面向海量软件的未知恶意代码检测方法

陈 恺[1,2], 王 鹏[2], Yeonjoon Lee[2], 王晓峰[2], 张 楠[2],
黄鹤清[3], 邹 维[1], 刘 鹏[3]

[1]中国科学院信息工程研究所 信息安全国家重点实验室 北京 中国 100093
[2]美国印第安纳大学伯明顿分校
[3]美国宾夕法尼亚州立大学

摘要 软件应用市场级别的安全审查需要同时具备准确性和可扩展性。然而，当前的审查机制效率通常较低，难以应对新的威胁。我们通过研究发现，恶意软件作者通过对几个合法应用重打包，将同一段恶意代码放在不同的应用中进行传播。这样，恶意代码通常出现在几个同源应用中多出的代码部分和非同源应用中相同的代码部分。基于上述发现，我们开发出一套大规模的软件应用检测系统——MassVet。它无需知道恶意代码的代码特征或行为特征就可以快速的检测恶意代码。现有的检测机制通常会利用一些复杂的程序分析，而本文方法仅需要通过对比上传的软件应用与市场上存在的应用，尤其关注具有相同视图结构的应用中不同的代码，以及互不相关的应用中相同的部分。当移除公共库和一些合法的重用代码片段后，这些相同或不同的代码部分就变得高度可疑。我们把应用的视图结构或函数的控制流图映射为一个值，并基于此进行 DiffCom 分析。我们设计了基于流水线的分析引擎，并对来自 33 个应用市场共计 120 万个软件应用进行了大规模分析。实验证明我们的方法可以在 10 秒内检测一个应用，并且误报率很低。另外，在检测覆盖率上, MassVet 超过了 VirusTotal 中的 54 个扫描器(包括 NOD32、Symantec 和 McAfee 等)，扫描出近 10 万个恶意软件，其中超过 20 个为零日(zero-day)恶意软件，下载次数超过百万。另外，这些应用也揭示了很多有趣的现象，例如谷歌的审查策略和恶意软件作者躲避检测策略之间的不断对抗，导致 Google Play 中一些被下架的应用会重新出现等。

关键词 恶意代码；MassVet；重打包；视图结构
中图法分类号 TP309.5

Scalable Detection of Unknown Malware from Millions of Apps

CHEN Kai[1, 2], WANG Peng[2], Yeonjoon Lee[2] , WANG Xiaofeng[2], ZHANG Nan[2],
HUANG Heqing[3], ZOU Wei[1], LIU Peng[3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100093, China
[2] Indiana University, Bloomington, USA
[3] The Pennsylvania State University, USA

Abstract An app market's vetting process is expected to be scalable and effective. However, today's vetting mechanisms are slow and less capable of catching new threats. Based upon a key observation that Android malware is constructed and disseminated typically through repackaging legitimate apps with similar malicious components, we developed a new technique, called MassVet, for vetting apps at a massive scale, without knowing what malware looks like and how it behaves. Unlike existing detection mechanisms, which often utilize heavyweight program analysis techniques, our approach simply compares a submitted app with all those already on a market, focusing on the difference between those sharing a similar UI structure (indicating a possible repackaging relation), and the commonality among those seemingly unrelated. Once public libraries and other legitimate code reuse are removed, such diff/common program components become highly suspicious. We implemented MassVet over a stream processing engine and evaluated it over 1.2 million apps from 33 app markets around the world, the scale of Google Play. Our study shows that the technique can vet an app within 10 seconds at a low false detection rate. Also, it outperformed all 54 scanners in VirusTotal (NOD32, Symantec, McAfee, etc.) in terms of detection coverage, capturing over a hundred thousand malicious apps, including over 20 likely zero-day malware and those installed millions of times. A close look at these apps brings to light intriguing new observations: e.g., Google's detection strategy and malware authors' countermoves that cause the mysterious disappearance and reappearance of some Google Play apps.

通讯作者： 陈恺，博士，副研究员, Email: chenkai@iie.ac.cn。本课题得到国家自然科学基金(No.U1536106, 61100226)资助。
本文是论文 Finding Unknown Malice in 10 Seconds: Mass Vetting for New Threats at the Google-Play Scale(发表于 USENIX Security 2015)的扩展。收稿日期：2015-10-25；修改日期：2015-12-01；定稿日期：2016-01-07

Key words malice; MassVet; repackage; view

1 简介

安卓设备的不断增加促进了移动应用市场的繁荣, 安卓用户从世界各地的应用市场下载应用并安装(如典型的应用市场 Google Play、Amazon Appstore 等)。随着这些第三方应用市场的不断发展, 整个安卓生态系统正在不断地被恶意软件所影响。这些恶意软件通过对合法的应用进行重打包, 伪装成正常的应用出现在第三方市场上, 其恶意行为包括窃取短信与个人信息、发送付费短信等。然而, 阻止这些恶意软件并不容易, 尤其对于 Android 软件: 最近的一份报告[8]指出 99%的恶意软件应用运行在安卓设备上。

应用审查面临的挑战 当前的应用市场都会采取必要的审查手段对上传的应用进行分析, 鉴定其行为是否可疑。例如, Google Play 的审查引擎 Bouncer[24]通过静态扫描匹配已知的恶意代码, 并在谷歌云提供的一个虚拟环境中执行应用以判断是否存在潜在的恶意行为。但其问题在于静态扫描不能检测未知的恶意行为。另外, 应用可以通过识别虚拟环境的特征, 从而在动态执行的时候隐藏恶意行为[30]。此外, 动态分析通常很难覆盖到应用所有的执行路径。

近期相关研究团队[57,28]提出的审查方法多针对已知的可疑行为进行检测, 例如从不可信的网站动态加载二进制代码[57]、组件挟持相关的操作[28]、Intent 挟持[12]等行为。这些方法都需要复杂的信息流分析以及一系列启发式方法对恶意行为进行分类。此外还可能需要进行动态分析[57]甚至是人工参与分析[14]; 且多数动态检测是在模拟器上进行的, 可被恶意软件检测到并绕过[23]。另外, 这些方法都没有使用市场规模的海量应用去作性能评估。

