# 基于信息流关系特征的恶意软件检测方法

杨保山, 杨　智, 张红旗, 韩　冰, 陈性元, 孙　磊

信息工程大学 郑州 中国 450000

**摘要**　移动互联网的发展使得移动设备已经影响到了我们生活的方方面面, 这导致了个人信息在移动设备的集中。由于 Android 系统的开放性和其自身安全机制的不完善, 软件非法窃取隐私信息已是普遍存在的问题。信息流分析技术以保证信息的安全性为目标, 通过分析应用程序中数据传播的合法性来检测隐私数据是否遭到泄露, 当前利用信息流分析来检测恶意软件的方法已成为研究热点。但是 Android 应用程序的功能复杂性在不断增加, 同时伴随着代码复杂性的增加, 使得良性应用和恶意应用在敏感信息流行为模式上的相似度越来越高, 粗粒度的信息流特征描述很难对良性应用和恶意应用做出区分, 这会在很大程度上影响检测的准确率。为此本文提出了一种新的基于信息流关系特征的恶意软件检测方法, 该方法在提取应用敏感信息流的基础之上进一步挖掘了信息流之间的关系特征, 我们对敏感 API 调用序列之间的关系进行了详细的形式化描述, 并通过动态规划方法分析得到敏感 API 调用序列之间的关系特征和它们的连续公共子序列, 我们将关系特征表述为五元组, 对连续公共子序列中的 API 进行分类后表述为六元组, 最后将这两方面的特征融合后输入到卷积神经网络(Convolutional Neural Networks, CNN)中来实现恶意软件的检测。实验结果表明, 我们在 MalGenome 和 AndroZoo 数据集下分别达到了 98.5%和 97.6%的准确率, 可以看出更加细粒度的敏感信息流之间关系特征表述对于良性应用和恶意应用的区分起着重要的作用。

**关键词**　Android 恶意软件检测; 关系特征; 信息流; 特征融合

**中图法分类号**　TP309　DOI 号 10.19363/J.cnki.cn10-1380/tn.2024.11.11

# Malware Detection Method Based on Information Flows Relationship Features

YANG Baoshan, YANG Zhi, ZHANG Hongqi, HAN Bing, CHEN Xingyuan, SUN Lei

Information Engineering University, Zhengzhou 450000, China

**Abstract**　The development of the mobile internet has made mobile devices affect all aspects of our lives, which has led to the concentration of personal information in mobile devices. Due to the openness and the imperfect security mechanism of the Android system, the illegal theft of private information by software has become a common problem. Information flow analysis technology aims at ensuring the security of information. It detects whether private data has been leaked by analyzing the legitimacy of data transmission in applications. The method of using information flow analysis technology to detect malware has become a current research hotspot. However, the functional complexity of Android applications is increasing, along with the increase of code complexity, the similarity between benign and malware in the behavior patterns of sensitive information flows is getting higher and higher. It is difficult to distinguish between benign and malware by coarse-grained information flow feature description, which will greatly affect the accuracy of detection. In this paper, we propose a new malware detection method based on the relationship features of information flows. This method further excavates the relationship features of information flows based on the extraction of application sensitive information flows. We have made a detailed formal description of the relationship between sensitive API call sequences, and obtained the relationship features and continuous common subsequences between sensitive API call sequences through dynamic programming analysis. We have expressed the relationship features as five-tuples, and the API in the continuous common subsequence is classified as six-tuples. Finally, the features of these two aspects are fused and input into the convolutional neural networks (CNN) to realize malware detection. The experimental results show that we have achieved 98.5% and 97.6% accuracy respectively in MalGenome and AndroZoo datasets. It can be seen that the more fine-grained expression of the relationship between sensitive information flows plays an important role in distinguishing between benign and malware.

**Key words**　Android malware detection; relationship features; information flows; feature integration

**通讯作者：**杨　智, 博士生导师, 教授, Email: zynoah@163.com。

本课题得到国家自然科学基金(No. 62176265, No. 61972040)资助。

收稿日期: 2022-11-02; 修改日期: 2023-02-10; 定稿日期: 2024-09-05

## 1 引言

近年来, 随着移动互联网的发展, 用户可以通过移动设备获取互联网丰富的资源和完善的服务, 智能手机得到了广泛的普及和应用, 成为人类生活工作中不可或缺的重要工具。目前移动设备主要采用来自 Google 公司的 Android 操作系统, 据 statcounter 统计, 截至到 2022 年 9 月, Android 占据全球移动操作系统市场份额的 71.62%[1]。

恶意应用程序以系统中用户的隐私数据为攻击目标, 对用户数据的保密性和完整性产生威胁。现如今, 智能设备配备了相机、麦克风、陀螺仪、GPS 等复杂的传感器, 移动应用已经可以满足我们办公、社交、娱乐、出行、购物和支付等需求, 但与此同时也产生了包含个人隐私信息的海量数据[2], 一些恶意的应用程序会在用户无感知的情况下来窃取用户的隐私信息。由于 Android 系统本身的安全机制不够完善[3], 用户可以在自己的移动设备上自由下载安装来自第三方市场甚至任意渠道的软件[4], 而且由于 Android 应用开发的便捷性, 一些个人开发者在个人网站上便能公开提供应用的下载, 这些软件缺乏监管和审核, 其安全性无法得到保证, 一定程度上也导致了恶意软件的出现和传播[5]。近年来, 如何准确地检测 Android 恶意软件来保护用户隐私信息已成为信息安全领域的研究热点。

信息流分析技术[6]以保证信息的安全性为目标, 通过分析程序中数据传播的合法性来防止隐私数据在传播的过程中遭到泄露或者篡改, 在恶意软件检测和隐私保护领域有着广泛的应用。它不需要运行程序和修改源代码, 一般将应用的字节码文件反编译成中间代码, 然后对中间代码进行分析并建模, 在标记污点数据后通过一定的规则来跟踪其是否从污点源(Source)传播到污点汇聚点(Sink), 从而来判断该数据是否泄露或者篡改, 从而有效的保证了系统中数据的保密性和完整性。但是由于现如今应用程序的复杂性, 良性应用程序和恶意应用程序可能收集相同的信息, 例如应用程序会通过收集手机通讯录信息来给用户推荐可能认识的人, 获取位置信息来给用户推荐附近的景区、酒店和餐厅等, 但是这类应用程序是本着为用户提供便利的原则, 与恶意应用程序有着本质上的不同, 需要将其区分开来, 然而基于粗粒度的信息流特征描述来检测恶意软件的方法则可能会产生误报。

为了进一步挖掘并细化信息流之间的关系特征, 刻画出良性应用程序和恶意应用程序不同的行为模式和信息流特征, 提高检测的准确率, 我们提出了一种新的基于信息流之间关系特征的恶意软件检测方法。该方法首先通过静态分析提取出 Android 应用程序的敏感 API 调用序列, 其次通过动态规划的方法分析得到 API 序列之间的关系特征和连续公共子序列, 在对连续公共子序列转换成特征向量后, 我们将这两个特征融合到一个矩阵中作为代表该应用程序的特征向量。最后, 我们将固定大小的矩阵输入给 CNN 模型。

本文的主要贡献如下:

(1) 提出了一种新的基于信息流之间关系特征的恶意软件检测方法, 本方法给出了敏感 API 调用序列之间关系的详细的形式化描述, 并使用动态规划来分析敏感 API 序列之间的关系, 同时提取出连续公共子序列。

(2) 提出了将 API 之间的关系特征表示为一个五元组, 同时为了解决连续公共子序列长度不一致的问题, 我们重新定义了六种 API 类别, 将连续公共子序列中的每个 API 进行分类, 进而将其表示到一个六元组中, 最后将它们融合到一个矩阵向量中。

