2020 年全国大学生信息安全竞赛

作品报告

作品名称： 基于权限对图和广义敏感 API 的安卓恶意软件联合检测器

电子邮箱： 1572517198@qq.com

提交日期： 2020.6.14

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用A4纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

目录

摘要

随着智能终端的普及，基于智能操作系统Android的应用也层出不穷，随之而来的，就是针对安卓平台的恶意软件在急剧增加，严重威胁安卓系统用户的信息安全。目前国内外学者对安卓恶意软件检测技术进行了广泛的研究，他们主要从应用程序的权限、组件、用户界面(UI)、安卓软件安装包(APK)结构、API函数调用序列等特征着手，结合静态分析技术或动态分析技术进行研究。

通过分析恶意软件可以发现，其开发者的技术也在不断提高，安卓恶意软件的功能越来越多，隐蔽性越来越强，导致现有的检测技术中使用的应用程序特征无法快速准确地识别出恶意软件。已有的基于权限的检测方法大部分是通过提取恶意软件和普通应用软件中使用的权限并分析危险的权限，而没有考虑恶意软件的权限之间的关系。

针对以上问题，我们分析各种应用程序特征，研究了不同家族恶意软件样本的恶意功能，提取和分析恶意软件中成对出现的权限模式，将传统的特征检测与机器学习中的随机森林算法结合，提出了一种基于权限对图和广义敏感API的安卓恶意软件检测方案，并且实现了一个B/S架构的安卓恶意软件联合检测器——AnDetector。

检测系统主要从两方面进行设计：权限检测与API调用检测。在权限检测模块，提取待检测文件所申请的权限信息，利用这些权限信息生成两两对应的权限图，使用以权限对图为基准的检测，充分利用不同权限之间的隐含关联。在API调用检测模块，提取出待检测文件中的API信息并加以筛选，使用随机森林算法进行判别，可以增强对权限整体内部联系的聚类分析；在API提取过程中，增强对上下文关联语法特征与API本身语义特征的处理。此外，还利用APK其他特征，使用图形神经网络进行辅助检测。

本作品最终实现了一个安卓恶意软件多特征联合检测器。通过测试不同的数据集，检测的准确率达到98%以上，体现了该检测平台具有较高的准确性与良好的适应性。此外，本作品实现了安卓恶意软件的批量检测和分析，并且能够对安卓软件进行风险评估。

关键字：Android 静态检测 特征提取 权限对图 机器学习

第一章 作品概述

1.1 背景分析

近年来，移动终端给人们的工作与生活带来了越来越多的便利，也极大地改变着人们的生活娱乐方式。这一方面归因于智能终端的硬件和软件系统越来越强大，同时成本越来越低，另一方面也归功于移动应用的爆炸式增长，而且很多应用是免费的。据中国互联网信息中心发布的第 44 次《中国互联网发展状况统计报告》报告统计，截止 2019 年 6 月，我国移动互联网网民数量突破 8.47 亿，占我国网民总数量的 99.2%。金融服务、生活服务以及社交娱乐等全面面向移动互联网迁移。一方面这些行业领域给我们带来了生活的便利，物质和精神需求的满足；另一方面这些行业也遭受到移动恶意软件的攻击，不仅侵害了移动用户的合法利益，同时破坏了正常的行业领域发展。

随着移动互联网市场规模的变大，移动应用也越来越丰富，这也为一些恶意应用和风险应用的出现提供了极好的条件。

根据 360 烽火实验室的报告[1]，2019 年全年，360 安全大脑共截获移动端新增恶意程序样本约 180.9 万个，平均每天截获新增手机恶意程序样本约 0.5 万个。新增恶意程序类型主要为资费消耗，占比 46.8%；其次为隐私窃取（41.9%）、远程控制（5.0%）、流氓行为（4.6%）、恶意扣费（1.5%）、欺诈软件（0.1%）。

过去五年，360 安全大脑每年截获的移动端新增恶意软件样本数量均达数百万个。纵观 2019 年全年恶意样本增长情况，如下图 1-1 所示。

图 1-1　2019 年全年恶意样本增长情况

2019 年全年移动端新增恶意软件类型主要为资费消耗，占比 46.8%；其次为隐私窃取（41.9%）、远程控制（5.0%）、流氓行为（4.6%）、恶意扣费（1.5%）、欺诈软件（0.1%）。如下图 1-2 所示。

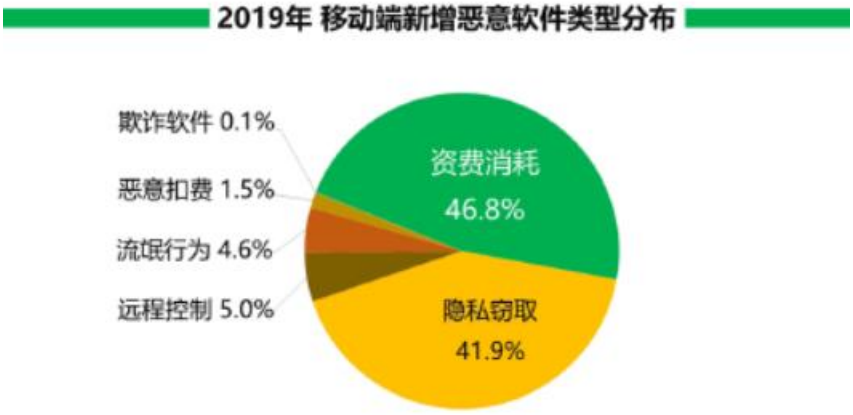

图 1-2　2019 年移动端新增恶意软件的类型分布

目前，在主流的智能操作系统中，安卓(Android)系统是市场占有率最高的。而且，各终端厂商基于安卓系统也定制了各种新的操作系统，例如小米手机的 MIUI 系统等。2019 全年从漏洞数量看，Android 系统以 414 个漏洞位居产品漏洞数量榜首。从系统更新情况看，截至 2019 年 5 月，Google 发布的 Android 系统版本分布统计，Android Oreo（Android 8.0/8.1）达到 28.3%，占比第二的是 Android Nougat（Android 7.0/7.1）总占比已达 19.2%。

安卓系统的开放性得到了应用程序开发者的青睐，巨大的市场份额也吸引着恶意攻击者的注意，各种恶意应用不断出现。移动互联网信息安全的形势是严峻的，而且普通用户对安全防范的意识还有待提高。因此，如何对大量未知的安卓恶意软件进行有效的检测具有重要的研究价值。基于此，我们设计并开发了作品——基于权限对图和广义敏感 API 的安卓恶意软件多特征联合检测器。

1.2 相关工作

安卓恶意应用的传统检测方法目前主要分为下列两种：1）动态检测方法[2]：检测时需要运行待检测应用，进行动态追踪。但是在此方法中需要运行该应用，将带来较大的风险，同时会占用更多的系统资源。2）静态检测方法[3]：在检测过程中不需要运行应用，相关资源的消耗量少，可以节省更多的系统资源，并且代码覆盖率高。但是分类特征的选取就变得极为重要。

随着数据量的与日俱增，基于机器学习的检测方法应运而生。机器学习算法可以通过训练大量的数据集，并利用恰当的特征信息而得出最佳分类模型。

在依据权限申请清单而进行的分类探索中，Anshul Arora 等人[4]提出了一种新颖的检测方法：将权限结合成权限对并将之图表化。在此篇论文中，检测者同样利用权限清单进行检测；不同以往的是，测试者并没有将权限清单中的权限列表作为一维特征向量直接作为输入进行分类预测，而是以每个权限作为节点，将清单中的每个节点两两相连，以此构成一份权限对图。通过将权限两两组合图化的方式，将一维特征向量转化为二位特征向量。此种表示方法，更可以突出权限之间所存在的内部联系；经过大量的训练后，可以提出相关的无效边，通过删除检测时无用的权限信息，既可以提高检测效率，又可以提高准确率。此项方法的具体原理以及实现流程将在下文介绍。

通过评测应用内部 API 信息而进行分类，在以往的测试中也有良好的效果；随着近些年来人工智能算法的普及，使用机器学习相关算法进行大量数据的处理而得到了良好的应用。在 Na Huang 等人的实验中[5]，测试者选取了 API 信息为特征向量，以 CNN 分类器作为基准进行判别，成功检测率为 94.3%；在王聪等人的实验中[6]，测试者使用关联权限方法与贝叶斯算法进行实验，最终检测率为 93.7%；在李孔渤等人的实验中[7]，测试者选取集成学习方法，最终达到了 95.8%的准确率。综上可见，使用机器学习相关算法回取得比较好的准确率。而随机森林算法在分类问题上有良好的应用效果，所以在实现过程中选择随机森林算法。

1.3 作品特色描述

一、基于权限对图的恶意软件检测方法

本作品采用的检测方法基于Android系统权限控制机制，提取待检测软件所申请的权限信息，利用这些权限信息生成两两对应的权限图，使用以权限对图为基准的检测，充分利用不同权限之间的隐含关联。通过多次优化算法将每条边计算出权重值，并由此形成权限图字典用于待测软件的检测，累加所有权限对的权重，评估软件的威胁程度，生成安全检测报告。整体流程压缩到10秒之内，快于现有的大部分在线云沙箱，同时具有很高的准确性。

二、双方案联合判定

本作品除了实现快速静态检测，还为用户提供深度检测方案。通过反编译源文件，

提取出待检测软件中的API信息并加以筛选，并且使用随机森林算法进行判别，可以增强对权限整体内部联系的聚类分析；在API提取过程中，增强对上下文关联语法特征与API本身语义特征的处理。此外，还利用APK文件其他特征，使用图形神经网络进行辅助检测，提高了平台的检测正确率。

三、多维可视化报告

AnDetector平台通过对用户上传的待检测软件进行本地化存储，并在后台调用检测模块对文件安全性进行打分以及重点权限对标记，前端采取可视化技术直观展示权限对之间的关系，对软件行为进行监控，为用户提供更详细易懂的报告，全面剖析待检测软件的风险情况。

