4^{th} 과제: 안드로이드 에뮬레이터 탐지 앱 구현

(2020년 2학기: 운영체제보안)

분반: 2분반

이름	한예진	유하연	박진아	양지원
학번	32164881	32182861	32181912	32182595
학과	무역학과	컴퓨터공학과	컴퓨터공학과	컴퓨터공학과

제출일: 2020년 12월 15일

1. 안드로이드 루팅 탐지 – 한예진, 박진아 수행

1-1 루팅 탐지 알고리즘 및 구현 코드 설명

1) su 명령어로 확인

안드로이드는 리눅스 기반 운영체제이기 때문에 su 명령어 실행이 가능하다. 리눅스에서 su 명령어는 사용자를 root로 변경하는 명령어이다. 일반적으로 루팅이 되지 않은 안드로이드 디바이스는 su 명령어가 존재하지 않는다. 따라서 안드로이드에서 su 명령어 실행이 된다면 루팅이 되어 있다고 할 수 있다.

```
70      //su 명령어로 확인
71      public boolean isRootedBySUCmd() {
72          return canExecCmd( command: "su") || canExecCmd( command: "/system/bin/which su")
73              || canExecCmd( command: "/system/xbin/which su") || canExecCmd( command: "which su");
74      }
75      public boolean canExecCmd(String command){
76          boolean isExecuted;
77          try{
78              Runtime.getRuntime().exec(command);
79              isExecuted = true;
80          } catch(Exception e){
81              isExecuted = false;
82          }
83          return isExecuted;
84      }
```

위의 코드를 보면 isRootedBySUCmd()함수에서 su 명령어를 인자로 하여 canExecCmd()를 호출한다. canExecCmd()는 Runtime.getRuntime().exec() API를 이용하여 인자로 들어오는 명령어를 실행한다. 만약 su 명령어를 실행했을 때 Exception이 발생하면 루팅이 되지 않은 디바이스고, 정상적으로 실행이 되면 루팅된 디바이스로 탐지한다.

2) su 바이너리 파일 이름 기반 존재 유무 확인

또한 su 이름을 갖는 바이너리 파일을 검색하여 루팅을 탐지할 수 있다.

-파일 목록

"/sbin/su", "/system/su", "/system/sbin/su", "/data/data/com.noshufou.android.su", "/system/bin/.ext", "/system/xbin/.ext", "/system/bin/su", "/system/xbin/su", "/system/bin/.ext/.su", "/system/usr/su-backup","/system/usr/we-need-root/su-backup","/system/app/Superuser.apk"

일반적으로 루팅을 하고 나면 추후에 다시 root권한을 이용하기 위해 su이름을 갖는 바이너리

파일을 만들어 놓는다. 이렇게 루팅시 생성되는 기본적인 su파일, apk파일 이름을 기반으로 존재 유무를 확인한다.

```
86      // root 관련 바이너리 이름 기반 존재 유무 확인
87      public boolean isRootedByFilename(){
88          Boolean check = false;
89          String[] arrayOfFile = {"/sbin/su", "/system/su", "/system/sbin/su",
90                  "/data/data/com.noshufou.android.su", "/system/bin/.ext", "/system/xbin/.ext",
91                  "/system/bin/su", "/system/xbin/su", "/system/bin/.ext/.su", "/system/usr/su-backup",
92                  "/system/usr/we-need-root/su-backup","/system/app/Superuser.apk"};
93          for(String path:arrayOfFile){
94              if(new File(path).exists())
95                  check = true;
96          }
97          return check;
98      }
```

위의 코드를 보면 isRootedByFilename()함수에서 arrayOfFile의 파일명을 for문을 돌려서 File.exists()로 하나씩 체크하여 만약 존재한다면 루팅된 디바이스로 탐지한다.

3) build.prop 파일 기본값 확인 - tag: test-keys

기본적으로 안드로이드에서 /system/build.prop 파일의 ro.build.tag 옵션을 비롯한 여러 옵션들은 release-keys로 설정되어 있다. release-keys는 커널이 컴파일될 때 공식 개발자의 official 키로 서명되었음을 의미한다. 그런데 루팅이 되는 경우 이 값이 test-keys로 바뀌는데, 이는 제3자에 의해 만들어진 AOSP(Android Open Source Project) test keys로 서명되었음을 의미한다. 따라서 이 값을 확인하여 루팅을 탐지한다.

```
100     // build tag 확인
101     public boolean isRootedbyBuildTag(){
102         boolean TestKeysFlag = false;
103         String buildTags = Build.TAGS;
104         if(buildTags != null && buildTags.contains("test-keys"))
105             TestKeysFlag = true;
106         return TestKeysFlag;
107     }
```

위의 코드를 보면 isRootedbyBuildTag()함수에서 build.tags를 불러와서 만약 test-keys가 존재할 시에는 루팅된 디바이스로 탐지한다.

4) busybox 및 관련 명령어 탐지

안드로이드는 리눅스 기반임에도 불구하고 리눅스에서 주로 사용되는 일부 명령어들이 존재하지 않는다. 이러한 불편함을 해결하기 위해 busybox 명령어 모음 패키지나 필요에 따른 명령어 바이너리를 설치하여 사용하게 된다. Busybox에 있는 명령어의 대부분은 리눅스와 호환되기 때문에 busybox를 설치하면 안드로이드에서 리눅스 명령어를 자유롭게 사용할 수 있다. 따라서 루팅

된 장치에 종종 설치되는 busybox 유무를 확인하여 루팅 여부를 감지할 수 있다.

```
110     //busybox 체크
111     public boolean isRootedByBusyBox(){
112         String [] binaryPaths= {
113                 "/data/local/",
114                 "/data/local/bin/",
115                 "/data/local/xbin/",
116                 "/sbin/",
117                 "/su/bin/",
118                 "/system/bin/",
119                 "/system/xbin/",
120                 "/system/bin/failsafe/",
121                 "/system/bin/.ext/",
122                 "/system/sd/xbin/",
123                 "/system/usr/we-need-root",
124                 "/system/app/Superuser.apk",
125                 "/cache",
126                 "/data",
127                 "/dev"
128         };
129
130         for(String path:binaryPaths){
131             File f = new File(path,  child: "busybox");
132             boolean busyboxFlag = f.exists();
133             if(busyboxFlag)
134                 return true;
135         }
136         return false;
137         }
```

위의 코드를 보면 isRootedByBusyBox()에서 for문을 통해 binaryPaths에 있는 폴더 밑에 busybox 파일을 확인한다. Busybox가 존재할 시에는 루팅된 디바이스로 탐지한다.

