



基于 ClamAV 和 RF 的安卓恶意代码检测器
2021-2022 计算机病毒原理课程作业文档

组号：第九组

专业：信息安全

指导教师：刘功申

学院：电子信息与电气工程学院

本组成员：冯驰，姚思悦，何梓欣，韩雨虹

目录

| | |
|---------------------------------|----|
| 1 项目介绍 | 4 |
| 1.1 项目环境 | 4 |
| 1.1.1 openEuler 操作系统 | 4 |
| 1.1.2 UKUI 桌面环境 | 4 |
| 1.1.3 运行环境与部署 | 5 |
| 1.2 项目概述 | 5 |
| 2 安卓 APK 恶意代码检测 | 6 |
| 2.1 检测方法概述 | 6 |
| 2.2 开发环境和数据集 | 6 |
| 2.3 模块实现-RF 训练分类器 | 7 |
| 2.3.1 反汇编 | 7 |
| 2.3.2 字节编码映射 | 8 |
| 2.3.3 n-gram 特征提取 | 10 |
| 2.3.4 RF 随机森林训练和预测 | 10 |
| 2.4 模块实现-RF 检测器 api | 12 |
| 2.4.1 反汇编 | 12 |
| 2.4.2 字节编码映射 | 13 |
| 2.4.3 n-gram 特征提取 | 13 |
| 2.4.4 随机森林检测器 | 13 |
| 2.4.5 调用接口 | 13 |
| 3 ClamAV 的安装、修改与应用 | 14 |
| 3.1 ClamAV 安装与部署 | 14 |
| 3.2 ClamAV 源码修改 | 16 |
| 3.2.1 定位 APK 检测 | 16 |
| 3.2.2 用 c 调用 python 检测 | 18 |
| 3.2.3 cl_scandesc_callback 函数改写 | 18 |
| 3.3 修改后的 APK 文件检测效果 | 19 |

| | |
|---------------------------|-----------|
| 4 ClamTK 的安装、修改与应用 | 19 |
| 4.1 ClamTK 安装 | 19 |
| 4.2 ClamTK 前端修改 | 19 |
| 5 总结 | 21 |
| 5.1 项目成果总结 | 21 |
| 5.2 不足与改进 | 21 |
| 5.3 个人总结 | 22 |
| 6 致谢 | 25 |

1 项目介绍

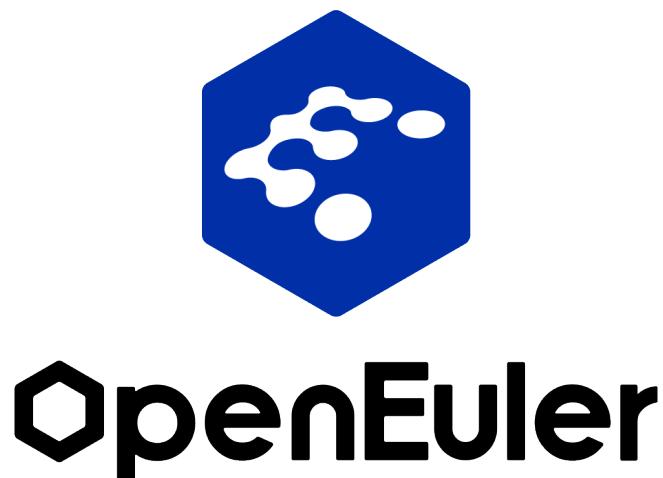
1.1 项目环境

1.1.1 openEuler 操作系统

openEuler 一般称为 EulerOS，是华为自主研发的服务器操作系统，能够满足客户从传统 IT 基础设施到云计算服务的需求，ARM64 架构提供全栈支持，打造完善的从芯片到应用的一体化生态系统。

openEuler 以 Linux 稳定系统内核为基础，支持鲲鹏处理器和容器虚拟化技术，是一个面向企业级的通用服务器架构平台。

与其他操作系统相比，openEuler 具有一定的平台优势，其在系统高性能、高可靠、高安全等方面积累了一系列的关键技术，为我们的大作业提供了一个稳定安全的基础软件平台。



1.1.2 UKUI 桌面环境

UKUI 是基于 Linux 和其他类似 Unix 发行版的可插入框架的轻量级桌面环境，使用 GTK 和 Qt 开发，可以使浏览、搜索和管理计算机提供更简单。

UKUI 具有直观易用、简洁强大、智能通用、友好实用、方便稳定等优点，同时具有丰富的应用生态，为操作系统提供更愉悦轻松的体验。

1.1.3 运行环境与部署

| | 名称 | 版本号 |
|------|-----------|---------------|
| 操作系统 | openEuler | 20.03-LTS-SP3 |
| 桌面环境 | UKUI | 2.0.2-9 |
| 杀毒软件 | ClamAV | 0.103.5 |
| 前端 | ClamTK | 6.15 |

操作系统安装：在 openEuler 官网选择合适的镜像仓和 ISO 文件，本组选择了阿里云镜像仓与 openEuler20.03-LTS-SP3.iso。在 VMware 新建虚拟机并安装该操作系统。

桌面环境安装：安装好操作系统后，在终端输入 `yum install ukui`, 一直 `y` 回车，直到显示“complete!” 表示下载成功，再输入 `yum group install fonts` 安装字体库。确认正确安装后，输入 `systemctl set-default graphical.target` 设置默认图形开机。重启之后即可看到图形界面如下图：

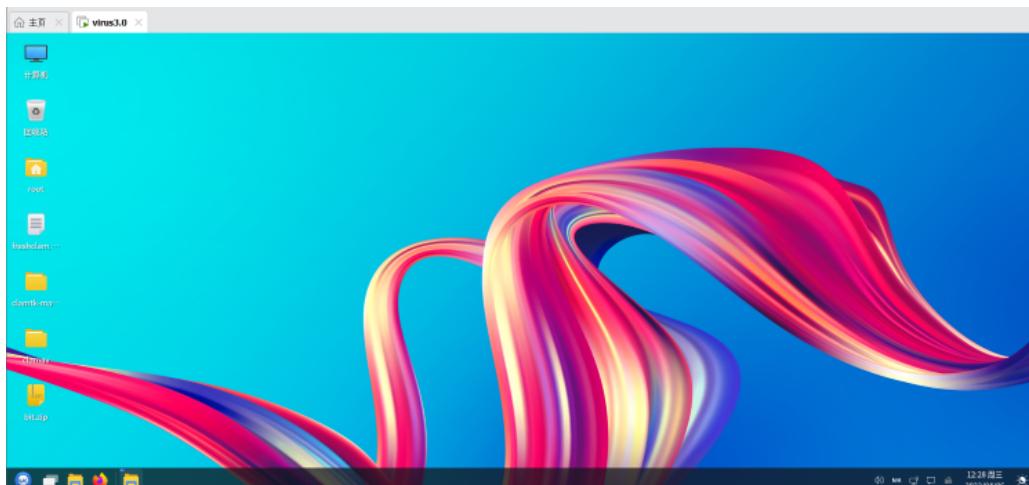


图 1: openEuler 图形界面

ClamAV 和 ClamTK 的安装部署见章节3.1和章节4.1。

1.2 项目概述

我们的项目是《基于 ClamAV 和 RF 的安卓恶意代码检测器》。本项目中我们基于 ClamAV，增加了 APK 检测功能，调用事先训练好的随机森林模型

对目标 APK 文件进行检测，同时在前端使用经过我们修改的 ClamTK 实现对 APK 文件的检测和最终判断结果的输出。

2 安卓 APK 恶意代码检测

2.1 检测方法概述

在进行了对 APK 格式恶意软件检测方法的调研后，我们选择了将软件反编译，然后提取字节编码作为特征，最后使用随机森林算法进行分类。

算法步骤如下：

- 将安卓软件反编译，提取字节编码。
- 将字节码分为 MRGITPV 七类，并进行分类映射。
- 使用滑动平均窗口提取 3-gram 特征。
- 使用随机森林训练分类器。

