# 基于AOP的Android Studio恶意

# 代码检测

# 郑文浩/1 白璠/1 丹增晋美/2

北京理工大学

501320528@qq.com

# 摘要：近年来随着智能手机的发展，基于Android Studio的恶意代码逐渐增多，防止和检测恶意代码就显得尤为重要，本文提出了一种通过AOP获取代码的CPU占用率与内存使用等情况，再通过机器学习的算法，用训练集来生成决策树模型，通过测试集来检测恶意代码。

# **1. 导言**

# Android由于最底层使用Linux内核，使用的是GPL许可证，也就意味着相关的代码是必须开源的。2007年11月，Google与84家硬件制造商、软件开发商及电信营运商组建开放手机联盟共同研发改良Android系统，随后Google以Apache开源许可证的授权方式（比GPL协议稍严格些），发布了Android的源代码。自此之后，Android的发展可以说是突飞猛进。开源带来的是快速流行的能力与较低的学习成本。一方面，各个手机厂商无需自行开发手机[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/watsonliang/article/details/_blank)，因此纷纷采用Android系统，甚至可以按照自己的目的进行深度定制（虽然有时候会负分优化）。另一方面，开源也促进了学习研究社区的迅速兴起，对于开发者来说，相比[iOS](http://lib.csdn.net/base/ios" \o "iOS知识库" \t "http://blog.csdn.net/watsonliang/article/details/_blank)，开源使得[安卓](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/watsonliang/article/details/_blank)是一个更适合研究与魔改的系统，而不受到不开源系统的限制。

# 开源带来的一个极大的好处就是手机厂商成本的降低。除却了操作系统开发的高成本，Android厂商的手机价格可以控制在很低的水平；或者在同样价位中相对iOS拥有更高端的硬件配置。因此在中低端市场，安卓有着绝对的统治地位；在高端市场也与iOS有一较之力。可以说是安卓实现了普通消费者也能使用[智能](http://lib.csdn.net/base/aiplanning" \o "人工智能规划与决策知识库" \t "http://blog.csdn.net/watsonliang/article/details/_blank)机的梦想。近年来随着智能手机成为生活得必备，Android系统也已经占有了很大的市场份额，并且基于Android系统的应用也是呈现爆炸式的增长，达到了惊人的81%。但是随着Android系统的快速增长，恶意代码也随之出现，并且移动恶意代码的发展具有较强的爆发性，使得检测Android系统的恶意代码成为一个受关注的问题。

# 其中较为典型的恶意行为如下：

# 1通过疯狂截图来获取聊天信息、短信内容等，以窃取用户隐私信息;利用手机僵尸网络“挖矿”——CoinKrypt家族木马；通过手机架设Web服务器，窃取用户隐私，反向链接的行为更加隐蔽——Gandspy家族木马；将移动设备加密锁屏后，对用户进行勒索——simplelock家族；

# 2 编程语言多样化Android应用支持多种开发语言，除常规的Java与C/++，还有易语言、VB、C#、HTML5等。2014年开始捕获到使用易语言和C#开发的恶意应用，其中易语言开发的恶意代码应用已超过200余个。

# 3 恶意软件加壳技术

# 2014年，更多恶意软件采用加壳技术躲避安全软件的查杀。2013年到2014年仅一年时间，加壳恶意软件数量就增长了约18倍。

**2. Aspectj介绍**

AspectJ（也就是[AOP](http://baike.baidu.com/item/AOP" \t "http://baike.baidu.com/_blank)）的动机是发现那些使用传统的编程方法无法很好处理的问题。@AspectJ 使用了Java5 的注解，可以将切面声明为普通的Java类。

[面向方面编程](http://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E6%96%B9%E9%9D%A2%E7%BC%96%E7%A8%8B" \t "http://baike.baidu.com/_blank)的出现正好给处于黑暗中的我们带来了光明，它针对于这些横切关注点进行处理，就好像面向对象编程处理一般的关注点一样。而作为AOP的具体实现之一的AspectJ，它向Java中加入了连接点（Join Point）这个新概念，其实它也只是现存的一个Java概念的名称而已。它向Java语言中加入少许新结构：切点（pointcut）、通知（Advice）、类型间声明（Inter-type declaration）和方面（Aspect）。切点和通知动态地影响程序流程，类型间声明则是静态的影响程序的类等级结构，而方面则是对所有这些新结构的封装。