(3) 为了验证方法的准确率和稳定性, 我们将得到的融合特征矩阵输入 CNN 模型进行训练, 并在 MalGenome 和 AndroZoo 数据集下进行实验, 分别达到了 98.5%和 97.6%的准确率。

本文的其余部分组织如下: 第二节介绍相关研究工作, 第三节提出了我们的 Android 恶意软件检测方法。第四节我们进行了实验并评估了实验结果。最后, 第五节总结了本文。

## 2 相关工作

### 2.1 Android 信息流分析

目前针对 Android 的信息流分析工具大多是致力于提取出更全面、更精确的敏感信息流。在静态信息流分析中, Soot[7]是常用的 Java 代码优化框架, 在简化语句方面有很大作用, 它将一些复杂的细节进行了屏蔽, 从而有利于在此基础上进行分析。FlowDroid[8]是静态信息流分析工具的代表之一, 它基于 Soot 框架和 DEXPLER[9], 通过抽象一个 dummyMain 作为程序入口, 模拟 Android 应用程序的生命周期, 利用 SUSI[10]在 Android API 调用序列中自动识别 Sources 和 Sinks, 从而发现是否有污点传播路径, 但仅仅在单个组件中进行信息流分析。IccTA[11]在 FlowDroid 的基础上利用 Epicc[12]和 IC3[13]工具提取组件间通信(Inter-Component Communication, ICC)链接, 然后对 ICC 进行分析, 扩展了组件间

通信的信息流分析, 并对生命周期和回调方法进行建模, 构建了完整的组件内、组件间和应用程序间模型。Amandroid[14]构建了应用程序组件的高精度的过程间控制流图, 相比IccTA, Amandroid实现了远程过程调用(Remote Procedure Call, RPC)分析, 基于字符串分析以推断 RPC 参数, 并基于流敏感匹配算法找到ICC源和ICC目标之间的对应关系。MaMaDroid[15]则首先提取出应用程序的 API 调用序列, 并将每个API调用抽象为包、家族或者类其中的一种, 然后构建马尔可夫链来建模每个应用程序的行为并构建用于分类的特征向量。DroidSafe[16]基于 Android 开放源代码项目(Android Open Source Project, AOSP)构建出了较为完整的 Android 执行模型——Android 设备实现(Android Device Implementation, ADI), 并在此基础上实现了对象敏感分析和流敏感分析, 通过准确的分析存根来提高 ICC 建模的精度。另外也有通过动态分析来提取信息流的方法。TaintDroid[17]是一个知名的针对 Android 应用程序的动态信息流分析工具, 它定制了 Dalvik VM 虚拟机来实现污点标签的存储和污点传播的跟踪, 通过集成变量、消息、方法和文件四个污点传播粒度, 一定程度上提高了检测的效率, 但是最高仅支持 Android 4.3 系统。从 Android 5.0 之后, Google 改变了策略, 开始引入 Android Run Time(ART)来取代 Dalvik VM, TaintART[18]则解决了这个问题, 它是一个多级的动态信息流分析技术, 可以通过插入跟踪日志来跟踪数据, 它采用了一种多级污点标记方法来最小化污点标签的存储, 这样标记就可以存储在处理器寄存器中以便快速访问, 并且通过定制 ART 编译器来优化系统的性能和运行速度, 而且因为编译器和调用约定在不同版本之间是稳定的, 所以 TaintART 是持久的, 并且可以很容易地更新以支持未来的版本。AppsPlayground[19]则是在 Android SDK 提供的 Android 模拟器的基础上进行改进, 使其支持真实移动设备的一些功能。该方法集成了动态污点跟踪、API 监控和内核级系统调用监控等多种检测技术, 根据系统调用和内核级数据结构将已知的 Android 漏洞描述为可用于动态分析的签名。

对单条的信息流分析已无法适应 Android 应用程序日益复杂化的情况, 有学者通过分析良性应用和恶意应用的信息流特征的区别来提高检测的准确率。Yang 等人[20]将具有相同部分的调用序列通过约简和合并得到一条最长的敏感 API 调用序列, 然后挖掘加权支持度的频繁序列模式从而更有效地表示应用程序的行为语义, 最后构建机器学习模型实现恶意软件检测。Avdiienko[21]等人在对比良性应用和恶意应用的敏感信息流时发现, 恶意应用的信息流流向总是集中在少量几个典型的 Sinks 中, 这些 Sinks 在良性应用和恶意应用之间有很大的区别, 并在此基础上提出了利用良性应用中正常的信息流集合来分析检测其他应用中是否存在异常信息流的方法 MUDFLOW。MUDFLOW 首先使用 FlowDroid 将应用程序的所有敏感信息流提取出来, 仅保留信息流中的 Sources 和 Sinks, 并且对 Sources 和 Sinks 进行分类聚合, 然后根据恶意应用与良性应用的信息流差异来对恶意软件进行检测, 实验结果表明, 该方法对恶意软件检测的准确率达到86.4%, 对泄露敏感数据的恶意软件检测的准确率达到 90.1%。Shen 等人[22]提出了一种基于信息流分析的 Android 恶意软件检测方法, 通过分析应用程序中的敏感信息流, 将恶意应用与良性应用区分开来。该方法提出了复杂流(Complex-Flows)的概念来寻找良性应用程序和恶意应用程序信息流之间的差异, 在提取应用的敏感信息流时, 判断信息流之间是否存在交叉共享的路径, 这些存在共享路径的敏感信息流将会被提取出来, 然后通过 N-gram 滑动窗口的方式对敏感信息流进行处理。在生成特征向量阶段, 该方法将数据集中所有样本的 gram 特征取并集, 形成一个全局的特征向量空间, 基于这个特征空间, 来为每一个应用生成特征向量。当应用的某个 gram 特征对应在全局特征空间的第 $n$ 个位置时, 就将该应用的特征向量第 $n$ 位置的值置为 $m$, 其中 $m$ 为这个 gram 特征在当前应用中出现的次数, 其余没有在全局特征空间中存在对应关系的便将其对应位置设置为 0。但是这样造成了信息流之间关系信息的丢失, 特征向量仅表示某些敏感信息流之间是存在共享路径, 而不能确定是哪两条路径之间存在共享关系, 因此需要对敏感信息流之间的关系特征做进一步的研究。

## 2.2 CNN 在 Android 恶意软件检测的应用

CNN 是广泛使用的机器学习框架, Hubel 等人[23]在 1968 年通过研究猫和猴子的视觉皮层细胞提出了一种视觉神经系统的层次模型, 被看作是 CNN 发展的开端。Fukushima 等人[24]基于 Hubel 等人提出的层次模型的基础上提出了一种被称为 Neocognitron 的视觉模式识别机制的神经网络模型, 该模型由多个模块化结构的级联连接组成, 通过无监督学习的方式, 获得了一种基于几何形状相似性的模式识别能力, 而且不受其位置或形状改变的影响。LeCun 等人[25]在 1990 年提出了将反向传播网络(Back Propa-

gation, BP)应用在书写体的数字识别上面, 该网络无需任何预处理, 可以直接将图像作为输入, 在对美国邮政局提供的邮政编码进行识别时, 错误率仅为 1%, 拒绝率约为 9%, 证明了 BP 网络处理大量低级别信息的能力。Lecun 等人[26]在 1998 年进一步提出用BP算法训练多层的神经网络, 并首次提出了CNN的概念。Krizhevsky 等人[27]提出了一个更加庞大、深层的 CNN 模型, 它包含 8 个学习层, 其中 5 个卷积层和 3 个全连接层, 同时为了减少全连接层中的过拟合, 采用了一种称为 Dropout 的正则化方法, 把识别错误率进一步降低。目前 CNN 已广泛应用于图像识别[28]、视频处理[29]和自然语言处理(Natural Language Processing, NLP)[30]等领域。

