目录

[**1.** **题目** 1](#_Toc505690097)

[**2.** **作者** 1](#_Toc505690098)

[**3.** **关键字** 1](#_Toc505690099)

[**4.** **摘要** 1](#_Toc505690100)

[**5.** **介绍** 2](#_Toc505690101)

[**6.** **方法** 3](#_Toc505690102)

[**6.1机器学习的流程** 3](#_Toc505690103)

[**6.2静态分析** 4](#_Toc505690104)

[**6.2.1 提取manifest文件的特征** 4](#_Toc505690105)

[**6.2.2 提取Dex中的信息** 4](#_Toc505690106)

[**6.3随机森林算法** 4](#_Toc505690107)

[**7.** **评估** 4](#_Toc505690108)

[**7.1数据集和特征集** 5](#_Toc505690109)

[**7.2检测性能** 6](#_Toc505690110)

[**7.2.1 分类效果** 7](#_Toc505690111)

[**7.2.2 与AV引擎相比** 7](#_Toc505690112)

[**7.3 运行时间性能** 8](#_Toc505690113)

[**7.4检测家族** 8](#_Toc505690114)

[**7.4.1 恶意家族分类** 8](#_Toc505690115)

[**7.4.2 分类测试** 9](#_Toc505690116)

[**7.4.3 小结** 10](#_Toc505690117)

[**8.** **讨论** 10](#_Toc505690118)

[**9.** **相关工作** 11](#_Toc505690119)

[**9.1基于权限组合** 11](#_Toc505690120)

[**9.2基于污点跟踪** 11](#_Toc505690121)

[**9.3基于机器学习** 11](#_Toc505690122)

[**10.** **总结** 12](#_Toc505690123)

[**11.** **参考文献** 12](#_Toc505690124)

1. **题目**

**Learning APK malware at scale**

1. **作者**

**Yunbin Yang, Guobin Huang, Wei Jiang**

1. **关键字**

**Machine Learning, Android Malware, Static Analysis**

1. **摘要**

随着智能手机的快速发展，极大的刺激了移动恶意软件的传播，特别是在Android这种流行的平台上，到2017第一季度为止，全球智能手机Android系统占有率已经达到85.0%[1]，截止到2017年中旬Android恶意软件的数量已经到达1,000万[2]，而通过纯手动分析恶意软件已经不能胜任这个数量的任务了，所以需要一种有效、能够快速地检测出恶意软件的方法，许多研究者也尝试借助机器学习方法[6-14]进行恶意软件的分类与检测，证明了机器学习的方法能够满足这个需求。~~本文基于Daniel Arp等人[31]的研究，改善了其基于SVM算法检测恶意APK的方法，~~本文旨在检测性能和运行时间性能上取得平衡，致使模型能够适用于时间性能要就较高的应用场景下。本文在Daniel Arp等人[31]的研究基础上，经过对大量的Android恶意软件的分析，使用静态分析的方法重新提取了特征集，并使用了效率更高的随机森林算法对特征集进行分类。结果表明，改善的方法在良性样本 21,203个，恶意样本 20,976个的实验中，分类器的准确率达到了97% 以上，并且检测的速度平均提高了50%。~~由于大量的应用程序使用混淆和加固的技术，这迫使我们进行更加深入的工作，在进行了大量手工分析和统计分析后，我们增加了N-gram算法对字节码进行特征提取，最后将准确率提高了到98%以上。~~

1. **介绍**

Android系统是目前市场上占有率最大的手机平台。由于其开源特性使得恶意软件的发展迅速，所以其安全问题也变得十分重要。很多Android恶意应用千方百计地利用Android平台安全机制的弱点来~~获取用户额隐私信息~~危害用户，如有些恶意应用会访问恶意网站或偷偷记录用户的隐私信息[3]，有些恶意应用后台运行在用户不知情的情况下发送短信[4]，更有甚者，一些恶意应用通过Android系统的漏洞可获取到超级用户权限[5]。

为了遏制恶意应用对Android生态环境的破坏,Google于2012年推出了 Bouncer服务。Bouncer首先根据后台中已知恶意软件、流氓程序、木马的特征数据对上传到Google Play的应用程序进行扫描,同时Bouncer还会将待检测的应用安装到在线的Android模拟器中,通过实时运行,动态地收集应用程序的行为特征，然后与已知恶意应用、木马进行比对。尽管Google在为确保应用程序安全可信的目标上做出了相当大的努力，但研究表明[35]，Bouncer服务仍然可以被绕过。

不仅如此，Android平台也提供了多种安全措施，加强恶意软件的安装，特别是Android权限系统。要在设备上执行某些任务（例如发送SMS消息），每个应用程序必须在安装期间或运行期间明确请求用户的许可。然而，许多用户往往盲目授予未知应用程序的权限，从而破坏权限系统的目的。因此，恶意应用程序实际上很难受到Android权限系统的限制。

因此，大量的研究已经研究了在安装之前分析和检测Android恶意软件的方法。这些方法可以大致分为使用动态和静态分析的方法。例如，TaintDroid[25]、DroidScope[22]、FLASKDroid[30]、Droidward [29]和Andromaly[11]等是可以在运行时监视应用程序行为的方法。尽管在识别恶意活动方面非常有效，但运行时监控却有很大的开销。相比之下，静态分析方法，如~~DroidAPIMiner~~ ~~[10]~~DroidMiner[10]、DroidSIFT[19]、Kirin [18]、 RiskRanker [9] 和Drebin[31]，通常只会产生一个很小的运行时间开销。

为了能够部署到防火墙等一些时间性能要求较高的平台上，我们选择了静态分析的方法。本文在Daniel Arp等人[31]的研究基础上，~~采用静态分析的方法，~~对Android应用进行反编译操作，提取权限、代码、字符串等特征，通过随机森林算法对实验样本进行分类。~~本文改善的地方在于：~~本文的贡献有以下几点：

1. 样本更加有效。

实验使用的样本在时间上和空间上都是均衡的，时间上收集于2010-2017年之间，空间上良性样本和恶意样本大体相等。良性样本不仅经过VirusTotal服务的筛选，还经过了如下载量、评分、开发者等其他因素的判断。并且最终我们训练模型的恶意样本到达了万级别，这在学术中比较罕见，大部分论文都是使用几百或者几千的样本。以上3点说明了我们的样本更加有效。详情在7.1.1节说明。

**~~良性样本：~~**~~与Daniel Arp等人[31]收集的10多万良性样本不同，本文是在普通应用商店下载，通过排名、评分、下载量等特征进行筛选后选取的良性样本，这样筛选出来的结果，可以将应用商店大部分恶意软件或者潜在的恶意软件剔除，防止良性样本中存在恶意样本，尽量减少良性样本被污染的程度。~~

**~~恶意样本：~~**~~本文使用的Android恶意样本达到万级以上，而公开的学术论文一般只使用千级甚至百级数量的样本作为训练数据。在小数据集中，如果特征数量大于样本数量时，很容易造成过拟合的情况，导致实验的结果与生产环境的结果有出入。~~

2. 对提取的特征进行了筛选。根据Daniel Arp等人的方法，在不进行任何特征提取的筛选下，我们提取了100多万数量的特征，这是因为提取了很多无用的信息，如Activity、Service、Content Provider和Broadcast Receiver等的名称，而这些名称是可以随意更改，如Felt等人[15]研究表明，不同的恶意软件可以通过对其包名、类名和方法名进行变形，来逃过恶意软件检测工具的检测。本文实验中使用的特征数量一共1,664个。

