南京郵電大學
Nanjing University of Posts and Telecommunications

Android Malware Detection using Complex-Flows

汇报人：厉剑豪

1 INTRODUCTION

分析信息流是检测某些恶意应用程序家族的有效方法。当恶意应用程序收集敏感信息时，主要目的是泄露它，这不可避免地会在应用程序代码库中创建信息流。

identifying the existence of simple information flows – i.e. considering an information flow as just a (source,sink) pair present within the program. 源（source）通常是一个读取敏感数据的API调用，而汇（sink）是一个将从源读取的数据写到程序之外的API调用。

1 INTRODUCTION

区分恶意应用和良性应用的关键在于发现应用行为和对敏感数据的计算的区别。我们提出了一种新的信息流表示，称为复杂流(Complex-Flows)，用于更有效的恶意软件检测分析。

本文的目标是通过分析利用敏感数据的信息流中的应用行为来区分恶意软件和良性应用程序。首先，我们提取应用程序内部的复杂流，专注于与敏感数据操作相关的应用程序行为。其次，我们分析API序列以提取复杂流中存在的信息流中的应用程序行为特征。最后，我们通过机器学习技术利用这些特征进行分类。

2 MOTIVATION

为了说明现代良性应用和恶意应用如何通过信息流混淆恶意软件检测器，考虑一个良性应用和一个恶意应用，它们包含相同的（source，sink）信息流，如表1所示。

TABLE 1
Information Flows in Both Benign and Malicious Apps

Source	Sink
TelephonyManager:getDeviceId	HttpClient:execute
TelephonyManager:getSubscriberId	HttpClient:execute
LocationManager:getLastKnownLocation	Log:d
TelephonyManager:getCellLocation	Log:d

2 MOTIVATION

让我们检查一下我们的两个示例应用程序如何访问敏感数据，看看我们是否可以区分它们。

```
1  public static String getLmMobUID(Context context){
2    ...
3    TelephonyManager tm= (TelephonyManager)
4        context.getSystemService("phone");
5    if (isPermission(context,
         "android.permission.READ_PHONE_STATE"))
6      localStringBuffer.append(tm.getDeviceId());
7    ..
8  }
9  public static String getImsi(Context context){
10   TelephonyManager tm = (TelephonyManager)
11       context.getSystemService("phone");
12   param = tm.getSubscriberId();
13   ...
14 }
```

Fig. 2. Data Access Code Snippet in Benign App

```
1  private void execTask(){
2    ...
3    this.imei = localObject2.getDeviceId();
4    this.imsi = localObject2.getSubscriberId();
5    str2 = "http://" + Base64.encodebook(
6    "2maodb3ialke8mdeme3gkos9g1icaofm", 6, 3) +
7    "/mm.do?imei=" + this.imei;
8    localStr2 = str2 + "&imsi=" + this.imsi;
9    ...
10   paramString1 =
         ((HttpClient)localObject).execute(localStr2);
11   ...
12 }
```

Fig. 3. Data Access Code Snippet in Malware App

2 MOTIVATION

系统调用序列有效地捕获了程序中完成的计算。因此，我们检查在良性和恶意应用程序中沿着流发生的API调用序列，并对它们进行比较。

In particular, we use the flow TelephonyMan-ager:getSubscriberId -> HttpClient:execute as an example to illustrate the differences in benign and malicious apps.

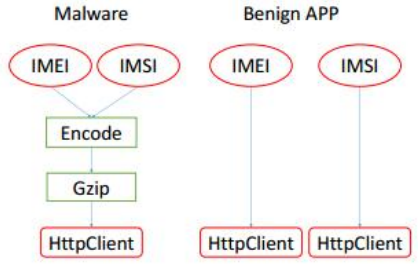

Fig. 1. App Behavior Comparison in Benign and Malware Apps

```
<Context: getSystemService(String)>
<TelephonyManager: getSubscriberId()>
<TelephonyManager: getDeviceId()>
<BasicNameValuePair: <init>(String,String)>
<URLEncodedUtils: format(List,String)>
<XmlServerConnector: byte[] zip(byte[])>
<HttpGet: void <init>(String)>
<DefaultHttpClient: void <init>()>
<HttpClient: getParams()>
<HttpParams: setParameter(String,Object)>
<HttpClient: getParams()>
<HttpParams: setParameter(String,Object)>
<HttpClient: execute(HttpUriRequest)>
```