1.4 作品应用前景

安卓系统的开放性得到了应用程序开发者的青睐，海量的安卓软件涵盖了办公学习、社交通讯、理财购物和旅游娱乐等，给人们的工作与生活带来了更多的便利和乐趣。但是，各种恶意软件不断出现。恶意软件是目前移动智能终端上被不法分子利用最多、对用户造成危害和损失最大的安全成胁类型。移动互联网信息安全的形势是严峻的，而且普通用户的安全防范意识还有待提高。基于此，我们设计并实现了作品——基于权限对图和广义敏感 API 的安卓恶意软件联合检测平台。

在用户使用该检测平台时，只需要将待检测的安卓软件通过浏览器上传界面传输到服务器后端。在后端，检测模块内部存在已封装好的检测模型——此检测模型有服务器定期维护更新，以实现准确率最大化。检测过程迅速，成功率高，使用便捷；并且在使用过程中减少了用户端资源的消耗，具有轻量级性质。

作为移动应用发布的源头，移动应用商店很有必要对上架的安卓软件进行批量的检测。本作品无论是对已知恶意应用还是未知的恶意应用，都有很好的检测和评估效果，同时在检测过程中，检测速度快，人工操作少，能为用户提供详细易懂的可视化报告，十分适合移动应用商店使用。因此，本作品具有较高的应用价值。

第二章 作品设计与实现

2.1 系统方案

2.1.1 作品的整体架构

本作品 Android 恶意软件检测平台采用 B/S 架构，由云服务器 CS（Cloud Server）、用户（User）两部分组成，如图 2-1 所示。

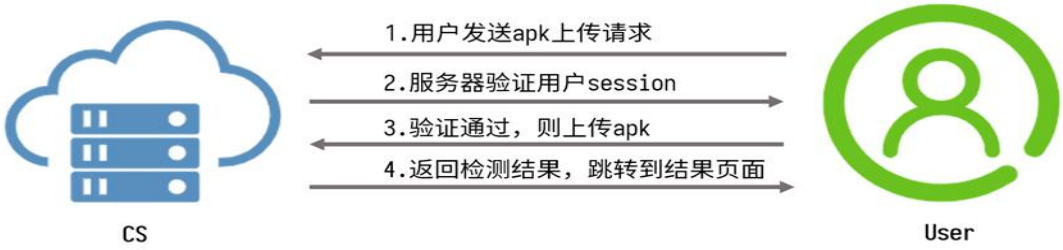

图 2-1 恶意软件检测平台构架

云服务器（Cloud Server) CS：Android 恶意软件检测平台的网站搭建在云服务器(Cloud Server)上，目前使用的是阿里云的 ECS 服务(Elastic Compute Service)，是阿里云提供的稳定可靠、弹性扩展的 IaaS（Infrastructure as a Service）级别云计算服务。云服务器免去了采购 IT 硬件的前期准备，让运营者更便捷、高效地使用服务器，实现计算资源的即开即用和弹性伸缩。云服务器是实现业务的主体，它负责用户信息的存储、验证，以及 APK 文件检测等功能。

用户（User）：Android恶意软件检测平台的用户为待检测APK文件的所有者。用户访问采用响应式设计方法设计的网页，并通过网站客户端上传待检测的APK文件，最终获得该文件的检测结果。

2.1.2 作品的网络结构

Android 恶意软件检测平台采用 B/S 结构（Brower/Server，浏览器/服务器），这是 Web 兴起后的一种网络结构模式。Web 浏览器是客户端最主要的应用软件。这种模式将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用；

客户机上只需要安装一个浏览器，服务器上安装 SQL Server, Oracle, MySql 等数据库；浏览器通过 Web Server 同数据库进行数据交互。

B/S 结构有三层结构，分别是：

1) 第一层表现层：主要完成用户和后台的交互及最终查询结果的输出功能；

2）第二层逻辑层：主要是利用服务器完成客户端的应用逻辑功能；

3）第三层数据层：主要是接受客户端请求后独立进行各种运算。

具体如图 2-2 所示。

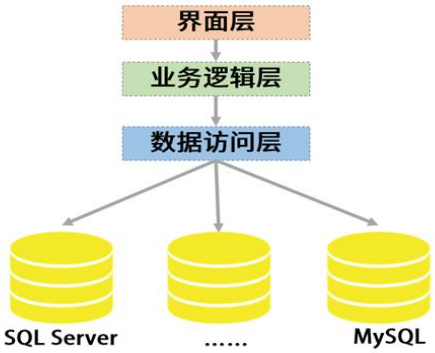

图 2-2　B/S 三层结构

Android 恶意软件检测平台具体的 B/S 架构模式为 WEB 浏览器—云服务器—内置数据库模式：

1）WEB 客户端向云服务器 CS 发起 Https 请求；

2）云服务器 CS 中的 web 服务层能处理 Https 请求；

3）云服务器 CS 中的应用层部分调用业务逻辑，调用业务逻辑上的方法；

4）云服务器 CS 和内置数据库进行数据交换，然后将模版＋JSON 数据渲染成最终的结果页面，返送给客户端。

具体形式如图 2-3 所示。

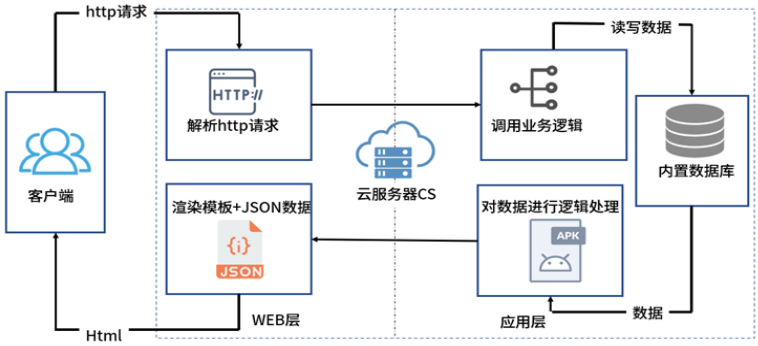

图2-3　WEB浏览器—云服务器—内置数据库构架

2.2 实现原理

2.2.1 权限对图化方法

本作品主要采用了主流的APK反编译工具，基于权限的安卓恶意软件静态检测技术，以及构造**权限对图**的检测模型，并进行图形合并和图形化简等思想。我们的平台实现了安卓恶意应用的单个或批量检测和分析，并且能够对应用做出风险评估。我们平台的检测率优于大多数类似的安卓恶意软件静态检测工具。

2.2.1.1 Android Permission安全控制机制

在 Android 系统中，用户自行安装的应用程序不具备可信性，在默认的情况下，Android 应用程序没有任何权限，不能访问受保护的设备 API 与资源。因此，权限控制机制是 Android 安全机制的基础，决定允许还是限制应用程序访问受限的 API 和系统资源。应用程序的权限需要明确定义，在安装时被用户确认，并且在运行时检查、执行、授予和撤销权限。

具体而言，应用程序在安装时都分配有一个用户标志（UID）以区别于其他应用程序，保护自己的数据不被其他应用获取。Android 根据不同的用户和组，分配不同权限，这些 Android 权限在底层映射为 Linux 的用户与组权限。权限机制的实现层次简要概括如下：

应用层显式声明权限：应用程序包（.apk文件）的权限信息在 Android 应用程序的清单文件 `AndroidManifest.xml` 中通过`<permission>`，`<permission-group>`与`<permission-tree>`等标签指定。如果 `package` 需要申请使用某个权限，那么需要使用 `<use-permission>` 标签来指定。`<uses-permission>` 是 Android 预定义的权限。而我们的平台就是着重于提取此类权限来构造权限对图的检测模型。

2.2.1.2 基于权限对的静态检测方法

恶意软件分析和检测技术可以分为三类：静态分析，动态分析和混合方法。静态检测利用相应的反编译工具提取程序的静态特征如语法语义、签名等特性等进行分析，评估软件安全。我们采用了的是基于权限对的静态分析方法，该方法不同于之前的各种基于权限的静态分析方法。之前的基于权限的检测方法大多是仅专注于通过提取恶

意软件和普通应用软件中广泛使用的权限并分析危险的权限，而没有考虑在正常和恶意软件中哪些权限模式成对出现，即使有的考虑到了安卓恶意软件的许可模式，但是他们也没有考虑普通应用程序，也没有提供任何检测模型。

恶意软件中存在一些权限往往是成对出现的，这与正常软件会存在差异，因此我们决定考虑权限之间的关系。分析两个以上的权限直接的关系比较复杂，会导致生成大量的权限模式，因此我们采用了权限对的思想，构建两个一组的权限模型。我们主要采用的方法的通过提取恶意软件和正常软件的权限，所有的权限构成顶点，所有顶点间形成边，每条边就是一组权限对，从而来构建正常图和恶意图训练模型，这些模型用来检测已知或未知的软件的恶意得分和正常得分，从而评估软件的风险。与此同时，我们也采用了图形合并算法，以及边消除算法，在下面会具体介绍。

2.2.1.3 检测系统的实现

本平台主要功能是对用户上传的 APK 文件进行扫描，给出正常得分和恶意得分，进行风险评估。

检测系统主要分为以下四个阶段：

1.权限的提取；

2.权限对图模型的构建；

3.根据检测模型进行风险评估；

4.优化权限对图模型。

1. 权限的提取

APK 安装包是检测 Android 应用程序安全性的主要数据源。APK 是 Android Package 的缩写，即 Android 安装包。APK 文件是字节码文件、资源文件、清单配置文件以及签名文件的压缩包集合。如下图 2-4 所示，经解压后的 APK 文件包含的具体内容。