受 CNN 在 NLP 领域取得突破的启发, 研究学者开始将 CNN 应用于 Android 恶意软件的检测和分类。McLaughlin 等人[31]通过反汇编 Android 应用程序包(Android Application Package, APK)来提取操作码序列以表示应用程序的特征, 它将操作码指令序列编码为 one-hot 向量, 并提供给操作码嵌入层, 为 CNN 提供操作码的语义信息, 然后利用卷积层提取抽象特征, 使用最大池(max-pool)将任意长度的操作码序列表示为固定长度的特征向量, 最后将特征向量传递给带有全连接隐藏层和全连接输出层的多层感知器(Multi-Layer Perceptron, MLP)。该方法与基于 N-gram 的恶意软件检测方法相比, 实现了更高的检测效率。Ye 等人[32]首先从 Android 应用程序中将 API 调用序列提取出来, 用于合成更高层次的语义特征, 以更全面地描述 Android 应用程序。具体来说, 通过构造异构信息网络(Heterogeneous Information Network, HIN)结构来模拟不同类型的实体, 包括 IMEI、API、从属关系和签名等, 并提出了一种基于元路径的方法来根据语义信息描述应用程序之间的关系。在内部, 一个 Android 应用程序对应于一个样本内节点或者一个样本外节点, 该节点位于语义信息的 HIN 中, 在获得样本内节点嵌入后学习样本外节点的表示, 随后将获取到的 HIN 提供给由 CNN 和 Inception 组成的深度神经网络中, 实现了 Android 恶意软件的检测和分类。MalDozer[33]使用 CNN 自动检测 Android 恶意软件, 并通过 API 调用序列的输入来检测恶意软件。API 调用序列是从 classes.dex 文件中提取出来的序列, 然后提取的 API 调用序列通过用标识符来替换每个 API 调用, 这样每个 API 调用序列就变成了一个数字序列, 然后使用 word2vector[34]和 Glove[35]技术将序列中的标识符嵌入到语义相关的向量中。在 CNN 模型设计中, MalDozer 方法的第一层是使用激活函数为 ReLU 的卷积层, 随后是一个最大池化层和全连接层; 然后连接输出层, 并利用损失函数防止过拟合, 提高检测性能。

## 3 提出的方法

我们提出的方法框架如图 1 所示, 该方法主要包括敏感 API 调用序列预处理、特征向量和机器学习等阶段。

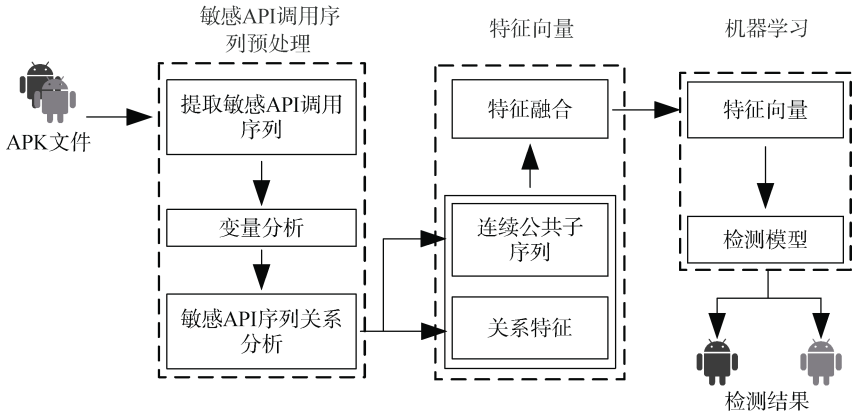


**图 1 方法框架图**

**Figure 1 Methodology framework**

### 3.1 敏感 API 调用序列预处理

#### 3.1.1 提取敏感 API 调用序列

本文使用 FlowDroid 来提取 APP 中存在的敏感 API 调用序列。Android 应用程序需要通过调用 Android API 来调用系统功能和获取设备信息。我们以 SendSMS.apk 为例, 如图 2 所示, Button1Listener 类中的 Onclick()方法通过调用 android.telephony.TelephonyManager 中的 getDeviceId()方法来获取设备的国际移动设备标识(International Mobile Subscriber Identity, IMEI)信息并赋值给字符串 *str*, 然后存放

```
public class Button1Listener implements View.OnClickListener {
    private final MainActivity act;

    public Button1Listener(MainActivity paramMainActivity) {
      this.act = paramMainActivity;
    }

    public void onClick(View paramView) {
      Intent intent = new Intent("android.intent.action.SEND");
      intent.setType("text/plain");
      String str = ((TelephonyManager)this.act.getSystemService("phone")).getDeviceId();
      intent.putExtra("secret", str);
      Log.i("SendSMS: ", "Sending implicit Intent with MIME data type text/plain: DeviceId " + str);
      this.act.startActivityForResult(intent, 0);
    }
}
```

**图 2　SendSMS.apk 中 Button1Listener 类代码片段**

**Figure 2　Code snippet of Button1Listener class in SendSMS.apk**

键为“secret”的键值对中，最后调用 android.util.Log 中的 i()方法写到日志文件中。因此该类中存在一条 android.telephony.TelephonyManager:getDeviceId() → android.util.Log:i()的 Source 点到 Sink 点敏感信息传播路径。

图 3 MainActivity 类中 onActivityResult()方法通过调用 sendSMSMessage()方法，利用 API 调用 android.telephony.SmsManager 将“secret”对应的值发送出去。因此存在一条 android.telephony.TelephonyManager:getDeviceId()→android.telephony.SmsManager:sendTextMessage()的 Source 点到 Sink 点敏感信息传播路径。

```
public class MainActivity extends Activity {
  protected void onActivityResult(int paramInt1, int paramInt2, Intent paramIntent) {
    if (paramInt2 == 0 && paramInt1 == 0 && paramIntent != null) {
      if (paramIntent.hasExtra("secret")) {
        if (paramIntent.getExtras().getString("secret") != null) {
          Log.v("In SendSMS: ", "Data recieved");
          sendSMSMessage(paramIntent.getExtras().getString("secret"));
          return;
        }
      } else {
        return;
      }
      Log.i("In SendSMS: ", "Data recieved");
      return;
    }
    Log.i("In SendSMS: ", "No data recieved");
  }

  protected void onCreate(Bundle paramBundle) {
    super.onCreate(paramBundle);
    setContentView(2130903040);
    ((Button)findViewById(2131230720)).setOnClickListener(new Button1Listener(this));
  }

  public boolean onCreateOptionsMenu(Menu paramMenu) {
    getMenuInflater().inflate(2131165184, paramMenu);
    return true;
  }

  @SuppressLint({"UnlocalizedSms"})
  protected void sendSMSMessage(String paramString) {
    try {
      SmsManager.getDefault().sendTextMessage("1234567890", null, paramString, null, null);
      Toast.makeText(getApplicationContext(), "SMS sent!", 1).show();
      return;
    } catch (Exception exception) {
      Toast.makeText(getApplicationContext(), "Couldn't send SMS!", 1).show();
      exception.printStackTrace();
      return;
    }
  }
}
```

**图 3　SendSMS.apk 中 MainActivity 类代码片段**

**Figure 3　Code snippet of MainActivity class in SendSMS.apk**

### 3.1.2　变量分析

程序在调用 API 获取到敏感信息之后，往往通过赋值给变量对其进行传输，这些变量可以是全局变量，也可以是局部变量，当敏感信息到达 Sink 点，敏感信息就通过变量之间的传递赋值传输了出去。我们仍以 SendSMS.apk 为例，并将核心代码提取出