Fig. 4. API Call Sequence in Malware App

```
<Context: getSystemService(String)>
<TelephonyManager: getSubscriberId()>
<PackageManager: checkPermission(String,String)>
<WifiManager: getConnectionInfo()>
<WifiInfo: getMacAddress()>
<TextUtils: isEmpty(CharSequence)>
<TextUtils: isEmpty(CharSequence)>
<TextUtils: isEmpty(CharSequence)>
<HttpGet: <init>(String)>
<BasicHttpParams: <init>()>
<HttpConnectionParams:
    setConnectionTimeout(HttpParams,int)>
<HttpConnectionParams:
    setSoTimeout(HttpParams,int)>
<DefaultHttpClient: <init>(HttpParams)>
<HttpClient: execute(HttpUriRequest)>
```

Fig. 5. API Call Sequence in Benign app

南京郵電大學 Nanjing University of Posts and Telecommunications

3 COMOLEX-FLOWS

3.1 Muti-Flows

我们把通过数据流或控制流在计算上相互关联的简单流称为多流。抽象地说，一个多流是由多个简单流组成的，这样多流中的任意两个简单流共享其计算的子集。

假设我们有以下简单流：

简单流1 (SF1)

- **SRC**：从传感器获取数据 (getSensorData)
- **S1**：处理数据 (processData)
- **S2**：转换数据格式 (convertDataFormat)
- **SNK**：发送数据到服务器 (sendToServer)

表示为：$SF1 = getSensorData \sim processData \sim convertDataFormat \sim sendToServer$

简单流2 (SF2)

- **SRC**：从传感器获取数据 (getSensorData)
- **S1**：处理数据 (processData)
- **S2**：记录数据到本地日志 (logDataLocally)
- **SNK**：发送通知 (sendNotification)

表示为：$SF2 = getSensorData \sim processData \sim logDataLocally \sim sendNotification$

简单流3 (SF3)

- **SRC**：从用户输入获取数据 (getUserInput)
- **S1**：处理数据 (processData)
- **S2**：存储数据到数据库 (storeInDatabase)
- **SNK**：发送确认信息 (sendConfirmation)

表示为：$SF3 = getUserInput \sim processData \sim storeInDatabase \sim sendConfirmation$

在上述简单流中，$SF1$ 和 $SF2$ 共享公共子序列：$\bar{S} = getSensorData \sim processData$。因此，这两个简单流可以构成一个多流。

- **公共子序列** $\bar{S}$：$getSensorData \sim processData$
- **多流** $F'(\bar{S})$：
  - 包含简单流 $SF1$：$getSensorData \sim processData \sim convertDataFormat \sim sendToServer$
  - 包含简单流 $SF2$：$getSensorData \sim processData \sim logDataLocally \sim sendNotification$

表示为：$F'(\bar{S}) = \{SF1, SF2\}$

Let SRC be the data source an app accesses. Let SNK be the sink point the data flows into. Let S_n be an intermediate statement in the program where the source data or data derived from the source data is used (i.e. a data flow).

Definition 1. A simple flow, $SRC \rightarrow SNK$, is composed of a sequence of statements $\bar{S}$, which includes SRC and SNK: $\bar{S} = SRC \rightsquigarrow S_1 \rightsquigarrow S_2 \ldots \rightsquigarrow S_{n-1} \rightsquigarrow S_n \rightsquigarrow SNK$. We say that a sequence $\bar{S}$ is a subsequence of a flow F, written as $\bar{S} \subset F$, if $\bar{S}$ is contained within F.

Definition 2. A Multi-Flow represents multiple simple flows that share common computation within a program. Let $\bar{F}$ be a set of all simple flows in a program. A Multi-Flow for a sequence $\bar{S}$, $\bar{F}'(\bar{S})$, is a set of simple flows in $\bar{F}$ that share $\bar{S}$ as a common subsequence. It is defined as:

$$\bar{F}'(\bar{S}) = \{F_i | F_i \in \bar{F} \text{ and } \bar{S} \subset F_i\}$$

3.2 Complex flows

Definition 3. Let $\bar{S}$ be a simple flow $SRC \rightsquigarrow S_1 \rightsquigarrow ... \rightsquigarrow S_n \rightsquigarrow SNK$, where SRC is a source, SNK is a sink, and S_i is a program statement. We define an API sequence of $\bar{S}$ as a filtered sequence over $\bar{S}$ that only contains API call statements. Note that both the source and sink are API calls by definition.