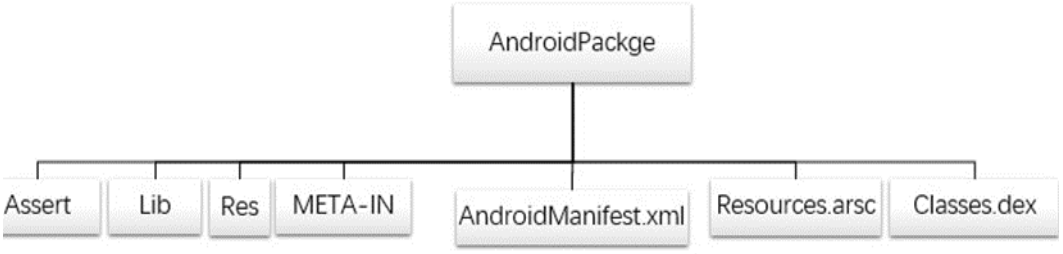

图 2-4　APK 文件结构

其中比较重要的是 `AndroidManifest.xml`：这个文件是每个 Android 应用程序都必须有的，该文件包括了四大组件的配置信息，声明了应用程序向系统申请的权限列表、应用程序的当前版本号、编译程序的 SDK 版本号等。

我们利用 APK 反编译工具 apktool 从 APK 安装包中得到权限清单及其中所申请的权限。我们利用`"aapt dump permissions + apk 路径"`命令提取出每个 apk 文件所需申请的权限。

2. 权限对图模型的构建

我们考虑了近三年的安卓恶意软件样本以及 `Andro-Dumpsys` 数据集中恶意软件部分作为数据集[12]以及我们从 GooglePlayStore 下载了普通的 Android 应用程序作为正常软件的数据集。权限对图模型的构建主要分为如下两个部分。

（1）图形构建阶段

在此阶段，我们生成四个权限对图，具体的**图构造算法**如下。

在 `code_final.py` 模块，我们使用 `getCurrentDirApk()` 函数生成特定路径下软件集的权限对图。

```python
def getCurrentDirApk(filepath):
    global w
    n = 0
    files = os.listdir(filepath)
    # 遍历给定路径下的文件
    for filename in files:
        if os.path.splitext(filename)[1] == '.apk':
            # 如果是apk文件则打印出来
            # print('find apk:', filename)
            n = n + 1  # 计算apk个数
            # 得到apk文件的绝对路径
            apk_file = filepath + '/' + filename
            # 调用函数开始提取清单中的权限，返回权限列表
            str_list = getAppBaseInfo(apk_file)
            # 调用函数，开始生成权限对字典
            w = draw_map(str_list)
    # 归一化
    for key in w.keys():
        w[key] = (float)(w[key]) / n
    return w
```

图 2-5 `getCurrentDirApk()`函数

技术原理：该模型使用图来表示每个应用程序中各个权限之间的关系。该算法用于构造图 `G(V,E)`，通过用节点 `V` 来表示唯一的权限，并用一个加权边 `E` 连接一对权限，每个边的权重随着来自不同应用程序的相同权限对的出现而增加。具有单一权限或者

无权限的应用程序不会对任何分析和检测造成影响。

如果一个应用程序有 N 个唯一权限，那么算法会创建 N 个顶点，每个顶点代表唯一权限。被加到图中的权限对的数量是C_N^2。在一个应用程序中，如果有两个权限 P_1 和 P_2，如果之前不存在这两个权限之间的边的话，那么在它们之间创建一个带有权重为 1 的边，否则创建边并且权重增加一。

一旦我们处理了数据集中的所有的应用程序，我们将每条边的权重除以数据集中 apps 的总数。这是为了规范不同图形中的权限对的权重。

算法 1　图形构建算法

```
输入：数据集 D 中的应用程序集 N
输出：权限对图 G（V,E）
初始化：V = ∅ and E = ∅
for　任意应用 N∈D　do
    使用 apktool 提取它的清单文件 M
    设 P =　（P1，P2，P3...Pm ）为 M 中的权限集
    if ( |P|==0 or |P|==1，即无任何权限或只有一个权限)　then
        V=V+∅　and E=E+∅
    else
        for 每个权限 Pi ∈M ，创建一个节点 Vi if Vi ∉ V
        为来自文件 M 中的每个权限对（Pi，Pk）添加一条边
        if　边（Pi，Pk）∉ E　then
            边权重 W（Pi，Pk）= 1
        else
            W（Pi，Pk）++
        end if
    end if
end for
每条边的权重除以应用的总数 N
返回图 G（V,E）
```

图 2-6　图形构建算法伪代码

（2）图形合并阶段

我们需要对生成的三个恶意图进行合并，此时需要用到**图形合并算法**。

技术原理：将三个恶意软件图（GG, GD, GK）合并成一个通用的恶意图（GM），通过组合它们的边。有两种类型的边：公共和不相交。

将所有不相交的边连同它们的边权一起插入GM是简单的。关键是如何确定公共边的权重。考虑一个公共边e，其权重分别是GG, GD和GK图中的ew_1，ew_2，ew_3。在最后的图GM中，我们可以用不同的方法考虑边 e 上的权重，例如取它们的平均值，最

小值或最大值。

但是我们采用了多目标优化算法，原因如下：

寻找公共边权的重要合并准则是：①大量 True Positive （真阳，TP）即被模型预测为正的正样本；②少量的 False Positive （假阳 FP），即被模型预测为正的负样本。

加权和法是最适合且经典的算法在解决我们赋予所有目标权重的问题时。加权和方法适用于每个目标的标量权重α_i，使所有权重和为 1。然后每个目标与其相应的权重相乘，并且将所有的目标加起来形成一个单一的目标函数，该目标函数将被优化。

图形合并问题有两个目标函数：一个是最大化真阳值（TP）$F1$，另一个用于最小化假阳值（FP）$F2$。加权算法需要考虑两个目标的权重。考虑一个边e有权重 ew_1（来自图GG ），ew_2（来自GD ），ew_3（来自GK ），和ew_n（来自GN） 。
分配给函数$F1$的权重为α_1。

α_1 =所有恶意图中任意边e的边权之和/(所有恶意图中任意边e的边权之和+正常图中边权)。

同样的分配给$F2$的权重α_2，$\alpha_2 = 1 - \alpha_1$。我们将完整的边集划分为公共边集$Common\ edge - set$和不相交边集合$Disjoint\ edge - set$。

我们需要做如下三个步骤：

① 将完备边集划分为在两个及两个以上图中都出现过该边的共用边集和只在一个图中出现过该边的不相交边集；

② 将不相交边集的所有边都放入合并后的最终恶意软件图中；

③ 将共用边集中的边的几个权重值与正常软件图中的权重进行比较，选用一个权重值作为该边在最终恶意软件图中的权重值。

权重值选择过程如下：

① 如果这些边的权重值的最小值大于正常软件图中这条边的权重值，则将拥有最小权重值的这条边放入最终图；

② 如果这些边的权重值的最大值小于正常软件图中这条边的权重值，则将拥有最大权重值的这条边放入最终图；

③ 如果这些边的权重值的有的大于正常软件图中这条边的权重值，有的小于正常软件图中这条边的权重值则需要计算出一个最优值。

计算最优值的方法：

$$Maximize\ Z = a_1(m_i - w_n)(w_1 + w_2 + w_3) + a_2(w_n - m_i)(w_n)$$

应满足条件$\{a_1 + a_2 = 1,\ m_i < \max(w_1,\ w_2,\ w_3)\ and\ m_i > (w_1,\ w_2,\ w_3)\}$，最终选取一个能使 Z 最大的$m_i$。

伪代码如下：

算法 2　图形合并算法

```
1:  输入: 图构造算法构造的恶意软件图 G_G(V_g,E_g),G_D(V_d,E_d),G_K(V_k,E_k)
2:  输出: 最终的恶意软件图 G_M(V_m,E_m)
3:  将三个图中的所有边集划分为两个集合: Common edge-set 和 Disjoint edge-set
4:  for 每条边 ei ∈ Disjoint edge-set do
5:      将( ei, W(ei) )加入到 E_m
6:  end for
7:  for 每条边 ej ∈ Common edge-set do
8:      把在 G_G(V_g,E_g),G_D(V_d,E_d),G_K(V_k,E_k) G_N(V_n,E_n) 中 ej 的权重分别赋值给 w1,w2,w3,wn
9:      if  minimum{w1,w2,w3}>wn then
10:         W( ej ) = minimum{w1,w2,w3}
11:         将边( ej, W(ej) )加入到 E_m
12:     else if  maximum{w1,w2,w3} < wn then
13:         W( ej ) = maximum{w1,w2,w3}
14:         将边( ej, W(ej) )加入到 E_m
15:     else
16:         设 mj 为最终图 G_M 中边 ej 的权重
17:         求 mj 满足以下条件:
            Maximize Z=α1(mj-wn)(w1+w2+w3)+α2(wn- mj)(wn)
            其中 α1+α2 = 1 and mj <= max(w1, w2, w3) and mj >= min(w1, w2, w3)
18:         W (ej) = mj
19:         将边( ej, W(ej) )加入到 E_m
20:     end if
21: end for
22: 返回图 G_M(V_m,E_m)
```

图 2-7　图形合并算法伪代码

为何使用加权和算法：

我们应用加权和法求公共边的权，设 m_i 是 G_M 中公共边 e_i 的权重。

$F_1 = (m_i - w_n)(w_1 + w_2 + w_3),\ F_2 = (w_n - m_i)w_n$;

当 G_M 中任意权限对的权重大于 G_N 时，样本被归为恶意软件的可能性增加。如果 m_i 大于 w_n，那么样本 $(w_1 + w_2 + w_3)$ 被归类为恶意软件的概率增加 $(m_i - w_n)$。如果 w_n 大于 m_i，样品被归类为正常值的概率增加$(w_n - m_i)$。因此，我们选择 m_i 使得 F_1 和 F_2 都是最优的。任何单一的最小二乘法都不能使两个目标达到最优。因此，采用加权和法求解，将其简化为单目标优化。

