논문 2015-1-3

난독화를 고려한 안드로이드 앱 유사성 비교

박종화*, 임경환*, 이경묵*, 김동진*, 조성제*† , 정영기*

Similarity Analysis of Android Apps Considering Code Obfuscation

Jonghwa Park*, Kyeonghwan Lim*, Kyungmook Lee*, Dongjin Kim*,
Seong-je Cho*† , Youngki Jeong*

요 약

코드 난독화는, 어떤 프로그램을 동일한 기능을 가지면서 역공학하기 어려운 버전으로 변환시켜주는 기법이다. 코드 난독화 도구를 사용하여, 소프트웨어 개발자는 자신의 코드를 난독화시켜 코드 내의 지적재산권을 감추어 역공학 및 변조 공격을 방어할 수 있다. 한편으로, 악의적인 사용자는 자신이 도용하거나 표절한 프로그램 코드를 숨기기 위해 난독화 기법을 악용할 수 있다. 만약 불법복제된 소프트웨어가 난독화된다면 소프트웨어 도용 여부를 탐지하기 어렵다. 본 논문에서는 안드로이드 앱을 난독화시켜 주는 3가지 도구를 비교하고, 이들 도구들이 소프트웨어 유사도 측정에 미치는 영향을 조사한다. 이 3가지 도구는 소스코드를 입력받아 기능은 동일하지만 분석하기 어려운 형식의 실행 파일로 변환시켜 준다. 이 도구들의 영향을 분석하기 위해, 난독화된 프로그램의 디컴파일 버전과 원본 소스파일 간의 유사도를 측정하였다. 본 연구는 불법복제된 소프트웨어가 난독화된 경우에도, 소프트웨어 도용 여부를 탐지하는데 도움을 줄 수 있다.

Abstract

Code obfuscation is a technique to transform a program into an equivalent one that is harder to reverse engineering. Using code obfuscation tools (or, obfuscators), software developers obfuscate their codes to conceal intellectual properties of the codes, in order to deter reverse engineering or prevent tampering. On the other hand, malicious users may also make use of obfuscation techniques badly to hide the program codes that they have thieved or plagiarized. It would be difficult to detect software theft if pirated software was obfuscated. In this paper, we compare three obfuscators for Android apps, and investigate the effects of the tools on measuring software similarity. To analyze the effects of them, we have measured software similarity between an original source program and the decompiled version from obfuscated program. This study is helpful to detect software theft using similarity measures even though pirated software is obfuscated.

한글키워드 : 난독화, 유사도, 특징정보, 안드로이드, 애플리케이션
keywords : Code obfuscation, similarity, feature code, andriod, application

※ 이 논문은 2015년 춘계학술대회에서 발표된 '안드로이드 앱 난독화 도구들의 효과 분석' 논문을 확장한 것임

* 단국대학교 컴퓨터학과
† 교신저자:조성제(email: sjcho@dankook.ac.kr)

접수일자: 2015.5.20. 심사완료: 2015.6.2.
게재확정: 2015.6.20.

※ 본 연구는 미래창조과학부 및 한국인터넷진흥원의 "2014년도 고용계약형 정보보호 석사과정 사업"의 연구 결과(과제번호 H2101-15-1001)와 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음 (IITP-2015-H8501-15-1012)

1. 서 론

스마트폰 및 태블릿과 같은 다양한 모바일 기기의 보급의 증가와 함께 모바일 애플리케이션(이하 앱과 동일) 마켓도 성장하였다. 이에 따라 모바일 앱의 불법 도용 및 표절에 따른 저작권 피해가 급증하고 있다. 또한 안드로이드 앱은 자바 기반 프로그램으로 역공학을 통해 소스코드의 추출이 가능하다. 따라서 공격자들은 추출한 소스코드에서 앱 개발사의 핵심기술을 탈취하거나 악의적인 행동을 하는 소스코드를 삽입하는 등의 악성 앱들로 인한 피해가 발생하고 있다[1].