来, 如图 4 所示。Onclick()方法通过调用 getDeviceId()获取到敏感信息之后赋值给变量 *str*, 然后将信息传给变量 *intent* 中键为“secret”的键值对中, 随后又传递给形式变量 *paramIntent*, onActivityResult()方法获取到 *intent* 中的信息后, 传递给形式变量 *paramString*, 最终在 sendSMSMessage()中将敏感信息发送出去。因此在 API 调用序列 android.telephony.TelephonyManager:getDeviceId() → android.telephony.SmsManager:sendTextMessage()中存在一条敏感信息在变量中传输的序列 *str* → *intent* → *paramIntent* → *paramString*。如果在传输过程中“secret”对应的值被赋为 null, 或者被赋值了其他无关的信息, 那该条 API 调用序列虽调用了敏感 API, 但没有传出任何有意义的信息, 如果再对该 API 调用序列分析就失去了意义。因此我们通过分析敏感信息在变量中的传输来筛选真正传出敏感信息的 API 调用序列, 以提高分析的准确性和减少待分析的数据量。

```
public class Button1Listener implements View.OnClickListener {

    public void onClick(View paramView) {

      String str = ((TelephonyManager)this.act.getSystemService("phone")).getDeviceId();
      intent.putExtra("secret", str);
    }
}
```

```
public class MainActivity extends Activity {

    protected void onActivityResult(Intent paramIntent) {

        sendSMSMessage(paramIntent.getExtras().getString("secret"));
    }

    protected void sendSMSMessage(String paramString) {

        SmsManager.getDefault().sendTextMessage("1234567890", null, paramString, null, null);
    }
}
```

**图 4　Button1Listener 类中的敏感信息传输**

**Figure 4　Sensitive information transmission in Button1Listener class**

### 3.1.3　敏感 API 调用序列之间关系分析

良性应用和恶意应用在敏感信息流的行为特征上存在一定的差异[22], 我们在此基础上进一步研究分析发现, 敏感信息流之间的关系对于恶意软件检测起着重要的作用。

图 5 所示的是良性应用和恶意应用获取 IMEI 的敏感行为。良性应用在获取到设备的 IMEI 后会存储到文件中, 同时展示到用户界面上, 以后获取 IMEI 便可以直接读取文件中的信息, 但是始终不会通过网络将信息发送出去, 而恶意应用将 IMEI 展示到用户界面的同时, 便通过网络将 IMEI 信息发送了出去, 虽然这两种行为的敏感信息流之间均存在着共享的路径, 但是它们的行为有着本质的区别。Complex-Flows 方法仅仅标识出了这些敏感信息流之间存在着共享的路径, 但是无法标识出哪些流之间存在共享, 那么它在特征的表述上面便无法对这两种情况做出区分。

另外, 如图 6 所示, 良性应用和恶意应用均会在获取用户的隐私信息后通过同一个网络接口发送出去。良性应用在收集 IMEI 信息的同时, 最多会伴随着对国际移动用户识别码(International Mobile Subscriber Identity, IMSI)的收集, 但是恶意应用会同时收集设备的 IMEI、IMSI 和位置等多个信息, 由于 Complex-Flows 方法丢失了敏感信息流之间的关系信息, 因此它们在特征表述上是一致的, 无法区分出哪些流之间存在汇聚关系, 所以便无法区分出这种情况。针对这些问题, 我们对敏感信息流之间的关系特征做了进一步的研究。首先, 我们给出了 API 调用序列之间关系的描述。

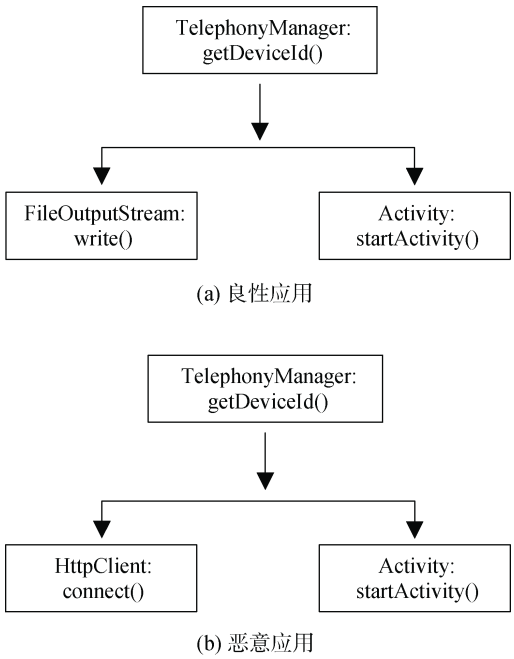


**图 5　良性应用和恶意应用获取 IMEI 的行为**

**Figure 5　The behavior of benign and malware to acquire IMEI**

**定义 1.** 定义 $F$ 为一个敏感 API 调用序列, $F =< SOURCE, A_1, A_2, \ldots, A_{n-1}, A_n, SINK >$, 序列中由若干个 API 函数组成, 其中 *SOURCE* 为污点传播源, *SINK* 为污点汇聚点, $A_n$ 表示该序列中第 $n$ 个 API 调用函数。

**定义 2.** 定义 $FS$ 为一个 Android 应用程序的敏感 API 调用序列集合, $FS = [F_1, F_2, \ldots, F_{n-1}, F_n]$, 其中 $F_n$ 表示其中第 $n$ 个敏感 API 调用序列。

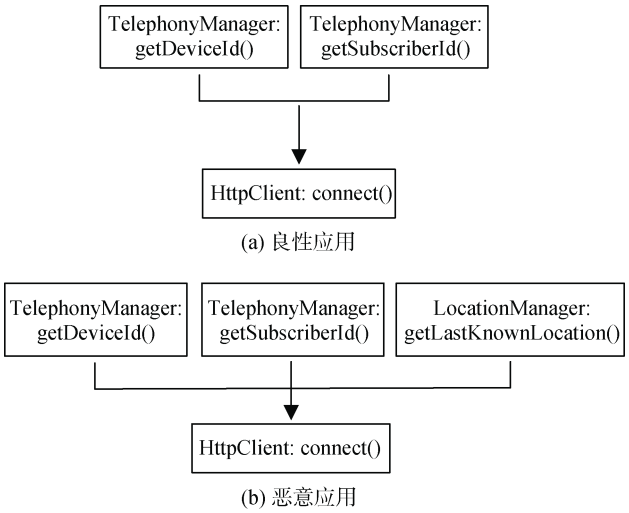


**图 6　良性应用和恶意应用获取多个隐私信息的行为**

**Figure 6　The behavior of benign and malware to acquire multiple private information**

**定义 3.** $|F|$为敏感 API 调用序列 $F$ 的长度。

**定义 4.** 在集合 $FS$ 中，如果存在 $F_i$ 和 $F_j$，$F=F_iF_j$，$F_iF_j$ 表示两个 $F_i$ 和 $F_j$ 的连结，$F$ 的长度为 $|F_i|+|F_j|$，其中 $F_i$ 为 $F$ 的前缀，记为 $F_i \sqsubset F$，$F_j$ 为 $F$ 的后缀，记为 $F_j \sqsupset F$。

**定义 5.** 在集合 $FS$ 中，存在 $F_i$ 和 $F_j$，如果 $F_i$ 和 $F_j$ 为相同的两个序列，则记为 $F_i=F_j$。

**定义 6.** 在集合 $FS$ 中，如果存在 $F_{pub}$，$F_i$ 和 $F_j$，$F_{pub}=F_i \cap F_j$，其中 $F_{pub}$ 仍为连续的 API 调用序列，则称 $F_{pub}$ 为序列 $F_i$ 和 $F_j$ 的连续公共子序列。