在 `permission_mergy.py` 模块中定义 `GraphMerge()` 函数，其中核心代码为，

```
for edge,value in commonEdge.items():
    #权重选择
    if(edge in normalDict):
        if(value['min'] > normalDict[edge]):
            finalEval[edge] = min
        elif(value['max'] < normalDict[edge]):
            finalEval[edge] = max
        else:
            #计算线性规划,调用numpy包
            a1 = (value['max'] + value['mid'] + value['min'])/(value['max'] + value['mid'] + value['min'] + normalDict[edge])
            a2 = 1 - a1
            z = np.array([(1)*(a1*(value['max'] + value['mid'] + value['min']) - a2*normalDict[edge])])
            mi_bound = (value['min'],value['max'])
            #线性规划
            res = optimize.linprog(z,bounds=mi_bound)
            finalEval[edge] = res.x[0]
    else:
        finalEval[edge] = min
```

图 2-8 GraphMerge()函数部分代码

3. 根据检测模型进行风险评估

恶意应用程序检测算法:判断应用程序是否恶意的过程。

技术原理：对于每个测试应用程序，使用图形构造算法提取它的权限对并得到权限对图。该过程可以计算待测应用程序两个得分：正常得分和恶意得分。这些分数可以通过在生成的正常图和恶意图中搜索测试应用程序的每一个权限对并添加来计算相应的边权。如果恶意得分大于正常得分，则该应用程序很可能是恶意的。这种技术是基于这样一个事实，即在恶意图中权限对的权重越大，任何包含该权限对的应用程序被恶意攻击的可能性就越大。

算法 3　检测算法

```
1:  输入: 需要测试的应用集 (A1, A2,...An)
2:  输出: 每个应用是恶意的还是正常的
3:  for each Ai do
4:      使用 apktool 提取清单文件 M
5:      设 P=(P1,P2,P3...Pm)是 M 中权限的集合
6:      If (|P|==0 or |P|==l) then
7:          Return
8:      else
9:          Malscore=0;Normscore=0
10:         for 每一个权限对 (Pi,Pj) do
11:             Malscore + = W(Pi,Pj)∈GM
12:             Normscore + = W(Pi,Pj)∈GN
13:         end for
14:         if Maliciousscore > Normalscore then
15:             Return Ai 是恶意软件
16:         else
17:             Return Ai 是正常软件
18:         end if
19:     end if
20: end for
```

图 2-9 检测算法伪代码

4. 优化权限对图模型

为了优化我们的权限对图模型，我们需要进行边消除的工作。

边缘消除：图G_M和G_N，可能包含几个不有助于检测的边，例如在G_M和G_N中具有相似权重的边缘。我们认为这些边缘不能改变检测结果。考虑一个边E_1，它在这两个图中都有 0.1 的权重。从两个图中删除 E_1 可能不会改变准确性，因为使用 E_1 对恶意分数和正常分数的贡献相等。如果去除这些边，真阳性(TPR)和假阳性(TNR)不降低，则可以删除上述的边。我们称这个过程为边消除。边缘消除包括以下两个部分：无关边识别和寻找最大无关边集。

无关边识别：对于G_M和G_N的联合中存在的每一条唯一边，我们每次从两个图中删除一条共有或不相交的边，并应用算法 3 检验检测的准确性。除非两者都有 TPR 和 TNR 没有减少，那么边缘被认为是不适当的，并且它被添加到$E_{reduced}$。假设一个无关边分别在G_M和G_N中有权，那么它与这两个图的权差为$|\lambda - \mu|$。我们将所有这些不相关边的权差和(SWD)定义为 β 。我们将所有这些权重差异插入到一个单独的列表中，列表上写着 E_{list} ，按照递增的顺序排序。现在，我们需要找到最大可能边集，这是 E_{list} 的子集，我们可以从图中删除不降低 TPR 或 TNR 的边。

```
for edge,value in list(evaldic.items())[lowlimit:uplimit]:
    count = count + 1
    #因为是无向图，相邻的两个元素属于一条边
    if(count%2==0):
        continue
    evaltest = copy.copy(evaldic)
    TestEdgeValue = evaltest.pop(edge)
    TestTPR = evalDetection(evaltest,normaldic)
    TestTNR = NormalDetection(evaltest,normaldic)
    if(TestTPR >= TPR and TestTNR >= TNR):
        Ereduced.append(edge)
        if(edge in normaldic):
            #如果在正常图中存在
            Elist[edge] = math.fabs(normaldic[edge] - TestEdgeValue)
        else:
            Elist[edge] = TestEdgeValue
    else:
        continue
```

寻找最大无关边集：我们的目标是寻找图中可以消除的最大无关边集。首先，我们删除所有的不相关边，也就是说，我们删除的 SWD 为 β 的边缘并检查检测率。如果 TPR 和 TNR 都没有减少，那么我们消除所有的不相关边，算法结束。然而，如果它导致 TPR 或 TNR 的降低，那么我们需要找到最大可能的边集，它是E_{list}，我们可以从图表中删除。

```
normaltest = copy.copy(normaldic)
evaltest = copy.copy(evaldic)
weight_sum1=round(0,precision)
del_edge=[]
#用来得到本轮需要删除的边
for edge in Elist:
    if (weight_sum1>up_limit1):
        break
    else:
        weight_sum1+=Elist[edge]
        weight_sum1=round(weight_sum1,precision)
        del_edge.append(edge)
#删除这些边
for edge in del_edge:
    if edge in normaldic.keys():
        normaltest.pop(edge)
    if edge in evaldic.keys():
        evaltest.pop(edge)
```

我们应用一种类似于**二分法**的技术来寻找最大可能子集。设置一个下限，用δ_1表示，将其初始化为 0，同时设置一个上限，用δ_2表示，等于 β 。我们的目标是找到权重总和的最大值，在δ_1和δ_2之间，这样 TPR 和 TNR 保持相同或更高并删除所有的边，其权重差加起来达到最大。最初，我们将 δ_{max} 设置为 2。如果从图中删除集 E_{list} 的所有边，无论是 TPR 还是 TNR，我们都减少了上限的一半，即$\delta_2 = \frac{\beta}{2}$ ，而下限δ_1仍然为 0。当δ_1和δ_2都收敛到同一个值(我们称之为 δ_{max})时，过程终止。

```
    #计算删除后的TPR和TNR
    TestTPR = evalDetection(evaltest,normaltest)
    TestTNR = NormalDetection(evaltest,normaltest)
    if(TestTPR >= TPR and TestTNR >= TNR):
        #如果TPR和TNR都没有下降并且为第一轮，则直接返回，说明可以删除所有无关边
        if(count==0):
            up_limit_final=up_limit1
            return
        #如果不是第一轮，由于都没有下降，因此下限变为本轮传入的上限，上限变为上一轮的上限，即二倍
        else:
            low_limit2=round(up_limit1,precision)
            up_limit2=round(2*up_limit1,precision)
    #如果下降了，那么下限依旧不变，上限变为之前的一半
    else:
        low_limit2=round(low_limit1,precision)
        up_limit2=round((float)(up_limit1)/2,precision)
count += 1
Deletedge(low_limit2,up_limit2)
```

我们删除图中的边，这些边 SWD 总和为δ_2，并测试检测结果。可能出现两种情况：

如果检出率没有下降，那么δ_2就是新的下限，以前的δ_2就成为新的上限；如果检出率下降，那么零仍然是下限，$\frac{\beta}{4}$是新的上限。上面的步骤重复直到$\delta_1 = \delta_2$ ，我们把这个点命名为 δ_{max}。最后，我们从G_M和G_N中去除了这些权重差最大的边集合，即完成权限对图的优化。

2.2.2 敏感 API 提取

2.2.2.1 敏感API的选取

Android 应用程序通过某些函数接口调用可能会访问并获取用户的一些敏感数据，或者做出一些敏感行为，这样的行为会对用户的隐私和安全造成威胁，这类函数接口调用被称为敏感 API。

Android 应用程序的开发者在使用某些 API 时需要在应用的程序清单 `AndroidManifest.xml` 中申请所需要的权限，例如当开发者使用发送短信的敏感 API `android.telephony.SmsManager.sendTextMessage()` 时需要申请对应的敏感权限 `android.permission.SEND_SMS`。所以敏感权限与敏感 API 是紧密相关的。PScout[8]给出了 Android 系统中权限与 API 的对应关系，在 Android 官方文档中给出了 9 类 24 种危险权限，通过查找映射关系可以找到与之对应的 95 种敏感 API。

除了与敏感权限相关的敏感 API 外，我们还搜集了 48 种恶意软件中经常使用的 API，去除两种敏感 API 中重复的部分后剩余 138 种敏感 API。提取 API 并进行数据分析后发现，某些 API 在实际使用中的使用率较低，并且一些 API 的区分度不高，在去除这类 API 后最终保留了 46 种 API 作为敏感 API。

2.2.2.2 API的提取

在特征向量的提取过程中使用了 androguard 工具和 apktool 工具。当 apk 文件小于 1MB 时，使用 androguard 可以更快的提取出 API。而当 apk 文件大于 1MB 时，使用 apktool 可以更快的提取出 API。所以我们在对 apk 进行 API 提取时，使用了两种工具相结合的方法。

在使用 androguard 时主要使用 `get_methods()`方法对 API 进行提取。在使用 apktool 时首先对 apk 文件进行反编译得到 apk 的各类基本信息以及 `smali` 中间代码，然后在 `smali` 文件夹下遍历所有反编译出的 `smali` 字节码文件，对每个 `smali` 文件都逐行进行正则匹配，如果包含 API 信息，则写入与 apk 同名的 txt 文件。在提取的过程中需要注意的是去除重复信息，这样可以达到加快提取速度，降低数据冗余量的目的。

之后将提取得到的 API 规定好的 46 个敏感 API 相比对，如果该应用使用了某个敏感 API，则相应的特征属性位置用 1 表示，否则用 0 表示。之后将得到的数据整理为逗号分隔值文件，用于机器学习。

