논문 2013-1-2

안드로이드 스마트폰에서 Dex 파일들의 유사도 비교 연구

고정욱*, 강성욱*, 문정오**, 김동진***, 조성제***†

A Study on Comparing Similarity of Dex files on Android Smartphones

Jeonguk Ko*, Seongwook Kang*, Jeongoh Moon**, Dongjin Kim***, Seong-je Cho***†

요 약

안드로이드 앱은 역분석되기 쉽기 때문에 불법 도용 및 표절에 의한 저작권 침해가 급증하고 있다. 본 논문에서는 안드로이드 앱 저작권 침해 여부를 탐지하기 위한, 앱들 간의 유사성을 비교하는 3가지 방법을 제안한다. 제안기법은 안드로이드 앱의 실행 파일인 Dex 파일을 정적으로 분석하여 유사도를 측정한다. 첫 번째 기법에서는, Dex 파일의 바이트코드를 자바 소스코드로 역컴파일한 후, 역컴파일된 소스코드 간의 유사도를 측정한다. 두 번째 기법에서는, Dex 파일에 포함된 문자열들을 추출하고 문자열 비교를 기반으로 앱 간의 유사도를 비교한다. 마지막으로 Dex 파일에 정의된 메서드들로부터 특징정보를 추출하여 유사도를 비교한다. 실험을 통해 각 제안기법의 성능을 신뢰성 및 강인성 측면에서 평가하였다.

Abstract

Copyright infringement in Android applications is a growing problem because Android applications can be easily reverse engineered and pirated. In this paper, we propose three techniques for comparing similarity between Android applications in order to detect the theft of the applications. The proposed techniques statically analyze the Dalvik VM executable files in the Dalvik Executable (.dex) format, and measure similarity between the executable files (DEX files). The first technique decompiles DEX (VM bytecode) files into corresponding Java source codes and computes similarity between the decompiled source codes. The second technique extracts string information contained in DEX files and calculates similarity between applications based on the string information. The last technique extracts methods' features defined DEX files and compares similarity between applications based on the features. We evaluate the performance of the proposed techniques in terms of credibility and resilience by carrying out some experiments.

한글키워드 : 안드로이드 애플리케이션, 소프트웨어 유사성, 정적 분석

※ 이 논문은 2013년 제40회 한국컴퓨터종합학술대회에서 발표된 '안드로이드 애플리케이션들의 유사도 비교 연구' 논문을 확장한 것임.

* 단국대학교 컴퓨터학과 소프트웨어보안전공
** 단국대학교 소프트웨어학과
*** 단국대학교 컴퓨터학과
† 교신저자 : 조성제(Email: sjcho@dankook.ac.kr)

※ 본 연구는 문화체육관광부 및 한국저작권위원회의 2013년도 저작권기술개발사업의 연구 결과, 그리고 미래과학창조부 및 한국인터넷진흥원의 "고용계약형 지식정보보안 석사과정 지원사업"의 연구 결과로 수행되었음 (과제번호 H2101-13-1001)
접수일자: 2013.6.5 수정완료: 2013.6.24.

1. 서 론

스마트폰 애플리케이션(이하 '앱) 마켓이 급성장하면서, 스마트폰 앱에 대한 저작권 침해도 함께 급증하고 있다. 특히, 안드로이드 앱은 역컴파일 및 역어셈블을 통한 역분석이 쉽고, 변조 후 다시 리패키징하여 배포하기 쉽기 때문에 더 많은 저작권 피해가 발생하고 있다.