捕获未知恶意代码片段 事实上, 大多数安卓恶意应用都经过重打包, 即恶意作者把一些攻击代码附在几个不同的合法应用中。通过这种方式不仅可以把恶意代码隐藏在看似正常的应用中, 还可以自动化地制作和发布大量的恶意程序。另一方面, 一般的重打包应用只是包含几个广告库[2], 但重打包的恶意应用与之不同, 恶意代码片段会出现在几个毫不相关的应用中, 而这些应用除了一些库外没有其他公共部分。

上述发现引出了一个新的方法去检测重打包类型的恶意软件, 它主要是针对安卓恶意应用, 且不需要建立恶意行为模型。我们仅需比较同源应用(一个应用和它的重打包版本或者由同一个应用重打包后的几个应用)中的不同部分, 以及非同源应用中的相同部分来检查可疑的代码片段(函数级别)。一旦发现这些代码片段中有难以解释的代码(如非公用库中的敏感操作), 那么它们就很有可能是恶意代码。我们称这种方法为 DiffCom 分析, 它可以用来检查未知的恶意行为, 并且不需要重量级的信息流分析。

大规模的检测 基于该基本思想, 我们开发了一个跨市场的安卓重打包恶意软件检测工具——MassVet。MassVet 不用匹配恶意代码特征或已知的恶意行为模型, 它仅仅依赖市场上已存在的应用去检测一个新上传的应用, 具体来说, 它通过在整个应用市场进行一个高效的 DiffCom 分析来检测一个应用。市场中任何与新上传应用相关的应用(如同源应用)都会被很快的定位到, 之后对于不同签名的应用做如下处理: 有相同视图结构的应用分析它们的不同代码部分, 对不同视图结构的应用分析其公共代码部分。通过这种方法得到的代码片段, 在去掉那些重用的代码(如公共库等)后进行进一步分析, 依据一些特征(如敏感 API 调用等)来判断其是否存在恶意行为。

这些大规模的检测背后是一系列高效的视图级别和函数级别的比较过程。为实现对视图的高效可扩展分析, 我们把应用的视图结构(用户界面之间的联系)抽象成一个向量来表示, 把这个向量作为视图的几何中心, 称为 v-core。为方便进行二分查找, 所有应用的 v-core 都按序保存, 这使得整个方法具有可扩展性。

我们的发现 我们收集了来自 33 个安卓市场, 约 120 万个应用,在云平台实现了 MassVet。实验证明了 MassVet 可以在 10 秒内对一个应用进行检测, 并且误报率很低。更重要的是, 我们从 120 万个应用中发现了 127,429 个恶意应用(其中有 20 个是未知恶意行为的应用, 另外还有 34,026 个恶意应用没有被 VirusTotal[43]检测出)。进一步证明了 MassVet 的检测覆盖率比 VirusTotal 集成的所有扫描器(如 Kaspersky、Symantec、McAfee 等)都要高。除此之外, 我们还发现有 30,552 个恶意应用来自 Google Play, 恶意软件作者不断地同谷歌的应用审查策略对抗, 使得 Google Play 上的部分应用在被移除后, 又不断地重新出现。

本文的贡献总结如下:

(1) 我们开发了 MassVet, 对安卓应用进行大规模的恶意代码检测。上传到 MassVet 的应用仅需要和市场上存在的应用进行对比。MassVet 利用恶意代码重打包的特点, 检测出恶意应用, 甚至未知恶意行为的应用。MassVet 基于云平台且可以快速地进行视图级别和函数级别的比较, 从而具有很好的可扩展性。由于 v-core 和 m-core 数据集(120 万个应用仅占 100G 的空间)是从多个应用市场得到的, 所以 MassVet 还可以利用来自其他市场的应用来检测来自另一个市场的应用。

(2) 我们实现了 MassVet 并基于 120 万个应用对其进行评估, 据我们所知, 这在安卓恶意代码检测领域的数据样本中具有最大的规模。Google Play 作为世界上最大的安卓市场, 其规模有 130 万个应用[39]。我们的系统发现了十万恶意应用, 其中部分恶意应用能够躲避当前已有的恶意程序扫描器。MassVet 的检测覆盖率超过了 VirusTotal 中所有主流的扫描器, 并且可以在 10 秒内完成对一个应用的扫描。部分恶意软件有上百万的安装量, 其中 5000 个软件被安装超过 10000 次。实验部分也分析了 Google Play 如何审查上传的应用, 以及恶意软件作者是怎样隐藏和传播恶意代码。

2 背景

安卓应用市场 在市场上发布一个应用需要经过应用商店的审查, 包括质量控制、安全保护等。从 2012 年开始, Google Play 推出了名为 Bouncer 的应用审查技术,在一定程度上减少了恶意软件的传播, 如 F-Secure[15]发现的恶意软件仅占总数的 0.1%。另一方面, 恶意软件可以根据模拟器的指纹来隐藏恶意行为[33], 从而逃避检测。比起官方的安卓市场, 第三方市场很少有这种审查机制。据 F-Secure 的数据所示, 著名的市场如木蚂蚁、安智、百度等都受到恶意软件的威胁[16]。

当前安全审查机制主要依赖传统的恶意代码检测方法。大多数的方法, 如 VetDroid[52]通过追踪应用中的信息流以及建立恶意行为模型进行检测, 对于那种未知行为的恶意代码, 这种方法是无效的。另外, 信息流分析需要分析每一条指令的语义, 而且需要特殊的方法(如重量级程序分析)来减少误报率。种种限制导致以上方法难以用于大规模的分析。

重打包 重打包是指对原作者的应用进行修改, 加入自己的代码再重新发布到应用市场上的过程。趋势科技的一份研究数据表明(2014 年 7 月), Google Play 上的 Top50 的免费应用近 80%都有对应的重打包版本[49]。谷歌自己也表示有 1.2%的应用经过了重打包, 我们之前的研究也发现, 这个数字达到了 5%到 13%[55]。这些打包后的应用有两个目的, 一个是获取广告收入[7], 如对“愤怒的小鸟”这款应用重打包, 加入广告库绑定自己的广告 ID, 从广告商获取利益。第二, 恶意软件作者通过对这些流行的应用进行重打包, 提高其恶意代码传播的便捷性。相比制造一个功能丰富的病毒来说, 通过使用 smali/baksmali[36]这样的工具进行重打包节省了很多时间。更重要的是, 他们可以利用这些流行的应用感染更多的用户。事实上, 研究表明 86%的恶意软件都是重打包应用。