~~3. 与Daniel Arp等人的研究选取的支持向量机（SVM）方法不同，本文采取了随机森林的机器学习方法，其优势在于：[28]~~

* ~~算法具有很好的抗噪能力，稳定性好。~~
* ~~速度更快，并且可以并行计算。~~
* ~~可以处理高维数据，算法通过投票决策。~~
* ~~实现比较简单，易于理解。~~

~~4. 效率更高。基于2和3的优点，由于需要提取的特征更少并且使用了速度更快的随机森林算法，致使本文的检测时间控制在更短。在普通PC的计算力下，单线程能够达到平均每个文件0.56秒检测出结果。~~

~~5. 准确率更高。基于1和2的优点，我们的实验样本更加真实并且提取的特征集更加符合实际情况，所以本文的准确率也更高。实验结果表明，在良性样本 21,203个，恶意样本 20,976个中，分类器的准确率达到了97% 以上。~~

3.本文对模型进行了大量的实验分析，评估了模型的运行时间性能和对恶意软件的检测性能，最后对未知恶意家族的识别进行了分析。

4.众所周知，使用混淆和加壳技术的恶意程序是静态分析的一大克星，我们的分类器也不例外。在文末我们对该情况进行了大量手工分析和统计分析，并尝试提出解决方法。

1. **方法**

为了更加高效地检测Android手机上的恶意软件，我们使用了应用广泛的静态分析方法，从不同的数据源提取特征，并且使用随机森林算法进行建模，最后对模型进行了评估，该节具体安排如下：

1.机器学习的流程：介绍了我们使用机器学习算法检测Android恶意软件的流程。（第6.1节）

2.静态分析：介绍了我们从Android应用程序中提取的manifest和dex code特征集。（第6.2节）

3.随机森林算法：简单介绍了随机森林算法。（第6.3节）

**6.1机器学习的流程**

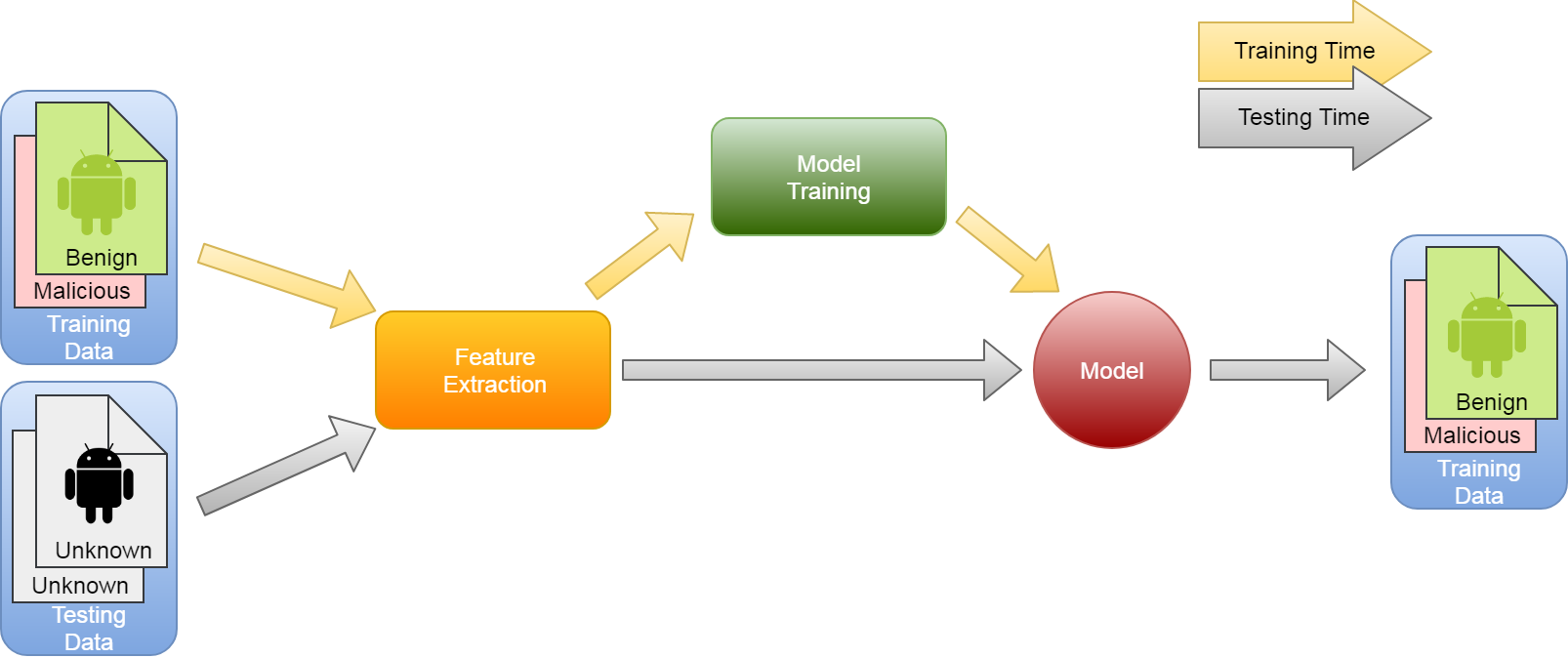
****

Figure 1. Schematic depiction of the Machine Learning steps performed

当我们创建一个Android恶意软件检测模型时，首先我们需要收集大量的各种恶意和良性应用程序。然后，从每个应用程序中提取各种特征，我们将在6.2节介绍相关的特征工程。最后，模型的训练是将所有这些特征提供给它，并选择合适的机器学习算法，使得模型能够找到对数据有很好的拟合。这样，当一个未知标签的应用程序的特征被呈现给模型时，它可以返回这些特征与恶意和良性集合的相似程度的置信度分数。图1很好的描述了我们模型的训练和预测过程。

**6.2静态分析**

我们对实验中的Android应用程序执行轻量级的静态分析，这是非常重要的一步。我们主要使用了应用程序的manifest文件和反汇编的Dex代码，通过线性扫描的方式对这些文件进行特征提取。为了分析过程的通用性和扩展性，我们首先将所有提取的特征表示为字符串集，例如权限、Intent和API调用。我们一共提取了以下5组字符串。

**6.2.1 提取manifest文件的特征**

每个Android应用程序中都必须包含一个名为AndroidManifest.xml的manifest文件，开发者使用该文件配置应用程序运行所必要的硬件、组件和权限等相关信息。我们可以使用开源的工具Androguard[27] 提取这些信息，包括：

**F1 硬件组成：**如果应用程序请求访问相机、触摸屏或智能手机的GPS模块，则需要在AndroidManifest.xml文件中声明这些硬件功能。当应用程序要求访问某些硬件组合时，通常会反映出有害的行为。例如，有权限访问GPS和网络模块的应用程序能够收集用户的位置数据并通过网络将其发送给攻击者。

**F2 请求权限：**在Android中引入的最重要的安全机制之一是权限系统。在Android 6.0之前，权限在安装时由用户主动授予，并允许应用程序访问与安全相关的资源。以前的工作[7,13]表示，恶意软件倾向于比无害应用程序更频繁地请求某些权限，但是本文的研究表明，最新的应用程序相比恶意软件申请了更多的权限，但是这并不妨碍我们提取这些权限作为特征。例如，当前恶意软件的大部分发送短信消息，需要请求SEND\_SMS权限。因此，我们使用manifest中列出的所有权限作为特征。