1. **数据挖掘**

本次研究采用动态分析，用过切面程序基于权限、CPU占用率与内存使用情况分析项目。

通过设置切点，编写切面程序，在项目当中获取所需的数据，但值得注意Aspectj在Android当中的功能丰富，需要认真掌握。

以下为本次实验调取权限、CPU占用率与内存使用情况的切面程序的部分代（主要部分）。其中获取系统各项数据的方法被我们封装在SystemInformation类中。

@Aspect  
public class TraceAspect {  
 private static final String METHOD\_EXECUTION = "execution(\* \*..MainActivity.\*(..))";   
 @Pointcut(METHOD\_EXECUTION)  
 public void methodExecution() {  
 }  
 @TargetApi(Build.VERSION\_CODES.LOLLIPOP)  
 @Before("methodExecution()")  
public void beforeMethodExecution(JoinPoint joinPoint) {

//获取方法名

getgetSignature().getName();

//获取开始时间

SystemInformation.getstartTime()  
 }  
@TargetApi(Build.VERSION\_CODES.LOLLIPOP)  
@After("methodExecution()")  
public void SystemInformation.afterMethodExecution(JoinPoint joinPoint) {

//获取结束时间

SystemInformation。GetEndTime();

//获取CPU使用率  
SystemInformation.getProcessCpuRate();

//获取可用物理内存

SystemInformation.getFreePhysicalMemory()

//获取软件获得的系统权限

SystemInformation.getPermission(Context context)   
 }

}

**4. 数据分析**

对提取的数据本次实验采用机器学习的方法，机器学习理论主要是设计和分析一些让计算机可以自动学习的算法。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。因为学习算法中涉及了大量的统计学理论，机器学习与统计推断学联系尤为密切，也被称为统计学习理论。

此次采用当中的j48决策树进行分析决策树生成算法，算法决策树是一种通过对历史数据进行测算实现对新数据进行分类和预测的算法。简单来说决策树算法就是通过对已有明确结果的历史数据进行分析，寻找数据中的特征。并以此为依据对新产生的数据结果进行预测。

**(1)决策树生成**

本次借助开源数据挖掘工具weka的jar架包，通过api调用其中的决策树生成算法，并生成图形显示，部分代码如下所示

Classifier m\_classifier = new J48();

File inputFile = new File("train.arff"); // 训练语料文件

ArffLoader atf = new ArffLoader();

atf.setFile(inputFile);

Instances instancesTrain = atf.getDataSet(); // 读入训练文件

instancesTrain.setClassIndex(0);m\_classifier.buildClassifier(instancesTrain); // 训练

inputFile = new File("test.arff"); // 测试语料文件

atf.setFile(inputFile);

Instances instancesTest = atf.getDataSet(); // 读入测试文件

// 设置分类属性所在行号（第一行为0号，instancesTest.numAttributes()可以取得属性总数

instancesTest.setClassIndex(0);

double sum = instancesTest.numInstances(); // 测试语料实例数

double right = 0.0f;

// 测试分类结果

for (int i = 0; i < sum; i++) {

// 如果预测值和答案值相等（测试语料中的分类列提供的须为正确答案，结果才有意义）

if (m\_classifier.classifyInstance(instancesTest.instance(i)) == instancesTest.instance(i).classValue()) {

// 正确值加1

right++;

}

}

算法中用到的主要[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/roger__wong/article/details/_blank)

（1）Instances对象

一个Instances代表一张表，可以对应一个arff文件或者是一个csv文件，通过Instances对象可以取某一列的均值方差等，主要就是若干行记录的一个封装。

（2）Instance

一个Instance代表一行记录，换言之一个Instances的数据包含多个Instance。每个Instance会有一个特殊的ClassIndex，该列值代表该Instance属于哪一类，具体来说就是图一里面的Golf。