适用范围和假设 MassVet 是用来检测重打包恶意应用的。我们认为恶意作者不会把恶意代码放在应用的核心功能函数里, 因为这需要花费更多的精力去分析软件逻辑。另外, MassVet 也可以用来应对大多数应用的代码混淆。我们假定代码没有被混淆到不能重打包的程度(在该高度混淆的情况下, 其他大多数静态分析方法都会失败)。最后, 我们假定

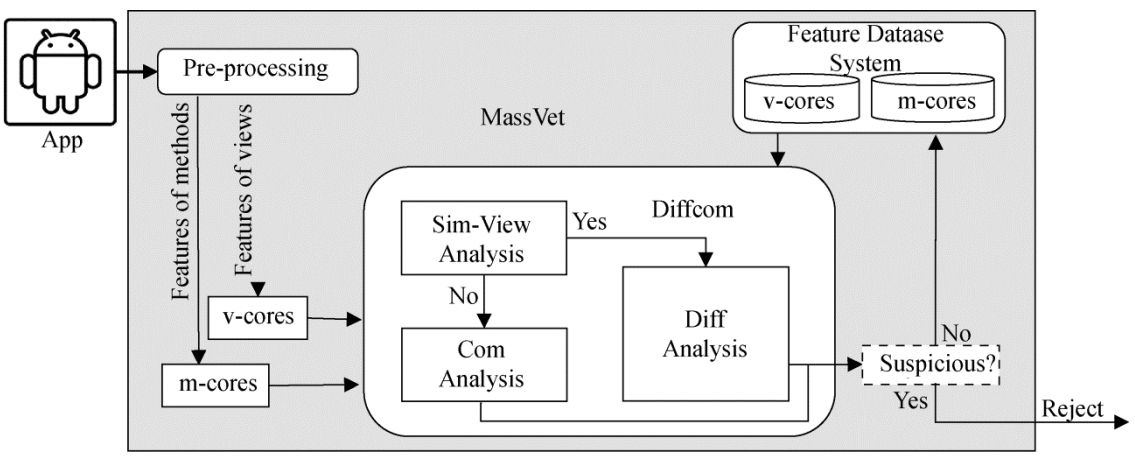


图 1 MassVet 架构图

应用市场的应用分布具有多样性, 对于一个上传的应用, 或者存在其同源应用, 或者存在跟它有相似恶意代码的应用。为增加假设的可能性, 不同的应用市场可以共享应用特征数据集(如 v-cores 和 m-core)。这个数据集较小, 120 万个应用产生的数据集只有 100G。

3 MassVet 的设计和实现

3.1 综述

设计和架构 为大规模的检测恶意软件, 我们设计了如图 1 所示的架构, 总共分三部分: 预处理模块、特征数据库系统以及 DiffCom 模块。预处理模块自动分析上传的应用, 包括提取应用的视图结构特征和函数特征, 并得出对应的 v-cores 和 m-cores。对于这些特征, DiffCom 模块在由所有应用产生的 v-cores 和 m-cores 数据库中进行查找。对于查找到的结果进一步筛选, 从而定位到可能包含恶意代码的部分。

工作原理 我们用一个例子来讲述整个系统的工作原理。MassVet 首先对一个市场上的所有应用进行处理, 将其视图结构特征存储在 v-core 数据库中, Java 函数特征存储在 m-core 数据库中。为方便二分查找, 两个数据库都按序存储。这里以重打包的 AngryBird 为例, 上传之后会被反编译为 Smali 代码, 之后基于 Smali 代码提取应用的特征(如用户界面、小部件、事件、控制流等), 计算视图结构和控制流图的几何中心, 并将其分别映射到 v-core 和 m-core 数据库中。接着, 通过二分查找的方法查找 v-core 数据集, 一旦存在匹配, 即存在一个和 AngryBird 具有相同视图结构的应用, 就会用它同新上传的应用进行函数级别的比较, 找出他们之间不同的部分。在排除广告库的干扰下, 不同部分的代码即为可疑代码。如果找不到相同视图结构的应用, MassVet 会在 m-core 数据集中进行匹配, 当找到一个相似的函数时, 排除代码重用(如库函数)的情况, 当两个相似函数所属的应用毫不相关时, 这个函数就是可疑函数。最后, MassVet 根据上述情况判断应用是否可能为恶意应用。上述所有的过程都是自动化实现的, 不需要任何人为参与。

3.2 用户界面快速分析

MassVet 检测上传的应用主要通过判断市场上存在的应用与之是否相关来进行的, 这种相关是指两个应用是否具有相同的视图结构。当两个视图相似的应用不是来自官方渠道时(例如来自第三方市场), 他们被称为同源应用, 那么两个应用不同的代码片段就很可能存在恶意代码。用视图结构来定义相关性要比用代码定义相关性更准确, 因为重打包的应用可能会混淆代码或者加入一些垃圾代码, 这使得两个应用在代码层面变得完全不同。另一方面, 改变一个应用的视图结构需要恶意作者更多的精力, 因此大多数重打包应用会保持原始的视图结构。我们在研究中发现, 很多重打包应用添加了很多新代码, 甚至多于原应用的功能, 但对视图结构却未做任何改动。

用视图结构去检测重打包应用在之前的研究中已有提到过[50], 他们用图的同构算法来定义两个应用之间的相似度, 但该算法分析的时间较长: 比较两个应用平均花费 11 秒。如果要分析类似 Google Play 这种市场数量规模(假设 130 万个应用)的应用时, 需要 165 天的时间。下边将阐述一种准确高效的应用视图结构分析方法。

特征提取 应用的界面由一系列的元素组成, 如按钮、列表、文本控件等, 这些元素会响应用户的输入。当发生输入事件时, 可能会跳转到其他界面或使得本界面的元素内容发生变动。这些界面彼此之间的联系, 可以用来提取应用的视图特征。

我们用一个带权重的有向图来描述应用的视图结构, 它包含了所有的用户界面以及界面之间的跳转关系。图中, 每一个节点代表一个界面, 界面中有效元素(可以产生响应事件的元素)的数量作为其权重, 两个节点之间的有向边表示界面之间的跳转关系, 跳转是由事件触发的, 如 onClick、onFocus-Change、onTouch 等, 可以根据事件的类型来区分不同的边。