假设在 $|F_i| \leqslant |F_j|$ 的情况下，我们将敏感 API 之间的关系定义如下:

(1)如果 $|F_{pub}| \lhd |F_i|$，$F_{pub} \sqsupset F_i$ 且 $F_{pub} \sqsupset F_j$，则称 $F_i$ 和 $F_j$ 存在汇聚关系。

(2)如果 $|F_{pub}| \lhd |F_i|$，$F_{pub} \sqsubset F_i$ 且 $F_{pub} \sqsubset F_j$，则称 $F_i$ 和 $F_j$ 存在分散关系。

(3)如果 $|F_{pub}| = |F_i|$，即 $F_i$ 是 $F_j$ 的连续子序列，则称 $F_i$ 包含于 $F_j$，即 $F_i \subseteq F_j$，其含义表示 $F_i$ 出现在 $F_j$ 的执行过程中，$F_i$ 和 $F_j$ 存在包含关系。

(4)如果 $F_{pub}=F_i[n...|F_i|]F_j[1...m]$，其中 $n>1$，$m<|F_j|$，此时 $F_i[n...|F_i|] \sqsupset F_i$，$F_j[1...m] \sqsubset F_j$，则称 $F_i$ 和 $F_j$ 存在串行关系。

(5)如果 $|F_{pub}| \lhd |F_i|$，$F_{pub}=F_i[n...n+|F_{pub}|]=F_j[m...m+|F_{pub}|]$，其中 $n>1$，$m>1$，此时 $F_{pub}$ 可能会存在着多个，因此我们将其统一表达为非特定交叉关系。

我们将分析出来的 API 之间的关系用五元组 *Re* 来表示，*Re*=[*Convergence*, *Dispersed*, *Contain*, *Connection*, *Cross*]，如果两个敏感 API 调用序列存在其中一种关系，就将该关系在元组 *Re* 中所对应的位置赋值为 1，其余位置为 0。

动态规划(Dynamic Programming, DP)是用来解决多阶段决策过程最优化的一种数量方法，常常适用于解决重叠子问题和具备最优子结构性质的问题，这与我们求解连续公共子序列问题相适应，因此我们使用 DP 方法来判断 API 序列之间的关系并求解出任意两个敏感 API 调用序列之间的连续公共子序列。状态转移公式如公式(1)所示，其中 $C[n][m]$ 表示某一状态下连续公共子序列的长度，$F_i$ 和 $F_j$ 表示敏感 API 调用序列 $i$ 和 $j$。从公式(1)可以看出该方法是将一个问题分解为子问题后递归求解，当 $F_i[n-1]=F_j[m-1]$ 时，子问题就变成了求解 $F_i[1...n-1]$ 和 $F_j[1...m]$ 的连续公共子序列，然后依次进行，最后一个子问题所得的最优解，就是整个问题的最优解，具体如算法 1 所示，对于两个敏感 API 调用序列 $F_i$ 和 $F_j$，通过循环依次判断比对 $F_i$ 的第 $n$ 个 API 调用和 $F_j$ 的第 $m$ 个 API 调用是否相同，如果存在相同的情况，则当前公共连续子序列的长度 $|F_{pub}|$ 加 1，并记录当前 $F_{pub}$ 的第一个元素在序列 $F_i$ 和 $F_j$ 中的位置 *index*，那么当最后一次循环结束后，如果 $|F_{pub}|$ 的值不为 0，我们便通过 *index* 和 $|F_{pub}|$ 的值提取到了 $F_i$ 和 $F_j$ 的连续公共子序列。我们得到连续公共子序列 $F_{pub}$ 后，便可进一步根据我们定义的关系的形式化描述得到关系特征。

$$C[n][m]=\begin{cases} C[n-1][m-1]+1 & F_i[n-1]=F_j[m-1] \\ 0 & \text{其他} \end{cases} \tag{1}$$

当 $F_i$ 和 $F_j$ 的长度分别为 $M$ 和 $N$ 时，该算法将问题分解为了 $M*N$ 个子问题，因此该算法在分析任意两个敏感 API 调用序列时的时间复杂度为 $O(M*N)$，空间复杂度也为 $O(M*N)$。

**算法 1.** 敏感 API 调用序列关系分析.

输入：敏感 API 调用序列集合 $FS=[F_1,F_2,\ldots,F_{n-1},F_n]$

输出: 所有敏感 API 调用序列间的关系特征 $Re$ 和连续公共子序列 $F_{pub}$

INITIALIZE $C[n][m]$, $index \leftarrow 0$.
FOR EACH $F_i, F_j \in FS$ DO
$N \leftarrow |F_i|, M \leftarrow |F_j|$
FOR $n \leftarrow 1$ to $N$, m$\leftarrow 1$ to $M$ DO
//判断 $F_i$ 的第 $n$–1 个 API 和 $F_j$ 的第 $m$–1 个是否相同
IF $F_i[n-1] = F_j[m-1]$ THEN
$C[n][m] = C[n-1][m-1]+1$
IF $C[n][m] > index_i$ THEN
//记录当前公共连续子序列的长度
$|F_{pub}^{ij}| \leftarrow C[n][m]$
//获取公共连续子序列在 $F_i$ 的位置
$index_i \leftarrow n-|F_{pub}^{ij}|$
//获取公共连续子序列在 $F_j$ 的位置
$index_j \leftarrow m-|F_{pub}^{ij}|$
//提取当前公共连续子序列
$F_{pub}^{ij} \leftarrow F_i[index_i, index_i+|F_{pub}^{ij}|]$
END IF
END IF
END FOR
//$|F_{pub}|$不为 0 表示 $F_i$ 和 $F_j$ 存在公共连续子序列, 即 $F_i$ 和 $F_j$ 存在着一定的关系
IF $|F_{pub}^{ij}|$!=0 THEN
IF $|F_{pub}^{ij}|$=$N$ THEN
$Re_{ij}$=[0, 0, 1, 0, 0] //包含关系
IF $index_i$ >1 && $index_i$ +$|F_{pub}^{ij}|$=$N$ THEN
$Re_{ij}$ =[1, 0, 0, 0, 0] //汇聚关系
IF $index_i$ =1 && $index_i$ +$|F_{pub}|$<$N$ THEN
$Re_{ij}$ =[0, 1, 0, 0, 0] //分散关系
IF $index_i$ >1 && $index_i$ +$|F_{pub}^{ij}|$=$N$ && $index_j$ =1 && $index_j$+$|F_{pub}^{ij}|$<$M$ THEN
$Re_{ij}$ =[0, 0, 0, 1, 0] //串行关系
ELSE
$Re_{ij}$ =[0, 0, 0, 0, 1] //非特定交叉关系
END IF
END IF
END FOR
RETURN $Re$ AND $F_{pub}$

## 3.2 特征向量

经过对各敏感 API 调用序列的分析, 我们获取到了序列之间的关系特征和各序列之间的连续公共子序列。由于机器学习算法一般将输入表示为固定长度的数值型特征向量, 因此我们还需要将文本形式的连续公共子序列转化为向量的表示形式。

但是通过我们对提取出的敏感 API 调用序列的连续公共子序列的长度进行统计时发现, 它们的长度是不一致的, 如图 7 所示, 其中长度在 1～5 范围内的样本约占总数的 84.26%, 6～10 范围内的样本比例约为 15.15%, 大于 10 的样本比例约为 0.5%。以上结果表明连续公共子序列的序列长度分布是不均匀的, 因此为了解决连续公共子序列长度不一致而使得矩阵不规则的问题, 我们将连续公共子序列中的 API 进行分类。