SW 불법 도용 및 표절로부터 저작권을 보호하기 위하여, 프로그램 간의 유사도를 분석하여 불법도용 여부을 탐지하기 위한 다양한 기법들이 연구되고 있다[1-11]. 기존의 SW 불법 표절 탐지 연구들은 소스코드 간의 유사도 비교를 중심으로 수행되었다. 하지만 저작권 분쟁 시에는 소스코드를 확보하기 어렵기 때문에 소스코드 수준의 유사도 비교 기법에는 한계가 존재한다. SW 워터마크(Watermark)는 프로그램에 저작권 정보를 추가로 삽입하기 때문에 공격자에 의해 제거 및 변조될 수 있다는 한계가 있다[11]. SW 버스마크(Birthmark)는 프로그램을 식별 가능한 프로그램의 특징정보로써, 최근 프로그램 도용 및 표절 탐지를 목적으로 연구되고 있다. 하지만 지금까지의 SW 버스마크 연구는 대부분 MS Windows 애플리케이션을 대상으로 하며, 스마트폰 앱을 대상으로 한 연구는 미비한 상태이다.

본 논문에서는 안드로이드 앱 불법 도용 및 표절을 탐지하기 위한 유사도 비교 기법 3가지를 제안한다. 제안 기법은 안드로이드 앱에서 실행파일에 해당하는 Dex(Dalvik Executable)을 정적 분석한 결과를 기반으로 한다. 첫 번째, 안드로이드 앱은 역컴파일이 쉽다는 특성을 활용하여, 앱을 역컴파일한 후, 역컴파일된 소스코드의 유사도를 비교한다. 두 번째, 앱 실행파일에 포함된 문자열들을 비교하여 유사도를 분석한다. 실행파일에 포함된 문자열은 소스코드의 특징 중 컴파일과정에서 변경되지 않는 정보로서, 프로그램의 중요 특징정보 중 하나이다. 마지막으로 자바 메서드(method)의 반환 값 및 호출 인자들의 자료형을 기반으로 앱 간의 유사도를 비교한다. 제안 기법들의 효율성 평가하기 위해, 기능별로 수집한 앱들을 대상으로 각 기법을 적용 및 실험하였으며, 이를 통해 각 기법들의 장·단점을 비교, 분석하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 설명하고 3장에서는 제안한 기법에 대해 설명 한다. 4장에서는 제안한 기법의 검증을 위한 실험 및 분석, 5장에서는 결론과 향후 연구 방향에 대해 기술한다.

2. 관련 연구

2.1 SW 불법 도용 탐지

대표적인 SW 불법 도용 탐지 기법은 SW 워터마크와 SW 버스마크 기법이다. SW 워터마크 기법은 프로그램에 저작권 정보를 추가적으로 삽입하여 저작권을 증명하는 기법이다. 하지만 추가된 저작권 정보가 난독화 등의 기법에 의해 훼손되거나, 공격자에 의해 제거될 수 있다. 또한 SW 전체 표절이 아닌, 부분 표절 및 도용 시에는 저작권을 증명하기 어렵다는 한계가 있다[11]. 특히 안드로이드 앱과 같이, 역공학에 취약한 경우에는 워터마크가 더 쉽게 훼손될 수 있다.

SW 버스마크 [1-10]는 프로그램의 고유한 특징정보로써, 프로그램을 유일하게 식별할 수 있는 특징정보 이다. 즉, 추가적인 정보의 삽입 없이도 불법 도용 및 표절을 탐지할 수 있는 기법으로 최근에는 다양한 SW 버스마크가 연구되고 있다.

대표적으로 Tamada와 Myles는 자바 애플리케이션을 위한 정적 버스마크를 제안하였다. Tamada [1]는 자바 바이트코드로부터 Constant

values, Sequence of Method calls, Used classes 등의 특징정보를 추출하여 이를 SW 버스마크로 사용하였다. Myles [2]는 자바 바이트코드를 역어셈블한 코드에서 Op-code의 서열(sequence)을 기반으로한 버스마크인 k-gram 기법을 제안하였다.