整体计划的检测流程框图如下：



图 2: 检测流程框图

2.2 开发环境和数据集

病毒检测的开发环境是 Windows10/MacOS，python 版本是 3.7, IDE 是 pycharm professioal 2022，在反汇编这一步选用的是基于 java 的反汇编工具 apktool。

关于训练所需要的数据集，我们选择的恶意软件数据来自 virusShare 网站，共 729 个恶意软件约 1.57G；良性软件数据来自来自 UNB2015 和 2017 (加拿大网络安全研究所)，共 686 个约 6.75G。

2.3 模块实现-RF 训练分类器

这个模块介绍使用数据集对随机森林分类器进行训练，得到一个检测效果良好的模型。

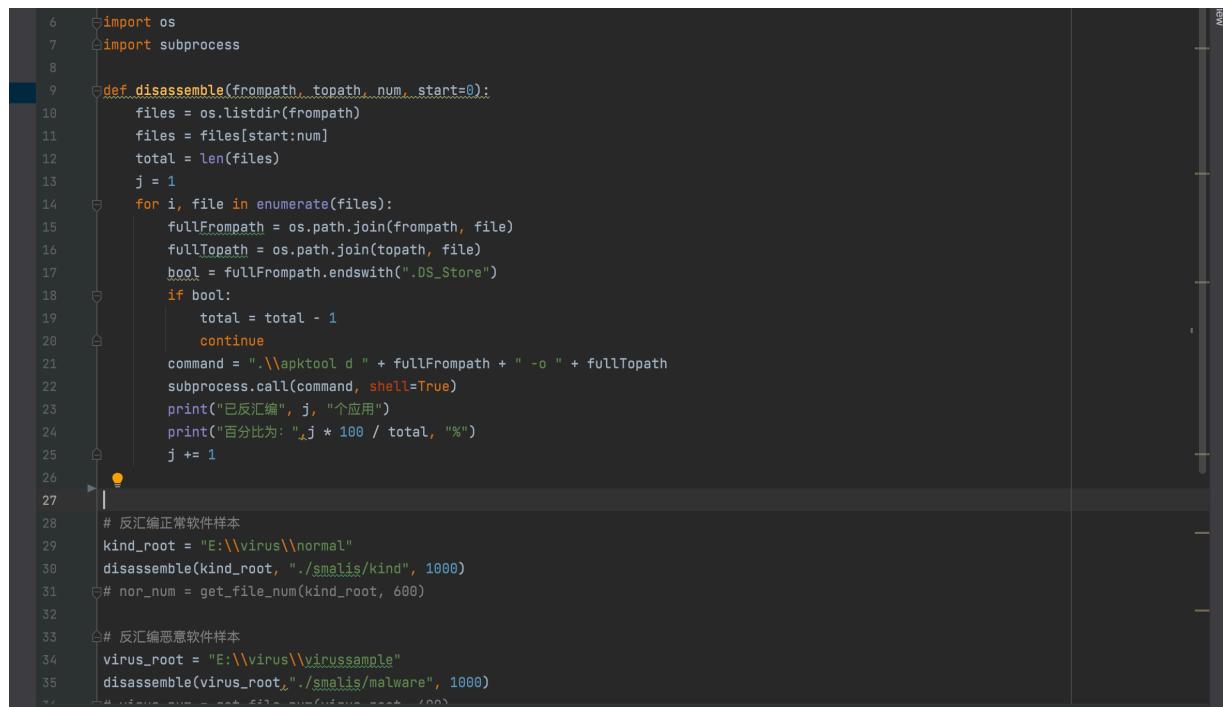
2.3.1 反汇编

区别与 JAVA 虚拟机 (JVM)，安卓的虚拟机称为 Dalvik 虚拟机 (DVM)。Java 虚拟机运行的是 Java 字节码，Dalvik 虚拟机运行的是 Dalvik 字节码。Java 虚拟机基于栈架构，而 Dalvik 虚拟机基于寄存器架构。

DVM 拥有专属的 DEX 可执行文件格式和指令集代码。smali 和 baksmali 是针对 DEX 执行文件格式的汇编器和反汇编器，反汇编后 DEX 文件会产生.smali 后缀的代码文件，smali 代码拥有特定的格式与语法，smali 语言是对 Dalvik 虚拟机字节码的一种解释。

apktool 工具是在 smali 工具的基础上进行封装和改进的，除了对 DEX 文件的汇编和反汇编功能外，还可以对 APK 中已编译成二进制的资源文件进行反编译和重新编译。所以我们直接使用 apktool 工具来反汇编 APK 文件，生成 APK 文件对应的 smali 源码文件。

实现反汇编的文件为 batch_disassemble.py。核心代码见下图：



```

6  import os
7  import subprocess
8
9  def disassemble(frompath, topath, num, start=0):
10     files = os.listdir(frompath)
11     files = files[start:num]
12     total = len(files)
13     j = 1
14     for i, file in enumerate(files):
15         fullFrompath = os.path.join(frompath, file)
16         fullTopath = os.path.join(topath, file)
17         bool = fullFrompath.endswith(".DS_Store")
18         if bool:
19             total = total - 1
20             continue
21         command = ".\\apktool d " + fullFrompath + " -o " + fullTopath
22         subprocess.call(command, shell=True)
23         print("已反汇编", j, "个应用")
24         print("百分比为: " + j * 100 / total, "%")
25         j += 1
26
27
28 # 反汇编正常软件样本
29 kind_root = "E:\\virus\\normal"
30 disassemble(kind_root, "./smalis/Kind", 1000)
31 # nor_num = get_file_num(kind_root, 600)
32
33 # 反汇编恶意软件样本
34 virus_root = "E:\\virus\\virussample"
35 disassemble(virus_root, "./smalis/malware", 1000)

```

图 3: 反汇编代码实现

2.3.2 字节编码映射

Smali 是对 DVM 字节码的一种解释，虽然不是官方标准语言，但所有语句都遵循一套语法规规范。由于 Dalvik 指令有两百多条，对此需要进行分类与精简，去掉无关的指令，只留下 M、R、G、I、T、P、V 七大类核心的指令集合，并且只保留操作码字段，去掉参数。(M、R、G、I、T、P、V 七大类指令集合分别代表了移动、返回、跳转、判断、取数据、存数据、调用方法七种类型的指令)。指令分类与描述的大致流程见下图：

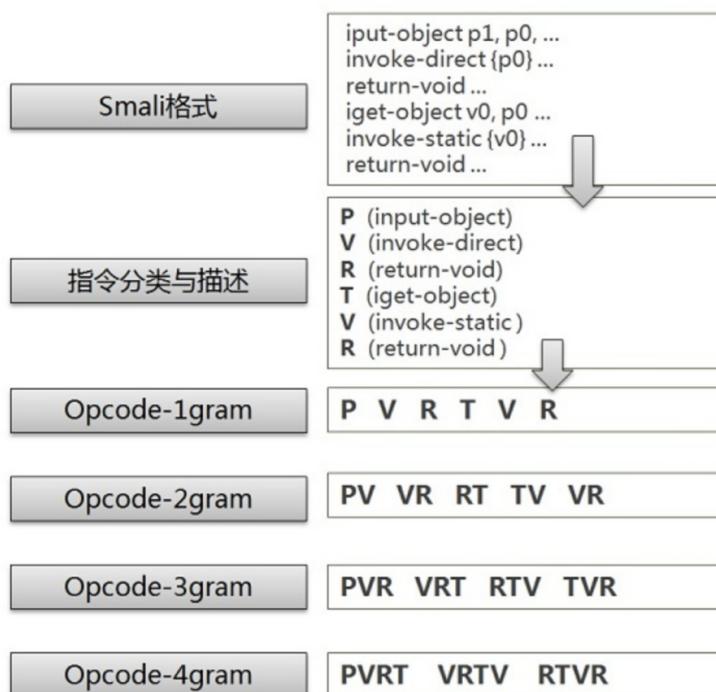


图 4: 指令分类与描述

因此我们要将字节码从 smali 文件中提取出来并映射成对应的分类，最终存储到当前目录下的 data.csv 中便于后续特征的提取。

对于映射规则，所有的字节码到其分类的映射规则都放在 `/infrastructure/map.py` 文件中。此外，`infrastructure.smali` 中的 Smali 类的每个实例代表一个 smali 文件，用于封装解析 smali 文件的逻辑。`infrastructure.ware` 中的 Ware 类实例代表一个安卓 APK，该类的实例会包含多个 Smali 实例，这些 Smali 实例都是从该 APK 反汇编得到的 smali 文件中得到的。