안드로이드 마켓에서 애플리케이션을 다운받을 경우 사용자의 스마트폰에 APK(Android application Packages)파일 형태로 배포되며, 앱의 실행파일인 classes.dex(이하 'DEX' 동일)를 포함하고 있다. DEX 파일은 자바 포맷을 안드로이드에 적합하도록 최적화한 것이다. 따라서 바이너리 실행 파일에 비해 역공학에 의한 소스 코드 추출이 용이하여 앱의 분석이 가능하다는 자바 프로그램의 단점이 존재한다. 이를 악용하여 앱의 불법 도용 및 표절, 악성코드가 내포된 앱 배포될 수 있다.

이러한 문제를 해결하기 위해 앱의 소프트웨어 버스마크(software birthmark)를 기반의 안드로이드 앱 유사도 분석 연구가 수행되고 있다[2,3,4]. 소프트웨어 버스마크(이하 '특징정보'와 동일)는 실행파일인 DEX로부터 추출된 고유한 특징정보이며 유사성 비교를 통해 앱 도용 탐지 및 식별이 가능하다.

난독화는 프로그램이 역공학을 어렵게 만들기 위해, 소스코드를 변형하는 기법이다. 안드로이드 앱 개발자는 소스코드를 보호하기 위해 난독화 기술을 적용한다. 하지만 특징정보 기반의 유사성 비교를 통한 도용 탐지를 우회하기 위해, 공격자가 난독화 기술을 사용할 수 있다. 하지만 기존의 특징정보 기반의 안드로이드 앱 유사도 비교 연구에서는 난독화에 대한 강인성을 고려하지 않고 있다.

본 논문에서는 난독화가 유사성 비교 연구에 사용되는 특징정보에 미치는 영향을 실험을 통해 분석한다. 실험 대상은 카테고리별 오픈 소스 앱을 선정하였으며, 문자열(string), API, 제어흐름(control flow) 난독화 기법 등을 적용하였다. 그리고 난독화 전/후 앱을 역컴파일하여 추출된 소스코드간의 유사도를 측정하였다. 역컴파일된 소스코드 간의 유사도 측정에는 MOSS[5]를 사용하였다. 이를 통해, 난독화 기법이 유사도 비교에 사용되는 특징정보에 영향을 미치는 것을 확인하였다.

2. 관련 연구

2.1 안드로이드 앱 난독화 기법

난독화는 프로그램의 소스코드가 역공학에 의한 분석을 어렵게 만드는 기술이다. 주요 난독화 기술은 식별자 이름 변경(Renaming Identifiers), 제어 흐름 난독화(Control flow obfuscation), 데이터 난독화(Data obfuscation), API 은닉(API hiding) 등이 있다. 식별자 이름 변경 기법은 클래스, 변수, 메소드 등의 식별자 이름을 의미없는 문자로 변경하여 역공학에 의해 추출된 소스코드 분석시 식별자 이름으로 유추할 수 있는 정보의 분석을 어렵게 만드는 기법이다. 제어 흐름 난독화는 제어 흐름을 변경하는 난독화 기술로 분기 명령어를 수정 및 추가하여 소스코드의 문맥 파악을 어렵게하는 기술이다. 데이터 난독화는 데이터의 구조나 형태를 변경하는 기법으로 사용되는 데이터들을 집합으로 재구성, 분리하거나 사용되는 문자열을 암호화된 문자열로 변경함으로써 데이터를 보호하고 역공학시 정적 분석을 어

렵게 만드는 기술이다. API 은닉은 외부에 의한 분석과 수정에 대해 앱이 사용하는 API 호출 정보를 은닉하여 프로그램의 주요 기능을 보호하고 파악하기 어렵게 하는 기술이다[6].