하지만 기존의 자바 바이트코드를 대상으로 한 SW 버스마크 기법을 안드로이드 앱에 그대로 적용하기는 어렵다. 안드로이드 앱의 경우, 개발 언어가 자바 프로그래밍 언어이고, 실행코드도 자바의 바이트코드와 일부 유사하다. 하지만 안드로이드 앱 실행파일인 Dex의 경우, 자바 프로그램의 실행파일을 JVM(Java Virtual Machine)이 아닌, 안드로이드 가상머신인 DVM(Dalvik Virtual Machine) 환경에서 실행 가능하도록 변형시킨 구조이기 때문이다.

2.2 안드로이드 앱 구조

안드로이드 앱은 실행파일인 classes.dex 파일, 앱의 권한 등이 정의되어 있는 AndroidManifest.xml과 이미지 파일 등의 리소스들로 구성된다. 그리고 구성요소들은 APK (Android Package) 형태로 패키징되어 배포된다.

앱 구성요소 중에서 AndroidManifest.xml에 포함된, 앱 권한 정보는 유사한 기능의 앱들에서 매우 유사하게 나타날 수 있다. 그리고 이미지 파일 등의 리소스도 실행에 아무런 영향을 미치지 못하기 때문에 쉽게 변경이 가능하다. 그렇기 때문에 AndroidManifest.xml의 정보와 리소스들은 앱 간의 유사도 비교 시에 사용할 특징정보로 부적합하다.

Classes.dex은 Dex(Dalvik Executable) 포맷으로 DVM(Dalvik Virtual Machine)에서 실행될 수 있도록 자바 클래스파일을 재구성한 것이다. Dex 포맷의 구조는 그림 1과 같으며, 'header'에는 파일의 크기 및 기타 다른 정보들의 위치 정보를 포함한다. 'string_id'는 Dex에 포함된 모든 문자열들의 인덱스 및 위치 정보를 포함한다. 'type_ids'는 실행코드에서 참조하는 클래스 및 배열 정보를, 'proto_ids'는 프로토타입 식별에 관한 정보를 포함한다. 'field_ids'는 모든 클래스의 필드 테이블을 포함하고, 'method_ids'는 메소드 이름, 메서드 형식 설명이며, 'class_ids'는 Dex파일에 포함된 모든 클래스 목록이다[12].

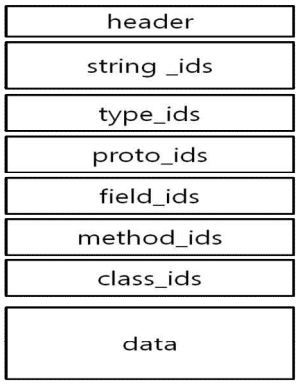


그림 1. Dex 파일의 구조

3. 제안 기법

3.1 역컴파일된 소스코드 수준 유사도 비교

안드로이드 앱의 classes.dex 실행파일은 역컴파일이 용이하다는 점을 활용하여 역컴파일을 통해 소스코드를 추출한다. 그리고 역컴파일된 소스코드 간의 유사도를 측정하고, 이를 앱간의 유사도 측정 결과로 사용한다.

안드로이드 앱의 컴파일 전의 소스코드와는 달리, 역컴파일된 소스코드에는 import하는 라이브러리들의 소스코드가 함께 포함되어 있다. 라이브러리 소스코드를 모두 포함하여 유사도를 측정할 경우 false-positive가 발생할 수 있다.

이를 개선하기 위해, 대다수의 앱에서 공통적으로 사용하는 라이브러리의 소스코드는 비교 대상에서 제외하거나, 유사도 값 계산 시에 가중치

를 낮추는 등의 고려가 필요하다.

역컴파일된 소스코드 수준의 유사도 비교를 위해서, 먼저 자바 역컴파일러인 JD-GUI [13]를 사용하여 역컴파일된 소스코드를 추출한다. 그 후에 잘 알려진 소스코드 간 유사도 비교 도구인 Moss[14]를 사용하여 유사도를 비교한다. Moss는 프로그램 소스코드를 토큰 단위로 분리하여 비교한다. 그러므로 소스코드가 난독화 기법 등에 의해 일부 변경되더라도 불법 도용 및 표절을 탐지 가능하다.