具体流程如图 2-10：

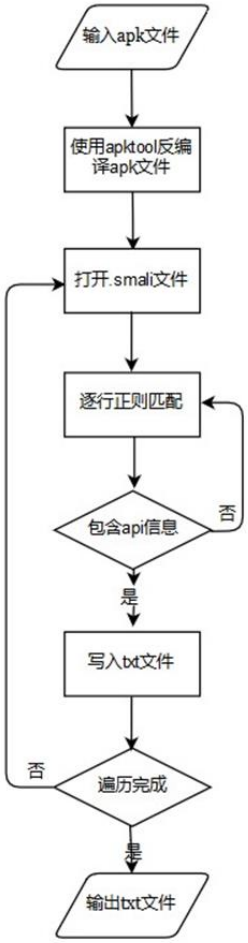

图 2-10 API 的提取流程

2.2.3 随机森林算法

2.2.3.1 随机森林算法的原理

在机器学习中，随机森林(Random Forest)是一个包含多棵决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。随机森林算法由 Leo Breiman 和 Adele Cutler[10]提出，而 "Random Forests" 是他们的商标。 这个术语是 1995 年由贝尔实验室的 Tin Kam Ho[11]所提出的随机决策森林（random decision forests）而来的。这个方法则是结合 Breimans 的 "Bootstrap aggregating" 想法和 Ho 的"random subspace method" 以建造决策树的集合。

决策树是一种基本的分类器，一般是将特征分为两类。构建好的决策树呈树形结构，可以认为是if-then规则的集合。具体来就是从数据的n维特征中无放回的随机抽取mtry维特征，在mtry维特征中，通过计算信息增益的方式找到分类效果最好的一维特征k，及其阈值 θ，小于 θ 的样本划分到左节点，其余的划分到右节点，继续训练其他

节点。重复训练所有的数据样本，得到n棵决策树。

2.2.3.2 算法流程

在随机森林算法中，建造每棵树的流程包含以下几步。如图 2-11 所示。

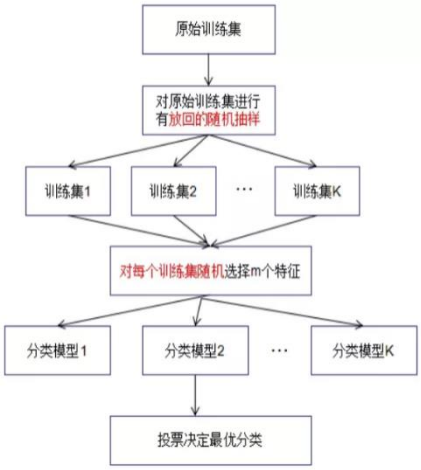

图2-11 随机森林算法图示

1) 用 N 来表示训练例子的个数，M 表示变量的数目；

2) 我们会被告知一个数 m ，被用来决定一个节点在做决定时，会使用到多少个变量(m 应小于 M)；

3) 从 N 个训练案例中以可重复取样的方式，取样 N 次，形成一组训练集（即 bootstrap 取样）。并使用这棵树来对剩余预测其类别，并评估其误差；

4) 对于每一个节点，随机选择 m 个基于此点上的变量。根据这 m 个变量，计算其最佳的分割方式。

5) 每棵树都会完整成长而不会剪枝（Pruning）（这有可能在建完一棵正常树状分类器后会被采用）。

随机森林便是所有决策树集合起来的森林，通过对每个子树的输出结果打分，进而评估每个特征的重要程度。

2.2.3.3 算法优点

随机森林具有很多优点，如下：

（1）在数据集上表现良好，训练和预测速度快，适合做分类问题。

（2）容错能力强，应对训缺失值的有效方法，大数据集存在大比例缺失值时，可以有效处理且保持精度不变。

（3）高维度数据上表现良好，可不做特征选择，数据集无需规范化，可处理离散

型和连续型数据。

（4）分类过程中可生成泛化误差的内部无偏估计。

（5）训练过称中可检测特征之间的相互影响及特征的重要程度（基于 OOB 误分率的增加量和基于分裂时的 GINI 下降量）。

（6）有效避免过拟合，有效抗噪（随机性引入）。

（7）实现简单，易实现并行化。

2.2.3.4 算法在作品中的应用

将 APK 文件的 api 调用信息作为特征值的输入，进行模型的建立与验证。首先将原始数据集分割为训练集与测试集：利用训练集填充到已有的随机森林框架中，得到训练好的随机森林模型；然后利用测试集代入模型，计算此模型的准确率。之后，将模型封装保存，再次使用时直接调用即可。

2.3 系统实现

2.3.1 用户 APK 文件上传过程实现

具体流程如下：

1）用户必须登录，获取生成的唯一`session`；

2）用户提交上传APK文件的请求：

用户通过PC端或者移动端向云服务器发送上传APK文件请求；

3）服务器验证用户身份：

在提交的请求中，服务器验证用户`sessionid`是否在数据库中对应着已注册用户；

4）使用用户`session`，上传APK文件：

服务器验证通过后，将`Form`表单中的APK文件上传至服务器，并生成唯一的`UUID`，并将生成文件的PATH写入`session`，用于判断是否重复上传，然后调用相应的检测模块对文件进行检测。

5）返回用户是否上传成功的结果：

服务器接收到提交的文件后，首先检查后缀，再检查文件头及文件完整性，若为完整APK文件，则发送上传成功的状态反馈，否则，返回上传失败的状态反馈。

6）服务器返回APK文件检测结果：

云服务器CS调用静态权限对检测模块对上传的文件进行检测，然后将检测结果封装在JSON数据里返回到结果页面，浏览器将HTML模板及JSON数据渲染成最终的检测结果；

7）用户提交机器学习深度检测请求：

用户如果对于静态检测的结果不满意，那么可以选择异步提交深度检测请求，使用机器学习方法，对于APK文件进行反编译，提取API，进行深度检测；

8）服务器返回APK文件深度检测请求：

服务器在进行机器学习检测之后，对APK恶意程度进行评估，并将结果、提取API、反编译文件列表封装成JSON数据返回到原检测结果页面。

9）服务器将上传的APK文件删除：

为减轻服务器压力，我们选择在进行检测之后，对用户上传的APK文件进行删除操作。

以上全部信息交互在HTTPS协议连接下传输，提供数据在传输过程中的机密性、可靠性、不可监听和篡改性。

2.3.2 网站应用开发层界面实现

网站应用界面主要是为用户提供可视化显示，方便用户了解平台的功能，进行注册、登录、使用。前端使用 `HTML5+Bootstrap+VUE+axios` 实现，后端使用 `Python Django` 搭建。

2.3.2.1 网站主页面的实现

网站主页需要实现介绍本 Android 恶意软件检测器的特色、功能，吸引用户，为用户提供注册/登录入口，联系团队等功能。因此，我们使用 `HTML5+Bootstrap` 实现单页面响应式布局，将一个页面分割成五部分：简介，特色，上传，团队简介，项目工作，产品预览，性能对比，联系方式，并提供注册登录的入口。

简介板块：用简短的语言介绍平台功能，并放置了一个“了解更多”的超链接按钮，点击之后移动到特色与上传板块。

特色板块：介绍了平台的四大优势：基于权限图的静态检测，结果精确，急速检测，可视结果。

上传板块：提供上传文件的功能区。

团队成员板块：介绍团队的各位成员，及各自的头像、座右铭与联系方式。

项目工作板块：展示了团队为项目付出的工作量。

对比板块：对于我们平台性能与其他主流在线文件检测平台进行的性能对比，突出优势。

联系板块：提供提交问题的表单区域，来获取用户遇到的问题和需求。

2.3.2.2 登录/注册页面的实现

如果需要获得进一步的服务，那么需要用户进行登录才能获取，因此需要实现平台的登录/注册功能及可视化页面。

注册页面：注册页面由左侧图片动画与右侧表单区域组成。左侧动画是为了提供更好的用户体验，右侧的区域由三个输入控件及两个按钮组成。这三个输入框为用户名，密码和再次输入密码，两个按钮为去登陆页面的超链接和提交的超链接。当完成所有输入之后点击提交按钮则会对注册请求进行操作，如果成功则自动跳转到登录页面，否则返回注册失败的原因，要求用户重新注册。

登录界面：登录界面由两个输入组件、一个登录按钮和一个超链接组成。输入框为输入用户名和密码，按钮用于确认信息填写完成后进行登录，另一个超链接为去注册页面完成注册新用户操作。

2.3.2.3 文件检测结果页面的实现

文件检测结果页面是用户完成上传操作，并且后台完成文件静态检测后显示的页面。它的功能是向用户直观的展示软件检测结果，提供详细的文件属性信息，并用中文解释属性信息的意义，便于普通用户对软件有更全面的理解。它由文件基本信息、危险权限对、权限对拓扑图、权限及中文、调用 API 信息、反编译文件目录几部分组成。

文件基本信息部分：列出 APK 文件的名称、SHA256、SHA1、MD5、文件上传时间、文件大小，以及评判结果与得分，在右侧提供了深度检测的按钮，如果用户希望用机器学习对文件进行进一步检测，点击它，将会提交异步请求，用户等待 30 秒

至 2 分钟，将得到机器学习检测结果，这期间页面并不会刷新。

危险权限对部分：给出 APK 文件得分前 20 的权限对名称与相应权值，以表格的形式展示。

权限对拓扑图部分：给出权限对之间直观的拓扑关系，以关系图的方式展示。

权限及中文部分：列出文件所申请的全部权限以及对应的中文，让用户对文件有着更轻松直观的认识与理解。

调用 API 信息部分：此部分及之后是点击深度检测按钮之后提供的信息，包括软件调用的 API 及相关解释，与包/类从属关系。

反编译文件部分：此部分给出了将 APK 文件反编译后的文件名称列表。

第三章 作品测试与分析

为了验证我们设计并实现的基于权限对图和广义敏感 API 的安卓恶意软件多特征联合检测系统的有效性和正确性，本章选取测试样本、设计测试方案对该系统进行实验测试，并对测试结果进行分析。