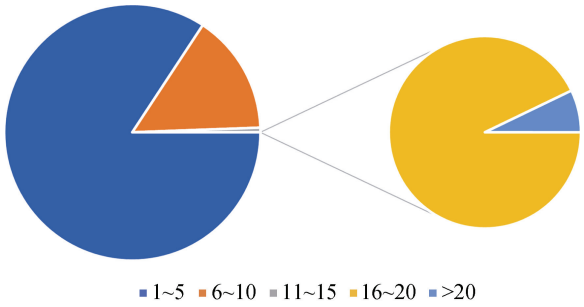


**图 7 连续公共子序列长度区间分布**

**Figure 7 Distribution of consecutive common subsequence length intervals**

**表 1 SUSI 中 Sources 和 Sinks 类别**

**Table 1 Sources and Sinks in SUSI**

| SOURCES | SINKS |
|---|---|
| NETWORK_INFORMATION<br>LOCATION INFORMATION<br>ACCOUNT INFORMATION<br>EMAIL INFORMATION<br>FILE INFORMATION<br>BLUETOOTH INFORMATION<br>VOIP INFORMATION<br>DATABASE INFORMATION<br>PHONE INFORMATION<br>CONTENT RESOLVER<br>NO SENSITIVE SOURCE | PHONE_CONNECTION<br>VOIP<br>PHONE_STATE<br>EMAIL<br>BLUETOOTH<br>ACCOUNT_SETTINGS<br>VIDEO<br>SYNCHRONIZATION_DATA<br>NETWORK<br>EMAIL_SETTINGS<br>FILE<br>LOG<br>INTENT<br>NO_SENSITIVE_SINK |

表 1 为 SUSI 定义的 11 种 Sources 和 14 种 Sinks 的类别。我们为了固定特征向量的大小, 减少特征向量的数据规模, 以提高检测方法的稳定性和运行效率, 我们在此基础上对 SUSI 中定义的类别进行分类和总结, 重新定义了 6 种 API 类别, 如表 2 所示。另外我们将这些 API 类别定义为 6 元组 *ACV*=[*NETWORK*, *LOCATION*, *ACCOUNT*, *DATA_PERSISTENCE*, *INFORMATION*, *INTENT*]。我们建立了一个 API 和 API 类别的映射字典 Dict, 当序列中的某个 API 映射到表 2 中的某个类别时, 就将 *ACV* 元组中该位置的值赋为 1。

**表 2 重新定义的 API 类别**

**Table 2 Redefined API categories**

| Category |
|---|
| NETWORK |
| LOCATION |
| ACCOUNT |
| DATA_PERSISTENCE |
| INFORMATION |
| INTENT |

特征融合的详细描述如算法 2 所示, 首先初始化元组 *ACV* 为[0, 0, 0, 0, 0, 0], 然后对连续公共子序列中的每一个 API 进行遍历, 找到其在字典 *Dict* 中的映射, 然后把映射到的 API 类别在 *ACV* 元组中对应的位置赋值为 1, 最后我们将上一阶段得到的关系特征 *Re* 和连续公共子序列特征 *ACV* 进行拼接融合, 存储到特征向量 *EM* 中。当关系特征的数量为 *M*, $F_{pub}$ 的值为 *N* 时, 该算法的时间复杂度为 *O(M*N)*。

**算法 2.** 特征融合.

输入: 各敏感 API 调用序列的连续公共子序列 $F_{pub}$, 敏感 API 调用序列的关系矩阵 ***Re***, API 类别映射字典 Dict, *ACV*=[0,0,0,0,0,0]

输出: 特征向量 *EM*

FOR $i, j \leftarrow 0$ to NUM of ***Re*** DO

FOR $A_k$ IN $F_{pub}^{ij}$ DO

//判断 API $A_k$ 是否在字典 Dict 中

IF $A_k \in$ Dict.keys() THEN

//找出类别 Dict[$A_k$]在 *ACV* 元组中的位置

$x \leftarrow$index of Dict[$A_k$] in tuple *ACV*

//将 *ACV* 元组下标为 *x* 的位置赋值为 1

$ACV_{ij}[x] \leftarrow 1$

END IF

END FOR

//将 *Re* 和 *ACV* 拼接后添加到 *RM* 中

$EM_{ij}$.append($Re_{ij}$+$ACV_{ij}$)

END FOR

RETURN *EM*

如图 8 所示, 根据我们的方法, 特征提取阶段收集的数据将表示为一个大小为 $K \times L$ 的矩阵 ***EM***, 其中 *K* 表示连续公共子序列的数量, *L* 的大小为 11, $EM_{ij}$ 表示敏感 API 调用序列 $F_i$ 和 $F_j$ 的融合特征向量。我们对每个应用程序的连续公共子序列数量进行了统计, 超过96.6%的应用程序连续公共子序列数量在 100 范围内, 因此我们设置 *K* 的大小为 100, 对于小部分连续公共子序列数量超出 100 的情况, 我们采取随机的策略对超出的部分进行删除, 小于 100 的则使用 0 进行补全。

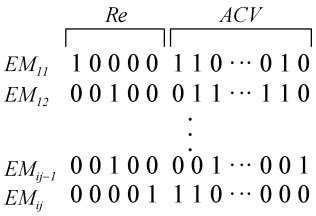


**图 8 特征向量**

**Figure 8 Eigenvector matrix**

## 3.3 机器学习模型

Keras 是目前主流的基于 Python 的机器学习库, 截至到 2021 年底, Keras 在行业和研究领域的应用率比其他框架更高[36], 因此我们使用 Keras 来构建机器学习模型。表 3 和图 9 分别描述了 CNN 模型的参数设置和模型结构。我们的 CNN 模型共分为输入层、卷积层、最大池化层和两个全连接层。第一层为输入层, 即为上一阶段生成的 $K \times L$ 的融合特征向量矩阵, 第二层为卷积层, 共有 256 个大小为 3 的卷积核, 激活函数为 ReLU, 第三层为大小为 3×3 的最大池化层, 紧接着为两个全连接层, 第一个全连接层的神经元数量为 128, 激活函数为 ReLU, 第二个全连接层为分类器, 用于输出检测的结果, 因为我们的检

**表 3 模型参数设置**

**Table 3 Model parameter settings**

| 结构 | 参数 | 激活函数 |
|---|---|---|
| 输入层 | $K \times L$ | / |
| 卷积层 | filters=256, kernel_size=3 | ReLU |
| 最大池化层 | maxpooling_size = (3,3) | / |
| 全连接层 | units=128, Dropout=0.5 | ReLU |
| 全连接层 | units=2 | Softmax |

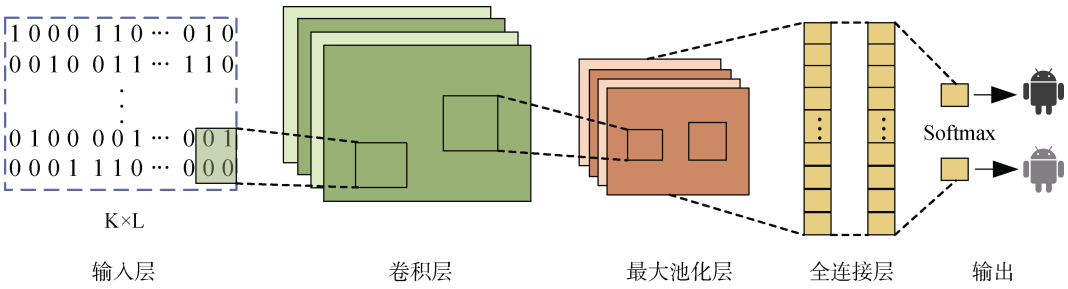


**图 9 机器学习模型**

**Figure 9 Machine learning model**

测结果为良性应用程序和恶意应用程序两种, 所以该层的神经元数量为 2, 激活函数为 Softmax。此外, 为我们在两个全连接层中间添加了 Dropout, 它会随机删除网络中的一些神经元, 降低神经元之间的相关性以防止模型过拟合。