5) adb 명령어로 확인

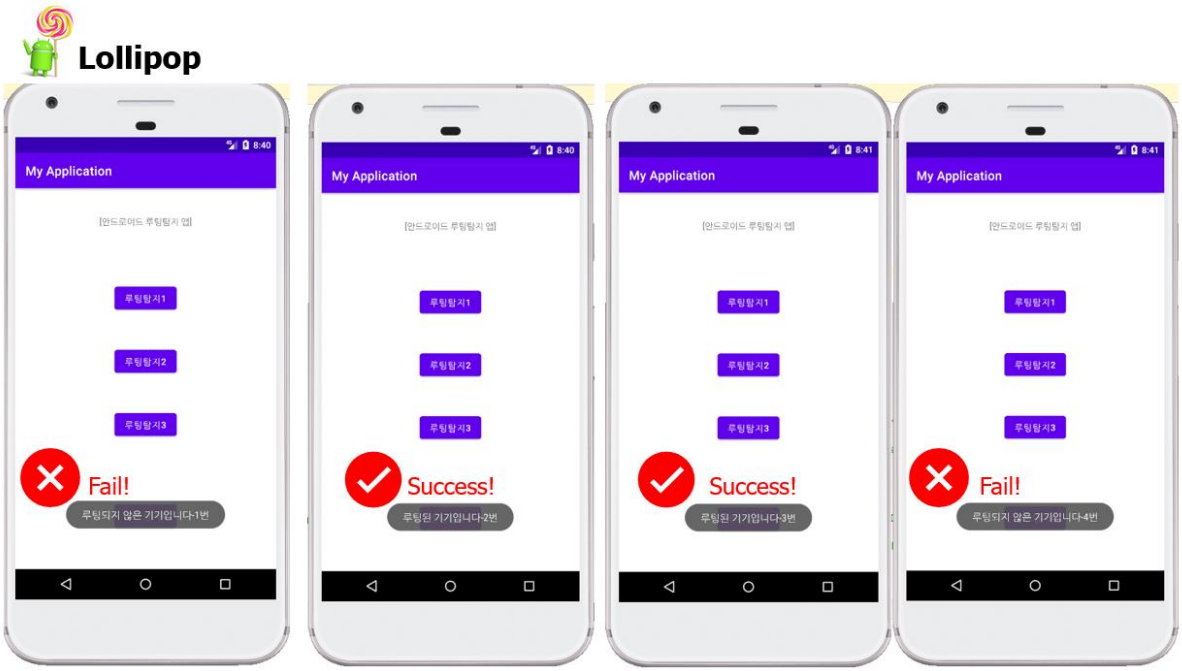
```
C:\Users\yunah\.android\platform-tools>adb shell
generic_x86_arm:/ # whoami
root
generic_x86_arm:/ # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard
),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=
r:su:s0
generic_x86_arm:/ #
```

adb shell 명령어로 현재 자신이 루트 권한을 가지고 있는지 확인한다. 위 캡쳐화면은 루팅이 된 상태임을 알 수 있다.

1-2 실험 환경 및 실험 결과 분석

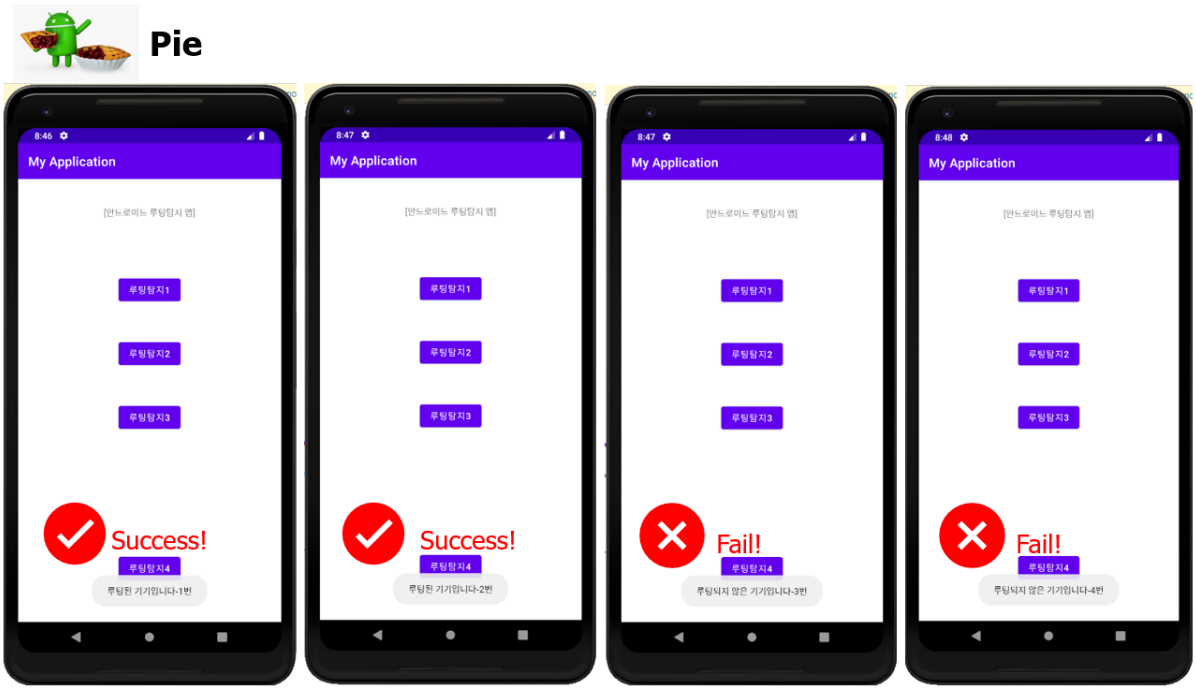
1) AVD(Android Virtual Device)

