가 classes.dex 파일의 해시 값을 생성하여 MANIFEST.MF 파일에 기록하기 때문에 새로 생성하지 않아도 된다.

3.4 DEX 파일의 동적 로딩

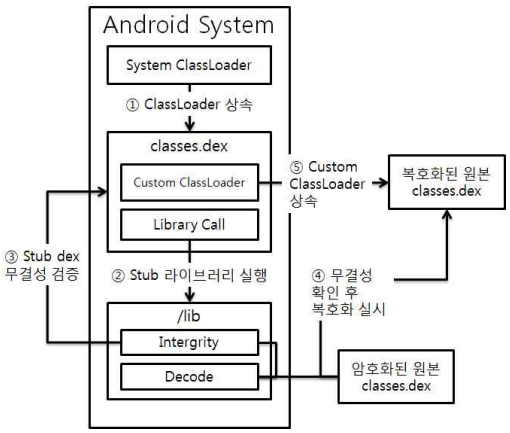


그림 5 Dex 파일의 동적로딩 과정

원본 DEX 파일이 암호화 되고, 스텁 코드를 대체하여 classes.dex 이름으로 삽입하였기 때문에, 복호화된 원본 DEX 파일로 실행 흐름을 변경해야 한다.

그러나 복호화된 원본 DEX 파일은 단순한 바이트 배열에 불과하기 때문에, 이 DEX 파일의 Class들을 직접 호출 할 수 없다. 따라서 스텁 코드에서는 클래스 로더(ClassLoader)를 상속하여 커스텀 클래스 로더(Custom ClassLoader)를 구현하였다. 커스텀 클래스 로더를 활용해 복호화된 DEX 파일의 Class들을 기존에 메모리에 적재되어 있던 것처럼 사용할 수 있다.

4. 실험 결과

본 논문에서 제안한 기법을 평가하기 위하여, 제안 기법이 적용됨에 따라 생성되는 시간 오버헤드와 각 앱 들의 크기와 원본 DEX 파일의 크기, 클래스 개수를 측정하였다.

구글 플레이에서 다운로드한 15개의 앱에 제안 기법을 적용하여 실험 셋을 구성하였다. 제안 기법이 적용된 앱이 구동된 환경은 삼성 갤럭시(Galaxy) S3(1.4 GHz quad-core Cortex-A9, 2 GB Ram, 안드로이드 버전 JellyBean 4.3)이다. 총 15개의 실험 앱 중, 13개의 앱에 본 논문에서 제안한 기법이 적용되어 정상적으로 작동함을 확인하였다.

총 15개의 앱 중, '쿠팡' 앱과 '캔디크러쉬 사가' 앱은 본 논문에서 제안한 기법으로 실행흐름을 제어할 수 없었는데, 분석 결과 모두 AndroidManifest.xml 파일에서 Application 클래스를 상속받아 사용하고 있음을 확인하였다.

실험 결과를 통해 본 제안기법은 평균적으로 0.81 초의 시간 오버헤드를 가지고, 시간 오버헤드는 APK 파일의 크기가 아닌 원본 DEX 파일에 비례하여 증가함을 확인할 수 있었다.

표 1의 실험결과에서 전체 시간 오버헤드에는 무결성 검증, 원본 DEX 파일 복호화, 동적 로딩의 3개 시간 오버헤드가 포함된 결과이다. 각 부분별 시간 오버헤드의 평균은 무결성 검증이 평균 0.01초, 원본 DEX파일 복호화가 평균 0.09초, 동적 로딩이 평균 0.71초로 동적 로딩에 소요되는 시간 오버헤드가 가장 큰 비중을 차지하고 있다.

'SCViewer' 앱의 경우 Dex 파일 크기가 'd shot' 앱 보다 0.2 MB 크지만, 전체 시간 오버헤드는 오히려 0.32초 적었다. 그 이유는 class 개수와 관련이 있다. 전체 시간 오버헤드에서 가장

큰 비중을 차지하고 있는 동적 로딩 부분에서는 실제 커스텀 클래스 로더에서 모든 원본 클래스를 메모리로 적재해야 한다. 따라서, DEX 파일의 크기 보다는 class의 개수가 전체 시간오버헤드와 밀접한 관련이 있는 것을 실험을 통해 확인하였다.

앱 이름 (앱 크기) (단위 : MB)	DEX 파일 크기 (단위 : MB)	class 개수 (단위 : 개)	전체 시간 오버헤드 (단위 : 초)
d shot (5.1)	1.7	1,475	1.13
traveldiary (3.6)	1.9	1,037	0.91
마약팅 (3.7)	2.1	1,627	1.16
연합뉴스 (4.6)	2.1	1,397	1.13
SCViewer (2.9)	1.9	936	0.81
할로윈 얼굴 만들기 (4.1)	1.4	991	0.8
풀어서 잠금해제 (5.2)	1.4	1,003	0.81
JOARA (1.7)	0.6	436	0.48
전국맛집 (1.1)	0.1	59	0.16
야구의 달인 (4.6)	1.1	914	0.73
블루 라이트 필터 (1.7)	0.7	358	0.43
Tribal Wars (22.1)	1.1	888	0.65
바운스 볼 (8.1)	3.2	2,528	1.39
쿠팡 (11.2)	3.7	2,340	X
캔디크러쉬 사가 (40.1)	1.6	1,215	X