3.2 문자열 기반 유사도 비교

실행파일에 포함된 문자열은 소스코드의 특징 중 컴파일 과정에서 훼손되거나 변경되지 않는 소스코드의 중요 특징정보이다. 본 논문에서는 문자열 기반의 유사도 비교를 위해 Dex에 포함된 문자열들을 추출하여 비교한다. 여기에는 상수 문자열, 클래스명, 타입명이 포함된다. 문자열 기반 유사도 값은 유사도 비교 기준 앱 A와 표절이 의심되는 앱 B에서 추출한 각 문자열들을 비교하여 정확히 일치하는 문자열의 개수를 A에서 추출한 전체 문자열의 개수로 나눈 값을 사용한다. 예를 들어, A와 B에서 추출한 문자열들의 개수가 각각 100개와 80개이고, 정확하게 일치하는 문자열의 개수가 40개일 때, 앱 A를 기준으로 한 A와 B의 유사도 값은 0.4(40/100) 이다.

3.3 메서드(method) 기반의 유사도 비교

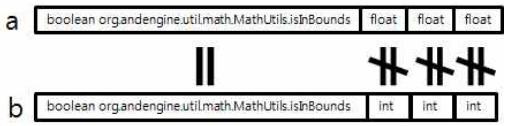
메서드를 기반으로 한 유사도 비교에서는 각 메서드의 반환 값과 호출 인자들의 자료형만을 추출하여 특징정보로 사용하고, 이 특징정보 간의 유사도를 전체 앱의 유사도로 사용한다.

메서드 명은 가장 기본적인 레이아웃 난독화 기법에 의해 쉽게 변조되기 때문에 메서드 명은 유사도 비교를 위한 특징정보로 사용하지 않는다. 실제로 앱 개발 시, 구글에서 제공하는 난독화 도구인 Proguard를 적용하면 모든 클래스 및 메서드 명이 의미 없는 값으로 난독화된다. 메서드 명과는 달리 반환 값과 인자의 자료형은 각 메서드의 기능과 관련 있기 때문에 쉽게 변조하기 어렵다. 그렇기 때문에 각 메서드별 인자의 개수와 반환 값 및 인자들의 자료형을 특징정보로 사용한다.

메서드 기반의 앱 간의 유사도 비교 단계는 다음과 같다.

- 1 단계: 앱 A의 메서드 a_i (1 <= i <= n) 와 B에 포함된 메서드 b_j (1 <= j <= m) 추출
- 2 단계: a_i와 b_j의 반환 값 및 인자의 자료형을 기반으로 각 메서드 간의 유사도 값 계산
- 3 단계: a_i와 가장 유사한 b_j을 찾고, 두 메서드 간의 유사도를 s_k (1 <= k <= n)으로 한 후에 s_k/n 을 최종적인 앱 간의 유사도로 사용

메서드 간의 유사도를 측정하는 방법은 그림 2와 같다. 두 메서드간의 동일한 반환 값 및 인자의 자료형이 존재하는 개수를 비교 기준인 메서드의 반환 값 및 인자의 개수로 나눈 값을 사용한다. 즉 그림 2의 경우, 유사도 측정 결과 값은 1/4 이다.



$$Similarity(a,b) = \frac{1}{4}$$

그림 2. 메서드간의 유사도 측정

4. 실험 및 평가

제안 기법을 검증하기 위해, 신뢰성(credibility)과 강인성(resilience)을 평가 및 분석하였다. 신뢰

성은 동일 또는 유사한 기능을 수행하는 서로 독립적으로 개발된 프로그램을 유사하지 않은 것으로 판단하는 능력에 대한 평가 기준이다. 강인성은 매우 유사한 방식으로 개발된 프로그램을 유사한 것으로 판단하는 능력을 평가하는 기준이다.