（3）Classifier接口

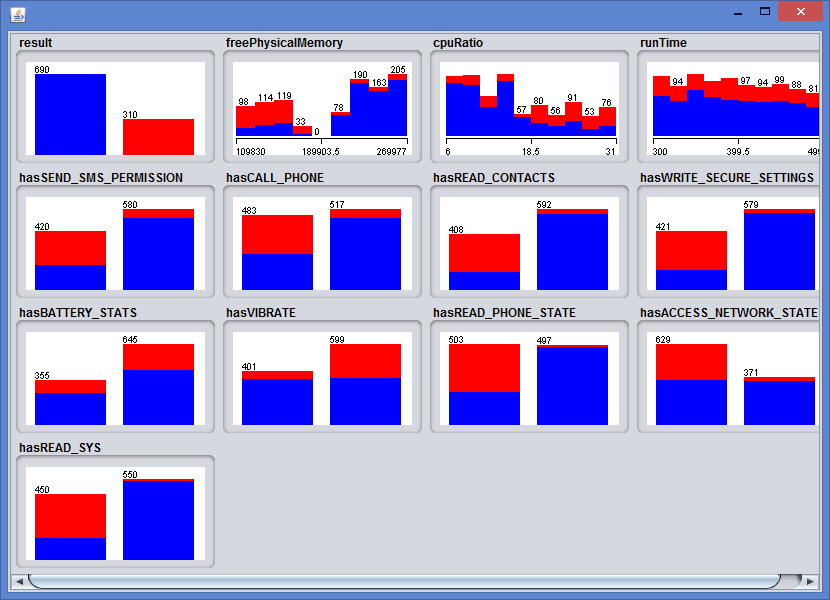
Weka中每一个分类器都继承与这个接口（虽然从意义上来说是个接口但其实是个子类），该接口提供一个buildClassifier方法传入一个Instances对象用于训练，还有classifyInstance方法用于传入一Instance来判断其属于哪个类。

（4）J48

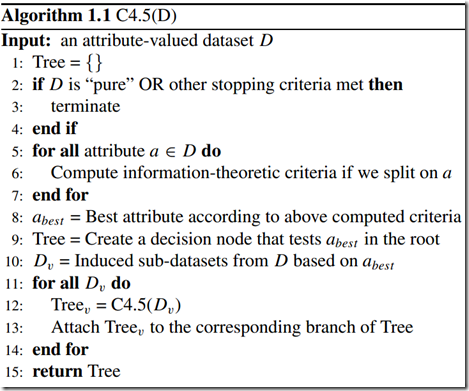
分类器主类，实现了Classifier接口。

（5）ClassifierTree接口代表树中的一个节点，维护和组成树的结构。其中J48用到C45PruneableClassifierTree和PruneableClassifierTree。