**6.2.2 提取Dex中的信息**

Android应用程序是用Java开发的，并被编译成Dalvik虚拟机的字节码。通过使用Androguard[27]工具可以高效地反汇编该字节码，并且获取应用程序中使用的API调用和数据信息等。我们使用这些信息来构建以下特征集：

**F3 受限制API调用：**Android权限系统会限制应用程序对一系列关键API调用的访问。 我们在反汇编代码中搜索这些调用的出现，并且将函数记录作为特征集。并且找出这些API对应的应用权限。

**F4 匹配API和权限：**F3中提取了受限制的API调用，而F4这个特征是由Felt等人[15]介绍的方法——匹配API调用和权限。当一个恶意软件使用了没有声明所需请求权限的API调用时，这可能表明恶意软件正在使用root漏洞，以越过Android平台的限制。

**F5 可疑API调用：**当应用程序访问智能手机的敏感数据或资源时，需要调用一些特定的API，这些API的调用可能导致恶意的行为，所以我们将这些API提取出来作为一组特征集。其中，提取的该特征集不与F3有任何交集。

**6.3随机森林算法**

随机森林算法通过重采用(bootstrap)方法，从样本里有放回地重复抽取个样本生成新的样本，然后重复以上步骤，生成个决策树，这些决策树之间是独立的，新分类的分类结果是根据分类树投票多少来决定的。

~~随机森林算法流程如下：~~

~~1.从样本空间有放回地随机采样选出个样本。~~

~~2.从所有提取出的特征中随机选择个特征，对选出的样本利用这些特征建立决策树。~~

~~3.重复前两步次，则生成棵决策树，即形成了随机森林。~~

~~4.对于测试数据，经过每棵决策树进行决策，然后通过投票决定分到哪一类。~~

随机森林算法的性质决定了它具有很好的抗噪能力，稳定性好，并且计算速度上比SVM算法更快。研究表明[36]，在实验样本均衡的二分类问题上，随机森林和支持向量机算法的泛化能力无显著差异，并且使用随机森林算法无需对数据集进行预处理。

在大量实验数据下计算速度更快和无需对数据集进行预处理2个随机森林算法的优点能够提高整个Android检测系统的速度，所以我们选择使用随机森林算法来训练我们的模型。

1. **评估**

详细介绍了我们的方法之后，我们实现了该方法并且对其性能进行了评估，为此，我们四个实验：

1. 数据预处理。首先，我们对实验中的数据进行了说明，并且对提取出来的特征集进行了分析和可视化。
2. 检测性能。其次，我们评估了模型在良性样本 21,203个，恶意样本 20,976个的数据集上的检测性能。并且将其与Drebin 和常见的反病毒厂家进行了对比。
3. 鲁棒性。接着，为了验证模型的鲁棒性，也就是对未知恶意软件的识别能力，我们对实验的数据集进行恶意家族分类，然后按照不同恶意家族的类别对模型进行评估。
4. 运行性能。最后，我们评估了模型在普通PC机上的检测效率。

**7.1数据集和特征集**

首先，我们对实验中的数据集和特征集进行了说明，接着我们对特征权重进行了计算并且尝试对top10的特征进行解释，最后我们可视化了实验中的特征集，更加直观的对特征集进行分析。

**7.1.1 说明**

**数据集：**对于实验所需要的数据集，我们希望能够收集到真正的Android应用程序和真正的恶意软件。最终我们收集到了包含了良性和恶意的154,113个应用程序，详细数据包括112,424个来自不同应用市场的样本和41,689个来自github项目、各安全论坛和博客以及VirusTotal网站的恶意样本。

为了确定良性软件，首先我们获取了应用市场良性样本的下载量、排名、评分和所属开发者等信息，然后对这些信息进行帅选，从而选出21,203个真正的良性软件。为了确定恶意软件，我们将每一个恶意样本发送到VirusTotal服务，并将4个以上的厂家的反病毒引擎扫描出恶意行为的程序标记为恶意样本。

为此，我们认为我们已经几乎正确地将数据集分成良性和恶意样本。最终模型训练的数据集包括良性样本 21,203个，恶意样本 20,976个。

同时，本文还收集了Drebin使用的5560个恶意样本库。然而我们只是拿Drebin恶意样本库来评估我们的模型，并不会在训练样本中使用它们。(详情见7.2.3)

**特征集：**根据我们第6节描述的方法论，从良性和恶意样本中静态分析提取的各类特征集总数如表1，F-ALL为所有特征集的总和，一共1,664个特征。从表中，我们发现请求权限F2有1,039个，而Android官网公开的系统权限只有150个[34]。深入分析，我们发现主要是由于某些应用程序使用了自定义的权限和使用权限不规范所造成，如大小写、拼写错误等。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Table 1 The number of features | | | TYPE | NUMBER | | F1 | 147 | | F2 | 1039 | | F3 | 346 | | F4 | 59 | | F5 | 73 | | F-ALL | 1664 | | |  |  | | --- | --- | | Table 2 The top10 of features | | | 1 | F5\_Landroid/util/Base64;decode | | 2 | F2\_android.permission.READ\_EXTERNAL\_STORAGE | | 3 | F2\_android.permission.CAMERA | | 4 | F3\_Landroid/provider/Settings$System;putString | | 5 | F5\_Landroid/view/KeyEvent;getDeviceId | | 6 | F4\_android.permission.BLUETOOTH | | 7 | F5\_Ljavax/crypto/Cipher;getBlockSize | | 8 | F2\_android.permission.RECORD\_AUDIO | | 9 | F2\_android.permission.SEND\_SMS | | 10 | F2\_android.permission.RECEIVE\_USER\_PRESENT | |

**7.1.2 特征权重**

首先，为了更好的理解提取出来的各类特征在整个特征集中的权重，我们使用卡方检验[32]对F-ALL特征进行特征选择计算，将计算结果的评分提取出来并进行统计分析。统计分析如下，从表3看出，F-ALL一共1664个特征数，前430个特征占的特征权重为99.00%，相当于，25.84%的特征是最重要的。另外，F3选取的特征数最多，而F2占的权重最大，F5每一个选取的特征的权重平均最高，F5特征占比最高，相当于F5的每个特征都非常重要。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table 3 The weight of features | | | | | |
| Type | NUM-1 | WEIGHT | AVG(WEIGHT/ NUM1) | NUM-2 | NUM1/NUM2 |
| F1 | 33 | 3.81% | 0.12% | 147 | 22.45% |
| F2 | 141 | 32.99% | 0.23% | 1039 | 13.57% |
| F3 | 171 | 31.36% | 0.18% | 346 | 49.42% |
| F4 | 32 | 10.24% | 0.32% | 59 | 54.24% |
| F5 | 53 | 20.61% | 0.39% | 73 | 72.60% |
| F-ALL | 430 | 99.01% | - | 1664 | 25.84% |

表2 是列出来的权重前10的特征，其中最重要的特征是F5\_Landroid/util/Base64，可能的原因是恶意软件很少使用系统自带的Base64算法类，以此逃过反病毒厂家对恶意软件字符串的解密，而良性软件没有这种顾虑。特征F2\_READ\_EXTERNAL\_STORAGE是用来申请读取外部存储资源的权限，特征F2\_SEND\_SMS是用来申请发送短信的权限， F5\_Landroid/view/KeyEvent 该类是用来监控按键事件。以上特征都属于比较敏感的特征。

