面向移动应用软件信息泄露的

模型检测研究

技术报告

——可规约代码生成算法

目录

一．算法介绍及生成规则

目标源码中的安全要素往往复杂难以操作，并且缺少初始化赋值，导致模型检测无法直接进行，因而需要首先生成可用于模型规约的中间代码，生成过程利用插装实现。

生成可规约系统需要对目标源码进行插装，首先根据诸如表 1 中的安全要素典型特征，扫描得到各安全要素的内容及位置，继而根据生成规则将源码转化为可规约代码，所定义的生成规则如下：

规则1 在 source 类要素之后生成新要素变量；而在 sink 类要素之前生成新要素变量。

规则2 如果 source 要素没有关联变量，则需生成一个新变量替代原要素。

规则3 对于每个 sink 要素的参数，生成一个关联变量。

规则4 对于 source 要素的关联变量，利用一个随机常量对其初始化。

规则5 对于 sink 要素的关联变量，利用参数原型对其赋值。

规则6 为每段待测源码生成一个驱动主函数，将 source 与 sink 要素分别按序调用。

二．规则示例

为了描述清晰，使用典型的示例代码对可规约系统的生成规则进行说明，每条规则都对应一个具体的示例代码。

（1）规则 1　在 source 类要素之后生成新要素变量；而在 sink 类要素之前生成新要素变量。

如图 1 所示该示例代码描述了一个简单的发送短信消息功能的 Activity，涉及的 source 要素是设备 ID,赋值给数组元素，在 source 要素之后添加新要素字符串变量 sourcedata，此处 source 要素有了关联变量 arrayData[1]，可以不定义新的变量；示例涉及的 sink 要素是 sendTextMessage()方法，在 sink 要素之前添加字符串变量 sinkdata。通过上述两步完成该示例代码的代码插装。

```
public class ArrayAccess1 extends Activity {
    public static String sinkdata;
    public static String sourcedata;
    public static String[] arrayData;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_array_access1);
        arrayData = new String[3];
        arrayData[0] = "element 1 is tainted:";
        arrayData[1] = ((TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE)).getDeviceId(); //source
        sourcedata =((TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE)).getDeviceId();
        //arrayData[2] is not tainted
        arrayData[2] = "neutral text";
        SmsManager sms = SmsManager.getDefault();
        //no data leak: 3rd argument of sendTextmessage() is not tainted
        sinkdata =arrayData[2];
        sms.sendTextMessage("+49 1234", null, arrayData[2], null, null);    //sink, no leak
    }
}
```

图 1　规则 1 示例

（2）规则 2　　如果 source 要素没有关联变量，则需生成一个新变量替代原要素。

如图 2 所示，该示例代码实现的是一个简单的发送短信消息功能的 Activity,涉及的 source 要素参数是设备 ID,涉及的 sink 要素是 sendTextMessage()方法，因为 source 要素对应的方法 getDeviceid()没有关联变量，需要生成一个新的变量 source 获得原要素包含的隐私信息，从而使检测过程顺利执行。

```
public class DirectLeak1 extends Activity{
      private static final String TELEPHONY_SERVICE = null;
      public String source;
      public String sink;
      @Override
    protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            TelephonyManager mgr =
                  (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);    //source
            SmsManager sms = SmsManager.getDefault();
            source = mgr.getDeviceId();
            sink = mgr.getDeviceId();
            sms.sendTextMessage("+49 1234", null, mgr.getDeviceId(), null, null);
      }
   public static void main(String[] args) {
         DirectLeak1 dire =new DirectLeak1();
         dire.onCreate(null);
   }
}
```

图 2　规则 2 示例

（3）规则 3　　对于每个 sink 要素的参数，生成一个关联变量。

如图 3 所示，该示例代码实现了该示例代码描述了一个简单的打印经纬度功能的 Activity，当用户的位置发生改变时，函数 onLocationChanged()将会被回调，当再次调用 onResume()时发生位置信息泄露。该示例涉及的 sink 要素是 Log.d()方法，对于每个 sink 要素的参数，生成一个关联变量，所以为参数生成两个新的要素变量 *sink1* 和 *sink2*，并插装于 sink 要素之前。而由于两个 source 要素已有对应的关联变量 *latitude* 和 *longtitude*，无需生成新的要素变量。

```
public class LocationLeak1 extends Activity {
    private String latitude = "";
    private String longtitude = "";
    public String sink1,sink2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_location_leak1);
        LocationManager locationManager =
                (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        LocationListener locationListener = new MyLocationListener();
        locationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER,5000,10,locationListener);
 }
protected void onResume (){
     super.onResume();
     sink1=latitude;
     Log.d("Latitude", "Latitude: " + latitude); //sink, leak
     sink2=longtitude;
     Log.d("Longtitude", "Longtitude: " + longtitude); //sink, leak
 }
 private class MyLocationListener implements LocationListener {
     public void onLocationChanged(Location loc) {    //source
         double lat = loc.getLatitude();
         double lon = loc.getLongitude();
         latitude =    Double.toString(lat);
         longtitude = Double.toString(lon);
         }
 }
```