[1] Lollipop, Android 5.0 x86, API level = 21



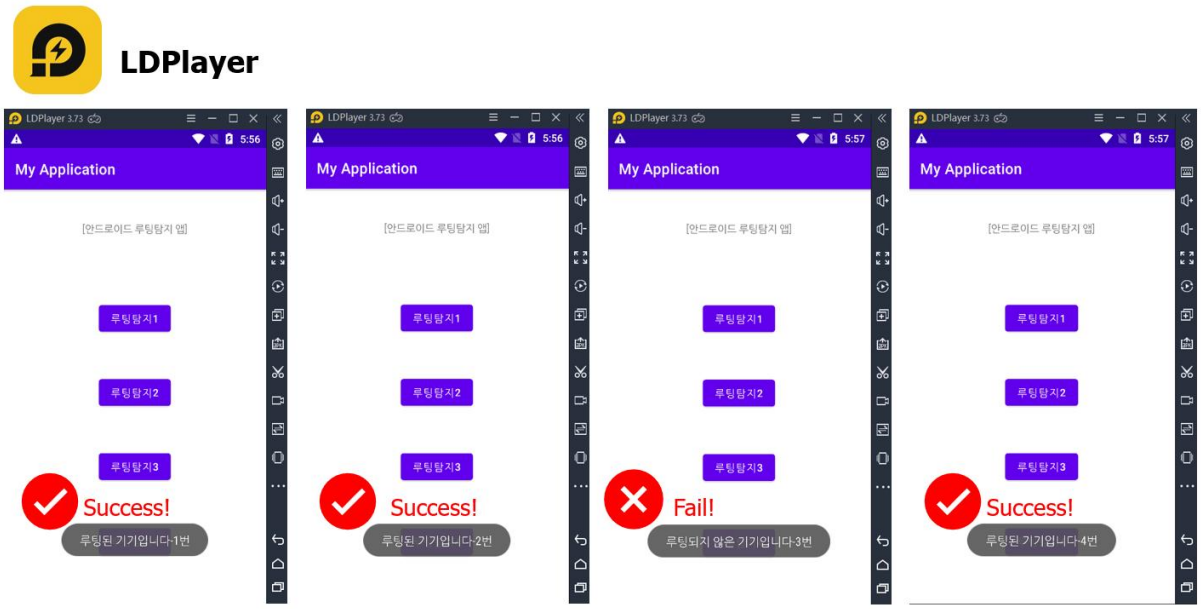
1. Su 명령어가 실행되지 않아 루팅이 되었음을 탐지할 수 없었다.
2. su이름을 갖는 바이너리 파일이 존재하여 루팅이 되었음을 탐지하였다.
3. Build.prop. 파일의 tag 옵션이 test-keys로 바뀌었기 때문에 루팅이 되었음을 탐지하였다.
4. Busybox가 설치되어있지 않아서 루팅이 되었음을 탐지할 수 없었다.

[2] Pie, Android 9.0 (Google X86_ARM), API level = 28

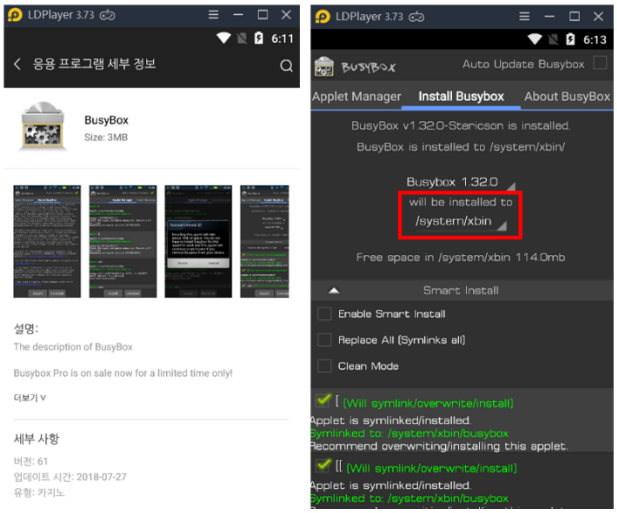


1. Su 명령어가 실행되어 루팅이 되었음을 탐지하였다.
2. su이름을 갖는 바이너리 파일이 존재하여 루팅이 되었음을 탐지하였다.
3. Build.prop. 파일의 tag 옵션이 test-keys로 바뀌지 않고 release-keys로 남아있어서 3번 알고리즘을 통해서는 루팅을 탐지할 수 없었다.
4. Busybox가 설치되어있지 않아서 루팅이 되었음을 탐지할 수 없었다.

2) LDPlayer 3.73



1. Su 명령어가 실행되어 루팅이 되었음을 탐지하였다.
2. su이름을 갖는 바이너리 파일이 존재하여 루팅이 되었음을 탐지하였다.
3. Build.prop. 파일의 tag 옵션이 test-keys로 바뀌지 않고 release-keys로 남아있어서 3번 알고리즘을 통해서는 루팅을 탐지할 수 없었다.
4. 4번 알고리즘은 busybox가 설치되었는지 확인하는 방식으로 루팅을 탐지한다. LDPlayer에서 탐지할 때는 이 방식을 성공시키기 위해 먼저 LD store에 들어가서 BusyBox를 설치하였다.