整个 `infrastructure` 目录下的 `fileutils.py`、`map.py`、`mydict.py`、`smali.py`、`ware.py` 文件都是字节编码这些内容的底层架构。

```
3 bytecode_map = { 3     import ...
4     #M指令集 4     class Smali:
5     "move": "M", 5
6     "move/from16": "M", 6     def __init__(self, path):
7     "move/16": "M", 7         self.path = path
8     "move-wide": "M", 8         with codecs.open(path, 'r', 'utf-8'):
9     "move-wide/from16": "M", 9             self.lines = f.readlines()
10    "move-wide/16": "M", 10            self.linenum = len(self.lines)
11    "move-object": "M", 11
12    "move-object/from16": "M", 12    def __to_next_method(self, begin):
13    "move-object/16": "M", 13        while begin < self.linenum:
14    "move-result": "M", 14            if self.lines[begin].startswith("M"):
15    "move-result-wide": "M", 15                return begin
16    "move-result-object": "M", 16            begin += 1
17    "move-exception": "M", 17    return -1
18    #R指令集 18
19    "return-void": "R", 19    def __analyze_line(self, line):
20    "return": "R", 20        words = line.split()
21    "return-wide": "R", 21        if words:
22    "return-object": "R", 22            cmd = words[0]
23    #G指令集 23            ctype = bmap.get(cmd, 0)
24    "goto": "G", 24            if ctype != 0:
25    "goto/16": "G", 25                self.featurelist.append(ctype)
26    "goto/32": "G", 26
27    #I指令集 27
28    "if-eq": "I", 28    def getFeature(self):
29    "if-ne": "I", 29        self.featurelist = []
30    "if-lt": "I", 30        cursor = 0
31    "if-ge": "I", 31        while True:
32    "if-gt": "I", 32            cursor = self.__to_next_method(cursor)
33    #C指令集 33            if cursor == -1:
34    "const4": "C", 34                return "".join(self.featurelist)
35    "const10": "C", 35            while True:
36    "const30": "C", 36            ...
```

图 5: infrastructure 字节编码底层架构部分代码

实现提取映射后的编码的文件为 `bytecode_extract.py`。核心代码见下图：

```
def collect(roottdir, isMalware):
    wares = os.listdir(roottdir)
    total = len(wares)
    for i, ware in enumerate(wares):
        warePath = os.path.join(roottdir, ware)
        ware = Ware(warePath, isMalware)
        print("文件名", ware.name)
        ware.extractFeature(f)
        print("已提取", i + 1, "个文件的特征")
        print("百分比: " + (i + 1) * 100 / total, "%")

# 1代表良性软件 0代表恶意软件 2代表测试软件

kindroot = "./smalis/kind"
virusroot = "./smalis/malware"
testwroot = "./smalis/test/white"
testbroot = "./smalis/test/black"

f = DataFile("./data.csv")

collect(kindroot, 1)
collect(virusroot, 0)
collect(testwroot, 2)
collect(testbroot, 2)
```

图 6: 字节编码映射提取

2.3.3 n-gram 特征提取

我们以 method (映射后的指令) 作为单位进行提取, 把每个 method 看成互不相关的“句子”。以 3-gram 为例, 如果 method 中的指令数目小于 3, 则忽略该 method。最终提取的特征是按照每种 n-gram 是否出现, 如果出现过为 1, 不出现就为 0。因为恶意软件往往不是从文件头开始写的, 大多数应该是通过重用之前的代码实现的, 因此通过判断一种恶意 n-gram 是否出现可以对恶意软件进行检测。

我们用 n_gram.py 对之前映射好的字节编码 data.csv 提取 n_gram 特征转换成 n_gram.csv。核心代码如下图:

```

8  n = int(sys.argv[1])
9  # n = 3
10 print("n = ", n)
11
12 origin = pd.read_csv("data.csv")
13 mdict = MyDict()
14 feature = origin["Feature"].str.split(" ")
15 total = len(feature)
16 for i, code in enumerate(feature):
17     mdict.newLayer()
18     if not type(code) == list:
19         continue
20     for method in code:
21         length = len(method)
22         if length < n:
23             continue
24         for start in range(length - (n - 1)):
25             end = start + n
26             mdict.mark(method[start:end])
27         print("已完成", i+1, "个应用")
28         print("百分比为: " , (i + 1) * 100 / total, "%")
29
30 result = mdict.dict
31 result['isMalware'] = origin.isMalware
32 pd.DataFrame(result, index=origin.index) \
33     .to_csv("./" + str(n) + "_gram.csv", index=False)

```

图 7: n-gram 特征提取代码

2.3.4 RF 随机森林训练和预测

在得到了软件的 n-gram 特征后, 我们需要对软件是否为恶意软件进行预测。在对项目进行初步实验设计时我们参考了一些其他项目资料来构建我们的检测器, 在考察了 SVM 向量机、BP 神经网络等常用的机器学习算法后, 我们最终决定使用随机森林模型来构建分类器进行检测。下图为随机森林模型简化结构图:

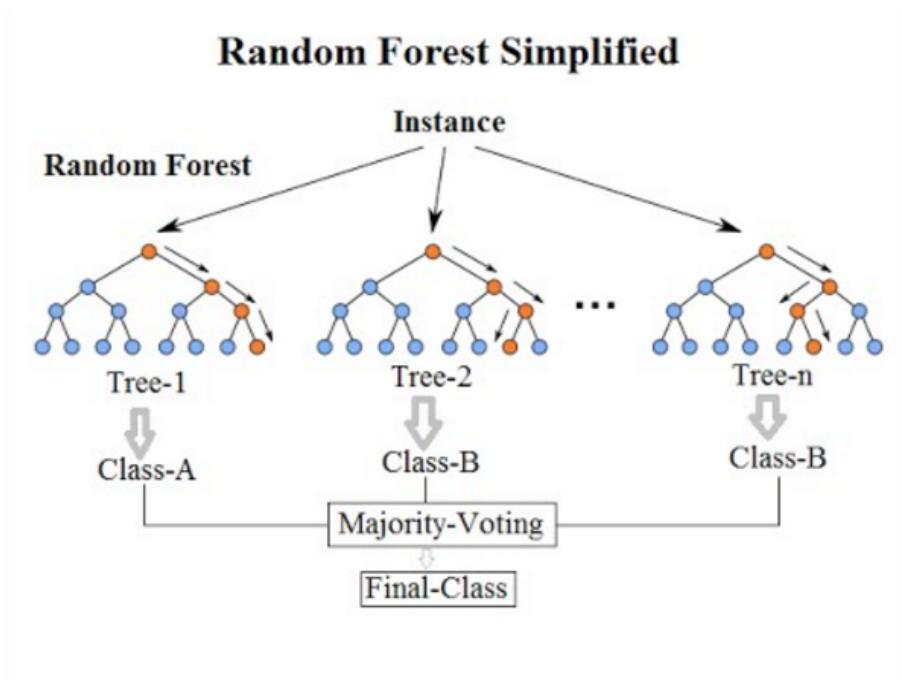


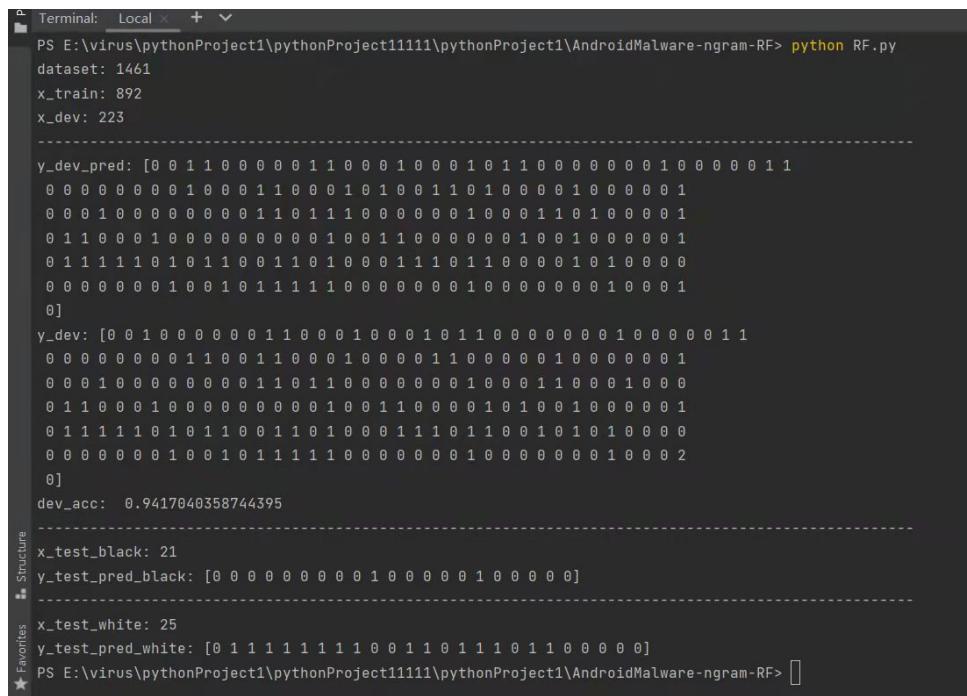
图 8: 随机森林模型结构

在训练部分，我们读入训练样本的 3-gram 特征矩阵文件 3_gram.csv，调用 sklearn 中的随机森林分类器进行训练。部分核心代码如下图：

```
7 dataset=pd.read_csv("3_gram.csv")
8 print("dataset:",dataset.shape[0])
9
10 x=dataset.iloc[300:(nor_num+virus_num),:-1].values
11 y=dataset.iloc[300:(nor_num+virus_num),-1].values
12
13
14 from sklearn.model_selection import train_test_split
15 x_train, x_dev, y_train, y_dev = train_test_split(x, y, test_size=.2, random_state=0)
16 print("x_train:",x_train.shape[0])
17 print("x_dev:",x_dev.shape[0])
18
19 # 特征量化
20 from sklearn.preprocessing import StandardScaler
21 sc = StandardScaler()
22 x_train = sc.fit_transform(x_train)
23 x_dev = sc.fit_transform(x_dev)
24
25 ...
26 from sklearn.ensemble import RandomForestClassifier as RF
27 # from sklearn.ensemble import RandomForestRegressor as RF
28 classifier = RF(n_estimators=500, n_jobs=-1)
29
30 # 训练分类器
31 classifier.fit(x_train,y_train)
32
33 # 预测验证集
34 y_dev_pred = classifier.predict(x_dev)
35 print("*"*100)
36 print("y_dev_pred:",y_dev_pred)
37 print("y_dev:",y_dev)
38 print("dev_acc: ",classifier.score(x_dev,y_dev))
```

图 9: 随机森林部分代码

在训练过程中我们使用了采集到的 1461 个样本，其中白样本 686 个，黑样本 729 个，随后在测试集上进行测试，经过训练的模型在验证集上的准确率为 94.17%。下图为模型最后输出结果：



```

Terminal: Local + 
PS E:\virus\pythonProject1\pythonProject11111\pythonProject1\AndroidMalware-ngram-RF> python RF.py
dataset: 1461
x_train: 892
x_dev: 223
-----
y_dev_pred: [0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1
0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1
0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1
0]
y_dev: [0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1
0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 2
0]
dev_acc: 0.9417040358744395
-----
x_test_black: 21
y_test_pred_black: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0]
-----
x_test_white: 25
y_test_pred_white: [0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0]
★ PS E:\virus\pythonProject1\pythonProject11111\pythonProject1\AndroidMalware-ngram-RF> []

```

图 10: 模型训练识别准确率

通过结果观察发现，我们训练的模型在识别准确率方面比较理想，但其中存在假阳现象，我们将在章节5.2对这个缺点的改进进行阐述。

2.4 模块实现-RF 检测器 api

在上一小节，我们得到一个检测效果较好的随机森林模型。在此小节，我们将把原来的代码进行重构和精简，构建一个实时检测 APK 恶意代码的接口。

2.4.1 反汇编

本模块的目的是将安卓 APK 从指定目录反汇编出对应的字节编码，因此将 frompath,topath 和文件名称设置为函数的参数。当在接口（章节2.4.5）的 test.py 中进行调用时，frompath 为我们需要测试的文件路径，指定 topath 为我们想要存放反汇编后生成的文件的地址。

```

import os
import subprocess

def disassemble(frompath,filename,topath, num, start=0):
    fullTopath = os.path.join(topath, filename)
    command = "apktool d " + frompath + " -o " + fullTopath + " -f "
    subprocess.call(command, shell=True)

```

图 11: 反汇编重构代码

2.4.2 字节编码映射

接下来需要对反汇编生成的文件进行提取处理，需要将字节编码取出并映射，保存映射后的文件。下图为重构后的代码：

```
from infrastructure.ware import Ware
from infrastructure.fileutils import DataFile
import os

def collect(rootdir,filename,isMalware,f):
    ware = filename
    warePath = os.path.join(rootdir, ware)
    ware = Ware(warePath, isMalware)
    ware.extractFeature(f)
```

图 12: 字节编码映射重构代码

2.4.3 n-gram 特征提取

本部分是将 data.csv 提取 n-gram 特征，转换成 n-gram.csv。在训练时对不同参数进行调试后，发现 $n=3$ 时效果最好，因此我们检测器也选用 3-gram 提取。此代码与训练时的代码基本一致。

2.4.4 随机森林检测器

在检测时，我们通过调用之前训练好的随机森林模型，读取待分析软件的 3-gram 特征文件进行分析，最终输出是否为恶意软件。

```
7  # load model
8  classifier = joblib.load("./train_model.m")
9
10 # load dataset
11 dataset=pd.read_csv("./result.csv")
12
13 x=dataset.iloc[1:2,:-1].values
14 y=dataset.iloc[1:2,-1].values
15
16 y_test = classifier.predict(x)
17 if(y_test == 0):print('0',end='')
18 else:print('1',end='')
```

图 13: 随机森林模型调用重构代码

2.4.5 调用接口

我们的检测 api 是传入一个文件名和路径，输出这个文件是否为安卓恶意代码，0 代表恶意软件，1 代表良性软件。核心代码如下：

```
12  test_root = sys.argv[1]
13  filename = sys.argv[2]
14  batch_disassembler.disassemble(test_root,filename,"/home/group9/smalis/test", 600)
15
16  testwroot = "/home/group9/smalis/test"
17  f = bytecode_extract.DataFile("/home/group9/data.csv")
18  bytecode_extract.collect(testwroot,filename, 2,f)
19  f.close()
20
21  n_gram.gram()
22  getfeature.get()
23  RF_usage.RF()
```

