LUNA: Quantifying and Leveraging Uncertainty in Android Malware Analysis

through Bayesian Machine Learning

Abstract—Android’s growing popularity seems to be hinderedonly by the amount of malware surfacing for this openplatform. Machine learning algorithms have been successfullyused for detecting the rapidly growing number of malwarefamilies appearing on a daily basis. Existing solutions alongthese lines, however, have a common limitation: they are allbased on classical statistical inference and thus ignore theconcept of uncertainty invariably involved in any predictiontask. In this paper, we show that ignoring this uncertainty leadsto incorrect classification of both benign and malicious apps.To reduce these errors, we utilize Bayesian machine learning– an alternative paradigm based on Bayesian statisticalinference – which preserves the concept of uncertainty in allsteps of calculation. We move from a black-box to a white-boxapproach to identify the effects different features (such assensitive resource usage, declared activities, services and intentfilters etc.) have on the classification status of an app. Weshow that incorporating uncertainty in the learning pipelinehelps to reduce incorrect decisions, and significantly improvesthe accuracy of classification. We achieve a false positive rateof 0.2% compared to the previous best of 1%. We presentsufficient details to allow the reader to reproduce our resultsthrough openly available probabilistic programming tools andto extend our techniques well beyond the boundaries of thispaper.

Android’s software stack was designed as a layeredarchitecture. The lowest layer is that of the Linux kernel,and all subsequent layers increase the level of abstraction while also enforcing security policies.The highest layer in the stack is that of applications, including those developed by the manufacturer and carrier as well as by third parties. Applications interact with the application framework layer through intents or API calls, which are protected by pre-deﬁned permissions. All sensitive resources such as network access, contacts and the ability to send and receive text messages are protected through different permissions[11].

If an application might need to use a resource at any point during its lifetime, the developer of the app must request the permission associated with that resource at compile time.

When a user installs the application, she is presented with a list of these permissions and must agree to grant all these permissions in order to be able to install the application.The most recent version of Android(6.0Marshmallow) allows users to restrict some of these permissions after installation of the app. Regardless, all applications must still declare, at compile time, all the permissions they might need during run-time.

These permissions are quite fine-grained, and as such give insights into the workings of the requesting application. One Of the first studies of this phenomenon was Kirin[12 ],which defined rules to match undesired combinations of requested permissions and warn users about them. For instance, if an app requested contacts information and the internet permission, the installer would raise a red flag since the app would have the ability to steal the user’s contacts data. Another view into the world of requested permissions was provided by Barrera et al. They used Self-Organizing Maps [14] to gain insights into permissions requested by apps belonging to different categories in the Play Store.

Deeper dynamic analyses beyond the requested permissions have also been carried out by several studies including TaintDroid[15],which used taint tracking to discover possible information leakage. Studies along these lines were carried out by DroidScope and FlowDroid among several others. However, these frameworks are difficult to implement and analyze. Moreover, they do not scale well to large-scale datasets and are infeasible to Deploy on mobile devices[3].

Machine learning techniques seem to be a more viable solution in this scenario. Several attempts have been made to classify Android apps by training machine learning models on available datasets of malware. One of the first attempts to make such a dataset available was by the Android Malware GenomeProject[21].The researchers in this project spent Around a year and a half collecting samples which have since been used in several studies. For instance, Peng et al. Introduced risk scores based on several variations of naive Bayes and achievedan area under the curve of 95.3% using requested permissions from this dataset. BayesDroid employs a slightly different approach by using naive Bayes to classify information flows of non-malicious apps as legitimate or illegitimate [22]. The word ‘naive’ in the model’s name comes from the assumption of conditional independence among features –asimpliﬁcation that isseldom true in realworld problems. Similarly,Garcia etal. [5]reported an obfuscation-resilient technique for accurate classification of Android malware.They too base their work on both static analysis along with applications’ meta data.

A recent and highly successful variation of these effortsis Drebin [ 3 ]. It analyzed different features of Androidapplications including their requested permissions, accessednetwork URLs, static analysis reports and several othermetrics to fit a Support Vector Machine (SVM) classifier.This achieved a detection rate of 94% with an error rate ofonly 1%. Dash et al. [ 24 ] use fine-grained dynamic analysisand virtual machine introspection for placing malware intospecific families using SVMs. Drebin and the work by Peng et al. [ 4 ] are closest to the concepts presented in this paper.

However, both of these, and indeed all the other worksusing machine learning for classification of Android apps todate, have ignored the uncertainty invariably present in thelearned model parameters. We, on the other hand utilize theunique strengths of Bayesian inference by quantifying andleveraging uncertainty in our model. The next section givesa brief review of Bayesian inference and the part played byuncertainty in this framework.