上述有向图可以用来描述一个应用的视图结构, 但是对于那些只有几个界面的小型应用, 这种方法是不可行的。为解决这个问题, 我们把一些其他的特征加入有向图中, 如其他的一些视图元素、元素的类型等。例如, 通过分析 Activity 的调用关系; 类似 AlertDialog(弹出对话框)类型的情况也被视为有向图中的一个节点, 对于自定义的对话框, 可以通过追踪类的继承关系判断。此外, 每一个元素的类型被赋予一个特殊值, 用来标识不同类型的元素, 这样我们可以根据特殊值来定义节点的权重, 使得节点较少的有向图之间有明显区别。图 2 给出了一个例子。

我们没有用文本标签、颜色、大小等属性作为界面的特征, 因为这些元素不同于按钮、事件等特征, 容易被开发者操控。如果开发者添加垃圾控件、修改控件类型或者弹出对话框等, 都会影响用户体验,

所以重打包应用一般不会改变这些特征。

为得到一个应用的视图结构，预处理模块自动化的分析其代码，恢复界面相关的进程间通信(IPC)，这些 IPC 调用包括 startActivity 和 startActivityForResult 等。对于每一个调用，首先定位到所属的界面，然后根据界面来寻找由其触发的调用。例如，IPC 调用位于一个和界面相关的类 v 中，它的参数是要调用的类 v'，这样在有向图中就会有一条有 v 到 v'的边，边的类型与触发事件相关，如 onClick 事件等。

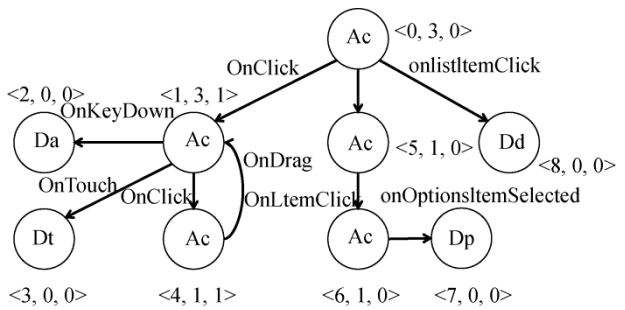


图 2　视图结构示例

(注: *Ac: Activity; Da: AlertDialog; Dt: TimePickerDialog Dp: ProgressDialog; DD: DatePickerDialog*)

大规模设计　当一个应用的视图结构特征被提取出来时，我们希望能够很快地同市场中其他的应用进行比较，从而发现与之相关的应用。比较操作需要非常高的性能，要求能在几秒内同 100 万个应用进行比较。为实现这个目标，我们运用了最近提出的一种图相似性比较算法 Centroid。Centroid 把一个程序的控制流程图的特征抽象为一个特征值，作为其几何的中心。这个特征值有一个特点，即当图中节点改动很小时，特征值的变动也很小，这在一定程度上能反映两个图的差异性。同时，这个特征会把全局的比较缩小在几个相邻节点的比较上，实现高性能而又不失准确率的目标。由于视图结构不同于控制流程图，这个方法不能直接用于视图结构的比较。应用的视图结构通常会比较碎片化，因为一个界面通常是由自己的模块来调用，如大多数广告界面是通过广告库中的 APIs 来调用的，这样他们的视图结构就会与主程序的视图分离。我们通过以下方法解决这个问题。

给定一个应用的界面集合 G_i=1⋯n，预处理模块首先计算每个界面的几何中心。对于一个子图 G_i，首先需要把每个节点的特征转化为一个三维向量 $\vec{c}=\{\alpha,\beta,\gamma\}$，其中$\alpha$代表每个节点的序号，按照图的深度优先遍历给出，即从主界面产生的节点开始，根据子树的大小依次选择节点去遍历。如果两个节点的子树大小相同，则按照节点的类型进行遍历。这样可以保证每一个节点拥有一个唯一的序列号。β代表节点的出度，γ 代表当前节点所处循环的层数。图 2 给出了一个例子，展示这个向量是如何建立的。

在每个节点 k 都得到一个向量$\overrightarrow{c_k}$后，我们通过如下方式计算其几何中心:

$$vc_i=\frac{\sum_{e(p,q)\in G_i}(w_p\overrightarrow{c_p}+w_q\overrightarrow{c_q})}{\sum_{e(p,q)\in G_i}(w_p+w_q)}$$

其中$e(p,q)$是指由 p 指向 q 的边，w_p是节点 p 的权重。由于 v-core 的单调性，我们对其进行排序，方便对大量的应用可以进行二分查找。通过这种方式，一个子图G_i可以很方便的同其他子图进行比较。特别的，给定一个图G_t和对应的 v-core vc_t，我们认为如果满足$|vc_i-vc_t|\leqslant\tau$时，它和图G_i相似。另外，给定的两个应用的视图共享一个子集 G_i(l=1⋯m)，我们认为当满足 $\sum_l|G_{i(l)}|/\sum_i|G_i|\geqslant\theta$时，两个应用是相似的，即一个应用的大多数视图出现在另一个应用中。这保证了即使新增了很多界面，两个重打包应用的视图关系依然可以建立起来。

我们通过对 5 万个应用进行训练得到了这些阈值，并设定了不同的阈值来评估对应的误报率和漏报率。对于误报情况，我们在每个阈值下随机选取了 50 个应用，并人工的判断其关系。对于漏报情况，我们利用已知的 100 对具有重打包关系的应用作为参考去发现不同的阈值。研究表明当$\tau=0$和$\theta=0.8$时，我们得到最低的误报率(4%)和漏报率(6%)。在这 5 万个应用中，我们发现了 26,317 对应用是重打包关系(约 3,742 个应用)。

视图结构分析效率　与基于代码的比较方法相比，基于视图结构的分析能更高效的检测同源应用。我们从 120 万个应用里找出了 10000 对重打包应用，这些重打包应用中的代码大部分是不相同的，在 14%的重打包应用分组中，代码的相似率低于 50%，这可能是因为在重打包时加入了庞大的广告库或者垃圾代码。由于代码层面上这些应用差异较大，所以不能通过其代码来进行比较，但通过视图的方法可以很准确的进行比较。