图 14: 整个检测器接口 api 代码

3 ClamAV 的安装、修改与应用

3.1 ClamAV 安装与部署

- 安装先决条件
 - yum install -y epel-release
 - yum install -y dnf-plugins-core
 - yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
 - yum install -y gcc gcc-c++ make python3 python3-pip valgrind
 - yum install -y bzip2-devel check-devel json-c-devel libcurl-devel libxml2-devel
 - yum install -y ncurses-devel openssl-devel pcre2-devel sendmail-devel zlib-devel
- 创建 ClamAV 目录，在 ClamAV 官网下载 ClamAV-0.103.5.tar.gz 包，解压源文件。
 - mkdir /usr/local/ClamAV
 - cd /Downloads
 - tar xzf ClamAV-0.103.5.tar.gz
 - cd ClamAV-0.103.5
- 设置 build，使用典型配置

- ./configure --prefix=/usr/local/ClamAV
- 编译 ClamAV
 - make -j2
- 安装 ClamAV
 - make install
- 创建日志目录和病毒库目录
 - mkdir /usr/local/ClamAV/logs
 - mkdir /usr/local/ClamAV/updata
- 修改配置文件
 - vim /usr/local/ClamAV/etc/clamd.conf
 - # Example
第 8 行, 注释掉 Example
 - LogFile /usr/local/ClamAV/logs/clamd.log
第 14 行, 删掉注释, 目录改为 logs 下
 - PidFile /usr/local/ClamAV/updata/clamd.pid
第 57 行, 删掉注释, 更改目录
 - DatabaseDirectory /usr/local/ClamAV/updata
第 65 行, 删掉注释, 更改目录
- 创建日志文件
 - touch /usr/local/ClamAV/logs/freshclam.log
 - chown ClamAV:ClamAV /usr/local/ClamAV/logs/freshclam.log
 - touch /usr/local/ClamAV/logs/clamd.log
 - chown ClamAV:ClamAV /usr/local/ClamAV/logs/clamd.log
 - chown ClamAV:ClamAV /usr/local/ClamAV/updata
- 升级病毒库

- /usr/local/ClamAV/bin/freshclam
- 查杀当前目录并删除感染的文件
 - /usr/local/ClamAV/bin/clamscan --remove

```
----- SCAN SUMMARY -----
Known viruses: 8609038
Engine version: 0.103.5
Scanned directories: 1
Scanned files: 45
Infected files: 0
Data scanned: 4.49 MB
Data read: 2.23 MB (ratio 2.01:1)
Time: 55.738 sec (0 m 55 s)
Start Date: 2022-04-06 13:15:27
End Date: 2022-04-06 13:16:23
[root@localhost clamav]#
```

图 15: 安装完 clamscan 扫描结果

3.2 ClamAV 源码修改

3.2.1 定位 APK 检测

ClamAV 确定文件类型的主要机制是将文件与 File Type Magic 签名匹配。这些文件类型签名会被编译到 ClamAV 中。但是通过查询发现，ClamAV 原生没有 APK 检测的 type。下图为 ClamAV 官网的 CL_TYPE 类型：

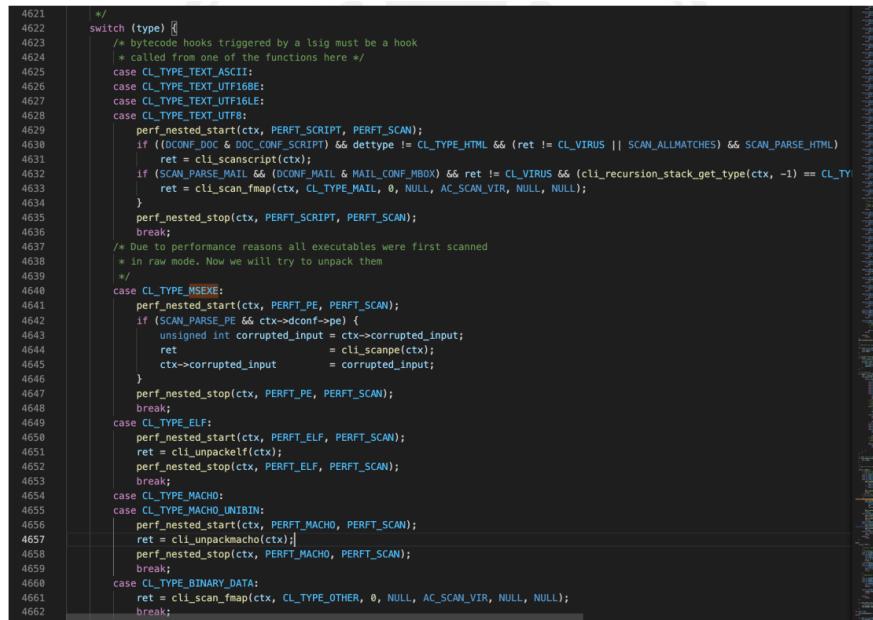
ClamAV Documentation

ClamAV File Types are prefixed with `CL_TYPE_`. The following is an exhaustive list of all current file types.

| CL_TYPE | Description |
|----------------------------------|--|
| <code>CL_TYPE_7Z</code> | 7-Zip Archive |
| <code>CL_TYPE_7ZSFX</code> | Self-Extracting 7-Zip Archive |
| <code>CL_TYPE_APM</code> | Disk Image - Apple Partition Map |
| <code>CL_TYPE_ARJ</code> | ARJ Archive |
| <code>CL_TYPE_ARJSFX</code> | Self-Extracting ARJ Archive |
| <code>CL_TYPE_AUTOIT</code> | AutoIt Automation Executable |
| <code>CL_TYPE_BINARY_DATA</code> | binary data |
| <code>CL_TYPE_BINHEX</code> | BinHex Macintosh 7-bit ASCII email attachment encoding |
| <code>CL_TYPE_BZ</code> | BZip Compressed File |
| <code>CL_TYPE_CABSFX</code> | Self-Extracting Microsoft CAB Archive |
| <code>CL_TYPE_CPIO_CRC</code> | CPIO Archive (CRC) |
| <code>CL_TYPE_CPIO_NEWC</code> | CPIO Archive (NEWC) |
| <code>CL_TYPE_CPIO_ODC</code> | CPIO Archive (ODC) |
| <code>CL_TYPE_CPIO_OLD</code> | CPIO Archive (OLD, Little Endian or Big Endian) |
| <code>CL_TYPE_CRYPTFFF</code> | Files encrypted by CryptFFF malware |
| <code>CL_TYPE_DMG</code> | Apple DMG Archive |
| <code>CL_TYPE_EGG</code> | ESTSoft EGG Archive, new in 0.102 |
| <code>CL_TYPE_ELF</code> | ELF Executable (Linux/Unix program or library) |
| <code>CL_TYPE_GIF</code> | GIF Graphics File, new in 0.103 |
| <code>CL_TYPE_GPT</code> | Disk Image - GUID Partition Table |
| <code>CL_TYPE_GRAPHICS</code> | Other graphics files; BMP, JPEG2000 |
| <code>CL_TYPE_GZ</code> | GZip Compressed File |