따라서 busybox를 감지하여 루팅을 탐지할 수 있었다.

1-3 루팅 탐지 실행코드

```java
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textview = findViewById(R.id.textView2);
        Button button1 = findViewById(R.id.root1);
        Button button2 = findViewById(R.id.root2);
        Button button3 = findViewById(R.id.root3);
        Button button4 = findViewById(R.id.root4);
        View.OnClickListener clickListener;
        clickListener = new View.OnClickListener(){
            @Override
            public void onClick(View v){
                switch(v.getId()){
                    case R.id.root1:
                        //roo1을 눌렀을 때의 처리
                        if(isRootedBySUCmd())
                            Toast.makeText(getApplicationContext(), text: "루팅된 기기입니다-1번", Toast.LENGTH_LONG).show();
                        else
                            Toast.makeText(getApplicationContext(), text: "루팅되지 않은 기기입니다-1번", Toast.LENGTH_LONG).show();
                        break;
                    case R.id.root2:
                        //root2를 눌렀을 때의 처리
                        if(isRootedByFilename())
                        Toast.makeText(getApplicationContext(), text: "루팅된 기기입니다-2번", Toast.LENGTH_LONG).show();
                        else
                        Toast.makeText(getApplicationContext(), text: "루팅되지 않은 기기입니다-2번", Toast.LENGTH_LONG).show();
                        break;
                    case R.id.root3:
                        //root3를 눌렀을 때의 처리
                        if(isRootedbyBuildTag())
                            Toast.makeText(getApplicationContext(), text: "루팅된 기기입니다-3번", Toast.LENGTH_LONG).show();
                        else
                            Toast.makeText(getApplicationContext(), text: "루팅되지 않은 기기입니다-3번", Toast.LENGTH_LONG).show();
                        break;
                    case R.id.root4:
                        //root4를 눌렀을 때의 처리
                        if(isRootedByBusyBox())
                            Toast.makeText(getApplicationContext(), text: "루팅된 기기입니다-4번", Toast.LENGTH_LONG).show();
                        else
                            Toast.makeText(getApplicationContext(), text: "루팅되지 않은 기기입니다-4번", Toast.LENGTH_LONG).show();
                        break;
                    default:
                        break;
                }
            }
        };
        button1.setOnClickListener(clickListener);
        button2.setOnClickListener(clickListener);
        button3.setOnClickListener(clickListener);
        button4.setOnClickListener(clickListener);
        }

```

```
<Button
    android:id="@+id/root1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="150dp"
    android:onClick="onClick"
    android:text="루팅탐지1"
```

버튼을 4개 만들어서 뷰에 **android:onClick** 속성을 추가하고 클릭 리스너로 사용할 메서드(**onClick**)의 이름을 넣었다. 그리고 findViewById()를 통해 Button 객체의 ID를 얻어와서 버튼이 눌릴 때마다 onClick을 호출한다. 호출되면 switch문을 통해 각 버튼 별 루팅 탐지 알고리즘으로 나눠진다. 그 후 if 문을 통해 해당하는 루팅 탐지 함수가 반환하는 값이 참인지, 거짓인지 확인하고 참이면 '루팅된 기기입니다-O번' 문구를 출력하고 거짓이면 '루팅되지 않은 기기입니다-O번' 문구를 출력한다.

[MainActivity.java]

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.TextView;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textview = findViewById(R.id.textView2);
        Button button1 = findViewById(R.id.root1);
        Button button2 = findViewById(R.id.root2);
        Button button3 = findViewById(R.id.root3);
        Button button4 = findViewById(R.id.root4);
        View.OnClickListener clickListener;
        clickListener = new View.OnClickListener(){
            @Override
            public void onClick(View v){
                  switch(v.getId()){
         case R.id.root1:
            //roo1 을 눌렀을 때의 처리
            if(isRootedBySUCmd())
                Toast.makeText(getApplicationContext(), "루팅된 기기입니다-1 번",
Toast.LENGTH_LONG).show();
            else
                Toast.makeText(getApplicationContext(), "루팅되지 않은 기기입니다-
1 번", Toast.LENGTH_LONG).show();
break;
```

```
case R.id.root2:
                    //root2 를 눌렀을 때의 처리
                    if(isRootedByFilename())
                    Toast.makeText(getApplicationContext(), "루팅된 기기입니다-
2 번", Toast.LENGTH_LONG).show();
                    else
                    Toast.makeText(getApplicationContext(), "루팅되지 않은
기기입니다-2 번", Toast.LENGTH_LONG).show();
                    break;
                case R.id.root3:
                    //root3 를 눌렀을 때의 처리
                    if(isRootedbyBuildTag())
                        Toast.makeText(getApplicationContext(), "루팅된
기기입니다-3 번", Toast.LENGTH_LONG).show();
                    else
                        Toast.makeText(getApplicationContext(), "루팅되지 않은
기기입니다-3 번", Toast.LENGTH_LONG).show();
                    break;
                case R.id.root4:
                    //root4 를 눌렀을 때의 처리
                    if(isRootedByBusyBox())
                        Toast.makeText(getApplicationContext(), "루팅된
기기입니다-4 번", Toast.LENGTH_LONG).show();
                    else
                        Toast.makeText(getApplicationContext(), "루팅되지 않은
기기입니다-4 번", Toast.LENGTH_LONG).show();
                    break;
                default:
                    break;
            }
        }
    };
    button1.setOnClickListener(clickListener);
    button2.setOnClickListener(clickListener);
    button3.setOnClickListener(clickListener);
    button4.setOnClickListener(clickListener);
    }

//su 명령어로 확인
public boolean isRootedBySUCmd() {
    return canExecCmd("su") || canExecCmd("/system/bin/which su")
        || canExecCmd("/system/xbin/which su") || canExecCmd("which su");
}
public boolean canExecCmd(String command){
    boolean isExecuted;
    try{
        Runtime.getRuntime().exec(command);
        isExecuted = true;
    } catch(Exception e){
        isExecuted = false;
    }
    return isExecuted;
}
```