3.3　大规模 DiffCom 分析

对于一个新上传的应用，首先提取其视图结构特征，判断其是否与市场中的其他应用相似。如果存在相似应用，进一步比较它们代码的不同部分，然后定位恶意代码。否则，提取该应用的函数特征，并与市场上的其他应用进行函数级别的比较，尝试去发现相同代码部分。查找不同或相同的代码部分需要排除代码重用的情况(如公共库等)，并找到某段代码是恶意代码的证据。上述的整个过程被称为

DiffCom模块, 下面我们会展示代码相似性分析的实体以及讨论对抗 DiffCom 分析的方法。

分析基础 为大规模的审查应用, DiffCom 需要高效的代码相似比较算法。我们使用 Centroid[7]算法来实现相似性分析, 如之前所述, 该方法将一个程序的控制流程图映射为一个特征值, 同样的方法可以用来分析视图结构。该算法把一个基本代码块作为节点, 基本块的权重为包含的语句数。图中每个节点由序号、出度和所处循环的层数来标识, 根据这三个参数来计算整个图的几何中心。

为实现大规模检测, 首先对市场上所有的应用以函数为单位划分, 在移除公共库后, 预处理模块会分析函数中的代码, 计算每一个函数的几何中心, 之后有序的存入数据库中。在检测时, 如果新上传的应用和数据库中的应用发现有相似视图结构, 就会用应用所产生的函数的几何中心同 m-core 数据库中相似应用函数的几何中心比较, 找到其代码不同的部分。这里采用了二分查找的算法去比较, 能够很快的找到结果。下面会阐述具体操作, 其性能评估会在 4.2 给出。

差异代码分析 当找到一个应用的同源应用时, 我们希望通过检查其不同的代码部分来发现可能的恶意行为。由于安卓的恶意代码主要通过重打包方式传播, 恶意代码通常也是通过工具自动化的注入, 对原始的应用代码并未作任何改动, 因此可以通过对比同源应用之间不同的代码部分来定位恶意代码。比较两个应用的不同可以通过比较它们的 m-core 实现。对于给定的两个有序的 m-core 序列 L 和 L', 找到两个序列的不同部分可以在 $\min(|L|, |L'|)$步内完成。

然而, 具有相似界面的应用并不一定是重打包关系, 例如同一种类型的库或界面可以用在不同的产品中, 甚至有可能一个应用是另一个应用的升级版本。另外, 流行的开源的界面库(如 Appcelerator[3])和开源的模板(如 Envatomarket[13])等会被不同的开发者采用, 这同样可以导致不同的应用具有相似的视图结构。即使两个应用是重打包关系, 其代码不同部分可能在广告库上, 但不一定就是恶意代码。如何处理这些特殊情况, 减少误报率是一个很重要的问题。

为解决这一问题, MassVet 首先对提交的应用函数作预处理, 移除广告库和其他一些第三方库。我们建立了一个合法广告库的白名单[6], 涵盖了如 MobWin、Admob 等主流的广告库。对于不太知名的广告库, 我们随机分析了 5 万个应用, 发现了有 34886 个函数被至少 27057 个应用所使用, 我们在 VirusTotal 上扫描每一个函数, 如果没有被报出恶意行为, 我们将其加入到白名单中。同理, 我们找到了流行的 UI 库, 并通过白名单的方式避免了视图分析时的误报。对于 5 万个应用的数据集, 得到的结果也并非完全准确, 有可能会放过部分恶意代码, 我们又随机选取了 50 个和广告相关的函数, 通过人工分析确定了他们确实是合法的广告库。尽管这种方法有一定的误报率, 但最终结果还是比 VirusTotal 中的所有扫描器覆盖率高。

对于同一个机构推出的应用, 其公用的代码由于不是主流的, 因此可能不能用上述方法来解决。我们通过检查应用的签名来判断: 这种方法在大多数情况下都是有效的, 因为合法的应用开发者通常会用同一个证书对其开发的应用签名。当然, 开发者有可能会用多个证书对不同的应用进行签名, 当出现这种情况时, 上述的方法有可能会报出恶意检测结果。为避免误报, 我们进一步检查这部分代码, 这部分代码通常会被其他部分的代码调用, 同时也会调用公共部分的代码, 而重打包的恶意代码通常独立于其他部分的代码, 很少会调用原始程序中的代码。

我们利用上述方法来区分合法代码和可疑代码, 对于发现的可疑代码, DiffCom 会检查其是否调用了程序其他部分的代码, 当仅仅发生了内部的调用时, 可以对其进行进一步的分析。这主要是由于恶意软件作者不会使他们的代码和原应用中的代码有调用关系(因为这需要更多的时间和精力去理解原应用中的逻辑)。

对于定位到的目标代码, DiffCom 模块对其进行进一步的分析, 判断可能存在的威胁。我们通过检查敏感 API 的调用来判断, 例如 getSimSerialNumber、sendTextMessage 和 getLastKnownLocation 等来判断。我们发现这部分可疑代码确实会对手机用户的个人信息带来威胁, 但并不知道具体的恶意行为。基于行为的恶意代码检测[27], 通常会匹配特定的行为, 如“读取通讯录并通过网络发送到第三方服务器”, MassVet 与之不同, 它不仅可以减小误报率, 还可以检测到未知行为的恶意代码。

公共代码分析 如果没有找到与待检测应用相关的同源应用, MassVet 会继续进行公共代码分析。为增加检测覆盖面, 在差异代码分析检测不到恶意结果时, 也可以再进行公共代码分析。公共代码分析把应用的每一个函数映射为 m-core 值, 然后通过二分查找的方式在 m-core 数据集中查找。一旦找到公共代码部分, DiffCom 会继续对其分析, 判断是否包含恶意代码, 给出报告。

同样, 这里的难点在于检测两个应用是否是不

相关的。通过检查签名的方法能够排除大部分同源应用但并非全部。通过检查一个代码段是否调用应用中其他部分代码的方法不适用于公共分析，问题在于两个重打包应用的相同代码片段不一定就是恶意的代码，这与上述差异代码分析不同。