图 16: ClamAV 官网的 CL_TYPE 类型

通过手动调试输出，发现 ClamAV 对 APK 的检测会将其碎片化，即拆成很多个小部分，然后分别调用 CL_TYPE_MACHO_UNIBIN 和 CL_TYPE_BINARY_DATA 类型的检测。下图为常规检测时调用的文件类型：



```

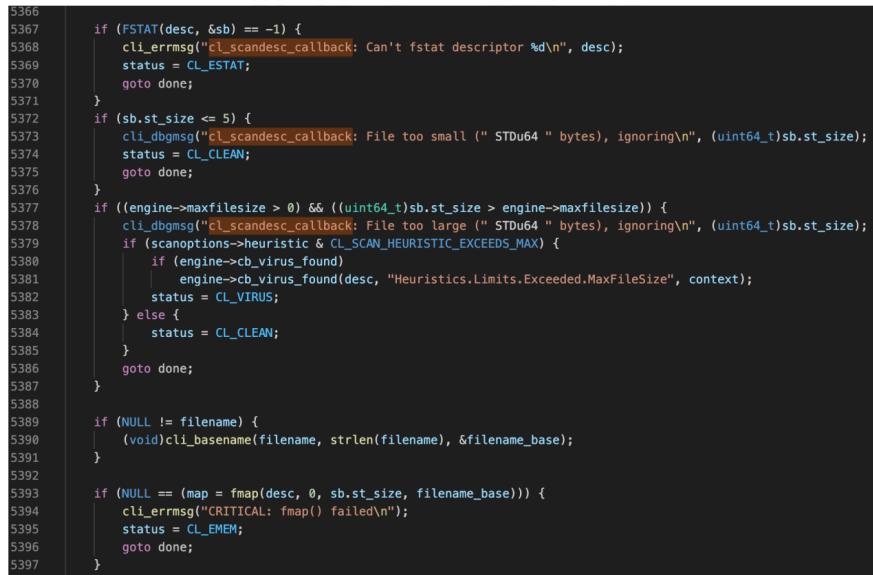
4621     /*
4622     switch (type) {
4623     /* bytecode hooks triggered by a lsig must be a hook
4624     * called from one of the functions here */
4625     case CL_TYPE_TEXT_ASCII:
4626     case CL_TYPE_TEXT_UTF16BE:
4627     case CL_TYPE_TEXT_UTF16LE:
4628     case CL_TYPE_TEXT_UTF8:
4629         perf_nested_start(ctx, PERFT_SCRIPT, PERFT_SCAN);
4630         if ((DCONF_DOC & DOC_CONF_SCRIPT) && dettype != CL_TYPE_HTML && (ret != CL_VIRUS || SCAN_ALLMATCHES) && SCAN_PARSE_HTML)
4631             ret = cli_scanscript(ctx);
4632         if (SCAN_PARSE_MAIL && (DCONF_MAIL & MAIL_CONF_MBOX) && ret != CL_VIRUS && (cli_recursion_stack_get_type(ctx, -1) == CL_TYPE_MAIL))
4633             ret = cli_scan_fmap(ctx, CL_TYPE_MAIL, 0, NULL, AC_SCAN_VIR, NULL, NULL);
4634     }
4635     perf_nested_stop(ctx, PERFT_SCRIPT, PERFT_SCAN);
4636     break;
4637     /* Due to performance reasons all executables were first scanned
4638     * in raw mode. Now we will try to unpack them
4639     */
4640     case CL_TYPE_MSXE:
4641         perf_nested_start(ctx, PERFT_PE, PERFT_SCAN);
4642         if (SCAN_PARSE_PE && ctx->dconf->pe) {
4643             unsigned int corrupted_input = ctx->corrupted_input;
4644             ret = cli_scanspe(ctx);
4645             ctx->corrupted_input = corrupted_input;
4646         }
4647         perf_nested_stop(ctx, PERFT_PE, PERFT_SCAN);
4648     break;
4649     case CL_TYPE_ELF:
4650         perf_nested_start(ctx, PERFT_ELF, PERFT_SCAN);
4651         ret = cli_unpackelf(ctx);
4652         perf_nested_stop(ctx, PERFT_ELF, PERFT_SCAN);
4653     break;
4654     case CL_TYPE_MACHO:
4655     case CL_TYPE_MACHO_UNIBIN:
4656         perf_nested_start(ctx, PERFT_MACHO, PERFT_SCAN);
4657         ret = cli_unpackmacho(ctx)];
4658         perf_nested_stop(ctx, PERFT_MACHO, PERFT_SCAN);
4659     break;
4660     case CL_TYPE_BINARY_DATA:
4661         ret = cli_scan_fmap(ctx, CL_TYPE_OTHER, 0, NULL, AC_SCAN_VIR, NULL, NULL);
4662     break;

```

图 17: 常规检测时调用的文件类型

通过向前定位函数调用，查找其他对文件类型进行分类的位置，我们发现在 cl_scandesc_callback 函数下，会先对文件是不是空、文件大小做限制。

于是我们在这个函数加一个条件判断文件是否为 APK，如果是 APK 就使用我们的方法进行文件检测，如果不是就继续使用原有的 ClamAV 检测。下图为我们定位到的可以添加判断条件的位置：



```

5366     if (FSTAT(desc, &sb) == -1) {
5367         cli_errmsg("cl_scandesc_callback: Can't fstat descriptor %d\n", desc);
5368         status = CL_ESTAT;
5369         goto done;
5370     }
5371     if (sb.st_size <= 5) {
5372         cli_dbgmsg("cl_scandesc_callback: File too small (" STDu64 " bytes), ignoring\n", (uint64_t)sb.st_size);
5373         status = CL_CLEAN;
5374         goto done;
5375     }
5376     if ((engine->maxfilesize > 0) && ((uint64_t)sb.st_size > engine->maxfilesize)) {
5377         cli_dbgmsg("cl_scandesc_callback: File too large (" STDu64 " bytes), ignoring\n", (uint64_t)sb.st_size);
5378         if (scanoptions->heuristic & CL_SCAN_HEURISTIC_EXCEEDS_MAX) {
5379             if (engine->cb_virus_found)
5380                 engine->cb_virus_found(desc, "Heuristics.Limits.Exceeded.MaxFileSize", context);
5381             status = CL_VIRUS;
5382         } else {
5383             status = CL_CLEAN;
5384         }
5385         goto done;
5386     }
5387     if (NULL != filename) {
5388         (void)cli_basename(filename, strlen(filename), &filename_base);
5389     }
5390     if (NULL == (map = fmap(desc, 0, sb.st_size, filename_base))) {
5391         cli_errmsg("CRITICAL: fmap() failed\n");
5392         status = CL_EMEM;
5393         goto done;
5394     }

```

图 18: 可以添加条件的位置

3.2.2 用 c 调用 python 检测

在章节2.4.5我们已经用 python 封装好了一个用 RF 分类器检测 APK 文件是否为病毒的脚本。于是我们在 scanners.c 中用 c 调用 python 脚本。并在终端读取脚本执行后的返回值，用于下一步的判断。detect() 函数用于调用 api 检测，exec_process() 函数用于处理终端输出值的回传。部分代码如下：

```

136 int exec_process(char *cmd, char *result, int size)
137 {
138     size_t ret;
139     FILE *fp = NULL;
140     if(cmd == NULL)
141     {
142         return -1;
143     }
144     fp = popen(cmd,"r");
145     if(fp == NULL)
146     {
147         printf("exec_process: error exec cmd: %s\n", cmd);
148         return -1;
149     }
150     if(result != NULL && size > 0)
151     {
152         memset(result, 0, size);
153         ret = fread(result, 1, size - 1, fp);
154         if(ferror(fp))
155         {
156             printf("exec_process: error\n");
157             pclose(fp);
158             return -1;
159         }
160         if(ret == 0) {
161             pclose(fp);
162             return -1;
163         }
164     }
165     pclose(fp);
166     return 0;
167 }
168 int detect(const char *file_name, const char *file_namebase){
169     char hhhhh[1000] = "python3 /home/group9/test.py ";
170     strcat(hhhhh, file_name);
171     strcat(hhhhh, " ");
172     strcat(hhhhh, file_namebase);
173     char popenret[100000] = {0};/*执行脚本后的返回值*/
174     memset(popenret, 0, sizeof(popenret));
175     if(exec_process(hhhhh, popenret, sizeof(popenret)) != 0) {

```

图 19: detect 和 exec_process 函数的部分代码

3.2.3 cl_scandesc_callback 函数改写

在章节3.2.1中我们得到要修改 cl_scandesc_callback 函数，同时我们发现结果是在 status 存储的，类型为 CL_VIRUS,CL_CLEAN，并会在 ClamAV 终端输出返回值。刚刚在章节3.2.2中，我们有了用 c 调用 python 的检测函数接口，所以我们对 cl_scandesc_callback 函数做以下修改：

```

5398 int temp = 0;
5399 if (is_apk(filename)){
5400     temp = detect(filename,filename_base);
5401     if (temp == 1){
5402         status = CL_VIRUS;
5403         engine->cb_virus_found(desc, "Group9 detected virus", context);
5404     }
5405     else if (temp == 2){
5406         status = scan_common(map, filename, virname, scanned, engine, scanoptions, context);
5407     }
5408     else status = CL_CLEAN;
5409 }
5410 else{
5411     status = scan_common(map, filename, virname, scanned, engine, scanoptions, context);
5412 }