## 4 实验

在本节中, 我们首先介绍了实验环境和数据集, 然后对机器学习训练模型的准确度、精确度、召回率和 F1 分数进行评估, 并在不同数据集下验证提出方法的稳定性。

### 4.1 实验环境

该实验使用运行在 Windows 11 professional edition 上的 Windows Subsystem for Linux(WSL) 2 作为运行环境, CPU 为 Intel(R) Core(TM) i7-11700@2.5GHz, GPU 为 NVIDIA GeForce RTX3090, 内存为 32GB, Python 版本是 Python 3.8。我们使用的来自 MalGenome[37] 和 AndroZoo[38] 的数据集。Android Malware Genome Project 是由北卡州立大学计算机科学系的两位研究人员发起, 致力于开发出更好的检测工具。AndroZoo 是由卢森堡大学发起, 目前已经收集包括来自 Google Play 应用市场、VirusShare 在内的多个来源的 Android 应用程序集合, 已经有超过 2 千万个 APK 文件, 且数量在不断增长。

### 4.2 实验结果

为了保证数据集的一致性, 我们首先使用来自 MalGenome 数据集的 1200 个 Android 恶意应用程序和 1200 个来自 Google Play 的良性应用程序。为了验证方法的有效性, 我们将此数据集划分为训练集和测试集, 其中 90%的样本是训练集, 其余 10%是测试集, 并从训练集中选取 20%作为验证集。如图 10 所示, 训练在 50 次迭代的情况下, 由于训练集的数据进行了正则化处理, 所以出现训练集的损失值大于验证集的损失值的情况, 但是可以看出训练集和验证集的损失值均在不断下降且都已经趋于稳定, 因此可以初步判定训练模型未出现过拟合的情况, 另外经过多次迭代后我们的模型在训练集的准确率为 98.1%, 测试集的准确率为 98.5%, 都表现了出较高的准确率且准确率相近, 由此可以得出模型未出现过拟合情况的结论, 且已经达到最优, 我们在此模型的基础上进行下一步的验证。

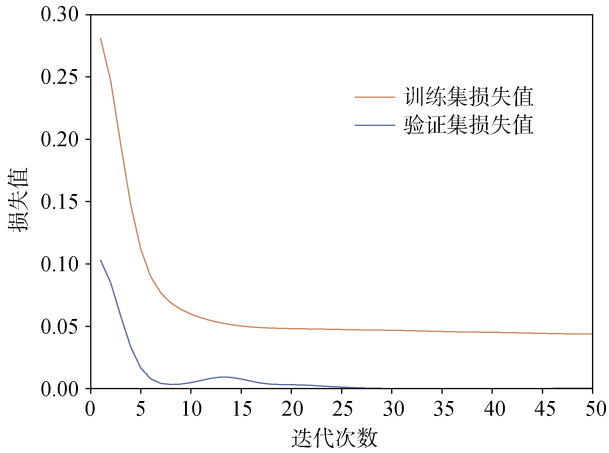


**图 10 Loss 曲线图**

**Figure 10 Loss curve**

最常见的统计标准包括真阳性(True Positive, TP)、假阳性(False Positive, FP)、真阴性(True Negative, TN)、假阴性(False Negative, FN)和准确率。其中 TP 是指预测结果为恶意应用且为真实恶意应用; FP 是指预测为恶意应用但实际为良性应用; FN 是指预测为良性应用但实际为恶意应用; TN 是指预测为良性应用且为真实良性应用; 准确率为所有预测正确的样本数量占所有应用数量的比例。

我们在对 Complex-Flows 方法进行复现时, 经过多次验证发现, 当参数 *gram_size* 为 1 的时候准确率达到最高, 这与 Shen 等人在 MalGenome 数据集下的验证结果一致, 因此我们选取 *gram_size* 为 1 的实验结果作为对比数据, 我们在以上五个方面将实验结果与 Shen 等人提出的 Complex-Flows 对比, 对比结果如表 4 所示。

表 4 MalGenome 下不同方法检测结果

Table 4 Detection results of different methods in MalGenome (%)

| 方法 | TP | TN | FP | FN | 准确率 |
|---|---|---|---|---|---|
| 本文方法 | 98.6 | 98.4 | 1.6 | 1.4 | 98.5 |
| Complex-Flows[22] | 86.8 | 98.2 | 1.8 | 13.2 | 92.5 |

通过对比分析我们可以看出，我们的 TP 为 98.6%, TN 为 98.4%, 均优于 Complex-Flows 的 86.8% 和 98.2%, 而且我们的漏报率 FN 仅为 1.4%, 远远低于 Complex-Flows 的 13.2%, 这说明对敏感信息流之间关系特征的表述对于良性应用和恶意应用的区分起着重要的作用。我们的方法在准确率方面比 Complex-Flows 也高了 6%, 这表明我们提出的更加细化的敏感信息流间的关系特征对于提升恶意软件检测的准确性起到了关键性的作用, 对于当前恶意应用和良性应用的行为越来越相似的情况下, 则更加突显了我们方法的优越性。

受试者工作特征曲线(Receiver Operating Characteristic Curve, ROC)曲线主要用于评价机器学习模型的预测能力。如图 11 所示, 蓝色虚线为随机猜测曲线 $y=x$, ROC 曲线在 $y=x$ 的上方则说明模型的性能较好, x 轴为假阳性率(False Positive Rate, FPR), y 轴为真阳性率(True Positive Rate, TPR), TPR 越高, FPR 越低, 则曲线越靠近左上角, 表明检测模型的性能越好, AUC(Area Under Curve)表示 ROC 曲线下的面积, 面积越接近 1, 检测模型性能越好, 我们提出的方法的曲线面积相比于 Complex-Flows 的更接近于 1, 具有更高的准确性。

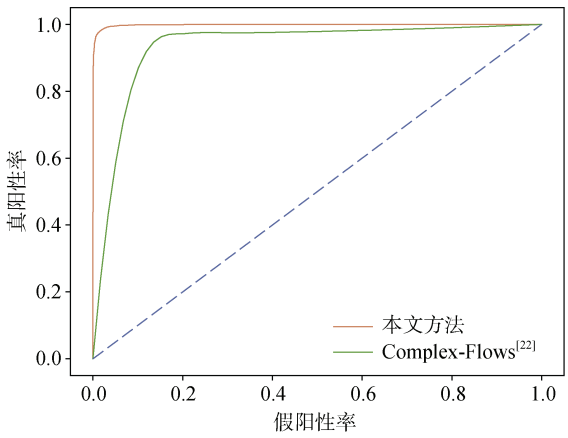


图 11 不同方法 ROC 对比曲线图

Figure 11 ROC comparison curve of different methods

为了进一步验证方法的稳定性，我们从 AndroZoo 选取了 2000 个恶意应用程序和来自 Google Play 的 2000 个良性应用程序进行实验, 从准确率、精确率、召回率和 F1-分数四个方面进行对比, Complex-Flows 方法的参数 *gram_size* 仍设置为 1 时准确率最高。由于 AndroZoo 的应用程序仍在不断更新, 随着程序的功能复杂性和代码复杂性的增加, 从表 5 可以看出各项指标相比 MalGenome 数据集来说略微有所下降, 但是依然达到了 97.6%的准确率, 精确率、召回率和 F1-分数也均在 97%以上, 均优于 Complex-Flows 方法。