另一个方法是分析这些不相关的应用之间是如何联系的。如前所述，出现这种情况通常是由于开发者用不同的证书对不同的应用签名，而这些应用里包含一些重用的代码，如私有的 SDK 等。在这种情况下，一旦移除所有的公共库或重用代码后，剩下的公共代码就变得高度可疑。我们采用下边的方法来发现这种隐含的关系。

从之前的数据集中，我们发现大多数代码重用是由于采用相同的界面产生的，开发者通常利用存在的界面设计去很快的建立一个应用。即使两个应用在视图结构上没有相似之处，即被之前的视图结构分析模块视为不相关应用，仔细观察其视图时会发现，它们的某个子图确实可能存在相似关系。例如，在移除公共库的 5 万个应用的训练集中，我们发现 30286 个应用里至少有 30%的相似视图，16500 个应用里共享 50%的视图，8683 个应用里共享不少于 80%的视图。通过随机选取 10 个应用并人工的去分析，我们发现当比例到达 50%时，几乎所有的应用不是来自同一个作者就是重打包关系。当比例达到 80%时，几乎所有的应用都是重打包关系。基于这个发现，我们得到如结果: DiffCom 对比两个应用的子图，如果相似度达到 50%，那么这两个应用是相关的。

在进行了相关性检测后，我们建立常用库的白名单，移除公共库代码或模板代码。剩余的公共代码部分，特别是有敏感 API 的代码，则很可能是恶意代码。

逃避 MassVet 检测 为逃避检测，恶意作者可能会混淆自己的代码，这种方式可能会逃过一些相似性检测，但 MassVet 在某种程度上能够检测到。如变量或函数名重命名的方式不会改变 Centroid 值[7]，只有当恶意作者对每一次加入重打包程序中的恶意代码进行深度混淆时，才有可能逃避 MassVet，但当前的工具如 ADAM[53]和 DroidChameleon[32]都无法做到这一点。另外，深度混淆往往在同未做混淆的应用对比时会引起怀疑。恶意作者也有可能去改变应用产生的视图结构，但这和代码混淆比起来是很困难的，因为 OnClick、OnDrag 等事件是在安卓框架层实现的，无法对其更改。增加应用的页面也是比较困难的，这需要深度掌握应用的内部逻辑且还要避免影响应用正常的功能。即使加入与原视图无关的页面，也不会影响 MassVet 的检测。

我们从 Google Play 随机选取了 100 个应用来分析 MassVet 对抗混淆的能力，著名的混淆工具如 DexGuard[37]和 ProGuard[38]等仅仅工作在 Java 代码层，并不能直接被攻击者应用到 Dalvik 字节码上。这里我们采用 ADAM[53]和 DroidChameleon[32]对应用进行混淆，我们发现混淆前后的 v-core 值未发生任何改动，这说明了我们的方法在一定程度上可以对抗混淆。

3.4 系统设计

我们用 C 和 Python 语言实现了 MassVet，并收集了 120 万个应用作为数据集，其中将近 40 个应用来自 Google Play。

系统启动和恶意检测 首先需要高效的分析一个市场的应用。所有这些 APK 文件被反编译为 Smali 代码，然后提取其视图特征和函数特征，并转化为 v-core 和 m-core。我们使用 NetworkX[29]去处理图形和查找循环。之后这些特征有序地保存到各自的数据库中。我们使用 Sqlite 保存这些数据(该数据库占用空间小且性能高)。所有的这些应用最后生成了 1.5Gb 的 v-core 数据集和 97Gb 的 m-core 数据集。

下一步是扫描这 120 万个应用里的恶意代码。一个直接的方法是通过二分查找依次检查，但这会造成数以千万次的比较和分析操作。我们采用了一个高效的方法来实现，通过一次完整的扫描 v-core 数据库，找出所有相等的值作为一个集合。

同一个集合中所有的子图都相同，但把它们组合起来去判断两个应用的相似性变得很棘手，这是因为每个应用的界面被分成多个子图分布在 v-core 数据集中。这样就需要完整地遍历数据库。最快的方法是维护一个包含子图编号的表，但这个表是会非常大，最坏的情况下需要维护(120 万*120 万/2)个元素的表，这难以被一次性载入内存中。在实现中，我们每次检查 2 万个应用与其他 120 万个应用的差异性来平衡空间和时间消耗，这需要扫描整个集合 60 次，每次需要 100G 的内存。

云平台支持 为支持高性能的应用检测，MassVet 运行在云端，如图 3 架构所示。我们基于 Storm[40]来实现 MassVet。Storm 是一个开源的流处理引擎，它同样支持主流的 web 服务如 WebMD、Alibaba 和 Yelp 等。Storm 通过一个工作单元集合来支持大规模的数据流分析。我们把 MassVet 整个工作流程转为一个拓扑结构: 对于一个待检测的应用，首先提取其视图和函数，这个过程需要结合白名单移除合法的库或模板；之后计算应用的 v-core 和 m-core 值，并通过二分查找的方式进行查找；根据

查找的结果, 决定先进行差异代码分析还是公共代码分析。每一个操作被封装成一个处理单元, 应用的数据处理也在一个数据流中完成。Storm 引擎同样可以用来支持多数据流处理, 这可以用来处理大规模上传的应用。

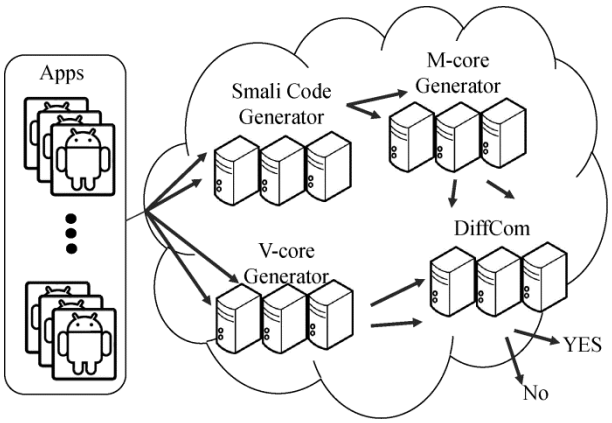


图 3 MassVet 云平台设计

4 实验

4.1 配置阶段

应用收集 我们从 33 个安卓市场收集了近 120 万个应用, 其中有 40 万个应用来自 Google Play, 596437 个应用来自中国的 28 个安卓应用市场, 61886 个应用来自欧洲的市场, 27047 个应用来自美国的市场, 详细数据如附录中表 3 所示。