```java
// root 관련 바이너리 이름 기반 존재 유무 확인
public boolean isRootedByFilename(){
    Boolean check = false;
    String[] arrayOfFile = {"/sbin/su", "/system/su", "/system/sbin/su",
            "/data/data/com.noshufou.android.su", "/system/bin/.ext",
"/system/xbin/.ext",
            "/system/bin/su", "/system/xbin/su", "/system/bin/.ext/.su",
"/system/usr/su-backup",
            "/system/usr/we-need-root/su-backup","/system/app/Superuser.apk"};
    for(String path:arrayOfFile){
        if(new File(path).exists())
            check = true;
    }
    return check;
}

// build tag 확인
public boolean isRootedbyBuildTag(){
    boolean TestKeysFlag = false;
    String buildTags = Build.TAGS;
    if(buildTags != null && buildTags.contains("test-keys"))
        TestKeysFlag = true;
    return TestKeysFlag;
}


//busybox 체크
public boolean isRootedByBusyBox(){
    String [] binaryPaths= {
            "/data/local/",
            "/data/local/bin/",
            "/data/local/xbin/",
            "/sbin/",
            "/su/bin/",
            "/system/bin/",
            "/system/xbin/",
            "/system/bin/failsafe/",
            "/system/bin/.ext/",
            "/system/sd/xbin/",
            "/system/usr/we-need-root",
            "/system/app/Superuser.apk",
            "/cache",
            "/data",
            "/dev"
    };

        for(String path:binaryPaths){
            File f = new File(path, "busybox");
            boolean busyboxFlag = f.exists();
            if(busyboxFlag)
                return true;
        }
        return false;
        }

}
```

2. 안드로이드 에뮬레이터 탐지 – 1),2) 유하연, 3) 양지원 수행

2-1 에뮬레이터 탐지 알고리즘 및 구현 코드 설명

1) 특정 sensor가 존재하는지 확인

실제 device 에서는 다양한 센서가 존재한다. 기기 위치 확인이나 통화 중 거리 확인, 핸드폰 움직임 감지 등을 위해 다양한 센서가 존재한다. 이러한 센서들이 실제로 있는지 확인하고 만약 존재하지 않는다면 에뮬레이터임을 탐지하는 방법이다.