안드로이드 앱 난독화를 위해 주로 사용되는 난독화 도구는 Proguard[7], DashO[8], DexProtector[9] 등이 있다. 표 1은 난독화 도구별로 제공하는 기법들을 정리한 것이다.

표 1. 안드로이드 앱 난독화 도구 비교
Table 1. Andriod opfuscation tools

Obfuscation	Proguard	DashO	DexProtector
Shrinking	○	○	○
Optimization	○	○	○
Renaming	○	○	○
String encryption	×	○	○
Class encryption	×	×	○
Control flow	×	○	×
API hiding	×	×	○

Proguard는 범용 도구로써 구글에서 Android SDK에 포함하여 기본으로 제공하는 난독화 도구이다. 주요 기능은 사용되지 않는 클래스, 필드, 메소드를 삭제하고(Shrinking), 바이트코드를 최적화하며(Optimization) 클래스, 필드, 메소드 등 식별자 이름을 변경(Renaming)한다.

DashO와 DexProtector는 상용 난독화 도구로써 Proguard의 주요 기능을 포함하며, 그 외에 문자열 암호화, 제어 흐름, 클래스 암호화, API 은닉을 제공한다.

위와 같은 안드로이드 앱 난독화 도구로 인하여 앱의 불법 도용 탐지에 사용되는 식별자 이름 정보, API, 문자열 등 특징정보가 달라지며, 이를 기반으로 한 유사도 비교 연구의 신뢰성은 떨어진다.

2.2 안드로이드 앱 유사성 비교 연구

기존 안드로이드 앱 유사성 비교 연구에서는 Third-Party 안드로이드 마켓에서 재패키징된 앱을 탐지하기 위해, 역공학을 통해 앱의 실행 명령어(opcode)와 fuzzing hashing 기반으로 유사성 비교를 수행하였다[2]. 또한, 역컴파일된 소스 코드 및 실행 명령어를 기반으로 MOSS와 k-gram을 통한 유사성 비교 기법이 제안되었다[3]. 하지만 실행 명령어는 표 1의 최적화(optimization) 및 제어흐름(control flow) 난독화에 등에 의해 쉽게 변경되는 것으로 알려져 있다. API k-gram 기반의 안드로이드 버스마크 논문[4]에서는 DEX에서 추출한 API 정보를 특징정보로 사용하여 k-gram 기반의 안드로이드 앱 유사성 비교 기법을 제안하였다. 하지만 API 정보를 특징정보로 하기 때문에 API 은닉 난독화가 적용될 경우 특징정보로써의 강인성이 떨어진다.

위와 같이, 기존 안드로이드 앱 유사성 비교 연구에서 사용되는 특징정보들은 난독화에 대한 강인성이 낮다. 이 외에 안드로이드 난독화 도구로 인하여 식별자 이름 정보, API, 문자열 등 특징정보의 강인성이 낮아질 수 있다.

3. 제안 방안

본 논문에서는 난독화가 앱 유사도에 미치는 영향을 분석하기 위해 난독화 도구를 이용하여 앱에 난독화를 적용한 후 MOSS로 원본 앱과 난독화된 앱의 역컴파일된 소스코드 간 유사도 측정을 한다. 그리고 각 난독화 도구가 제공하는 기법에 대한 유사성 비교를 통해 어느 기법이 유

사도에 많은 영향을 미치는지 확인한다. 또한, 난독화된 소스 코드를 확인하여 유사도 분석에 사용되는 특징정보가 난독화 기법에 영향을 받는 정도를 확인한다.

원본 앱과 난독화를 적용한 앱 간의 유사성 비교를 위해 오픈 소스 앱을 대상으로 난독화를 적용한다. 이때, Proguard에서 제공하는 기법인 shrinking, Optimization, Renaming은 선택 사항 없이 기본으로 설정되어 난독화를 수행한다. 따라서 위 3가지 기법을 동시에 제공하는 DashO와 DexProtector에서도 이를 기본으로 설정하여 난독화를 수행 하였다. 그 외에 DashO와 DexProtector에서 제공하는 String encryption, Class encryption, Control flow, API hiding을 개별적으로 수행하며, 각 도구 별로 제공하는 기법을 조합하여 난독화를 적용한다.