신뢰성 평가를 위해, 동일 기능의 제품군에 속한 서로 다른 앱 간의 유사도를 비교하였고, 강인성 평가를 위해서는 동일 앱의 다른 버전 간의 유사도를 비교하였다.

4.1 동일 제품군 다른앱 간의 유사도 비교

제안기법의 신뢰성 검증하기 위해 제안 기법의 기법 간 유사도를 비교하였다. 이를 위해, 대중적으로 많이 쓰이는 앱들 중에서 다섯 종류의 제품군을 선정하였고, 각 제품군 별로 2개의 앱을 선정하여 유사도를 측정하였다.

실험에서는 본 논문에서 제안한 3 가지 기법을 사용하여 실험을 진행하였다.

- 기법 1. 역컴파일한 소스코드의 유사도 비교
- 기법 2. 문자열 기반 유사도 비교
- 기법 3. 메서드 기반 유사도 비교

실험 대상으로 선정한 5개의 제품군 및 각 제품군별 대표 앱은 표 1과 같다. 또한, 위 기법 3가지를 이용하여 유사도를 비교한 결과는 다음 표 2와 같다.

표 1. 실험 대상 앱 정보

종류	이름	크기(MB)
Image Viewer	A comic viewer(1.4.1.4)	1.12
	Easy comic Viewer(1.6.7)	7.45
Subway	Paris metro(2.1.4)	0.03
	Sydney metro(new)	0.24
Camera	Camera 360(4.1.1)	6.62
	Camera ICS(1.2.1)	0.89
Map	OsmAnd Map(1.3.1)	12.9
	Qibla Map(1.0)	0.12
Video	Easy player(1.0.7)	9.01
	Real player(1.1.3.0)	3.20

실험 결과, 기법 1에 비해 기법 2의 유사도가 모두 더 높았다. Moss에서 사용하는 유사도 측정 기법은 패턴 매칭 기법으로써 어느 정도 유사한 패턴이 존재하면 소스코드가 일치하지 않더라도 유사하다고 판단하기 때문이다. 즉, 소스코드가 동일한 경우에도 패턴이 일치하지 않는다면

표 2. 제안 기법의 기법 간 유사도 측정 결과 (단위: %)

타 입	기준 프로그램	비교 프로그램	유 사 도 비교		
			기법1	기법2	기법3
Image Viewer	A comic viewer	Easy comic viewer	45%	54%	21%
	Easy comic viewer	A comic viewer	5%	19%	6%
Subway	Paris Metro	Sydney Metro	23%	29%	8%
	Sydney Metro	Paris Metro	3%	4%	2%
Camera	Camera360	CameraICS	3%	8%	3%
	CameraICS	Camera360	30%	43%	23%
Map	OsmAnd Map	Qibla Map	0%	1%	0%
	QiblaMap	OsmAndMap	21%	45%	28%
video	EasyPlayer	RealPlayer	6%	9%	5%
	RealPlayer	EasyPlayer	18%	19%	11%

유사도 값은 낮게 측정된다.

기법 2는 라이브러리를 포함하여 비교하였기 때문에 공통적으로 존재하는 문자열의 개수가 많았고, 이로 인해 전반적인 유사도는 높게 측정되었다.

기법 1과 2는 대체적으로 비슷한 결과를 보이고 있다. 그러나 Image Viewer 제품군과 Subway 제품군의 경우에는 유사도의 차이가 20% 정도로 큰 편차를 보이고 있다. 이를 분석한 결과, 기법 1에서는 소스코드의 알고리즘 부분에 비해 변수 선언부의 패턴 매칭이 많이 일어남을 알 수 있었다. 다음 그림 3은 기법 1 기반의 유사도 비교에서 변수 선언 부의 매칭이 많이 일어남을 보여주고 있다.