从 Google 上收集的应用包含娱乐、工具、社交以及聊天等 42 个分类, 对于每一个分类, 我们首先收集该分类下排名靠前(按安装量排名)的 500 个应用, 然后从其他分类应用中随机选取了 1000 到 3000 个。对于第三方的市场, 我们根据市场的规模随机选取了一定数量的应用。我们的数据集中包含了 Facebook、Skype、Yelp、Pinterest 和微信等较为流行的应用, 应用的大小由 1MB 到 100MB 不等。

验证 对于被 MassVet 检测出的可疑应用, 我们通过 VirusTotal 和人工分析的方式进行验证。VirusTotal 是比较高效的恶意检测系统, 其后台由 54 个主流的反病毒引擎组成, VirusTotal 也提供了对手机应用的扫描服务。VirusTotal 有两种模式, 完整扫描和缓存扫描。缓存扫描是非常快的, 它能够在每分钟扫描 200 个应用, 但是仅仅覆盖那些曾经被扫描过的应用, 对于从未上传到 VirusTotal 的应用, 其报告结果为“未知”。完整扫描的方式通过集成的 54 个扫描器最新特征库进行即时扫描, 但比较耗时, 平均每一个应用需要耗费 5 分钟左右。

为验证扫描出的几千个可疑应用, 我们首先通过缓存扫描的方式来确定大多数的恶意应用, 对于结果为“未知”的应用, 我们从中选取若干进行完整的扫描。对于 VirusTotal 没有检测出任何结果的应用, 我们进一步通过人工的方式进行验证。特别的, 对于公共分析模块检测出的可疑结果, 我们根据其公共的代码进行了聚类, 对于每个类, 无论多少同类应用被发现有恶意代码, 我们都将该类剩余的应用视为恶意应用, 即使它们没有被 VirusTotal 检测出。这些应用的公共代码或差异代码会进一步进行可疑行为分析, 如信息泄露等类型。我们根据代码手动的鉴定其对应的行为, 当代码对用户的信息或财产造成威胁时, 将其标注为恶意代码。

4.2 效率和性能

恶意检测和覆盖率 MassVet 从我们的数据集中检测出 127429 个可疑应用(占比 10.93%), 其中 10202 个由差异分析检测出, 剩余部分由公共分析检测出。这些可疑的应用来自不同的市场, 其中 30552 个应用来自 Google Play, 96877 个应用来自第三方市场, 详细分布如图 5 所示。我们首先用 VirusTotal 的缓存模式对这些应用进行验证, 最后有 91648 个应用被确认为恶意应用(占比 72%), 17061 个应用(13.38%)可能存在误报(即 VirusTotal 存在这些应用的缓存, 但未检测出恶意结果), 13492 个应用(10.59%)检测结果为“未知”(即 VirusTotal 中不存在这些应用的校验值)。为进一步验证, 我们从“未知”集合中随机选取了 2486 个应用, 从 “误报”集合中随机选取了 1045 个应用, 之后将这些应用上传至 VirusTotal 进行完整的扫描。结果显示, 2340(94.12%)个“未知”的应用和 349(33.40)个“误报”的应用被确定为恶意应用。据此, 我们计算出 MassVet 的误检率(即 FDR, 误检数量/所有检测出的数量)为 9.46%, 误报率(即 FPR, 误报数量/所有参与检测的数量)为 1%。我们看到公共分析比差异分析检测出更多的恶意结果, 因为差异分析是基于两个同源应用的差异代码进行分析, 而公共分析则是查找非同源应用中相同部分的恶意代码, 很多恶意的应用会使用相同的恶意 SDK, 这也使其更容易被检测出。

我们继续从误报的的应用里选取了 40 个进行人工验证, 发现其中 20 个为高度可疑应用。其中 3 个应用会动态加载和执行恶意代码, 1 个应用会偷偷拍照, 1 个应用会修改其他应用的启动顺序, 7 个应用会获取如 SIM 卡序列号、手机号码等敏感信息, 其他几个会放置钓鱼页面或诱导用户安装应用等。这些行为很有可能是零日(zero-day)软件, 我们把上述几个应用报告给 Norton、F-Secure 等 4 个反病毒软件厂

商, 如果这几个应用被确认为恶意应用后, MassVet 的误检率会减少到 4.73%。

为证明 MassVet 的覆盖率, 我们又从 Google Play 上随机抽取了 2700 个应用, 并上传至 VirusTotal 进行扫描。VirusTotal 的 54 个扫描器总共检测出 281 个恶意应用, 在这之中有 197 个应用也被 MassVet 检测出来。相对于 VirusTotal 的结果, MassVet 的检测覆盖率达到了 70.1%, 这超过了 VirusTotal 中任何一个单独的扫描器, 包括比较知名的扫描器如 NOD32 (60.8%)、Trend (21.0%)、Symantec (5.3%)和 MaAfee (16%)。更重要的是, 至少有 11%的恶意应用被 MassVet 检测出, 而未被其他几个扫描器检测出。详细实验数据如表 1 所示。考虑到 VirusTotal 本身可能存在一定程度的误报和漏报, 我们将 VirusTotal 检测结果列于图 7 供参考。 另一方面, 我们从 contagio 选取 50 个恶意程序用于测试覆盖率, 发现 45 个恶意程序能被检测出, 覆盖率达到 90%。

表 1 其他主流扫描器的检测覆盖率

AV Name	# of Detection	% Percentage
Ours (MassVet)	*197*	70.11
ESET-NOD32	*171*	60.85
VIPRE	*136*	48.40
NANO-Antivirus	*120*	42.70
AVware	*87*	30.96
Avira	*79*	28.11
Fortinet	*71*	25.27
AntiVir	*60*	21.35
Ikarus	*60*	21.35
TrendMi-cro-HouseCall	*59*	21.00
F-Prot	*47*	16.73
Sophos	*46*	16.37
McAfee	*45*	16.01