원본 앱과 난독화된 앱으로부터 소스 코드를 추출하기 위해, 두 APK 파일을 언패킹하여 실행파일인 classes.dex를 추출한 뒤, 역컴파일(decompile)을 통해 자바 소스 코드를 추출한다. 이에 사용된 도구는 JD-GUI[10]와 Dex2Jar[11]이다. 두 앱의 추출된 소스 코드를 기반으로 MOSS를 사용하여 유사도를 측정한다.

4. 실험 및 분석

본 논문에서 제안하고 있는 난독화가 유사도에 미치는 영향 분석을 위해, 오픈 소스 앱을 대상으로 주요 카테고리 별 앱을 수집하고 이를 대상으로 실험을 진행하였다. 실험 대상 앱은 표 2와 같다. 본 논문에서는 실험 대상 앱의 경우 자바 소스를 보유하고 있지만, Proguard의 경우 난독화 수행 과정 중 자바 소스에서 APK 파일로 패킹되는 과정 중에 난독화가 적용된다. 따라서 패킹 과정에서 발생되는 컴파일에 따른 최적화 등의 영향과 컴파일된 난독화 앱으로부터 소스 코드를 추출하기 위한 역컴파일 과정에서 사용되는 Dex2Jar, JD-GUI의 도구로 인한 영향은 고려하지 않는다. 마찬가지로 DashO, DexProtector 도구에 의한 난독화와 역컴파일 과정에서 사용되는 컴파일러 등 도구에 의해 발생되는 영향은 고려하지 않는다.

표 2. 실험대상 앱
Table 2. Experiment object App.

Category	Application
Weather	ADGWeather
	omsk-air-android
Bluetooth	BluetoothChat
	BluetoothSMS
Battery	BatteryIndicator
	SLWBatteryWidget
Keyboard	NorwegianIMESettings
	Softkeyboard-4.2
Notepad	mini_notepad
	notepad

표 3. Proguard 도구를 적용한 앱간의 유사성 분석
Table 3. Evaluation with Proguard

Category	Application	Similarity
Weather	ADGWeather	30%
	omsk-air-android	49%
Bluetooth	BluetoothChat	30%
	BluetoothSMS	26%
Battery	BatteryIndicator	39%
	SLWBatteryWidget	29%
Keyboard	NorwegianIMESettings	71%
	Softkeyboard-4.2	35%
Notepad	mini_notepad	17%
	notepad	34%

각 앱을 Proguard, DashO, DexProtector로 난독화하며, 이때 각 도구의 난독화 기법을 각기 적용하고 MOSS를 사용하여 난독화 적용 전, 후의 유사도를 측정하였다. 우선, 원본 앱을 기준으로 Proguard 도구를 이용해 난독화를 적용한 앱 간의 유사도 측정 결과는 표 3과 같다. 이를 통해 기본으로 제공되는 Proguard의 난독화를 적용할 시 원본 대비 최소 29%, 최대 83% 까지 유사도가 크게 떨어짐을 확인할 수 있다. 그림 1은 오픈 소스 코드의 클래스, 메소드, 변수를 무의미한 값으로 변경하는 등 식별자 이름 변경 기법이 적용된 결과를 보여준다. 위의 결과에 따르면, 메소드 정보, 식별자, 실행 명령어를 특징정보로 사용하는 유사도 측정 기법의 경우 난독화에 의한 강인성이 낮아질 수 있음을 알 수 있다.

```
public class SearchAdapter extends BaseAdapter {
Context context;
ArrayList<SearchObj> serArr = new
ArrayList<SearchObj>();
View child;
TextView loc;
public SearchAdapter(Context c,
ArrayList<SearchObj> so) {
        this.context = c;
        this.serArr = so;
    }
```