LUNA：通过贝叶斯机器学习对Android恶意软件分析进行量化和利用不确定性

摘要-Android越来越受欢迎似乎受到此开放平台恶意软件数量的影响。机器学习算法已经成功地用于检测每天出现的快速增长的恶意软件家族数量。然而，沿着这些线的现有解决方案有一个共同的局限性：它们都基于经典的统计推断，因此忽略了总是参与任何预测任务的不确定性的概念。在本文中，我们表明，忽略这种不确定性会导致对良性和恶意应用程序的错误分类。为了减少这些错误，我们利用贝叶斯机器学习 - 一种基于贝叶斯统计推断的替代范例 - 保留所有计算步骤中的不确定性概念。我们从黑盒子移动到白盒子方法，以识别不同功能（如资源使用情况，声明的活动，服务和意向过滤等）对应用程序分类状态的影响。 Weshow在学习管道中加入不确定性有助于减少不正确的决策，并显着提高分类的准确性。与之前的最佳1％相比，我们实现了0.2％的误报率。我们提供了足够的细节，使读者可以通过公开可用的概率编程工具重现我们的结果，并将我们的技术扩展到超出本文的范围之外。

Android的软件栈被设计为分层架构。最底层是Linux内核层，随后的所有层都增加抽象层次，同时执行安全策略。堆栈中的最高层是应用程序的最高层，包括由制造商和运营商以及第三方开发的应用程序。应用程序通过intent或API调用与应用程序框架层交互，这些调用受预定义的权限保护。所有敏感资源（如网络访问，联系人以及发送和接收文本消息的能力）都通过不同的权限受到保护[11]。

如果应用程序可能需要在其生命周期的任何时候使用资源，则应用程序的开发人员必须在编译时请求与该资源相关的权限。

当用户安装应用程序时，她会看到这些权限列表，并且必须同意授予所有这些权限以便能够安装应用程序。最新版本的Android（6.0Mhmahmallow）允许用户限制部分这些权限安装后的应用程序。无论如何，所有应用程序在编译时都必须声明它们在运行时可能需要的所有权限。

这些权限非常细化，因此可以深入了解请求应用程序的运行情况。对这种现象的最初研究之一是Kirin [12]，它定义了规则来匹配不需要的权限组合并警告用户。例如，如果一个应用程序请求联系信息和互联网许可，安装程序会提出一个红旗，因为该应用程序将有能力窃取用户的联系人数据。 Barrera等人提供了所请求的权限世界的另一种观点。他们使用Self-Organizing Maps [14]深入了解Play商店中属于不同类别的应用所请求的权限。

包括TaintDroid [15]在内的一些研究也进行了超出请求权限的更深入的动态分析，这些研究使用异常跟踪来发现可能的信息泄露。 DroidScope和FlowDroid在其他几项研究中沿着这些线进行了研究。但是，这些框架很难实施和分析。而且，它们不能很好地适应大规模数据集，并且不适合在移动设备上部署[3]。

在这种情况下，机器学习技术似乎是更可行的解决方案。已经通过在可用的恶意软件数据集上训练机器学习模型来尝试对Android应用进行分类。首先尝试制作此类数据集的机会之一是Android恶意软件GenomeProject [21]。该项目的研究人员花费了大约一年半的时间收集了已经用于多项研究的样本。例如，彭等人。引入基于朴素贝叶斯的几种变体的风险评分，并使用来自该数据集的请求权限在95.3％的曲线下实现了一个区域。 BayesDroid通过使用朴素贝叶斯将非恶意应用程序的信息流分类为合法或非法[22]，采用稍微不同的方法[22]。模型名称中的“天真”这个词来源于特征条件独立性的假设 - 在现实世界问题中很少真实的简单化。同样，Garcia等人。 [5]报道了一种用于精确分类Android恶意软件的模糊处理技术。他们的工作同样基于静态分析和应用程序的元数据。

这些努力的最近和非常成功的变化是Drebin [3]。它分析了Android应用程序的不同特征，包括它们所请求的权限，访问过的网址，静态分析报告以及其他一些度量，以适应​​支持向量机（SVM）分类器。这实现了94％的检测率，错误率仅为1％。 Dash等人[24]使用细粒度的动态分析和虚拟机内省来将恶意软件置于特定的家庭使用SVMs。 Drebin和Peng等人的着作。 [4]最接近本文提出的概念。

然而，这些以及所有其他使用机器学习来分类Android应用程序的工作都忽略了在模型参数中总是存在的不确定性。另一方面，我们通过对模型中的不确定性进行量化和平均来利用贝叶斯推断的独特优势。下一节将简要回顾一下贝叶斯推断和不确定性在这个框架中扮演的角色。