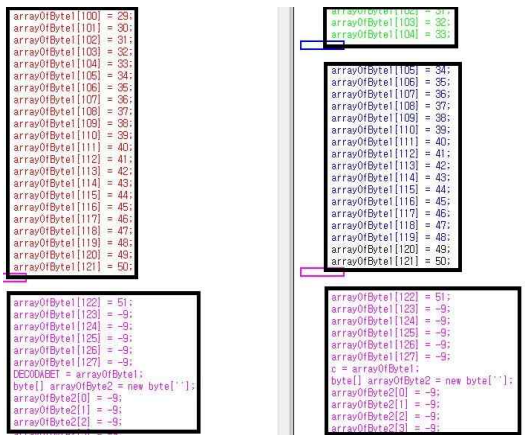


그림 3, 기법 1의 변수 선언부 패턴 매칭

표 3. 동일 앱 다른 버전 간 유사도비교 (단위 %)

프로그램	기준 버전	비교 버전	유사도 비교		
			기법 1	기법 2	기법 3
Camera 360	3.7	4.1.1	39	56.95	45
	4.1.1	3.7	38	52.51	41
	4.0	4.1.1	80	91.28	90
	4.1.1	4.0	91	95.64	96

4.2 동일 앱 다른 버전 기반의 유사도 비교

제안 기법의 강인성을 측정하기 위해 같은 앱의 다른 버전에 대한 유사도를 비교하였다. 표 3은 camera360 앱의 3가지 버전(3.7, 4.0, 4.1.1)에 대해 3.7버전과 4.1.1버전, 4.0버전과 4.1.1버전으로 나누어 비교한 결과이다.

실험 결과, 4.1 절의 실험 결과와 마찬가지로 기법 1과 비교하여 기법 2의 유사도가 더 높게 측정되었다. 또한, 기법 1과 3는 유사한 실험 결과를 보였다.

3.7 버전과 4.1.1 버전의 유사도가 4.0 버전과 4.1.1 버전 간의 유사도보다 낮게 측정되었다. 분석 결과, 3.7 버전과 4.1.1 버전은 소스코드의 추가 및 변화가 많이 존재하기 때문이다. 실험 결과 본 논문에서 제안한 3가지 기법 모두의 강인성이 높음을 확인하였다.

5. 결 론

안드로이드 앱 불법 도용 및 표절에 의한 저작권 침해에 따른 경제적 피해가 급증하고 있다. 하지만 PC 환경과 비교하여, 안드로이드 앱 저작권 보호를 위한 연구는 부족한 실정이다.

본 논문에서는 안드로이드 앱 불법 도용 및 표절을 탐지하기 위한, 앱들 간의 유사성 비교 기법을 3가지 제안하였다. 모두 안드로이드 앱의 실행 파일인 Dex 파일을 정적 분석 결과를 기반으로 한다. 첫 번째 기법에서는, Dex 파일의 바이트코드를 자바 소스코드로 역컴파일한 후, 역컴파일된 소스코드 간의 유사도를 측정하였다. 두 번째로 Dex 파일에 포함된 문자열들을 추출하고 문자열 비교를 기반으로 앱 간의 유사도를 비교하였다. 마지막으로 Dex 파일에 정의된 메서드들로부터 특징정보를 추출하여 유사도를 비교하였다. 실험을 통해 각 제안기법들이 신뢰성 및 강인성 측면에서 효율적임을 검증하였다.

향후에는 더 효율적인 안드로이드 앱 유사도 비교 연구를 위해, 다양한 기법들에 대하여 연구할 계획이다.

참 고 문 헌

[1] Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K. Design and Evaluation of Birthmarks for Detecting Theft of Java Programs. In Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004), pp.569-575, February 2004.

[2] Myles, G. and Collberg, C. K-gram based software birthmarks. In Proceedings of the 2005 ACM Symposium on Applied Computing (SAC'05), March 2005.

[3] Myles, G. and Collberg, C. Detecting Software Theft via Whole Program Path Birthmarks. In Proceedings of the Information Security Conference, September 2004.