For a formal definition of an API sequence, we write $S \in \overline{API}$, if the statement S is a call to an API function. Then an API sequence of $\bar{S}$ is produced by filtering $\bar{S}$ recursively using the following three rules, which essentially removes all non-API calls from a simple flow (below, S is a single statement, and $\bar{S}'$ is a sequence of statements):

$$Rule\ 1 : filter(S \rightsquigarrow \bar{S}') = S \rightsquigarrow filter(\bar{S}') \text{ if } S \in \overline{API} \quad (1)$$

$$Rule\ 2 : filter(S \rightsquigarrow \bar{S}') = filter(\bar{S}') \text{ if } S \notin \overline{API} \quad (2)$$

$$Rule\ 3 : filter(\emptyset) = \emptyset \quad (3)$$

Definition 4. We define a Complex Flow CF in terms of a Multi-Flow, $\bar{F}(\bar{S})$ as the set of filtered sequences (i.e., API sequences - $\overline{AS}$) for each flow in the Multi-Flow:

$$CF = \{AS | AS = filter(F), F \in \bar{F}(\bar{S})\}.$$

Definition 5. An N-gram API set is a set of API sequences of size N derived from an API sequence. Formally, a set of N-grams over a filtered sequence is defined as follows, where $|\bar{S}'|$ denotes the size of the filtered sequence $\bar{S}'$:

$$N\text{-}gram(\bar{S}) = \{\bar{S}' | \bar{S}' \subseteq \bar{S}, |\bar{S}'| = n\}$$

Definition 6. We define all N-grams for a Complex Flow CF as a set of N-gram API sets, one derived from each filtered sequence AS contained in the Complex Flow:

$$\{NG | NG = N\text{-}gram(AS), AS \in CF)\}.$$

4 SYSTEM DESIGN

我们建立了一个自动恶意软件检测系统，通过分析第2节和第3节中描述的复杂流的N-gram表示，将应用程序分类为恶意或良性。这个分类系统被集成到我们的BlueSeal编译器中.

BlueSeal编译器是一个静态信息流分析引擎，最初是为了从Android应用程序中提取信息流而开发的。它还可以处理由UI事件和传感器事件触发的信息流。BlueSeal对上下文敏感，但对路径不敏感。它将Dalvik Executable (DEX)字节码作为应用程序的输入，而不需要应用程序的源代码。BlueSeal建立在Soot Java优化框架之上，并利用过程内和过程间数据流分析。此外，BlueSeal能够解决不同的Android特定结构和反射。

我们的实现扩展了BlueSeal，不仅能够发现简单的信息流，还能够发现复杂流。自动分类组件执行以下四个分析阶段来生成特征并将应用程序分类为恶意或良性: (1) Multi-Flow discovery, (2) API call sequence extraction, (3) N-gram feature generation, and (4)Classification.

4 SYSTEM DESIGN

4.1 Multi-Flow Discovery

多流检测算法的目标是：(1) 为应用程序创建一个包含完整信息流路径的全局图；(2) 检测代表多流的各个信息流路径之间的交集。

多流检测算法通过输入BlueSeal本身检测到的各个信息流路径来工作，这些路径跟踪具有单一源和单一汇的简单流。

为了生成multiflow，我们对BlueSeal进行如下扩展:

1. 每当遇到包含敏感API调用(访问设备的敏感数据)的语句时，我们都会将该调用添加为全局图中的节点。这被认为是数据流路径的起点。

2. 接下来，我们检查每个程序语句，看看是否存在从当前语句到初始检测到的语句的数据流。如果存在，我们在全局数据流图中构建一个中间源节点，并从初始语句的节点添加一条边到这个中间源节点。这个步骤是递归的，如果从另一个程序语句到中间源节点存在数据流，我们就如上所述创建一个新的中间源节点。这些中间节点是关键的，因为它们将单个流连接在一起，创建多流。

4.1 Multi-Flow Discovery

3. 当我们找到一个汇聚点时，数据流的路径结束。这三种类型的点(即，源、中间和接收)能够捕获简单信息流的整个数据流路径，同时输出包含所有可能相互连接的数据流路径的全局图。

4. 通过迭代这个全局图来检测多流，找到简单的数据流和多流。

5.我们为所有multi - flow提取API调用序列。在此过程中，我们分析每个Multi-Flow中的控制流路径以提取API调用序列。

4.2 Complex-Flow Extraction with API Sequences

分析复杂流要求我们考虑包含分支和循环的控制路径，因为它们会生成不同的代码路径。

```
private void PhoneInfo(){
  imei = Object2.getDeviceId();
  mobile = Object2.getLine1Number();
  imsi = Object2.getSubscriberId();
  iccid = lObject2.getSimSerialNumber();
  url = "http://"+str1+".xml?sim="+imei+
      "&tel="+mobile+"&imsi="+imsi+"&iccid="+iccid;
  Object2 = getStringByURL(Object2);
  if ((Object2 != null) && (!"".equals(Object2))){
    sendSMS(this.destMobile, "imei:" + this.imei);
  }else{
    writeRecordLog(url);
  }
}
private void sendSMS(String str1, String str2){
  SmsManager.getDefault().sendTextMessage(str1,
      null, str2,null,null,0);
}
private void writeRecordLog(String param){
  Log.i("phoneinfo", param);
}
public String getStringByURL(String paramString){
    HttpURLConnection conn =
        (HttpURLConnection)new
        URL(paramString).openConnection();
    conn.setDoInput(true);
    conn.connect();
    return null;
}
```