**7.1.3 可视化特征集**

为了更好的观察各类特征的分类能力，我们使用PCA(Principal component analysis)算法[33]将各类特征映射成二维数据，图3是我们随机地从实验样本中选取了良性样本和恶意样本各200个，并使用PCA算法计算得到降维后的数据绘制而成。从图2可以看出，降维后的特征中，F1的特征分类效果并不是很明显，主要是因为恶意软件很少使用硬件组成，而导致降维后大部分显示的是良性软件的特征。其他类型的特征使用PCA降维后都能找到明显的分类曲线。

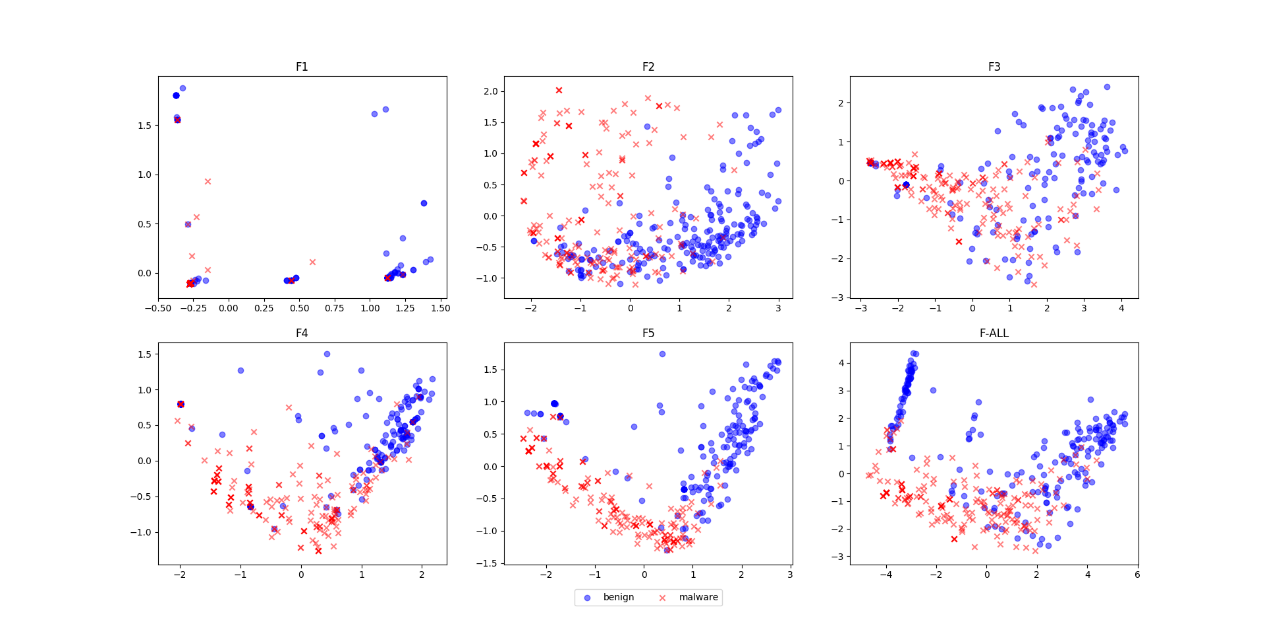


Figure 2 the 2D plots of the feature sets for both benign and malicious samples when using PCA dimensionality reduction

**7.2检测性能**

我们评估了模型的检测性能。对于这个实验，我们将数据集随机分成一个已知的训练组（70％）和一个未知的测试组（30％）。我们的检测模型及其参数在已知的训练组上确定，而未知的测试组仅用于测量最终检测性能。我们重复这个过程10次并取平均结果。接着我们列出了常见三种机器学习算法对5类特征集的检测性能，并进行了比较。最后我们将模型与常见的10个厂家的反病毒引擎进行了对比。

**7.2.1 分类效果**

本节使用了SVM算法、KNN算法和随机森林算法对各特征集进行了分类测试，结果如表3所示：

|  |  |  |  |
| --- | --- | --- | --- |
| Table 4 Detection rate of difference algorithm | | | |
|  | ACCURACY | | |
| TYPE | SVC | KNN | RF |
| F1 | 69.74% | 70.30% | 70.67% |
| F2 | 91.46% | 91.54% | 90.56% |
| F3 | 92.23% | 93.77% | 93.33% |
| F4 | 85.45% | 90.39% | 90.78% |
| F5 | 92.08% | 94.12% | 94.20% |
| F-ALL | 97.28% | 96.97% | 97.40% |

从表4可以看出，F1的分类效果是最差的，从7.1.3节中的图2也可能看出这个结论，恶意软件很少使用硬件组成信息，所提供的特征非常少，所以效果也比较差。而其他的特征基本都能够达到90%以上的分类准确率，特别是使用所有特征作为分类特征的时候，准确率到达了97%以上。

我们还通过分类器的ROC曲线检测了其分类效果，图3是使用随机森林算法分类产生的ROC曲线图。结果表明，在不同的阈值下，模型检测效果都非常好的。

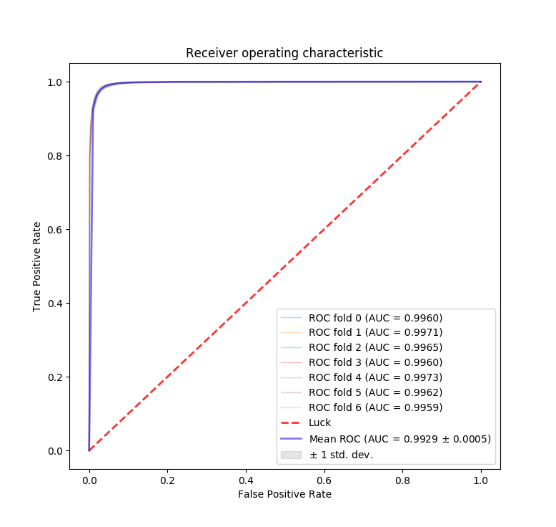
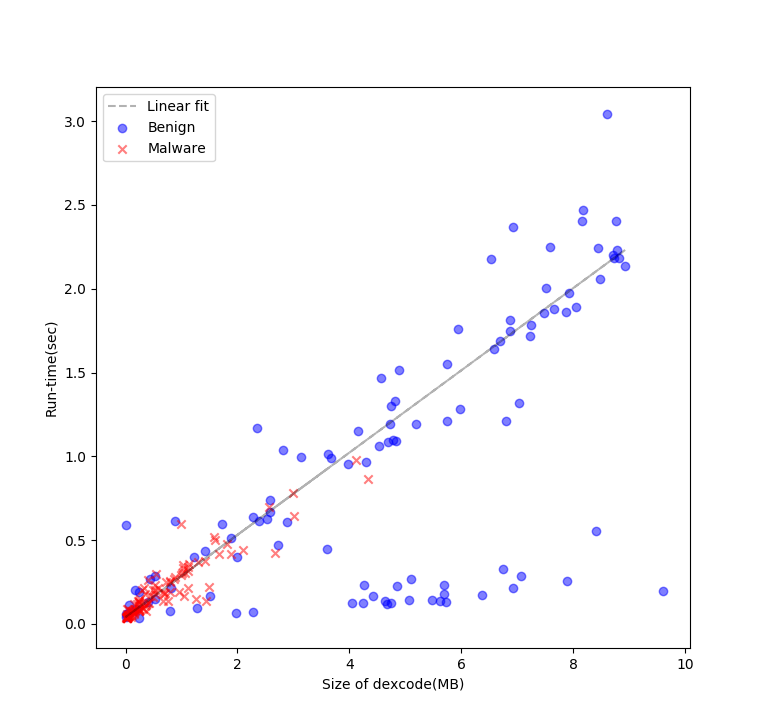
 ****