[4] Tamada, H., Okamoto, K., Nakamura, M., Monden, A. and Matsumoto, K. Dynamic Software Birthmarks to Detect the Theft of Windows Applications. In Proceedings of the International Symposium on Future Software Technology 2004 (ISFST 2004), October 2004.

[5] Wang, X., Jhi, Y., Zhu, S. and Liu, P. Detecting Software Theft via System Call Based Birthmarks, In Proceedings of the Annual Computer Security Applications Conference (ACSAC '09), 2009.

[6] Schuler, D. and Dallmeier, V. Detecting Software Theft with API Call Sequence Sets, In Proceedings of the 8th Workshop Software Reengineering, May 2006.

[7] Choi, S., Park, H., Lim, H. and Han, T. A Static Birthmark of Binary Executables Based on API Call Structure, In Proceedings of the 12th Asian Computing Science Conference (ASIAN), December 2007.

[8] Myles, G. Software Theft Detection through Program Identification. PhD thesis, Department of Computer Science, The University of Arizona, 2006.

[9] Zhou, X., Sun, X., Sun, G. and Yang, Y. A Combined Static and Dynamic Software Birthmark Based on Component Dependence Graph, In Proceedings of the 4th Int'l Conf. on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), pp. 1416-1421, August 2008.

[10] Tamada, H., Okamoto, K., Nakamura, M., Monden, A. and Matsumoto, K. Design and evaluation of dynamic software birthmarks based on api calls. Nara Institute of Science and Technology, Technical Report, 2007.

[11] Collberg, C. S. and Thomborson, C., Watermarking, Tamper-proofing, and Obfuscation-Tools for Software Protection, IEEE Trans. on Software Engineering, 28(8), Aug. 2002.

[12] Dex- Dalvik Executable Format, http://source.android.com/tech/dalvik/dex-format.html

[13] Java decompiler: JD-GUI, http://java.decompiler.free.fr/?q=jdgui

[14] A system for detecting software plagiarism - MOSS. [online] http://theory.stanford.edu/~aiken/moss/.

저 자 소 개

고 정 욱

2013년 제주대학교 컴퓨터교육과 학사
2013년 3월～현재: 단국대학교 컴퓨터학과 소프트웨어보안전공 석사과정
<관심분야 : 악성코드 및 취약점 분석, 웹(Front-end), 웹 보안 등>

강 성 욱

2010년 동신대학교 컴퓨터공학과 학사
2013년 3월～현재: 단국대학교 컴퓨터학과 소프트웨어보안전공 석사과정
<관심분야 : 안드로이드 애플리케이션, 악성코드 분석 등>

문 정 오

2006년 3월～현재: 단국대학교 컴퓨터학부 (소프트웨어학과) 학사 과정
<관심분야: 컴퓨터 보안, 소프트웨어 보안 등>

김 동 진

2009년 단국대학교 컴퓨터과학과 학사
2011년 단국대학교 컴퓨터학과 공학석사
2011년～현재 : 단국대학교 컴퓨터학과 컴퓨터과학전공 박사과정
<관심분야 : 컴퓨터보안, 소프트웨어 보증, 보안 테스팅, 소프트웨어 저작권 보호, 시스템소프트웨어 등>

조 성 제

1989년 서울대학교 컴퓨터공학과 공학사
1991년 서울대학교 컴퓨터공학과 공학석사
1996년 서울대학교 컴퓨터공학과 공학박사
2001년 미국 University of California, Irvine 객원연구원
2009년 미국 University of Cincinnati 객원연구원
1997년 3월～현재: 단국대학교 소프트웨어학과/컴퓨터학과 교수
<관심분야 : 컴퓨터보안, 소프트웨어 감정, 소프트웨어 보증, 시스템소프트웨어, 실시간스케줄링, 임베디드 소프트웨어 등>