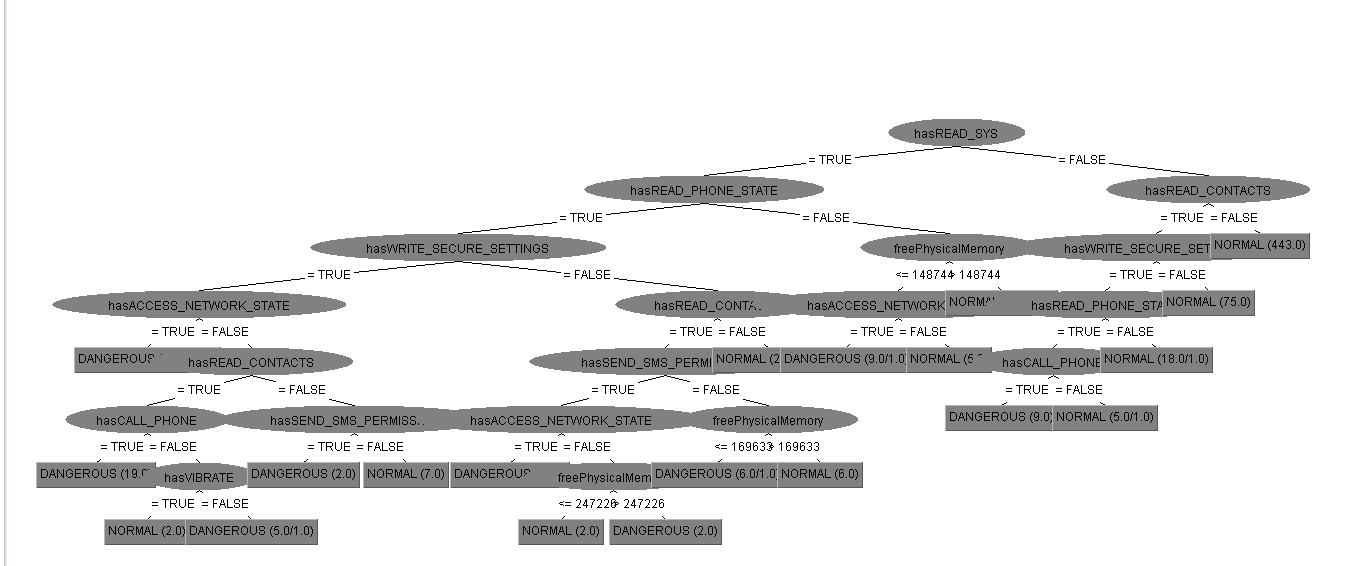
本次数据分析中，训练集由1000个样本组成，我们到网上查找了相关的论文并对他们的分析数据进行整理，构造成了训练集。以下为构造成的训练集的样本情况展示，其中左边表示拥有该项，右边表示没有该项。蓝色代表正常软件的个数，红色代表恶意软件的个数



J48决策树算法流程如下图所示:



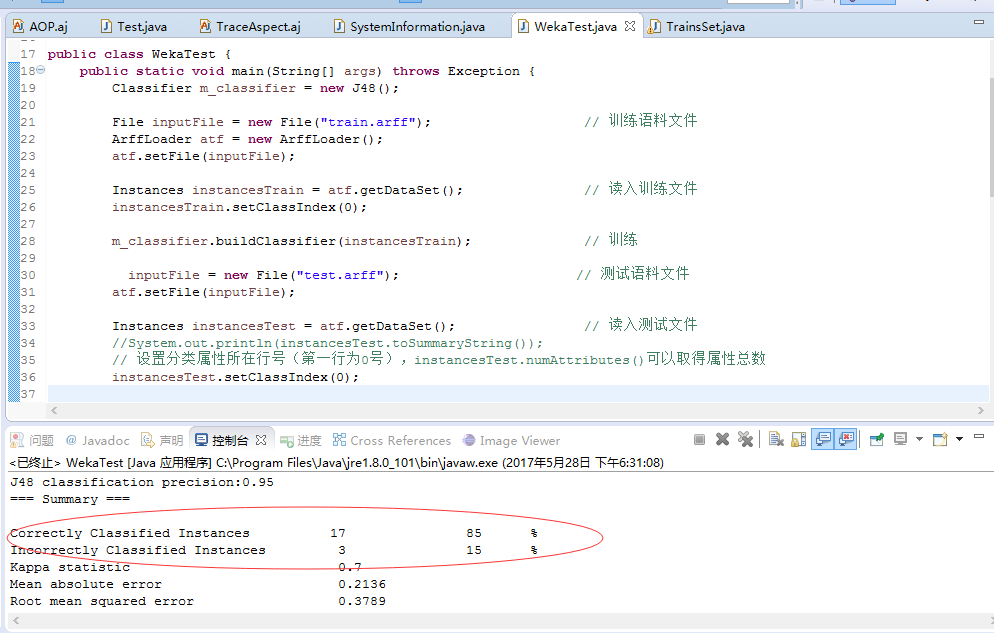
**本项目通过训练集的1000个样本由weka架包生成了j48决策树，接下来我们将用下图生成的决策树对测试集进行分析。**



**(2)决策树测试**

测试集由网上查找的20个项目通过上文中 的aspectJ切面代码进行切面获得，其中包含10个正常软件样本，10个恶意软件样本。本次在github上传了4个项目。

结果如下图所示：



其中17个预测正确，三个预测错误，正确率为85%，符合预期，模型建立成功！