Figure 3. The ROC curve of the classifier Figure 4. Run-time performance

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~Table 5 Detection rates of our model and anti-virus scanners.~~ | | | | | | | | | | |
| ~~Model~~ | ~~AV1~~ | ~~AV2~~ | ~~AV3~~ | ~~AV4~~ | ~~AV5~~ | ~~AV6~~ | ~~AV7~~ | ~~AV8~~ | ~~AV9~~ | ~~AV10~~ |
| ~~97.32%~~ | ~~94.57%~~ | ~~93.82%~~ | ~~93.16%~~ | ~~88.68%~~ | ~~88.01%~~ | ~~83.97%~~ | ~~79.68%~~ | ~~62.07%~~ | ~~54.92%~~ | ~~24.75%~~ |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 5 Detection rates of our model, Drebin and anti-virus scanners. | | | | | | | | | | | | |
|  | Model | Drebin | AV1 | AV2 | AV3 | AV4 | AV5 | AV6 | AV7 | AV8 | AV9 | AV10 |
| Our sets | 97.32% | - | 94.57% | 93.82% | 93.16% | 88.68% | 88.01% | 83.97% | 79.68% | 62.07% | 54.92% | 24.75% |
| Drebin sets | 99.81% | 94.00% | 83.80% | 99.38% | 98.67% | 94.67% | 99.03% | 96.02% | 97.87% | 96.18% | 98.09% | 79.85% |

**7.2.2 与Drebin模型相比**

由于我们并没有Drebin的模型，所以我们只能比较相同数据集下模型的准确度。为此我们拿到了Drebin所使用的5,560个恶意样本，在清理了与我们测试数据集重复的794个样本后，还剩下4,766个待测试样本。如表5的所示，使用我们的模型检测的结果准确度达到了99.81%，只有9个恶意样本是我们模型不能检测出来的。相比Drebin 94%的准确度，我们的模型大约提高了6%的准确度。

**7.2.3 与AV引擎相比**

虽然在上面的实验中我们获取了非常好的效果，但是为了更加突出我们模型的识别率，并且在现实中能够与常见的反病毒引擎进行竞争。因此，我们还将模型与著名的十个反病毒引擎进行比较。每一个反病毒引擎的检测结果都是在VirusTotal中获取的。

表5表示了我们的实验结果。从表中可以看出，对于我们的数据集，反病毒引擎的检测率差别非常大。在十个反病毒引擎中，准确率最高的是94.6%，而有的检测率非常低，只有24.8%，可能是该引擎对Android恶意程序不敏感。我们的模型的准确率97.3%，优于所有的反病毒引擎。

而对于Drebin数据集，反病毒引擎检测的结果非常好。非常可能的一个原因是Drebin样本集比较旧(收集于2010-2012年)，大部分的恶意标签都已经被记录到反病毒引擎的病毒库中。其中8个反病毒引擎的检测结果在95%以上，最低的也有79.85%。虽然如此，我们模型达到了99.81%的准确度，还是优于所有的反病毒引擎。

**7.3检测家族**

为了测试模型对未知样本的识别能力，我们还对恶意样本进行了家族分类测试，来评估机器学习在不同情况下的识别效果。

首先需要对恶意样本家族进行分类，我们还对恶意样本进行了家族分类测试，来评估机器学习在不同情况下的识别效果。本文使用开源的恶意样本家族分类软件AVClass[16]，通过将恶意样本的VirusTotal报告输入到AVClass中，AVClass将输出一个最有可能的恶意软件家族标签，该标签即代表恶意样本所属的家族。AVClass在大样本下的家族分类准确率高达88%。

接着进行机器学习分类测试，本章使用了标准的随机森林进行测试，并且根据样本家族信息对恶意样本不同程度的切分，分为4种情况，分别是：T1-普通分类测试、T2-前10的家族不提供训练样本、T3-前20的家族不提供训练样本、T4-前20的家族特供10%的训练样本。其中T1是作为参考的对照组。

**7.3.1 恶意家族分类**

使用AVClass对恶意样本进行家族分类，20,976 个恶意样本一共分为454个家族，其中有869个恶意样本AVClass不能识别其家族。表7按照各家族样本数量进行统计并排序，列出了前20个家族的信息，这些家族的恶意样本数量一共 15,997 个，占比为76.26%。

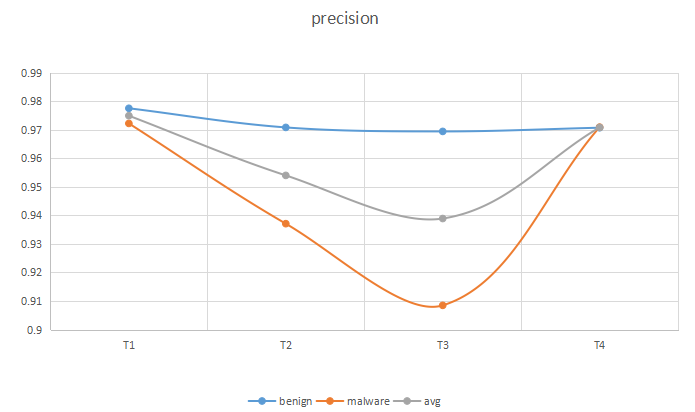
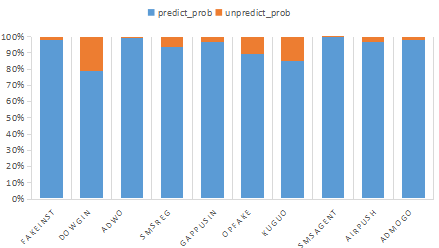
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table 7 ranks the top 20 malicious families | | | | | |
| index | family | count | index | family | count |
| 1 | fakeinst | 4444 | 11 | plankton | 369 |
| 2 | dowgin | 1878 | 12 | smspay | 355 |
| 3 | adwo | 1491 | 13 | youmi | 326 |
| 4 | smsreg | 1083 | 14 | mobwin | 306 |
| 5 | gappusin | 1044 | 15 | utchi | 305 |
| 6 | opfake | 991 | 16 | umpay | 266 |
| 7 | kuguo | 988 | 17 | lotoor | 250 |
| 8 | smsagent | 471 | 18 | wooboo | 222 |
| 9 | airpush | 441 | 19 | mseg | 193 |
| 10 | admogo | 384 | 20 | secapk | 190 |

在该测试样本中，Fakeinst恶意家族是包含最多恶意样本的家族，该家族在2012-2015年之间都是数量最多的家族，主要伪装成主流应用程序，然后后台控制手机发送扣费短信，并且会隐藏图标，而Zhou等人[17]的研究，他们的样本中有86%属于这个家族。之后的像airpush、smsagent、gappusin都是比较著名的恶意家族，这些家族的主要行为是数据窃取、恶意下载、恶意扣费等。

**7.3.2 分类测试**

对恶意样本家族进行分类后，接着我们使用随机森林算法对T1-T4的4种情况分别进行了测试。

图5是4次测试中输出的准确率，蓝色是指良性样本的准确率，橙色是恶意样本的准确率，灰色是平均的准确率。在T1-T3的使用过程中，我们是逐步减少了恶意家族的测试集，导致每次实验结果的准确率都有不同程度的下降，而在T4时，我们增加了10%的前20的恶意家族样本，其准确度又回到了T1的情况。

|  |  |
| --- | --- |
| Figure 5 the accuracy of T1-T4 | Figure 6 the classification of the TOP 10 families in T2 |