表 5 AndroZoo 下不同方法检测结果

Table 5 Detection results of different methods in AndroZoo (%)

| 方法 | 准确率 | 精确率 | 召回率 | F1-分数 |
|---|---|---|---|---|
| 本文方法 | 97.6 | 98.2 | 97.1 | 97.6 |
| Complex-Flows[22] | 90.8 | 91.0 | 90.5 | 90.7 |

另外如图 12 所示, 蓝色虚线为随机猜测曲线 $y=x$, 我们对假阳性率在 0~0.2 的区间内对图像进行局部放大, 可以看出这两个数据集的 ROC 曲线十分接近, 表明了我们方法在检测不同类型和复杂度的软件时体现出的稳定性。

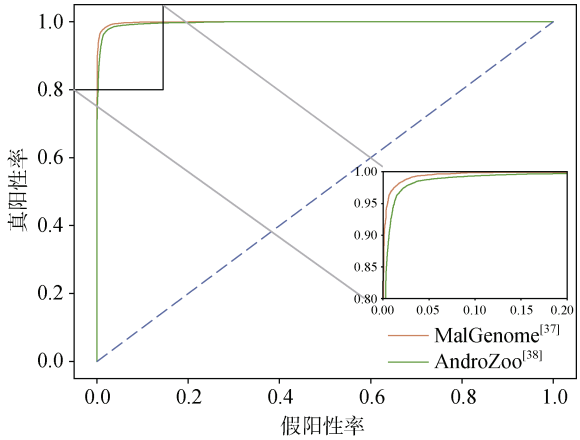


图 12 不同数据集的 ROC 对比曲线图

Figure 12 ROC comparison curve of different datasets

## 5 总结

本文针对应用程序复杂性的提高导致的基于粗粒度的信息流特征检测准确率低的问题, 提出了一种新的基于信息流关系特征的恶意软件检测方法。

我们对敏感 API 调用序列之间的关系特征给出了详细的形式化描述, 对其进行了更加深层、细化地挖掘, 完善了当前研究对信息流间关系特征分析的不足。另外我们结合了 CNN 机器学习模型, 提出了连续公共子序列中 API 的分类以及和关系特征的融合, 解决了敏感 API 公共调用长度不一致的问题, 也缩减了特征的数据量。我们在 MalGenome 数据集下进行对比验证, 结果表明, 我们的方法在恶意软件检测的准确率达到了 98.5%, 另外我们在较新的数据集 AndroZoo 上进行实验以验证方法的稳定性, 准确率也达到了 97.6%, 优于 Complex-Flows 方法。在下一步的工作中, 我们将进一步结合污点追踪等动态分析方法来应对软件加固和代码混淆等技术对我们方法的影响, 以便更加全面的提取敏感信息流, 扩充我们的关系特征库。

## 参考文献


[1] StatCounter Global Stats. Mobile Operating System Market Share Worldwide[EB/OL]. https://gs.statcounter.com/os-market-share/mobile/worldwide. 2022-10-27.

[2] Gibler C, Crussell J, Erickson J, et al. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale[M]. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 291-307.

[3] Qing S H. Research Progress on Android Security[J]. *Journal of Software*, 2016, 27(1): 45-71.
(卿斯汉. Android 安全研究进展[J]. *软件学报*, 2016, 27(1): 45-71.)

[4] Wang X Q, Sun K, Wang Y W, et al. DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices[C]. *Proceedings 2015 Network and Distributed System Security Symposium*, 2015.

[5] Li L, Bissyandé T F, Papadakis M, et al. Static Analysis of Android Apps: A Systematic Literature Review[J]. *Information and Software Technology*, 2017, 88: 67-95.

[6] Sabelfeld A, Myers A C. Language-Based Information-Flow Security[J]. *IEEE Journal on Selected Areas in Communications*, 2003, 21(1): 5-19.

[7] Arzt S, Rasthofer S, Fritz C, et al. FLOWDROID: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps[J]. *ACM SIGPLAN Notices*, 2014, 49(6): 259-269.

[8] Lam P, Bodden E, Lhotak O, et al. The Soot Framework for Java Program Analysis: A Retrospective[J]. *In Cetus Users and Compiler Infastructure Workshop*, 2011, 15(35).

[9] Bartel A, Klein J, Le Traon Y, et al. Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot[C]. *The ACM SIGPLAN International Workshop on State of the Art in Java Program analysis*, 2012: 27-38.

[10] Arzt S, Rasthofer S, Bodden E. Susi: A tool for the fully automated classification and categorization of android sources and sinks[J]. University of Darmstadt, Tech. Rep. TUDCS-2013-0114, Citeseer, 2013.

[11] Li L, Bartel A, Bissyandé T F, et al. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps[C]. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015: 280-291.

[12] Octeau D, McDaniel P, Jha S, et al. Effective {Inter-Component} Communication Mapping in Android: An Essential Step Towards Holistic Security Analysis[C]. *22nd USENIX Security Symposium*, 2013: 543-558.

[13] Octeau D, Luchaup D, Dering M, et al. Composite Constant Propagation: Application to Android Inter-Component Communication Analysis[C]. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015: 77-88.

[14] Wei F G, Roy S, Ou X M, et al. Amandroid[J]. *ACM Transactions on Privacy and Security*, 2018, 21(3): 1-32.

[15] Mariconti E, Onwuzurike L, Andriotis P, et al. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models[C]. *Proceedings 2017 Network and Distributed System Security Symposium*, 2017: 1-34.

[16] Gordon M I, Kim D, Perkins J H, et al. Information flow analysis of android applications in droidsafe[C]. *NDSS*, 015, 15(201): 110.

[17] Yang J Y, Fan J W, Zhou J, et al. Android Malware Detection Method Based on Behavior Pattern[J]. *Journal of Frontiers of Computer Science and Technology*, 2022, 16(8): 1792-1799.
(杨吉云, 范佳文, 周洁, 等. 融合行为模式的 Android 恶意代码检测方法[J]. *计算机科学与探索*, 2022, 16(8): 1792-1799.)

[18] Shen F, Del Vecchio J, Mohaisen A, et al. Android Malware Detection Using Complex-Flows[J]. *IEEE Transactions on Mobile Computing*, 2019, 18(6): 1231-1245.

[19] Team K. Keras documentation: Why choose Keras[EB/OL]. https://keras.io/why_keras/. 2022-10-28.

[20] Zhou Y J, Jiang X X. Dissecting Android Malware: Characterization and Evolution[C]. *2012 IEEE Symposium on Security and Privacy*, 2012: 95-109.

[21] Allix K, Bissyandé T F, Klein J, et al. AndroZoo: Collecting Millions of Android Apps for the Research Community[C]. *The 13th International Conference on Mining Software Repositories*, 2016: 468-471.


**杨保山** 于 2020 年在河南大学软件工程专业获得学士学位。现在信息工程大学软件工程专业攻读硕士学位。研究领域为软件安全分析。Email: yang201998@foxmail.com

**杨智** 现任信息工程大学教授, 研究领域为操作系统安全、云计算安全和隐私保护。Email: zynoah@163.com

**张红旗** 现任信息工程大学教授, 研究领域为网络安全、风险评估、等级保护和信息安全管理等。Email: zhq37922@126.com

**韩冰** 现任信息工程大学讲师, 研究领域为网络空间信息管理和评估。Email: yzdzjs@sina.com

**陈性元** 现任信息工程大学教授, 研究领域为网络与信息安全、大数据安全等。Email: chxy302@vip.sina.com

**孙磊** 现任信息工程大学教授, 研究领域为网络空间安全和操作系统安全。Email: 13523556215@139.com