```

图 20: cl_scandesc_callback 函数添加部分

3.3 修改后的 APK 文件检测效果

重新编译运行 clamscan，发现 apk 文件都能检测出来，并能输出我们的文字。检测截图如下：

```
[root@localhost example]# /usr/local/etc/bin/clamscan
/home/example/virusshare127489hfueoal.apk: Group9 detected virus FOUND
/home/example/virusshare04738921473829.apk: Group9 detected virus FOUND
/home/example/virussharefheuwoyq8979032up.apk: Group9 detected virus FOUND
/home/example/python: Symbolic link
/home/example/air.com.apk: OK
/home/example/tattoo.my.photo.apk: OK

----- SCAN SUMMARY -----
Known viruses: 8609038
Engine version: 0.103.5
Scanned directories: 1
Scanned files: 5
Infected files: 3
Data scanned: 0.00 MB
Data read: 41.87 MB (ratio 0.00:1)
Time: 81.192 sec (1 m 21 s)
```

图 21: APK 文件检测效果

4 ClamTK 的安装、修改与应用

4.1 ClamTK 安装

从 github 下载 ClamTK 源代码。直接执行 perl ClamTK 安装命令，不过这时由于缺失所需的包指令会提示错误，根据报错信息安装缺失软件包。

首先安装 cpan 软件包： yum install perl-CPAN*

安装完成后，可以通过 cpan 来安装相应模块。

4.2 ClamTK 前端修改

对 GUI.pm 进行修改，修改 ClamTK 前端界面。

- 将 ClamTK 界面标题修改为“ClamTK Virus Scanner of Group9”。

```
$hb->set_title( _( 'ClamTk Virus Scanner of Group9' ) );
$hb->set_decoration_layout( 'menu,icon:minimize,close' );
$hb->set_show_close_button( TRUE );
```

图 22: 修改标题的代码

```

sub about {
    my $dialog = Gtk3::AboutDialog->new;
    my $license
        = 'ClamTk is free software; you can redistribute it and/or
        · modify it under the terms of either:
        · · a) the GNU General Public License as published by the Free
        · Software Foundation; either version 1, or (at your option)
        · any later version, or
        · · b) the "Artistic License".';
    $dialog->set_wrap_license( TRUE );
    $dialog->set_position( 'mouse' );

    my $images_dir = ClamTk::App->get_path( 'images' );
    my $icon      = "$images_dir/Dee.png";
    my $pixbuf    = Gtk3::Gdk::Pixbuf->new_from_file( $icon );
}

```

图 23: 修改关于界面的代码

- 增加名为“Group9”的按钮，描述为“Group9 clamscan”。

```

    {
        link      => _('Group9'),
        description => _('Group9 clamscan'),
        image      => 'media-playback-start',
        button     => FALSE,
    },
}

```

图 24: 添加按钮的代码

- 将 ClamTK 的关于界面改成我们自己的图片。

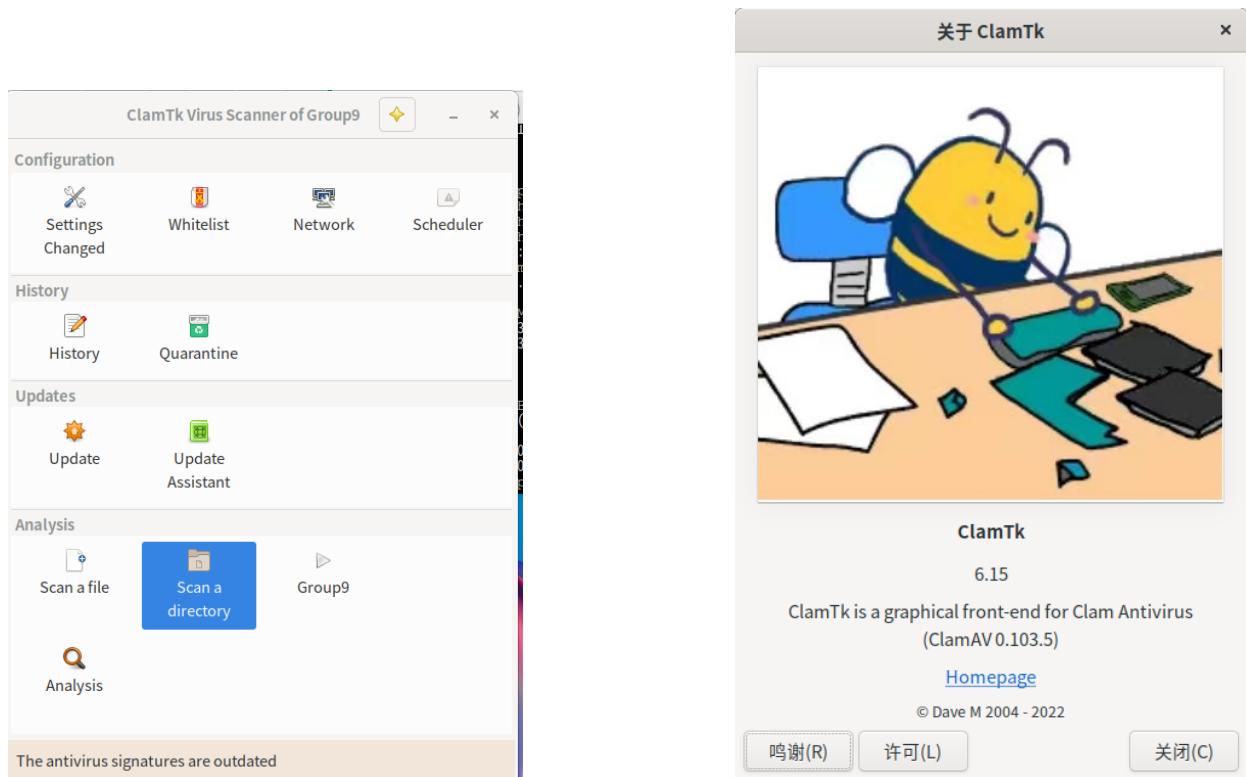


图 25: 修改后的 ClamTK 界面

- ClamTK 成功调用 ClamAV, 扫描 APK 文件。

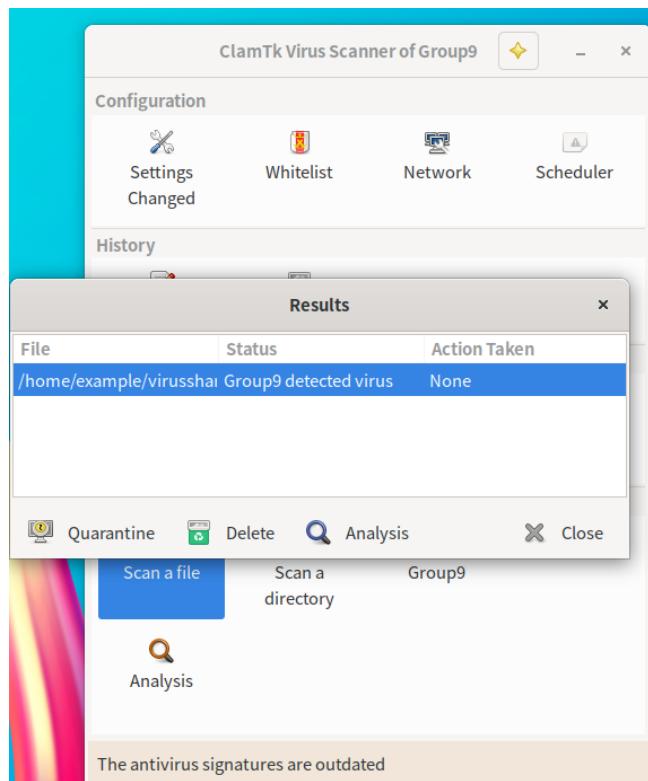


图 26: ClamTK 成功运行

5 总结

5.1 项目成果总结

本组基于 openEuler 操作系统, 对 ClamAV 源码进行了修改, 使用收集到的数据集训练了可以对 APK 格式的恶意代码文件进行检测的随机森林模型, 并在 ClamTK 中增添了相关功能界面, 最终展示结果良好。