```
28      if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) == null) {
29          newRating1++;
30          System.out.println("accelerometer TRUE");
31      } else {}
32
33      if (mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT) == null) {
34          newRating1++;
35          System.out.println("light TRUE");
36      } else {}
37
38      if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) == null) {
39          newRating1++;
40          System.out.println("magnetic TRUE");
41      } else {}
43       if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) == null) {
44           newRating1++;
45           System.out.println("gravity TRUE");
46       } else {}
47
48       if (mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY) == null) {
49           newRating1++;
50           System.out.println("proximity TRUE");
51       } else {}
52
53       rating1 = newRating1;
```

위의 코드에서 확인한 센서는 움직임(흔들기, 기울기 등)을 감지하는 TYPE_ACCELEROMETER 과 TYPE_GRAVITY, 화면 밝기를 제어하는 TYPE_LIGHT, 나침반 등 지자기장을 이용하여 방향을 감지하는 TYPE_MAGNETIC_FIELD, 통화 중 전화 위치를 확인(뷰 화면과의 근접도 측정, 휴대전화를 귀에 대고 있는지 확인한다.)하는 TYPE_PROXIMITY 센서이다. 실제 device 에서 가장 공통적으로 있는 센서들이 실제로 있는지 확인하고, 만약 이러한 센서들이 없다면 newRating 값을 1 씩 증가한다. 최종적으로 newRating(=rating1) 값이 일정 정수를 넘어가게 된다면, 실제 device 에서 있어야 할 센서가 없는 것으로 판단하고 에뮬레이터라고 판단한다.

2) OS build 정보 확인

OS build 정보를 확인한다. import android.os.Build 를 통해 현재 실행중인 OS 의 상품, 제조사, 브랜드, 모델, 하드웨어 등의 정보를 확인한다. 이러한 정보들이 에뮬레이터(녹스, LD 플레이어 등)의 정보와 일치하는 부분이 있다면 에뮬레이터라고 판단한다.

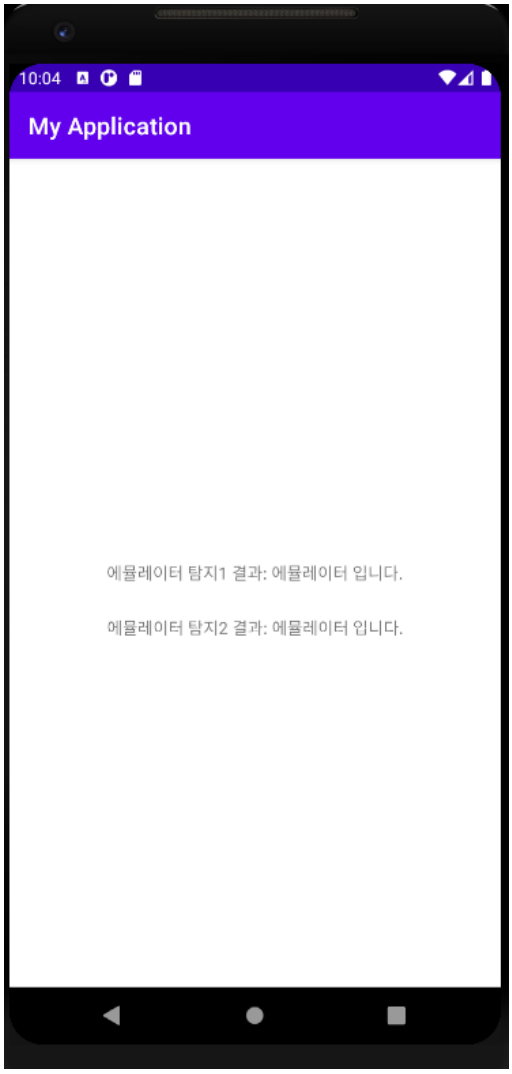
```
74    public static boolean OsBuildInfo() {
75        int newRating = 0;
76        if(rating2 < 0) {
77            if (Build.PRODUCT.contains("sdk") ||
78                    Build.PRODUCT.contains("Andy") ||
79                    Build.PRODUCT.contains("ttVM_Hdragon") ||
80                    Build.PRODUCT.contains("google_sdk") ||
81                    Build.PRODUCT.contains("Droid4X") ||
82                    Build.PRODUCT.contains("nox") ||
83                    Build.PRODUCT.contains("sdk_x86") ||
84                    Build.PRODUCT.contains("sdk_google") ||
85                    Build.PRODUCT.contains("vbox86p")) {
86                newRating++;
87            }
88
89            if (Build.MANUFACTURER.equals("unknown") ||
90                    Build.MANUFACTURER.equals("Genymotion") ||
91                    Build.MANUFACTURER.contains("Andy") ||
92                    Build.MANUFACTURER.contains("MIT") ||
93                    Build.MANUFACTURER.contains("nox") ||
94                    Build.MANUFACTURER.contains("TiantianVM")){
95                newRating++;
96            }
97
98            if (Build.BRAND.equals("generic") ||
99                    Build.BRAND.equals("generic_x86") ||
100                   Build.BRAND.equals("TTVM") ||
101                   Build.BRAND.contains("Andy")) {
102               newRating++;
103           }
104
105           if (Build.DEVICE.contains("generic") ||
106                   Build.DEVICE.contains("generic_x86") ||
```

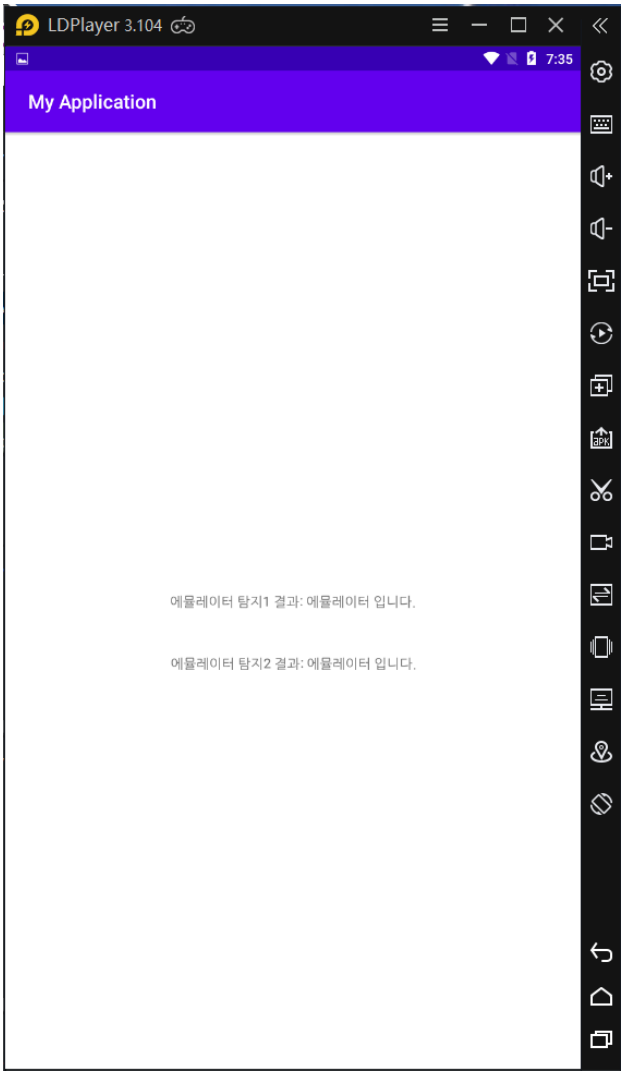
```
                Build.DEVICE.contains("generic_x86_arm") ||
                Build.DEVICE.contains("Andy") ||
                Build.DEVICE.contains("ttVM_Hdragon") ||
                Build.DEVICE.contains("Droid4X") ||
                Build.DEVICE.contains("nox") ||
                Build.DEVICE.contains("aosp") ||
                Build.DEVICE.contains("generic_x86_64") ||
                Build.DEVICE.contains("vbox86p")) {
            newRating++;
        }

        if (Build.MODEL.equals("sdk") ||
                Build.MODEL.equals("google_sdk") ||
                Build.MODEL.contains("Droid4X") ||
                Build.MODEL.contains("TiantianVM") ||
                Build.MODEL.contains("Andy") ||
                Build.MODEL.equals("Android SDK built for x86_64") ||
                Build.MODEL.equals("Android SDK built for x86")) {
            newRating++;
        }

        if (Build.HARDWARE.equals("goldfish") ||
                Build.HARDWARE.equals("vbox86") ||
                Build.HARDWARE.contains("nox") ||
                Build.HARDWARE.contains("ttVM_x86")) {
            newRating++;
        }