난독화 전

```
public class a extends BaseAdapter {
Context a;
ArrayList b = new ArrayList();
View c;
TextView d;
public a(Context paramContext, ArrayList
paramArrayList)
  {
    this.a = paramContext;
    this.b = paramArrayList;
  }
```

난독화 후

그림 1. Proguard 난독화 적용 전, 후
Fig. 1. Proguard Obfuscation results

표 4는 DashO 도구를 이용해 난독화를 수행한 결과이다. 이때, Proguard에서 제공하는 기법(Shrinking, Optimization, Renaming)이 동일하게 포함되므로 이를 기본(default)로 설정하여 난독화를 수행한다. DashO 분석 결과를 살펴보면, 문자열 난독화와 제어 흐름 기법과 기본적인 난독화 기법을 비교하였을 때 최대 41%의 유사도 차이를 보였다. 이를 통해 식별자 이름 변경, 최적화, 사용되지 않는 클래스와 변수 제거 등의 비교적 단순한 난독화 기법이 문자열 난독화나 제어 흐름보다 상대적으로 유사도에 영향을 미침을 알 수 있었다.

```
public class MainActivity extends Activity
    implements LocationListener{
    public class FavParsTask extends AsyncTask{
        protected transient Object
doInBackground(Object aobj[]){
            Log.i("Fave Parse URL", (new
StringBuilder()).append(parseURL).toString());
            parsingHandler = new
ParsingHandler(parseURL);
            urlBundle.putString("URL", parseURL);
          return null;      }
```

난독화 전

```
public class eval_a extends Activity{
    public class m extends AsyncTask{
        public static eval_a m(m m1){
            return m1.eval_f;
        }public transient Object
doInBackground(Object aobj[]){
            try{
Log.i(eval_dk.subSequence("S\177a}]{bZrl|arv", 21),
(new StringBuilder(eval_dk.subSequence("qwj=(",
4))).append(eval_f.eval_n).toString());
 eval_f.eval_b = new eval_n(eval_f.eval_n);
            }return null;    }
```

난독화 후

그림 2. DashO 난독화 적용 전, 후
Fig. 2. DashO Obfuscation results

또한, 모든 기법을 적용하였을 때 대체적으로 유사도가 크게 떨어지는 결과를 확인할 수 있다.

표 4. DashO 도구를 적용한 앱 간의 유사성 분석
Table 4. Evaluation with DashO Obfuscation

Category	Application	Similarity						
		default	String	Control	default & String	default & Control	String & Control	ALL
Weather	ADGWeather	78%	85%	86%	74%	76%	82%	73%
	omsk-air-android	43%	82%	72%	33%	29%	61%	19%
Bluetooth	BluetoothChat	63%	71%	73%	39%	48%	53%	27%
	BluetoothSMS	53%	72%	67%	42%	36%	56%	29%
Battery	BatteryIndicator	39%	80%	75%	29%	26%	65%	19%
	SLWBatteryWidget	63%	75%	40%	57%	27%	40%	26%
Keyboard	NorwegianIMESettings	71%	89%	77%	67%	58%	73%	54%
	Softkeyboard-4.2	76%	82%	68%	75%	54%	67%	53%
Notepad	mini_notepad	83%	88%	88%	82%	82%	87%	82%
	notepad	72%	66%	71%	49%	56%	51%	35%

그림 2는 모든 기법을 적용한 경우로, 소스 코드의 메소드 정보, 식별자, 문자열, 제어 흐름이 변경된 결과를 보여준다. 이를 통해, 난독화로 인해 변경된 정보를 특징정보로 사용하는 유사도 기법들은 난독화에 영향을 받음을 알 수 있다.