**T1——**普通测试

首先，我们将样本分成70%的训练组和30%的测试组，测试结果从图5的T1可以看出，该算法对样本的分类达到97%以上。

前20的恶意家族分类测试效果图7-A，绝大部分的恶意家族在提供70% 样本作为训练组时，识别的准确率都能达到95%以上，不过secapk家族的识别率只有71.19%，可能的原因是其训练的数量较少，导致识别的准确率下降，我们将在第8节中讨论该家族。

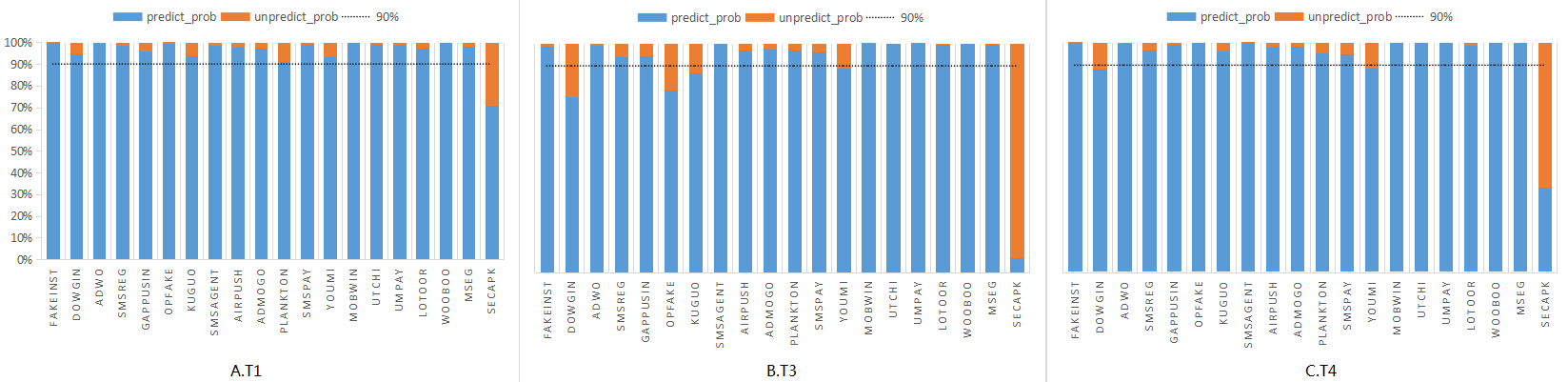


Figure 7 the top 20 families of classification results in T1, T3, T4

**T2——**前10的家族不提供训练样本

前10的家族不提供训练样本，全部作为测试组，其他恶意样本作为训练组，随机选取同等数量的良性样本作为测试组，剩余良性样本作为训练组，准确率如图5的T2，下降到了95%，当然测试组样本数量的减少也会对准确率造成一定的影响。分析其中的原因，主要是恶意样本的召回率（recall）下降了，说明部分恶意样本被识别成了良性样本。

前10的恶意家族分类效果如图6，这些家族的样本都没有拿去作为训练组，全是测试组，相当于评估模型对未知样本的识别能力。同T1相比，各家族的识别率都有不同程度的下降，其中dowgin家族的识别率下降了15.9%。

**T3——**前20的家族不提供训练样本

同T2，该节将家族的数量从10个增加到20个，也就是说，将前20的家族都不提供训练组样本进行评估分析，其准确率相比T2又降低了1.5%。在新样本或者新家族越来越多的情况下，如果不更新模型的话，我们可以预期到，准确率将会越来越低。

前20的恶意家族分类测试效果如图7-B，有5个恶意家族的识别率降低到90%以下，并且secapk家族识别率只有不到7%。

**T4——**前20的家族提供10%的样本

与T3不同，T4提供了前20家族随机抽取的10%的样本作为训练组。结果表明，相比完全不提供测试样本，提供10%的样本使得分类的准确度从94%提高到97%，而恶意样本的准确率从91%提升到97%。

前20的恶意家族分类测试效果如图7-C，相比T3，大部分恶意家族的识别率都达到90%以上，跟T1的结果相差无几，而且secapk家族识别率提升到37%。

**7.3.3 小结**

通过4种方式，将样本按照家族类型进行分组，使用默认参数的随机森林算法，对分组的样本进行分类识别。结果表明，在不提供相应家族的样本下，这些家族的识别率会大幅降低，而只需提供10%的该家族的样本，就能得到普通分类测试那么高的识别率。

虽然完全不提供前20家族的样本时导致这些家族的识别率大幅降低，但是也还有90%左右的识别率，说明我们的特征提取的效果不错。

在可预知的未来，如果不更新模型，由于新恶意样本新家族的出现，模型的识别率必然降低，为了应对这个情况，我们需要及时更新模型，而且只需提供极少数的新样本就能大幅提供模型识别率。

**7.4 运行时间性能**

为了更好地体现模型的效率，并且为以后能够更好的优化模型，本节我们统计分析了模型检测时的运行时间。我们从样本中随机抽取100个良性样本和100个恶意样本进行统计分析，主要记录了这200个样本检测过程中的运行时间和样本中dexcode的大小。我们测试的硬件配置是：CPU i5和8G内存，并使用了~~单线程~~单核进行运行。

其中，图4是样本的dexcode大小和运行时间的散点图，横坐标是dexcode大小，纵坐标是运行时间，蓝色是指良性样本，红叉是指恶意样本。从图中可以发现，样本的dexcode大小与检测时的运行时间成正相关，也就是说，当样本的dexcode越来越大，其花费的检测时间也会越来越多。其次，我们可以从图中看到有部分良性样本的dexcode大小很大，但是其检测时间却很少，不符合正相关关系，我们通过手动分析发现，这些样本的dexcode通过一些厂商的安全服务进行了加~~固~~壳，使得加~~固~~壳前的dexcode不能通过静态分析的方法进行解析，只能解析进行解密的代码，而这些解密代码占总的 dexcode大小是很小的，所以出现了部分偏离正相关的数据。最后，我们还发现恶意样本大部分分布在图中的左下角，而良性样本分布比较均匀，表明恶意样本的dexcode大小整体比较小，一方面恶意样本的功能相比良性样本会比较少，另一方面小体积的恶意样本更容易传播。

Table 6 Detailed run-time analysis of model

|  |  |  |
| --- | --- | --- |
|  | AVG Run-time(sec) | The number of samples detected daily |
| Benign | 0.936 | 92307 |
| Malware | 0.194 | 445360 |
| AVG | 0.565 | 152920 |

表6中，我们统计了样本检测时的平均运行时间和评估了模型一天检测样本的数量，从表中得知，良性样本的平均检测时间远超过恶意样本的，也说明了良性样本的dexcode相比恶意样本会比较大。Drebin模型平均每个样本需要0.75s的时间进行检查，预估一天能够检测100,000个样本。我们的模型平均每个样本需要0.57s的时间进行检查，平均一天能够检测150,000左右的样本。相比之下，我们的模型提高了50%的效率。

1. **讨论**

本文验证了使用机器学习算法静态检测Android恶意软件中的可行性，但是由于某些限制和研究时间上不足的原因，导致未来还有很多工作需要去做。该节我们讨论静态分析方法在遇到应用程序被混淆和加壳的情况下的检测能力。

**secapk家族：**因为实验中secapk家族的检测率不高，我们随机抽取了几个该家族的样本，手动去分析了这些样本的行为和特征。首先，我们分析了secapk家族命名的原因，这些恶意样本都是经过梆梆加固进行加壳过，而加壳后会出现secapk的类名，所以AVClass工具对这类样本命名为secapk家族，而加固的后的Dex代码都被加密了，静态分析提取的特征非常有限，导致模型对secapk家族的识别率不高。