Fig. 6. API Call Sequence Extraction Example

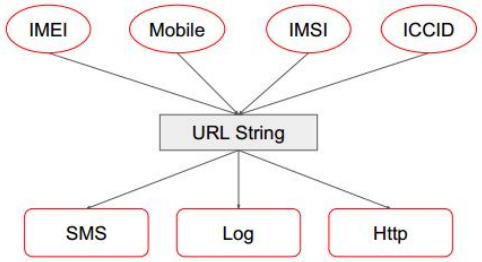

Fig. 7. Data Flow Structure of Example Code Snippet

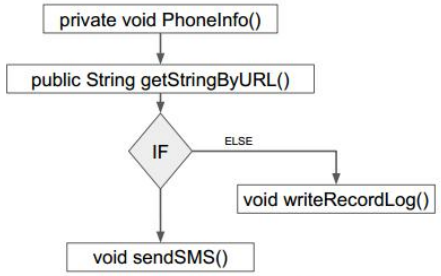

Fig. 8. Control Flow Structure of Example Code Snippet

TABLE 2
Final API Call Sequence Output

Sequence 0	TelephonyManager:getDeviceId() TelephonyManager:getLine1Number() TelephonyManager:getSubscriberId() TelephonyManager:getSimSerialNumber() java.net.URL:openConnection() HttpURLConnection:setDoInput() HttpURLConnection:connect() SmsManager:getDefault() SmsManager:sendTextMessage()
Sequence 1	TelephonyManager:getDeviceId() TelephonyManager:getLine1Number() TelephonyManager:getSubscriberId() TelephonyManager:getSimSerialNumber() java.net.URL:openConnection() HttpURLConnection:setDoInput() HttpURLConnection:connect() Log: int i()

4.3 N-gram Feature Generation

我们将每个gram视为给定API调用序列的一个子序列。序列N-gram是重叠的子字符串，以滑动窗口的方式收集，其中固定大小的窗口每次滑动一个API调用。序列N-gram不仅捕捉长度为n的API调用子序列的统计信息，还隐式地表示较长调用序列的频率。

TABLE 3
Example of API Sequence and its 2-grams

API Sequence	TelephonyManager:getDeviceId() TelephonyManager:getLine1Number() TelephonyManager:getSubscriberId()
2-grams	TelephonyManager:getDeviceId() TelephonyManager:getLine1Number()
	TelephonyManager:getLine1Number() TelephonyManager:getSubscriberId()

4.4 Classification

我们恶意软件分类工具的最后一步是在从复杂流表示中获得的N-gram特征之上利用机器学习技术。此分类的目标是识别恶意应用和良性应用之间显著的不同行为。在这个过程中，我们的系统使用良性应用和恶意应用进行训练。我们的恶意软件数据集包含来自不同家族的恶意应用，我们的良性数据集是多样化的，涵盖了流行应用以及来自不同类别的应用。我们的系统能够学习不同类型的恶意软件如何利用信息流以及包含哪些行为。此外，我们的系统还能学习当良性应用利用这些信息流时会出现哪些行为。我们的系统被训练来学习不同信息流上的行为模式，并将其作为分类的特征。通过这种方式，我们的系统可以判断一个应用是以良性还是可疑的方式利用信息流。

对于一个新应用，我们的分类系统会比较其行为模式（由应用的复杂流表示生成的特征向量）并决定它更类似于训练集中良性应用还是恶意应用。总体来说，我们的分类系统会根据应用在信息流上的行为来检测其信息流是否是恶意的。

5 EVALUATION

Play_2014: Apps collected from Google Play in Jan, 2014. The total number of apps is 800.

MalGenome: Malware apps collected from MalGenome project. The total number of apps is 800.

TP True positive rate—the rate of benign apps recognized correctly as benign.
TN True negative rate—the rate of malware recognized correctly as malicious.
FP False positive rate—the rate of malware recognized incorrectly as benign.
FN False negative rate—the rate of benign apps recognized incorrectly as malicious.

TABLE 4
Gram Based Classification Results of Play_2014 and MalGenome Apps

gram size	TP	TN	FP	FN	accuracy
1	0.975	0.852	0.148	0.025	0.913
2	0.950	0.699	0.301	0.050	0.822
3	0.980	0.688	0.312	0.020	0.831
4	0.549	0.948	0.052	0.451	0.753
5	0.485	0.952	0.048	0.515	0.725
1,2	0.81	0.92	0.08	0.19	0.868
1,2,3	0.886	0.84	0.16	0.114	0.863
1,2,3,4	0.759	0.889	0.111	0.241	0.825
1,2,3,4,5	0.696	0.938	0.062	0.304	0.819

THANKS!

感谢您的观看与聆听

汇报人：厉剑豪