표 5는 DexProtector 도구를 이용해 문자열 난독화, 클래스 난독화, API 은닉을 적용한 결과이다. 다른 도구들과는 달리 클래스 난독화를 제공하기 때문에 DEX 파일을 암호화하여 역컴파일을 불가능하게 하므로 유사도 측정에서 제외되었다. 따라서 문자열 난독화, API 은닉 기법에 대한 유사도를 측정한다. 위의 결과를 통해, API 은닉 기법이 문자열 난독화에 비해 유사도가 매우 크게 떨어졌다.

그림 3을 살펴보면, API 은닉 시 리플렉션(reflection)을 이용하기 위해 Java.lang.reflect의 메소드(.forName, .getMethod, method.invoke 등)의 코드 삽입으로 인해 유사도에 영향을 미친 것으로 판단된다. 두 기법을 적용한 결과 API 은닉을 적용한 경우에 비해 크게 떨어지지 않은 것으로 확인되었다. 문자열 난독화를 적용하였을 때 유사도가 절반 이하로 떨어진 것에 비해 두 기법을 적용하였을 때 API 은닉만을 적용한 유사도에 비해 최대 3% 차이에 불과하며, 유사도가 변하지 않는 경우도 존재했다. 이에 대한 분석은 향후에 소스 코드의 상세 분석을 통해 API 은닉과 문자열 난독화가 서로 어떤 연관성이 있는지 확인할 필요가 있다.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