**新的样本：**另外，我们从其它途径收集了8700个新的恶意样本，该样本集导致我们的模型准确率从97%下降到94%。初步分析表明，该数据集存在大量加壳的行为造成模型识别率的下降。当然这个是所有静态分析所面临的问题，而不只是我们模型。但是我们还是希望能够使用静态分析的方法去解决这个问题。

基于以上两~~点~~个方面，致使我们迫切的需要进一步的工作去分析加壳的应用程序，并且为了提供模型对加壳程序的识别，我们需要提取更加行之有效的静态特征。为此，我们做了以下尝试：首先，我们使用类似于7.3节的方法是统计了加壳程序对我们模型的影响。其次，我们尝试手工分析了部分加壳的应用程序。最后，我们尝试增加N-gram算法提取Dalvik虚拟机的字节码的特征去分类加壳的应用程序，并且增加新样本后模型的准确度从94%提高到98%。

由于篇幅原因，该节我们只是讨论为何N-gram算法对加壳恶意软件的分类行之有效。在我们的统计分析中，发现大部分安全厂家在提供对APK的加壳服务时，都会对该应用程序进行扫描，并且拒绝为恶意程序提供加壳服务。所以，在我们的样本中，只有5%的恶意软件使用了安全厂商的加壳服务，而良性样本使用了加壳服务的占了35%。一般的加壳过程都会使用Android动态加载的方法，具体就是使用DexClassLoader类加载原来的dex代码。当然，不只是恶意程序会使用动态加载的方法，良性程序也会使用。在我们的样本中，剔除使用了安全厂商加壳服务的程序，其中33%的良性样本和31%的恶意样本都使用了动态加载的方法。

不过很明显的是，他们使用动态加载的方法的目的不同。良性软件是为了实现插件化的开发方式或者由于项目巨大，需要多个classes.dex文件进行开发，而恶意软件是为了躲避反病毒引擎的检测才使用的动态加载方法。因此，恶意软件在动态加载恶意的dex文件前，会对其进行解密操作。一般相同家族的恶意软件，会使用相同的加密方法。所以，我们认为使用N-gram算法去分析字节码能够提取恶意样本解密恶意dex文件的行为特征，配合文章前面的静态分析方法，我们得到了很好的效果，模型的准确提高到了98%。

1. **相关工作**

Android恶意软件的分析和检测在过去几年中一直是一个生动的研究领域。已经提出了几个概念和技术来对付这种恶意软件的日益增长的数量和复杂性。Felt等人[15]和Zhou 等人[17]的研究提供了当前恶意软件的概况。

**~~9.1基于权限组合~~静态分析**

Android程序安全性的静态分析主要是指，将程序的apk包进行反编译操作，以便在不实际运行程序的情况提取特征。通常的提取特征如权限、代码、字符串等。进一步的，静态分析方法还能被用作恶意数据流分析。

基于权限的检测工具Kirin[18]是早期有代表性的基于权限的静态分析方法之一,它通过定义的一组规则识别危险的权限组合，它提供9条安全规则作为检测标准。这些安全规则是一些恶意应用族，频繁使用的高危权限集合。由于规则数量少，代表性不高，因此它存在检测率低下的问题。RiskRanker [9]旨在通过评估由不受信任的应用程序引起的潜在安全风险来识别0day安卓恶意软件。它筛选来自Android市场的大量应用程序，并检查它们以检测某些行为，如加密和动态代码加载，这些行为形成恶意模式，可用于检测隐形恶意软件。Wu等人[14]提出的DroidMat工具利用静态分析，获取Android软件的权限信息，组件信息以及API信息，作为软件的特征,并通过分类算法进行分类，得到分类模型，从而以此检测未知apk。

静态分析方法还被用作分析数据泄露和恶意数据流。如Hoffmann 等人[23]提出了一个适用于Android应用程序的静态Android分析框架SAAF，该框架在分析smali代码并且创建程序片后，能够执行数据流分析以及回溯给定方法使用的参数。能够自动方式识别可疑的代码段。该框架还实现了控制流程图的可视化和广告相关的代码识别。Arzt等人[24]提出了一种名为FlowDroid的分析工具，通过静态污点分析来检测Android上敏感信息的泄露。与动态方法不同，该静态分析的方法不会将运行时间开销引入目标应用程序的正常执行，并且可以在应用程序分发给用户之前检测到信息泄露，因此恶意应用程序可以在用户下载之前从市场上移除。DroidMiner[10]、DroidSIFT[19]等基于FlowDroid工具实现了通过Android应用程序的API控制依赖和相似性检测恶意程序的功能。基于数据分析的污点跟踪技术近年来有很多人进行了研究，虽然它能够对Android应用程序进行更详细的分析，但是其运行性能也是大大折扣的。例如DroidMiner 平均需要19.83s来检测一个Android应用程序，更进一步的，DroidSIFT 检测每个应用程序平均需要175.8s。

**~~9.2基于污点跟踪~~动态分析**

动态分析通常涉及程序的实时运行，通常在虚拟机或者受保护的环境中。明显的，动态分析比静态分析会消耗更多的时间性能。

Yang等人[29]提出了DroidWard 动态分析系统，并使用了6个新的动态特征，通过自动监控运行在虚拟机的应用程序，捕捉其特征并进行判定。运行时间需要花费120s每个应用程序。

Enck 等人[25]为了解决Android系统本身不能提供个人隐私数据的可视性，而提出了一个高效的系统级动态污点跟踪和分析系统TaintDroid，通过利用Android的虚拟执行环境进行实时分析，能够同时跟踪多个敏感数据源。TaintDroid在CPU绑定的微型基准测试中~~只~~会导致32％的性能开销~~，并且在交互式第三方应用程序上的开销可以忽略不计~~。

动态分析对混淆和加壳的应用的检测也是非常不错的，但是其性能却不是普通智能手机能够接受的，所以将动态分析的部分功能放到云端似乎是可行的。ParanoidAndroid [37]采用智能手机的虚拟克隆，在云中并行运行，并重播设备的活动。然而，即使最小的执行轨迹实际发送到云端，这仍然对电池寿命造成不可忽视的损失。

~~在各种Android污点跟踪和分析系统中，都需要手工标记大量的sources 和sinks 的Android API方法，而这些API名单非常难获得，因此Rasthofer 等人[26]提出了一种新颖的机器学习指导方法，能够自动识别任意版本的Android API中的sources和sinks。~~

**9.3基于机器学习**

为了弥补静态和动态分析工具的不足，机器学习的技术也被应用到了Android恶意程序的检测。

Moghaddam等人[20]提出了一种敏感性静态分析方法，这种方法改进了基于分类的Android恶意软件检测技术，通过敏感性静态分析方法，分析APK中的静态特征。这种方法在对字节码文件分析后，抽取manifest特征和类特征并按组划分每类特点，这样能找到在分类精度和分析开销上最优的分类特征，从而按照这种最优特征进行分类学习并检测恶意软件。Droidmat [14]使用API​​调用跟踪和显示文件来学习恶意软件检测功能。R.Dhaya等人[21]提出了一种基于N-gram机器学习算法的恶意软件检测系统，这种系统通过传统静态分析获取软件行为特征，利用N-gram算法进行恶意软件与正常软件的类型识别。Daniel Arp等人[31]提出了一种静态分析APK的方法，提取应用程序的权限信息，组件信息以及API信息作为特征，并使用SVM算法进行模型的训练，最后生成的Drebin模型能够在智能手机上运行。