图 3　规则 3 示例

（4）规则 4　　对于 source 要素的关联变量，利用一个随机常量对其初始化。

如图 4 所示，该示例代码实现了该示例代码描述了一个简单的打印经纬度功能的 Activity，当用户的位置发生改变时，函数 onLocationChanged()将会被回调，当再次调用 onResume()时发生位置信息泄露。该示例代码中涉及的 source 要素参数是包含位置的信息的 latitude 和 longtitude，为了模拟安卓程序的运行过程，需要利用一个随机常量对其初始化，简单的，可以利用一个转化函数 init()在初始化时将随机整型转化为随机字符型，完成对 source 要素关联变量的赋值。

```
public class LocationLeak1 extends Activity {
    private String latitude = "";
    private String longtitude = "";
    public String sink1,sink2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
      ****** }
protected void onResume (){
     super.onResume();
     sink1=latitude;
     Log.d("Latitude", "Latitude: " + latitude); //sink, leak
     sink2=longtitude;
     Log.d("Longtitude", "Longtitude: " + longtitude); //sink, leak
  }
  private class MyLocationListener implements LocationListener {
     public void onLocationChanged(Location loc) {    //source
        double lat = loc.getLatitude();
        double lon = loc.getLongitude();
        latitude =    Double.toString(lat);
        latitude = init(1,2);
        longtitude = Double.toString(lon);
        longtitude = init(1,2);
        }
  }
  Random random = new Random(42);
  public void init(domain,randN){
       String base = "abcdefghijklmnopqrstuvwxyz";
       StringBuffer ran = new StringBuffer();
       for (int i = 0; i <domain; i++) {
            int number = random.nextInt(randN);
            ran.append(base.charAt(number));
       }
  return ran.toString();
}
}
```

图 4　规则 4 示例

（5）规则 5　　对于 sink 要素的关联变量，利用参数原型对其赋值。

如图 5 所示，该示例代码实现的是简单的打印结果的 Activity，其中涉及的 sink 要素是 Log.i()，其关联变量值是通过 if 判断条件得到，与 source 要素之间存在一定的间接关联关系，需要利用参数原型对其进行赋值，以顺利完成检测过程。

```
public class ImplicitFlow2 extends Activity {
    static String sink,sink1,sink2;
    private boolean passwordCorrect = false;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_implicit_flow2);
    }
    public void checkPassword(View view){
        EditText mEdit = (EditText)findViewById(R.id.password);
        String userInputPassword=init(1,2); //source
        if(userInputPassword.equals(“b”))
            passwordCorrect = true;
        if(passwordCorrect)
        {   sink= "Password is correct";
            Log.i("INFO", "Password is correct");
        }//sink, leak
        else
            {
            sink ="Password is not correct";
            Log.i("INFO", "Password is not correct"); //sink, leak
            }
        }
  public static void main(String[] args){
      ImplicitFlow2 imp1 =new ImplicitFlow2();
      imp1.checkPassword(null);
      sink1= imp1.sink;
      ImplicitFlow2 imp2 =new ImplicitFlow2();
      imp2.checkPassword(null);
      sink2= imp2.sink;
  }
}
```

图 5　规则 5 示例

（6）规则 6　　为每段待测源码生成一个驱动主函数，将 source 与 sink 要素分别按序调用。

如图 6 所示，该示例代码实现的是一个简单的发送短信消息功能的 Activity,涉及的安全要素参数是设备 ID,涉及的 sink 要素是 sendTextMessage()方法，通过执行 onCreate()方法，造成用户设备信息隐私的泄漏，添加一个 main()函数驱动程序执行顺序，模拟安卓程序运行过程，如图 6 所示，在 main()函数中实现 onCreate()方法的调用执行。

```
public class DirectLeak1 extends Activity{
    private static final String TELEPHONY_SERVICE = null;
    public String source;
    public String sink;
    @Override
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TelephonyManager mgr =
            (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);    //source
        SmsManager sms = SmsManager.getDefault();
        source = mgr.getDeviceId();
        sink = mgr.getDeviceId();
        sms.sendTextMessage("+49 1234", null, mgr.getDeviceId(), null, null);
    }
  public static void main(String[] args) {
      DirectLeak1 dire =new DirectLeak1();
      dire.onCreate(null);
  }
}
```

图 6　规则 6 示例

三．算法总结

可规约代码生成算法.

输入：待测目标安全要素.

输出：可规约的系统代码.

```
1.  PROCEDURE RulableSystem(A)
      //A 为待测目标安全要素
2.    FOREACH(k in A)
3.      IF(source == k.type && NULL==(k.var& k.para)) THEN
4.        k.var.add(createNewVar());
          //为每个无额外变量的 source 生成新关联变量
5.        moveTo(k.pos+1);
6.        k.var_new=randomValue();
        //利用随机常量初始化 source 变量
7.      ELIF sink==k.type THEN
8.        FOREACH(p in k.para)
9.          k.var.add(createNewVar());
            //为每个 sink 参数生成新关联变量
10.         moveTo(k.pos-1);
11.         k.var_new=p.var;
            //将参数赋值给新关联变量
12.       ENDFOR
13.     ELSE
14.       output("invalid k");
          //无效的要素
15.     ENDIF
16.   ENDFOR
17.   IF(!createDrive()) THEN
18.     output("failed to generate driven");
        //无效的驱动
19.   ENDIF
20. ENDPROCEDURE
```

对于一个具备安全要素的目标系统，通过定义 2 利用模型检测验证该系统泄露与否， 其在形式上应符合下列条件：

1） 具备通用的 source 与 sink 要素变量，

2） 要素变量能够方便获取，

3） Source 要素变量值易于变更。

算法通过语句 4 对于每一个无关联变量的 source 要素生成一个新要素变量，并且利用语句 6 赋给一个随机初始值，该初始值通过函数随机变更，从而满足条件 3)；而语句 7-12，对于每一个 sink 要素根据其参数生成对应的关联要素变量。由于新生成的要素变量形式和名称被设计为通用已知的，因此易于获取，从而满足条件 1）和 2)。总体而言，算法的初始化插装使得程序中的安全要素变得易于修改和获取，为后续模型检测工作提供便利。