Class class3 = Class.forName(attachBaseContext
("\u2CC1\u0213\uEB99\u3F98\uCCB3\u91E7\uF327\...
\u9297\uAB42"));
String s = attachBaseContext("\u2CC7\u0218\ ...
\uCCB5\u91FC");
Class aclass[] = new Class[2];
aclass[0] = class1;
aclass[1] = Integer.TYPE;
Method method = class3.getMethod(s, aclass);
Object aobj[] = new Object[2];
aobj[0] = attachBaseContext("\u2CC4\...\uEB85");
aobj[1] = Integer.valueOf(0);
Object obj = method.invoke(this, aobj);
```

그림 3. DexProtector 난독화 후
Fig. 3. DexProtect obfuscation results

표 5. DexProtector 도구를 적용한 앱 간의 유사성 분석
Table 5. Evaluation with DexProtector tool

Category	Application	Similarity						
		String	Class	API	String & Class	Class & API	String & API	ALL
Weather	ADGWeather	43%	-	15%	-	-	14%	-
	omsk-air-android	45%	-	23%	-	-	22%	-
Bluetooth	BluetoothChat	41%	-	16%	-	-	13%	-
	BluetoothSMS	44%	-	15%	-	-	13%	-
Battery	BatteryIndicator	43%	-	24%	-	-	23%	-
	SLWBatteryWidget	49%	-	33%	-	-	33%	-
Keyboard	NorwegianIMESettings	47%	-	26%	-	-	25%	-
	Softkeyboard-4.2	49%	-	12%	-	-	12%	-
Notepad	mini_notepad	45%	-	14%	-	-	13%	-
	notepad	38%	-	15%	-	-	12%	-

5. 결론 및 향후 연구방향

본 논문에서는 난독화가 유사도 측정에 미치는 영향에 대해 살펴보았다. 오픈 소스 앱을 대상으로 3가지 안드로이드 앱 난독화 도구를 사용하였으며, 각 도구의 기법들을 각기 적용하여 어느 난독화 기법이 유사도 측정에 영향을 미치는지 확인하였다. 실험 과정 중 컴파일, 역컴파일, 난독화를 위한 도구에서 발생되는 코드 최적화 등의 영향은 고려하지 않았다.

기본적으로 구글에서 제공하는 Proguard만으로도 안드로이드 앱의 유사도 측정에 많은 영향을 줄 수 있음을 확인하였다. DexProtector는 클래스 난독화로 인하여 앱의 역컴파일을 불가능하게 하여 유사도 측정을 우회함을 알 수 있었으며, API 은닉 기법이 상대적으로 유사도에 가장 많은 영향일 미치는 것으로 판단되었다. 이와 같이, 난독화 적용시 유사도 비교의 한계를 보여주며, 유사도 측정에 사용되는 특징정보인 메소드 정보, 식별자, API 정보, 실행명령어 등이 난독화 기법에 영향을 받음을 알 수 있었다. 논문에서 보이는 실험 결과는 소스 코드 기반의 유사도만 제공하므로, 다양한 유사도 측정 기법에 대한 비교를 통해 난독화에 강인한 특정정보 추출 기법을 연구할 예정이다.

참 고 문 헌

[1] ITWorld, “불법 다운로드 앱 판치는 안드로이드, 개발자 시름 ‘깊어져’”, http://www.itworld.co.kr/news/91735., 2015.02.

[2] W. Zhou, et al., “DroidMOSS: Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces”, Proceedings of the second ACM conference on Data and Application Security and Privacy, 2012.

[3] Ko, Jeonguk, et al., “Measuring similarity of android applications via reversing and K-gram birthmarking”, Proceedings of the 2013 Research in Adaptive and Convergent Systems, 2013.

[4] 박희완, “API K-gram 기반의 안드로이드 버스마크”, 정보처리학회논문지, pp.177-180, 2013.

[5] Alex Aiken, “A System for Detecting Software Plagiarism - MOSS”, http://theory.stanford.edu/~aiken/moss/, 2015.

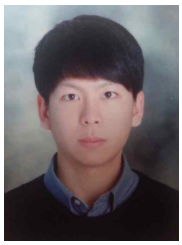
[6] Yuxue, Piao, 정진혁, 이정현, “모바일 난독화기술 동향”, 한국통신학회지, 29권 8호, pp.65-71, 2012.
[7] Proguard, http://developer.android.com/tools/help/proguard.html, 2015.
[8] DashO, http://www.preemptive.com/products/dasho, 2015.
[9] DexProtector, https://dexprotector.com/,2015.
[10] JD-GUI, http://jd.benow.ca/, 2015.
[11] Dex2jar, http://code.google.com/p/dex2jar/, 2015.

저 자 소 개

박종화 (JongHwa Park)

2015년 2월 : 단국대학교 소프트웨어학과 졸업
2015년 3월~현재 : 단국대학교 컴퓨터학과 소프트웨어학 석사과정
<관심분야 : 시스템 보안, 역공학>

이경묵 (Kyungmook Lee)



2015년 2월 공주대학교 정보통신학과 졸업
2015년 3월~현재 : 단국대학교 컴퓨터학과 소프트웨어학 석사과정
<관심분야 : 시스템 보안, 접근제어, 모의해킹 >

임경환 (Kyeonghwan Lim)

2015년 2월 : 단국대학교 소프트웨어학과 졸업
2015년 3월~현재 : 단국대학교 컴퓨터학과 소프트웨어학 석사과정
<관심분야 : 시스템 보안, 스마트폰 보안>

김동진 (Dongjin Kim)

2009년 2월 : 단국대학교 컴퓨터학과 이학사
2011년 2월 : 단국대학교 컴퓨터학과 공학석사
2011년 3월～현재 : 단국대학교 컴퓨터학과 박사과정
<관심분야> 컴퓨터보안, 소프트웨어 지적재산권 보호, 스마트폰 보안, 시스템 소프트웨어 등

조 성 제 (Seong-je Cho)

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 지적재산권 보호, 시스템 소프트웨어, 스마트폰 보안 등

정 영 기 (Youngki Chung)

1981년 : BS in Industrial Engineering Columbia University, NY USA
1989년 : MS in Computer Science New York University, NY USA
1986년 : Computer Network Systems
1989년 : AT&T Bell Labs
1997년 : Hyundai Electronics Industry
1999년 : Motorola
2005년~현재 : Samsung Electronics
2013년~현재 : 단국대학교 컴퓨터학과 박사과정
<관심분야> 스마트폰 보안 등