3.1 测试环境

作品测试的环境如下表 3-1 所示。

表 3-1 测试环境配置

实体	环境	配置
主机	Windows 10 x64 Chrome 版本 81.0.4044.138	Intel Core i5-6300HQ CPU 内存 SODIMM 8G 硬盘 HDD 1T
Web 服务器	Aliyun Cloud Server CentOS 7.7 x64	Ecs.t5-c1m2.large 2 Core 4G 1Mbps 硬盘 40G
后端服务器	Aliyun Cloud Server CentOS 7.7 x64	Ecs.t5-c1m2.large 2 Core 4G 1Mbps 硬盘 40G

3.2 测试方案

本系统基于 B/S 架构，实现了一个多用户注册并可管理的安卓恶意软件快速检测平台，提供可视化报告，并且保证用户数据安全。以下为结合前后端功能采取的测试方案。

3.2.1 系统的性能测试

评测原则

我们将 APK 样本提交到不同的平台进行检测，其中平台为：本产品，微步在线云沙箱，腾讯哈勃分析系统。对于检测评估，我们使用所用时间和检测率作为主要评测依据，在检测率尽可能高的情况下，耗时越少越好。

测试样本

我们采用的 APK 样本相关参数为：

基本信息	
文件名称	0630fc8424edc0e5f3e8cf...74072d44a7f9a3111b5bd.apk
文件大小	11.02MB
最低运行环境	Android 4.2, 4.2.2
SHA256	0630fc8424edc0e5f3e8cfb875276b5a5cba6e14e2374072d44a7f9a3111b5bd
包名	com.mobilebt.snakefight
是否恶意	是

3.2.2 交互功能测试

这部分测试包括如下四个方面：

1. Web 服务器界面展示是否正确。
2. 用户注册登录功能是否正常。
3. 后台数据库展示。
4. 用户上传功能是否正常。

3.2.3 扫描与检测功能测试

这部分测试包括如下两个方面：

1. 检测功能是否正常。
2. 个人中心功能是否正常。

3.3 测试结果及分析

3.3.1 性能测试结果及分析

本节将我们的作品、腾讯哈勃分析系统和微步在线云沙箱的性能进行对比测试，比较四个方面，分别为：对 APK 样本的检测，不同恶意样本量下的正确检测率，不同正常样本量下的正确检测率，不同样本量下的检测效率。

1. 对 APK 样本的检测

将 APK 样本提交到不同的平台进行检测，得到的结果如下表 3-2 所示。测试结果表明，在三个平台中，本作品 AnDetector 的测试速度最快，而且检测结果正确。

表 3-2　APK 样本的测试结果

检测平台	检测时间	检测结果
腾讯哈勃分析系统	1 分 5 秒	恶意软件
微步在线云沙箱	2 分 25 秒	恶意软件
本产品 AnDetector	6 秒	恶意软件

2. 不同恶意样本量下的正确检测率

针对不同的恶意样本量，我们进行了线下批量测试，得到的测试结果如下图 3-1。从图中可以看到，在恶意样本量分别为 300 个，450 个和 600 个的情况下，本作品 AnDetector 的正确检测率都超过了 98%，高于另外两个平台：微步在线云沙箱和腾讯哈勃分析系统。

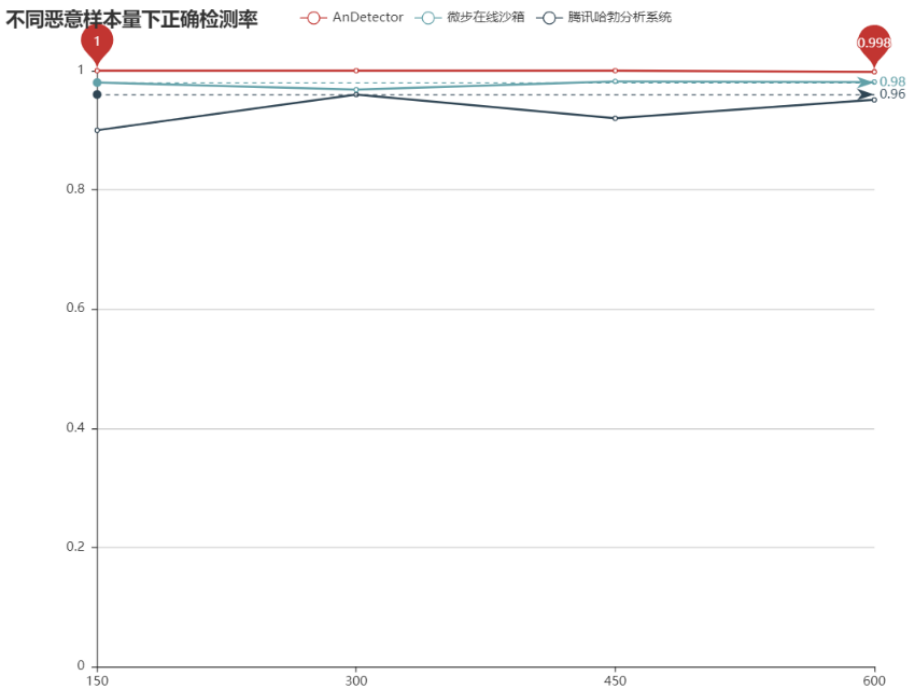

图 3-1 不同恶意样本量的正确检测率

3. 不同正常样本量下的正确检测率

针对不同的正常样本量，我们进行了线下批量测试，得到的测试结果如下图 3-2。从图中可以看到，在正常样本量分别为 500 个，1000 个，1500 个和 2000 个的情况下，本作品 AnDetector 的正确检测率都超过了 95%，略低于另外两个平台：微步在线云沙箱和腾讯哈勃分析系统。原因分析如下：目前正常软件中申请的权限过多，影响了检

测系统的评定，造成了少量误判。

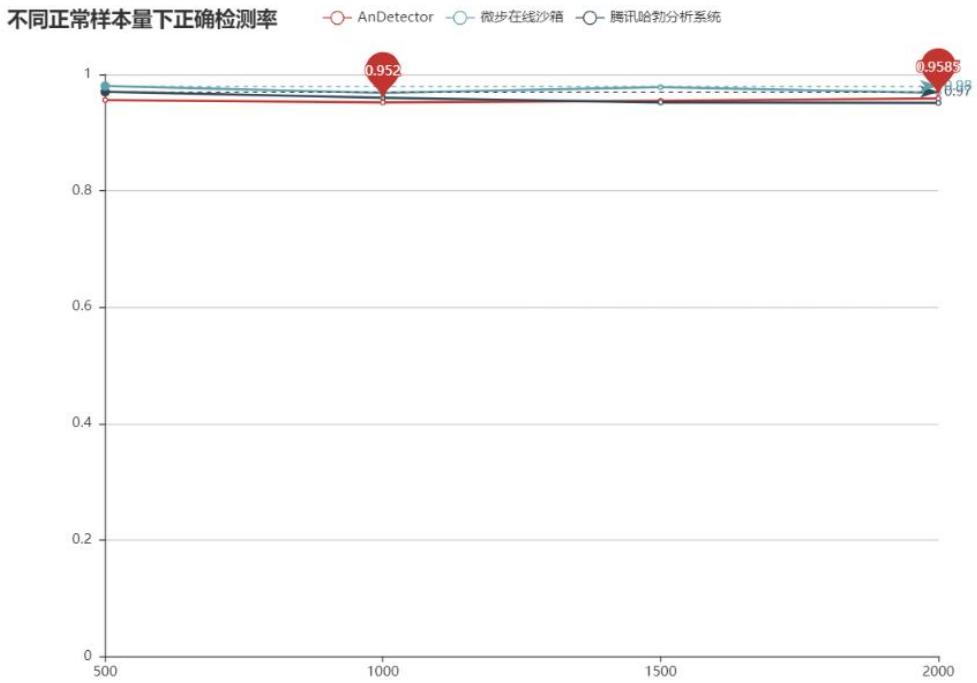

图 3-2 不同正常样本量的正确检测率

4. 不同样本量下的检测效率

针对不同的正常样本量，我们进行了线下批量测试，得到的测试结果如下图 3-3，在该图中，纵轴为样本数量，横轴为运行时间，单位为秒。

从图中可以看到，在正常样本量分别为 1 个、150 个、300 个、450 个和 600 个的情况下，本作品 AnDetector 的测试时间都明显小于另外两个平台：微步在线云沙箱和腾讯哈勃分析系统。