        if (Build.FINGERPRINT.contains("generic/sdk/generic") ||
                Build.FINGERPRINT.contains("generic_x86/sdk_x86/generic_x86") ||
                Build.FINGERPRINT.contains("Andy") ||
                Build.FINGERPRINT.contains("ttVM_Hdragon") ||
                Build.FINGERPRINT.contains("generic_x86_64") ||
                Build.FINGERPRINT.contains("generic_x86_arm") ||
                Build.FINGERPRINT.contains("generic/google_sdk/generic") ||
                Build.FINGERPRINT.contains("vbox86p") ||
                Build.FINGERPRINT.contains("generic/vbox86p/vbox86p")) {
            newRating++;
        }
        rating2 = newRating;
    }
```

Build 정보 중 PRODUCT, MANUFACTURER, BRAND, DEVICE, MODEL, HARDWARE, FINGERPRINT 정보를 비교해 확인한다. 각각의 정보에 에뮬레이터에서의 정보가 있을 경우 newRating값을 1씩 증가시킨다. 최종적으로 newRating(rating2)값이 양수라면 에뮬레이터에 대한 OS build 정보가 포함된 것으로 판단한다.

위의 탐지 결과를 바탕으로 rating1 값과 rating2 값이 0 보다 크다면 에뮬레이터라고 판단하고, 탐지 결과를 text 로 띄웠다. 에뮬레이터 탐지 1 은 센서 확인, 에뮬레이터 탐지 2 는 OS Build 정보 확인이다.

다음은 adb 에뮬레이터와 ld 플레이어를 사용하여 탐지에 성공한 화면을 보인 캡쳐화면이다. ld 플레이어에서는 apk 파일을 realese 버전으로 build 하여 설치하여 실행하였다.





[MainActivity.java] – 에뮬레이터 탐지 1,2

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;


import android.hardware.Sensor;
import android.hardware.SensorManager;
```

```java
import android.os.Build;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private SensorManager mSensorManager;
    private static int rating1 = -1;
    private static int rating2 = -1;

    @Override protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView textView1 = (TextView) findViewById(R.id.text1);
        TextView textView2 = (TextView) findViewById(R.id.text2);


        int newRating1=0;
        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

        if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) == null) {
            newRating1++;
            System.out.println("accelerometer TRUE");
        } else {}

        if (mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT) == null) {
            newRating1++;
            System.out.println("light TRUE");
        } else {}

        if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) == null) {
            newRating1++;
            System.out.println("magnetic TRUE");
        } else {}

        if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) == null) {
            newRating1++;
            System.out.println("gravity TRUE");
        } else {}

        if (mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY) == null) {
            newRating1++;
            System.out.println("proximity TRUE");
        } else {}

        rating1 = newRating1;
        System.out.println(rating1);



        if (rating1 > 0) {
            textView1.setText("에뮬레이터 탐지 1 결과: 에뮬레이터 입니다.");
        }
        else {
            textView1.setText("에뮬레이터 탐지 1 결과: 에뮬레이터가 아닙니다.");
```

```
        }


        if (OsBuildInfo()==true) {
            textView2.setText("에뮬레이터 탐지 2 결과: 에뮬레이터 입니다.");
        }
        else {
            textView2.setText("에뮬레이터 탐지 2 결과: 에뮬레이터가 아닙니다.");
        }
    }


//OS build 정보 확인
    public static boolean OsBuildInfo() {
        int newRating = 0;
        if(rating2 < 0) {
            if (Build.PRODUCT.contains("sdk") ||
                    Build.PRODUCT.contains("Andy") ||
                    Build.PRODUCT.contains("ttVM_Hdragon") ||
                    Build.PRODUCT.contains("google_sdk") ||
                    Build.PRODUCT.contains("Droid4X") ||
                    Build.PRODUCT.contains("nox") ||
                    Build.PRODUCT.contains("sdk_x86") ||
                    Build.PRODUCT.contains("sdk_google") ||
                    Build.PRODUCT.contains("vbox86p")) {
                newRating++;
            }

            if (Build.MANUFACTURER.equals("unknown") ||
                    Build.MANUFACTURER.equals("Genymotion") ||
                    Build.MANUFACTURER.contains("Andy") ||
                    Build.MANUFACTURER.contains("MIT") ||
                    Build.MANUFACTURER.contains("nox") ||
                    Build.MANUFACTURER.contains("TiantianVM")){
                newRating++;
            }

            if (Build.BRAND.equals("generic") ||
                    Build.BRAND.equals("generic_x86") ||
                    Build.BRAND.equals("TTVM") ||
                    Build.BRAND.contains("Andy")) {
                newRating++;
            }

            if (Build.DEVICE.contains("generic") ||
                    Build.DEVICE.contains("generic_x86") ||
                    Build.DEVICE.contains("generic_x86_arm") ||
                    Build.DEVICE.contains("Andy") ||
                    Build.DEVICE.contains("ttVM_Hdragon") ||
                    Build.DEVICE.contains("Droid4X") ||
                    Build.DEVICE.contains("nox") ||
                    Build.DEVICE.contains("aosp") ||
                    Build.DEVICE.contains("generic_x86_64") ||
                    Build.DEVICE.contains("vbox86p")) {
                newRating++;
```

```
            }

            if (Build.MODEL.equals("sdk") ||
                    Build.MODEL.equals("google_sdk") ||
                    Build.MODEL.contains("Droid4X") ||
                    Build.MODEL.contains("TiantianVM") ||
                    Build.MODEL.contains("Andy") ||
                    Build.MODEL.equals("Android SDK built for x86_64") ||
                    Build.MODEL.equals("Android SDK built for x86")) {
                newRating++;
            }

            if (Build.HARDWARE.equals("goldfish") ||
                    Build.HARDWARE.equals("vbox86") ||
                    Build.HARDWARE.contains("nox") ||
                    Build.HARDWARE.contains("ttVM_x86")) {
                newRating++;
            }

            if (Build.FINGERPRINT.contains("generic/sdk/generic") ||
                    Build.FINGERPRINT.contains("generic_x86/sdk_x86/generic_x86")
||
                    Build.FINGERPRINT.contains("Andy") ||
                    Build.FINGERPRINT.contains("ttVM_Hdragon") ||
                    Build.FINGERPRINT.contains("generic_x86_64") ||
                    Build.FINGERPRINT.contains("generic_x86_arm") ||
                    Build.FINGERPRINT.contains("generic/google_sdk/generic") ||
                    Build.FINGERPRINT.contains("vbox86p") ||
                    Build.FINGERPRINT.contains("generic/vbox86p/vbox86p")) {
                newRating++;
            }
            rating2 = newRating;
        }

        return rating2 > 0;
    }
}
```

3) 안드로이드 고유 식별자(unique identifier) 탐지

안드로이드 api는 모바일 기기의 전화기능, 하드웨어 통신, 로컬 저장소 등과 관련해서 다양한 인터페이스를 제공한다. 그중에서도 고유식별자(android_id) 사용하여 에뮬레이터를 탐지했다.