5.2 不足与改进

- 假阳性概率较高: 目前我们只使用了随机森林模型对分类器进行训练, 在后续改进时可以考虑使用层次结构更复杂、识别效果更好的其他神经网络模型。
- 运行所需反编译时间较长, clamscan 调用过程占用的内存空间较大: 目前我们使用的是基于 java 的 apktool, 这种方法的内存占用较高。在后续改进时可以使用 c 语言原生的 java 反编译库, 以提高效率。

- 判断是否为 APK 的方法不佳：我们现在判断 APK 文件的方法不够通用，可以进行改进。在反汇编时可以观测到一个文件是否反汇编成功，所以可以通过判断能否反汇编成功来判断是否为 APK。

5.3 个人总结

冯驰

在本次的项目中，我是小组的组长。我主要负责的部分是使用随机森林对 APK 文件进行检测和 ClamAV 代码改写部分。

总体收获：

- 在本次大作业中，我查找了许多资料，阅读了较多前人代码，因此学习到了很多机器学习有关的知识。
- 虽然学过 c++，但平时对 c 的使用并不多，所以对 c 和 c++ 的区别不是很清楚。这次的大作业让我对 c 语言的理解更加清晰了。
- 平时我对工程类项目的接触较少，所以这次课程作业中阅读 ClamAV 源码的环节增强了我理解大量代码项目的能力。

最开始我们查到了很多符合作业要求的选题，包括对 dll 文件检测、对挖矿病毒检测等等，让我了解到了很多对病毒检测的方法。

在使用随机森林检测安卓恶意代码环节，我学习了如何对 APK 文件进行反汇编，什么是 Dalvik opcodes，如何将反汇编的 smali 编码映射到 Dalvik 指令，如何通过滑动平均提取 n-gram 特征，如何构建随机森林分类器进行训练、验证和预测等。在调整网络结构和数据集时，我观测了验证集的准确率，发现会出现过拟合和欠拟合现象，所以还要通过调节来使准确率达到最佳。同时，我通过实验比较，提取不同的 n-gram 特征最后得到的准确率是不同的，在比较之后，我最后选取的是 n=3。

在修改 ClamAV 源码时，我通过层层向上查找，修改了原本 ClamAV 直接进行检测的函数。在这过程中，我阅读了 ClamAV 许多源代码，并查阅了 ClamAV 官方网站，才发现并没有专门的 APK 检测接口。同时，我也写了 c 调用 python 脚本对 APK 文件进行检测，读取终端检测返回值等一系列函数来完成 ClamAV 对 APK 的检测。

总之，本次小组大作业让我收获了很多知识，也调节了同学之间合作的友谊，让我的合作能力和项目能力都得到了提升。

姚思悦

在本次的项目中，我负责的是随机森林分类器的训练和调用相关部分。

在项目进行前期，我对该方面常用的神经网络算法模型进行了调研，从实现难度、实现效果等方面对不同方法进行了评估，在结合从网络上搜索到的其他项目材料的基础上，最终选定了随机森林模型。

在项目实施的过程中，我们参考其他项目想通过使用 C 语言改写的方式提高我们的运算效率，但最终因为随机森林模型的改写难以找到先例且工程量较大，最终选择通过 C 调用 Python 的方式实现。

在进行检测过程中，我们发现在进行 3-gram 特征提取时，最终的输出结果与输入样本数相关，导致在进行检测时检测样本的特征矩阵与模型中的特征矩阵特征数不同，进而无法正确识别调用。因此我添加了一步对输出的特征矩阵进行处理，以确保得到向量数与模型要求相同。

通过本次实验，我加深了自己对各种神经网络常用模型的理解，并对它们的结构和相关原理有了更清晰的认识。与此同时，通过前期阅读参考项目的源代码，我深化了自己的对于病毒的理解，提升了阅读代码的能力，锻炼了写代码的能力。

在小组合作中，我与同组同学在疫情情况下坚持讨论，克服了很多困难达成合作，最终成功实现了项目。这既磨砺了我们的意志，也让我们更深切地感受到了团队合作的可贵精神。

韩雨虹

在本次大作业之前进行其他课程的实验与作业时，我们使用的多是 Ubuntu 虚拟机，而 openEuler 系统是我们第一次使用，许多操作方式并不熟悉，这也造成了一开始在配置虚拟机环境、编译安装 ClamAV 和 ClamTK 的困难。

实验刚开始时，我们先选择了一个自带桌面环境的 kylinsec 系统，但在安装 ClamAV 的过程中发现这个系统缺少一些 ClamAV 需要的库，所以我们又换成了最后使用的 openEuler20.03-LTS-SP3 版本。而在编译安装 ClamAV 的过程中也遇到了一些问题：选择 0.103.1 版本后，系统提示版本较低，最

终选择 0.103.5 版本；缺少较多依赖库，需要手动安装；终端运行时无法识别 freshclam 及 clamscan 指令，需要软连接或者通过路径调用。

在实验过程中，我的虚拟机也多次“重生”。从崩溃、移除虚拟机、新建虚拟机等反复操作过程中，反复就同一个问题在网上搜索解决方法、无济于事，我的心态也一点一点被“磨平”，我能够更加平和地接受自己的失败、不足。

本次实验让我了解了 openEuler 操作系统的相关知识，也对 ClamAV 和 ClamTK 的使用、源码有了一定认识。同时，我还学习了基于随机森林进行安卓恶意代码检测的方法。

同时，感谢冯驰、姚思悦和何梓欣同学在实验过程中的帮助，也感谢刘功申老师的指导。

何梓欣

在本次实验中我主要负责了部分 RF 训练器模型和实时检测的代码实现。

在最初确立好要实现检测恶意安卓软件的目标后，我们搜索比较了很多资料，最终选用了将 APK 反汇编然后提取特征的方法。实际操作中发现 APK 反汇编速度较慢，会大大影响 clamscan 的时间，但是由于我的代码能力有限，就没有选用效果更好的方法，这也是我以后需要着重提高的地方。在完成 python 代码实现的过程中，我还发现我写代码的习惯并不是很好，主要有以下 3 点问题：

1. 代码中不习惯随手标注释，尤其是因为线上的原因，交流不便，导致一些参数和接口需要额外花费时间和其他同学进行统一；
2. 面向对象的抽象不够，每一个 py 文件都是简单地把实现过程封装在函数里，并没有实现各种接口的分离；
3. 在 debug 的时候用了很多输出语句，给后来调用的同学造成了很大不便。

不过总体来说，通过这个项目我对病毒和恶意代码有了进一步的理解，我的代码能力也得到了很大的提高，在实验过程中遇到了非常多的困难，也学会了使用不同的手段去解决。更重要的是，我在这个项目中体会到了团队配合的快乐。非常感谢和我同组的冯驰，韩雨虹和姚思悦，我们在疫情的严重影响下，依然努力配合，相互帮助，使我们的项目得以很好地完成，也让我有了很大的进步。

6 致谢

- 感谢刘功申老师在课程中教授关于计算机病毒和恶意代码的知识，同时感谢刘老师课程之余对我们提出的问题细心解答与耐心指导。
- 感谢助教的付出和在我们编程时遇到困难的指导，这让我们在开发过程中更有动力。
- 感谢冯驰，姚思悦，韩雨虹，何梓欣小组内四位同学的通力合作，出色的完成了这一项目的开发。感谢每个人的辛苦付出，我们项目的顺利进展离不开大家共同的努力。