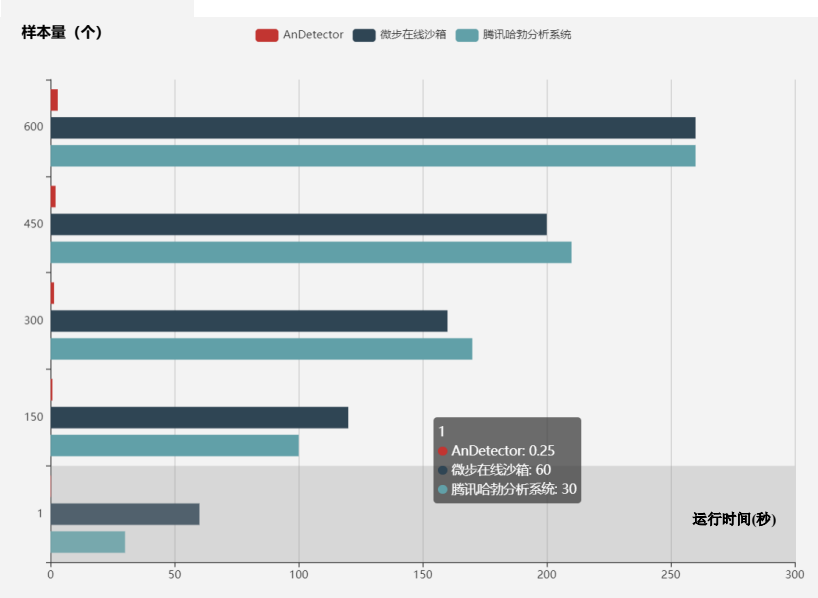

图 3-3　不同样本量下的检测时间

3.3.2 交互功能测试结果

1. 用户通过浏览器进入网址 https://ciscn.hackerjerry.top/，登录到平台首页，见图 3-4。：

图 3-4 平台首页

2. 用户注册界面，如图 3-5 所示。

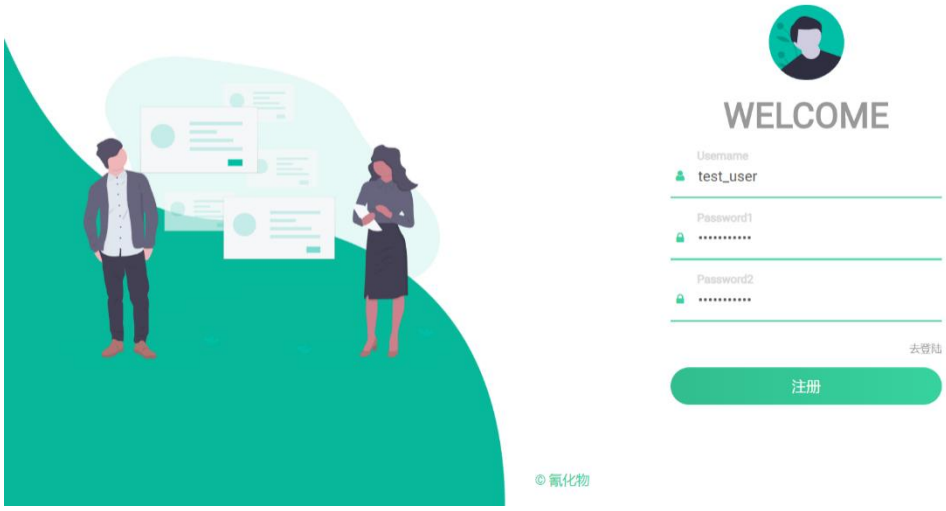

图 3-5 用户注册

注册后后台数据显示，如图 3-6 所示。：

图 3-6 后台数据库显示

2. 用户登录界面，如图 3-7 所示。

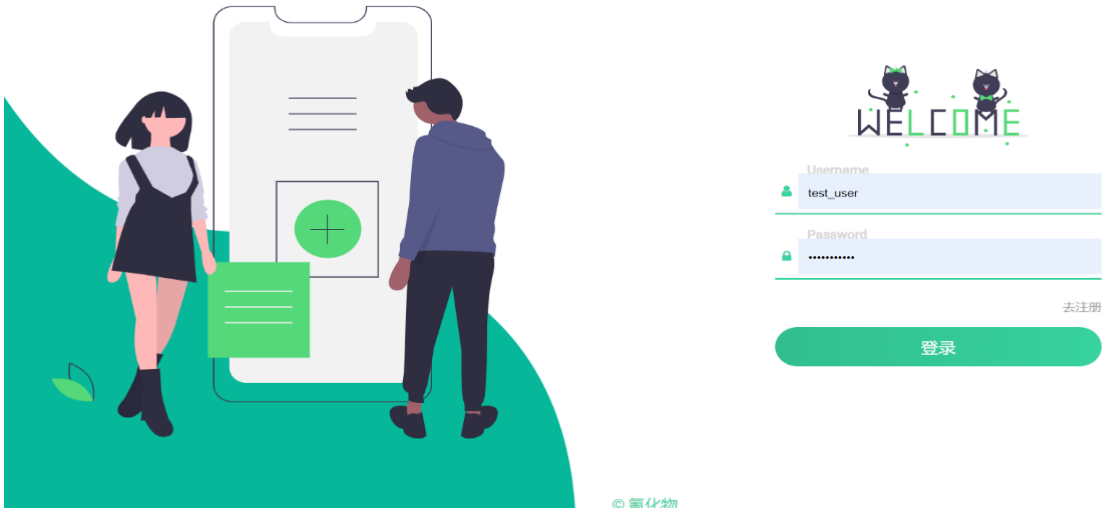

图 3-7 用户登录

3.3.3 扫描与检测功能测试结果

1. 文件上传界面，见图 3-8：

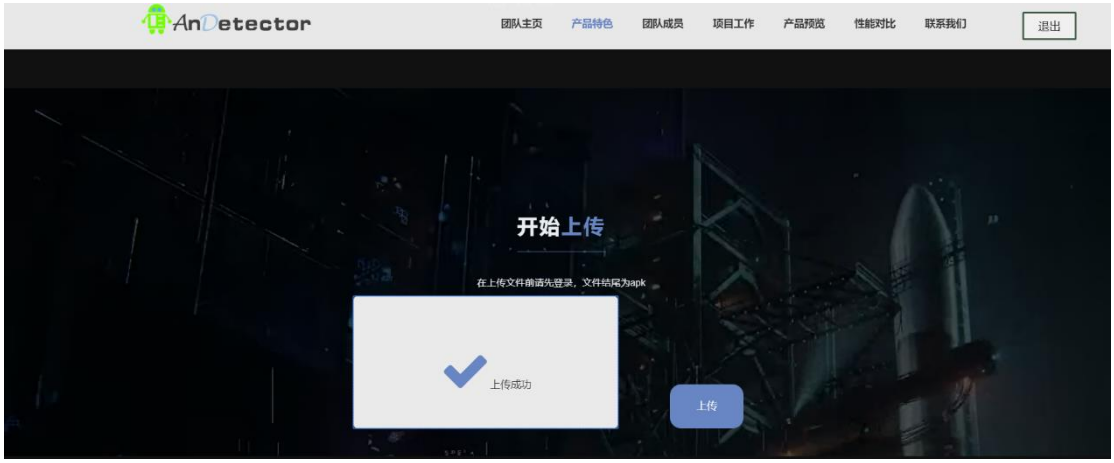

图 3-8 文件上传界面

2. 用户上传一个恶意软件，返回结果，见图 3-9：

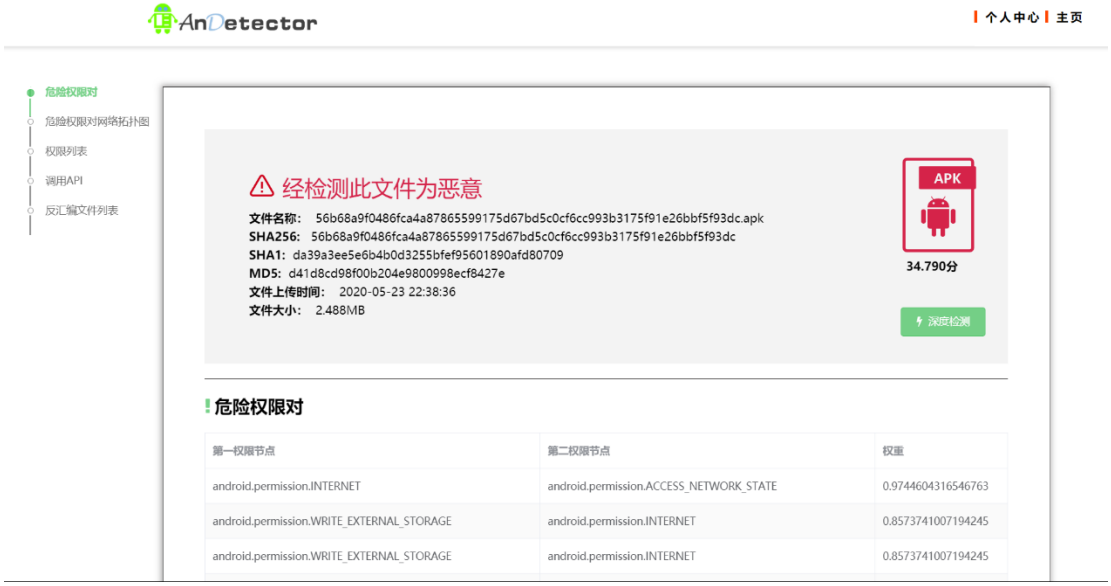

图 3-9 用户上传恶意软件的结果图

此时，查看历史信息显示为图 3-10：

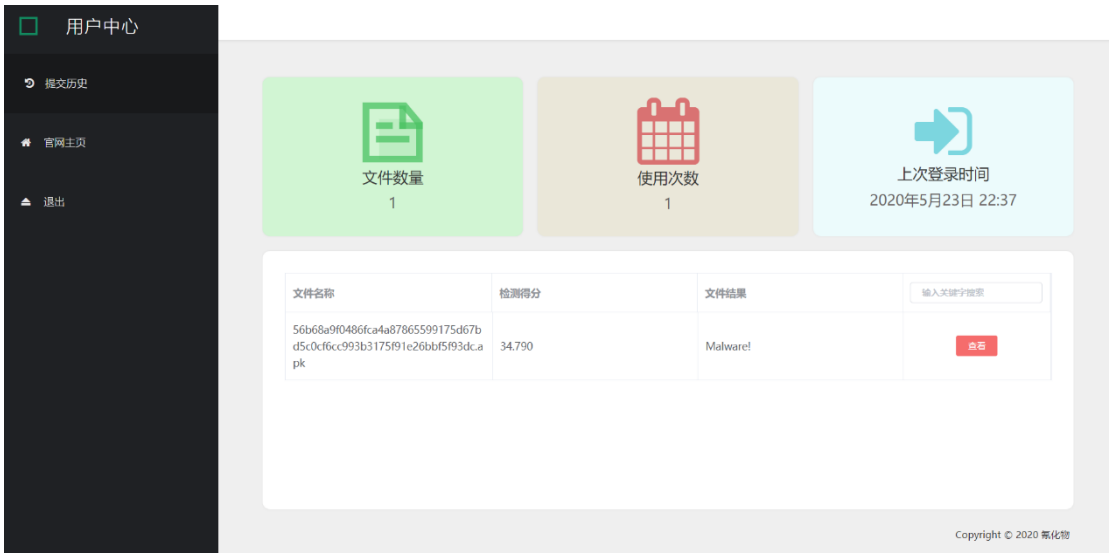

图 3-10

3. 用户上传正常软件并返回检测信息，见图 3-11：

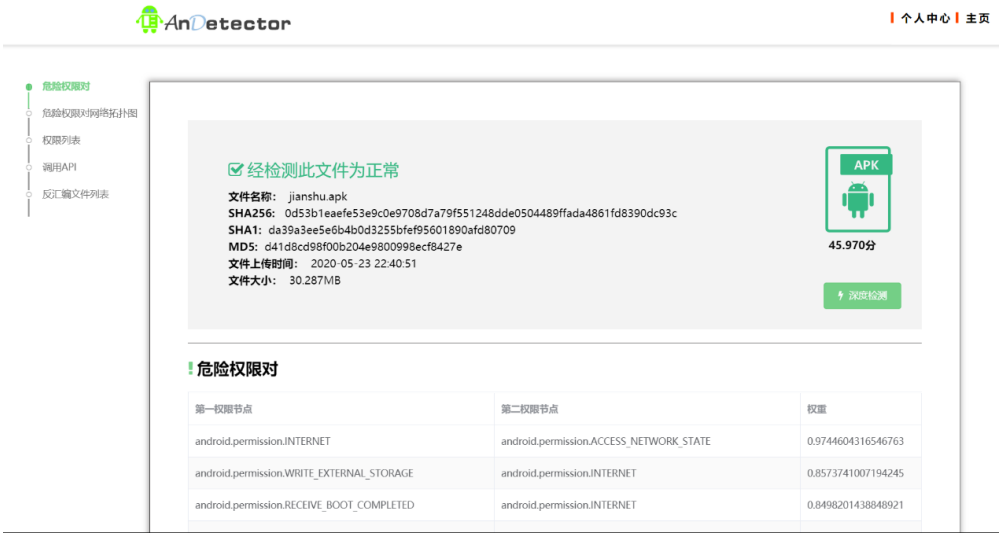

图 3-11 用户上传正常软件返回检测信息

此时，用户中心历史信息显示为图 3-12：

图 3-12

4. 至此，后台数据库记录为图 3-13：

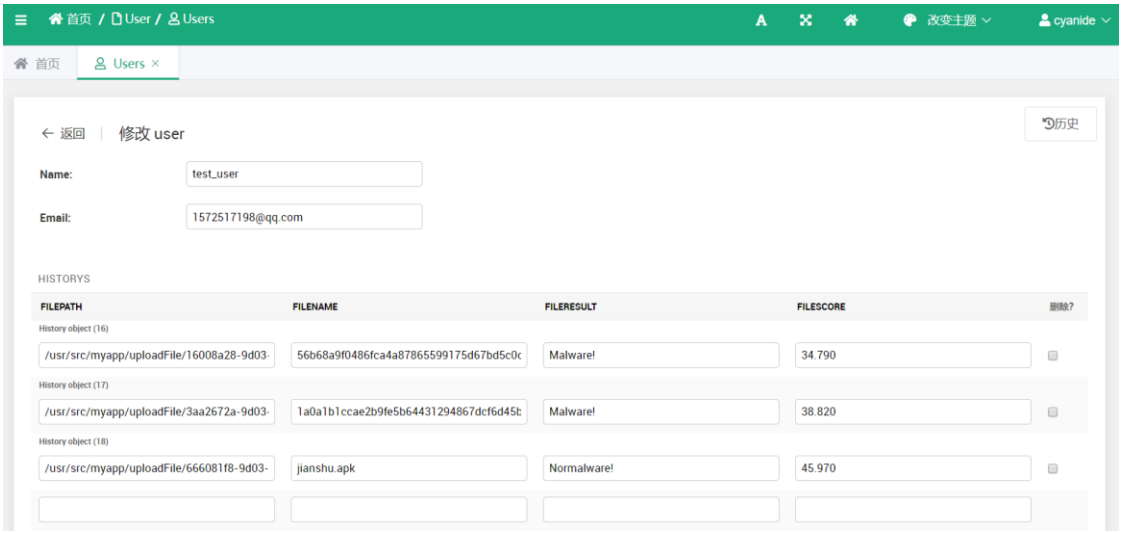

图3-13

5. 上传文件并使用深度检测功能。

图 3-14 深度检测结果

第四章　创新性说明

AnDetector--安卓恶意软件联合检测器实现了快速检测未知样本安全性的功能。我们通过分析各种应用程序特征，研究不同家族恶意软件样本的恶意功能，提取和分析恶意软件中成对出现的权限模式，将传统的特征检测与机器学习中的随机森林算法结合，提出了一种基于权限对图和广义敏感 API 的安卓恶意软件联合检测方案，并且实现了一个 B/S 架构的安卓恶意软件联合检测器。

作品的创新性如下：

一、 基于权限对图的恶意软件检测方案

本方案基于Android系统权限控制机制，提取待检测文件所申请的权限信息，利用这些权限信息生成两两对应的权限图，使用以权限对图为基准的检测，更加方便利用不同权限之间的隐含关联。通过多次优化算法将每条边计算出权重值，并由此形成权限图字典用于待测文件的检测，累加所有权限对的权重，评估文件的威胁程度，生成安全检测报告。整体流程压缩到10秒之内，快于市面大部分在线云沙箱，同时保证了很高的准确性。

二、双方案联合判定

平台除了快速静态检测，还为用户提供深度检测方案，通过反编译源文件，提取出待检测文件中的API信息并加以筛选，使用随机森林算法进行判别，可以增强对权限整体内部联系的聚类分析；在API提取过程中，增强对上下文关联语法特征与API本身语义特征的处理。此外，还利用APK文件其他特征，使用图形神经网络进行辅助检测，提高了平台的检测正确率。

三．批量化检测

本作品实现了线下批量测试的功能，用户可将待检测文件批量提交给后台，为需

求量大的用户提供更快速地检测，免去冗长的等待时间，保证了多场景下的广泛适用性。

四、多维可视化报告

AnDetector平台通过对用户上传的待检测软件进行本地化存储，并在后台调用检测模块对文件安全性进行打分以及重点权限对标记，前端采取可视化技术直观展示权限对之间的关系，对软件行为进行监控，为用户提供更详细易懂的报告，全面剖析待检测文件的情况。

综上，AnDetecor可以快速检测用户上传的待检测文件，为普通用户提供便捷的安全扫描服务。市面上的在线云沙箱基本是采取动态检测技术，运行维护成本大，并且检测周期较长，检测结果不直观，误报较多。我们的作品弥补了这些缺点。

第五章 总结

安卓系统的开放性得到了应用程序开发者的青睐，巨大的市场份额也吸引着恶意攻击者的注意，各种恶意应用不断出现。移动互联网信息安全的形势是严峻的，而且普通用户对安全防范的意识还有待提高。因此，如何对大量未知的安卓恶意软件进行有效的检测具有重要的研究价值。基于此，我们设计并开发了作品——基于权限对图和广义敏感 API 的安卓恶意软件联合检测器。