最与本文相似的是Drebin模型，在第7节我们已经与Drebin进行了比较。相比之下，我们收集了数量更多和覆盖率更广的样本，并对特征进行了大量优化，最后使用随机森林算法代替了SVM算法达到了更好的效果。最终我们模型的检测性能和运行时间性能都达到了更优。

总而言之，如何在检测性能和运行时间性能上取得平衡是我们未来重要研究方向之一。

1. **总结**

Android恶意程序正在快速爆发，需要一种快速、有效的检测方案来检测大量的恶意程序，本文验证了机器学习检测Android恶意程序的可行性，改善了一种基于传统的静态分析方法提取APK特征的方法，并使用随机森林算法对特征集进行分类。该方法在保证了分类的准确率达到97%以上的前提下，检测速度上快50%，而且能够使用分布式的方法进一步的提高检测的速度。

1. **参考文献**

[1] Smartphone OS Market Share, 2017 Q1

https://www.idc.com/promo/smartphone-market-share/os

[2] First Half of 2017 Mobile Security Report - Cheetah Mobile and ANTIY Mobile Security jointly released,http://www.cmcm.com/blog/cn/security/2017-08-10/1092.html

[3] IBanking Mobile Bot, http://blogs.rsa.com/ibanking-mobile-bot-source-code-leaked/

[4] MouaBad.p,https://blog.lookout.com/blog/2013/12/09/mouabad-p-pocket-dialing-for-profit/

[5] J.Forristal Android: One root to Own Them all. In: Black Hat USA 2013 https://mediablackhat.com/us-13/us-13-forristal-android-one-root-to-own-them-allSlides.pdf

[6] Liu X, Liu J. A two-layered permission-based Android malware detection scheme. In: Proc. of the 2014 2nd IEEE Int’l Conf. on Mobile Cloud Computing, Services, and Engineering (MobileCloud 2014). 2014. 142-148. [doi: 10.1109/MobileCloud.2014.22]

[7] Sarma BP, Li N, Gates C, Potharaju R, Nita-Rotaru C. Android permissions: A perspective combining risks and benefits. In: Proc. of the ACM Symp. on Access Control Models and Technologies (SACMAT 2012). 2012. 13-22. [doi: 10.1145/2295136.2295141]

[8] Wolfe B, Elish K, Yao DF. High precision screening for Android malware with dimensionality reduction. In: Proc. of the IEEE 2014 13th Int’l Conf. on Machine Learning and Applications (ICMLA 2014). 2014. 21-28. [doi:10.1109/ICMLA.2014.10]

[9] Grace M, Zhou Y, Zhang Q, et al. Riskranker: scalable and accurate zero-day android malware detection[C]//Proceedings of the 10th international conference on Mobile systems, applications, and services. ACM, 2012: 281-294.

~~[10] Aafer Y, Du W, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In: Proc. of the 9th Int’l ICST Conf. (SecureComm 2013). Springer-Verlag, 2013. 86-103. [doi:10.1007/978-3-319-04283-1\_6]~~

[10] Yang C, Xu Z, Gu G, et al. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications[C]//European symposium on research in computer security. Springer, Cham, 2014: 163-182.

[11] Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y. Andromaly: A behavioral malware detection framework for Android devices.Journal of Intelligent Information Systems, 2012,38(1):161-190. [doi: 10.1007/s10844-010-0148-x]

[12] Amos B, Turner H, White J. Applying machine learning classifiers to dynamic Android malware detection at scale. In: Proc. of the 2013 9th Int’l Wireless Communications and Mobile Computing Conf. (IWCMC). 2013. 1666-1671.[doi: 10.1109/IWCMC.2013.6583806]

[13] Peng H, Gates C, Sarma B, Li N, Qi Y, Potharaju R, Nita-Rotaru C, Molloy I. Using probabilistic generative models for ranking risks of Android apps. In: Proc. of the 19th ACM Conf. on Computer and Communications Security (CCS 2012). 2012. 241-252.[doi: 10.1145/2382196.2382224]

[14] Wu DJ, Mao CH, Wei TE, Lee HM. Droidmat: Android malware detection through manifest and API calls tracing. In: Proc. of the 7th Asia Joint Conf. on Information Security (Asia JCIS). 2012. 62-69. [doi: 10.1109/AsiaJCIS.2012.18]

[15] Felt A P, Chin E, Hanna S, et al. Android permissions demystified[C]//Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011: 627-638.

[16] Sebastián M, Rivera R, Kotzias P, et al. Avclass: A tool for massive malware labeling[C]//International Symposium on Research in Attacks, Intrusions, and Defenses. Springer International Publishing, 2016: 230-253.

[17] Zhou Y, Jin X. Dissecting android malware: Characterization and evolution. In: Proc. of the 2012 IEEE Symp. on Security and Privacy (SP). 2012. 95-109. [doi: 10.1109/SP.2012.16]

[18] Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification[C]//Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009: 235-245.

[19] Zhang M, Duan Y, Yin H, et al. Semantics-aware android malware classification using weighted contextual api dependency graphs[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014: 1105-1116.

[20] Moghaddam S H, Abbaspour M. Sensitivity analysis of static features for Android malware detection[C]//Electrical Engineering (ICEE), 2014 22nd Iranian Conference on. IEEE, 2014: 920-924.

[21] Dhaya R, Poongodi M. Detecting software vulnerabilities in android using static analysis[C]//Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on. IEEE, 2014: 915-918.

[22] Yan L K, Yin H. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis[C]//USENIX security symposium. 2012: 569-584.

[23] Hoffmann J, Ussath M, Holz T, et al. Slicing droids: program slicing for smali code[C]//Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013: 1844-1851.

[24] 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.

[25] 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): 5.

~~[26] Rasthofer S, Arzt S, Bodden E. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks[C]//NDSS. 2014.~~

[27] Desnos A. Androguard: Reverse engineering, malware and goodware analysis of android applications... and more (ninja!)[J]. 2013-03-26]. http://code, google, com/p/androguard, 2015.

[28] Breiman L. Random forests[J]. Machine learning, 2001, 45(1): 5-32.

[29] Yang Y, Wei Z, Xu Y, et al. Droidward: an effective dynamic analysis method for vetting android applications[J]. Cluster Computing, 2016: 1-11.

[30] Bugiel S, Heuser S, Sadeghi A R. Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies[C]//USENIX Security Symposium. 2013: 131-146.

[31] Arp D, Spreitzenbarth M, Hubner M, et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket[C]//NDSS. 2014.

[32] Moore D S. Chi-Square Tests[R]. PURDUE UNIV LAFAYETTE IND DEPT OF STATISTICS, 1976.

[33] Jolliffe I T. Principal component analysis and factor analysis[M]//Principal component analysis. Springer, New York, NY, 1986: 115-128.

[34]Manifest.permission,https://developer.android.com/reference/android/Manifest.permission.html#inhmethods

[35] N. J. Percoco and S. Schulte. Adventures in Bouncerland: Failures of Automated Malware Detection within Mobile Application Markets. Proc. of Black Hat USA12, 2012

[36] HUANG Y, ZHA W. Comparison on classification performance between random forests and support vector machine[J]. Software, 2012, 33(6): 107-110.

[37] Portokalidis G, Homburg P, Anagnostakis K, et al. Paranoid Android: versatile protection for smartphones[C]//Proceedings of the 26th Annual Computer Security Applications Conference. ACM, 2010: 347-356.