-안드로이드의 고유 id탐지기법(android_id)

androidID는 다음과 같은 방법으로 얻을 수 있다.

TelephonyManager()함수를 사용한 tm.getDeviceId();구현

```
package com.example.myapplication;

import android.annotation.SuppressLint;
import android.content.Context;
import android.os.Bundle;
import android.os.Build;
import android.provider.Settings;
import android.telephony.TelephonyManager;
import android.util.Log;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import java.util.UUID;
```

```
Settings.Secure.getString(context.contentResolver, Settings.Secure.ANDROID_ID)
```

```
final TelephonyManager tm= (TelephonyManager) getBaseContext().getSystemService(Context.TELEPHONY_SERVICE);
String uniqueID = UUID.randomUUID().toString();
final String tmDevice = "" + tm.getDeviceId();
final String tmSerial= " "+ tm.getSimSerialNumber();
final String android_id = Settings.Secure.getString(getContentResolver(), Settings.Secure.ANDROID_ID);//고유식별자 androidId

UUID deviceUuid = new UUID(android_id.hashCode(),  leastSigBits: ((long)tmDevice.hashCode() << 32 )| tmSerial.hashCode());
```

코드구현: MainActivity

adnroidmanifest.xml

activity_main.xml | AndroidManifest.xml | MainActivity.java

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.myapplication">

    <uses-permission android:name="READ_PREVILEGED_PHONE_STATE"/>
```

-READ_PRIVILEGED_PHONE_STATE 권한 사용하기

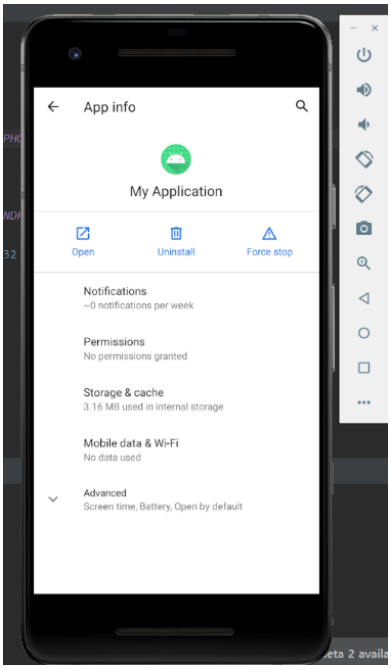
-에뮬레이터는 하드웨어가 존재하지 않으므로 기기의 식별번호를 읽어보는 TelephonyManager.getDeviceId()를 실행하면 결과값을 0으로 반환한다.
하지만 안드로이드10부터는 TelephonyManager에서 개인을 특정할 수 있는 정보를 가져올 수 없도록 변경되었다. 현재 안드로이드11버전이므로 사용이 불가하다.

TelephonyManager().getDeviced()

그리고 하드웨어의 시리얼 넘버도 사용이 불가능해졌다.

java.lang.RuntimeException: Unable to start activity ComponentInfo{dev.dnights.androiduniqueid/dev.dnights.a

실행시 위와 같은 securityexcpetion 발생.



-emulator 실행 시 앱이 실행되지 않는다.

-permission 을 바꾼결과 no permissions granted 라는 알림이 뜬다.

3. 과제를 수행함에 있어서 어려운 점 또는 건의사항

한예진)

이번 학기 안드로이드 과제를 통해 JAVA언어와 안드로이드 스튜디오를 처음 접하였습니다. 이제까지 주로 리눅스에서 C나 C++언어로만 코딩을 해왔기 때문에 익숙하지 않은 언어와 툴로 과제를 수행하려고 하니 많은 어려움이 있었던 것 같습니다. 그렇지만 과제를 수행해 나가면서 팀원들과 상의해 나가면서 잘 해결할 수 있었습니다.

박진아)

이번 학기에 보안을 처음 배우면서 색다른 실습을 할 수 있어 흥미로웠습니다. JAVA언어에 대한 기본적인 지식을 갖고 있고 안드로이드 스튜디오, LDplayer 등의 프로그램이 익숙했다면 이 과제를 좀 더 수월하게 해결할 수 있지 않았을까라는 아쉬움이 남습니다. 과제를 해결하기 쉽지 않았지만 함께 고민해주었던 팀원들 덕분에 잘 마무리할 수 있었습니다.

유하연)

이번 과제를 통해 안드로이드 스튜디오를 처음 사용해봤습니다. 처음 사용해보려고 하니 어려움도 있었고, 에뮬레이터 탐지라는 생소한 주제로 과제를 하게 되어서 생각하고 하기까지 많이 막막했습니다. 하지만 결론적으로 어느정도 알아가고 실제로 구현하면서 보안이라는 과목을 직접적으로 접했다는 생각이 들었습니다. 팀원들과 함께 상의하고 해결해 나가며 서로 도움을 주었던 과정에서 이 과제를 할 수 있었다는 생각이 듭니다.

양지원)

운영체제보안이라는 수업을 통해 보안이라는 분야를 처음 접해보았습니다. 마지막까지 비대면으로 팀플을 하게 되어 너무 아쉽습니다. 안드로이드 스튜디오를 이번에 처음 사용해보았는데, java를 너무 오랜만에 사용하게 되어 과제에 어려움이 있었습니다. 하지만 문제가 무엇인지 찾고 구현하면서 제 부족함을 찾게 되었습니다.