本平台基于Python Django、VUE、Bootstrap打造而成，可以实现快速检测安卓恶意软件的功能。在用户使用检测平台时，仅需要将待检测应用通过浏览器上传界面传输到服务器后端。在后端，检测模块内部存在及封装好的检测模型——此检测模型有服务器定期维护更新，以实现准确率最大化。采取双方案联合判定，权限对图快速检测配合机器学习深度检测，检测过程迅速、成功率高。并且在使用过程中减少了用户端资源的消耗，拥有着轻量级性质与便捷性，在生活中实用价值较高。最终展示给用户的多维可视化报告，直观地展示了权限对之间的关系，并且对软件行为进行监控、对文件安全性进行打分以及对重点权限对标记。

通过测试不同的数据集，检测的准确率达到98%以上，体现了该检测平台具有较高的准确性与良好的适应性。此外，本作品实现了安卓恶意软件的批量检测和分析，而且能够对安卓软件进行风险评估，具有良好的应用前景。

参考文献

[1] 360 烽火实验室，2019 年 Android 恶意软件专题报告[J/OL]，360 核心技术博客，2020-02-28.

[2] ENCK W, GILBERT P, HAN S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones[J]. ACM Transactions on Computer Systems(TOCS), 2014, 32(2): 1-29.

[3] ARZT S, RASTHOFER S, FRITZ C, ET AL. FlowDriod: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps[J]. Acm Sigplan Notices, 2014, 49(6), 259-269.

[4] Anshul Arora, Sateesh K. Peddoju, Mauro Conti，PermPair: Android Malware Detection Using Permission Pairs[J]，IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY，VOL. 15, 2020.

[5] Na Huang, Ming Xu, Ning Zheng , Tong Qiao, Kim-Kwang Raymond Choo；Deep Android Malware Classification with API-based Feature Graph[J]；IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering；2019.

[6] 王聪，张仁斌，李钢；基于关联特征的贝叶斯 Android 恶意程序检测技术[J]；计算机应用与软件；2017 vol 34.

[7] 李孔渤，王春东；基于集成学习的安卓恶意程序检测技术[J]；天津理工大学学报；2019 vol 35.

[8] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang,等. PScout:analyzing the Android permission specification[C]// Acm Conference on Computer & Communications Security. ACM, 2012.

[9] 谢念念等，多维敏感特征的Android恶意应用检测[J], 计算机科学，10.11896/j.issn.1002-137X.2019.02.015.

[10] Leo Breiman. Random Forests. Machine Learning 45 (1), 5-32, 2001.

[11] Tin Kam Ho. Random Decision Forests. In Proceedings of the Third International

Conference on Document Analysis and Recognition,Volume 1, 278. ICDAR ’95. Washington, DC, USA: IEEE Computer Society.
[12]Jang J W , Kang H , Woo J , et al. Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric informatio n[J]. Computers & Security, 2016, 58(May):125-138.