표 1 실험 앱의 전체 시간 오버헤드

5. 결론 및 향후 연구방향

2013년 4월 25일자 연합뉴스의 “스마트 앱 불법 1천774억원, 전체 23.1% 불법 앱 이용”이라는 기사에 의하면 2012년 스마트폰 앱 불법 시장 규모는 전체 스마트폰 앱 시장의 약 44%에 이른다고 보고하고 있다. 또한, 2013년 8월 1일자 데일리 시큐에 따르면 안드로이드 디바이스 99%가 앱 패키지 변조 위협에 취약하다고 한다. 따라서 앱 비즈니스가 앱 역공학 방지 및 앱 위변조 방지에 초점을 맞추고 있다.

이에 본 논문에서는, 안드로이드 앱 실행코드를 역공학 또는 위변조 공격으로부터 보호하기 위해, 실행코드 암·복호화 및 동적로딩, 무결성 검증 기법을 활용하는 앱 지적재산권 보호기법을 제안하고 구현하였다. 본 기법은 안드로이드 개발자에 의해 앱이 개발된 후에 적용 가능하다. 제안 기법은, 앱의 원본 실행코드를 암호화하여 배포·유통하며, 앱 실행 시 암호화된 코드의 복호화 및 동적 로딩, 무결성 검증을 위해 스텁 코드를 추가하여 구현하였다. 실험을 통해, 제안 기법의 효용성과 시간 오버헤드를 확인하였다. 향후, 앱 자체에서 Application 클래스를 상속받아 사용할 때에도 기법 적용이 되도록 연구를 진행할 예정이다.

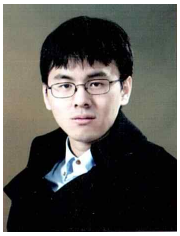
참 고 문 헌

[1] 이민형, “앱 비즈니스의 확대... 앱 위변조방지에 초점”, 디지털데일리, http://www.ddaily.co.kr/news/article.html?no=113236, January 2014.

[2] ProGuard Introduction, “http://stuff.mit.edu/afs/sipb/project/android/sdk/android-sdk-lin

ux/tools/proguard/docs/index.html#manual/introduction.html"
[3] 김태형, "난독화·리소스 암호화보다 높은 방어기술로 바이너리 암호화 부각", 보안뉴스, http://www.boannews.com/media/view.asp?idx=38059, October 2013
[4] Saikoa DexGurad, http://www.saikoa.com/dexguard
[5] "What kind of optimization does ProGuard support" http://proguard.sourceforge.net/#FAQ.html
[6] "A look inside DexGuard", http://www.android-decompiler.com/blog/2013/04/02/a-look-inside-dexGuard/, April 2013
[7] 김성렬, "공개키 기반 구조를 이용한 안드로이드 애플리케이션 보호 프레임 워크", 보안공학연구논문지, 제9권 제1호, Feb. 2012.
[8] Estourgie, "Analysis of Android Authenticators", Radboud Universiteit Nijmegen bachelor thesis, 2013

저 자 소 개



심형준

2013년 단국대학교 컴퓨터학과(공학사)
2014 - 현재 : 단국대학교 소프트웨어보안 석사 과정
<주관심분야 : 안드로이드 애플리케이션, 소프트웨어보안>

조상욱

2014년 단국대학교 컴퓨터학과(공학사)
2014 - 현재 : 단국대학교 소프트웨어보안 석사 과정
<주관심분야 : 운영체제, 시스템보안>

이찬희

2013년 단국대학교 컴퓨터학과(공학사)
2014 - 현재 : 단국대학교 컴퓨터학 석사 과정
<주관심분야 : 운영체제, 정보 보안, 모바일 시스템 보안>

정윤식

2012년 단국대학교 컴퓨터과학과(공학사)
2013년 단국대학교 컴퓨터과학과(공학석사)
2013 9월 - 현재 단국대학교 컴퓨터과학과 박사 과정
<주관심분야 : 컴퓨터보안, 소프트웨어보증, 시스템소프트웨어 >

한상철

1998년 연세대학교 컴퓨터과학과(공학사)
2000년 서울대학교 컴퓨터공학과(공학석사)
2007년 서울대학교 컴퓨터공학과(공학박사)
2008 - 현재 : 건국대학교 컴퓨터공학과 교수
<주관심분야 : 실시간 시스템, 스케쥴링 알고리즘>

조성제

1989년 서울대학교 컴퓨터공학과(공학사)
1991년 서울대학교 컴퓨터공학과(공학석사)
1996년 서울대학교 컴퓨터공학과(공학박사)
2001년 미국 University of Califonia, Irvine 객원 연구원
2009년 미국 University of Califonia 객원연구원
1997년 3월 - 현재 단국대학교 소프트웨어학과/컴퓨터학과 교수
<주관심분야 : 컴퓨터보안 (취약점 탐지 및 분석, 악성코드 분석), 스마트폰 보안, 소프트웨어 지적재산권 보호, 소프트웨어 보증>