检测时间 我们在一个 260GB 内存、40 核 2.8GHz 主频、28TB 硬盘的服务器上对 MassVet 的性能进行了评估。基于 Storm 流水线处理引擎, 我们针对 1 到 500 个并发上传的应用进行了测试。从应用提交到整个过程完成, 我们观测到的平均时间为 9 秒。整个检测过程是基于 120 万的应用分析的。

表 2 给出来不同阶段的时间, 包括预处理阶段、v-score 数据集的搜索、差异分析等。实验表明, MassVet 可以扩展用于真实市场的实时应用审查。

4.3 恶意代码分布与防御

基于 MassVet 检测出的 127429 个恶意应用, 我们进行了一些分析, 分析结果使我们更加深入地了解了恶意软件对整个安卓生态系统带来的威胁。

表 2 性能随并发上传应用数量变化表

Apps	预处理	v-core 搜索	差异分析	m-core 搜索	总计
10	*5.84*	0.15	0.33	1.80	8.12
50	*5.85*	0.15	0.34	1.99	8.33
100	*5.85*	0.14	0.35	2.23	8.57
200	*5.88*	0.16	0.35	3.13	9.52
500	*5.88*	0.16	0.35	3.56	9.95

分布情况 就恶意代码的分布而言, 中国应用市场以 12.90%占据首位, 其次是美国市场, 比例为 8.28%。这说明了中国市场与其他国家的市场相比, 可能缺少合适的安全保护策略。即使是从 Google Play 收集的应用, 也有 7.61%的应用存在恶意行为; 而先前发布的恶意代码比例仅有 0.1% [15]。恶意应用的具体数字在附录表 3 中给出。

我们观察到大部分扫描器对新出现的恶意软件反应较慢, 由 VirusTotal 确认的 91648 个恶意应用, 仅有 4.1%的恶意应用被 25 个扫描器提示, 详细结果如图 7 所示。这同样证明了 MassVet 能够很好的检测未知类型的恶意代码。

我们发现超过 5000 个应用的下载量超过 10000 次(如图 4 所示), 部分热门应用的安装量达到了 1000000 次。另外, 分布在 Google Play 中恶意 APK 的比例较高(大多数在 3.6 到 4.6 之间), 这当中的应用平均下载量非常高(由 100000 到 250000 不等), 如图 6 所示。

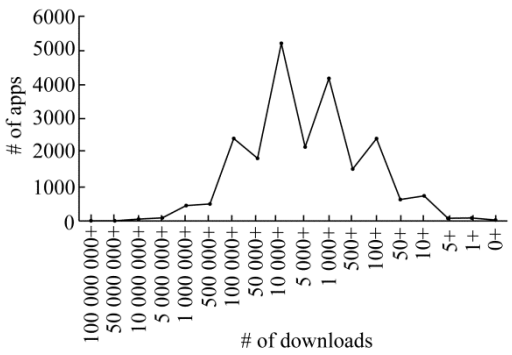


图 4 Google Play 恶意应用下载分布情况

存在的防护以及消失的应用 表面上看, Google Play 确实在检测恶意代码上采取很多措施, 然而, 我们的研究发现了 Google Play 审查机制的一些问题。如图 8 所示, 大多数发现的恶意软件是在过去的 14 个月上传的, 这些应用出现的越近就越有可能有问题。这说明了 Google Play 在持续的检测其已下架的可疑应用。

有趣的是, 在我们上传了 3711 个应用到 VirusTotal 后的 40 天内, 其中有 250 个应用从 Google

Play 消失了。90 天之后, 又有 129 个应用下架。在这 379 个下架的应用中, 其中 54 个(14%)是由 VirusTotal 检测出的。

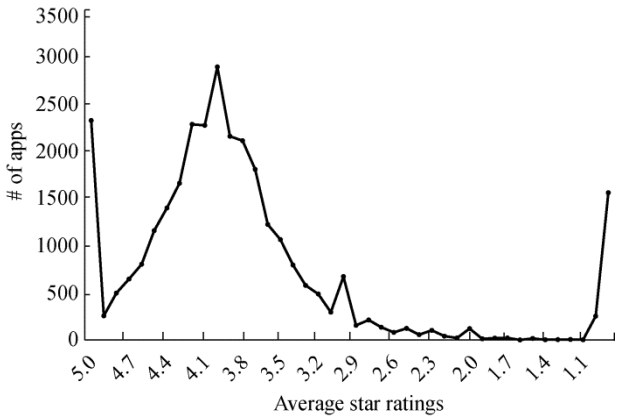


图 5 Google Play 恶意应用星级评分分布情况

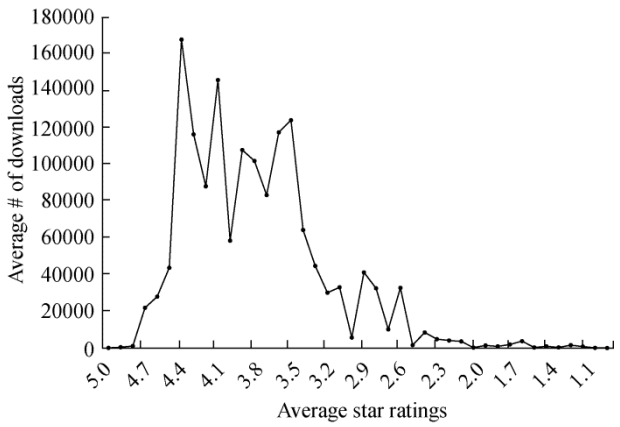


图 6 Google Play 恶意应用平均下载量分布情况

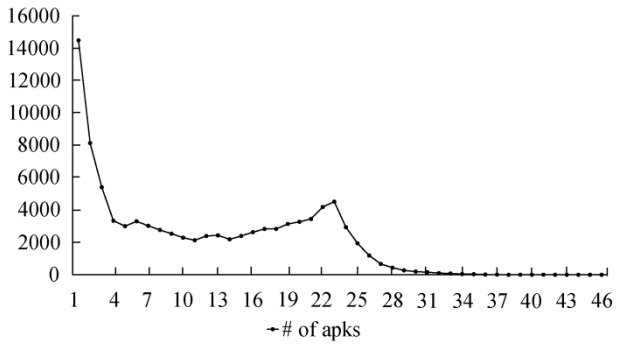


图 7 VirusTotal 检测出来的恶意软件数量

我们继续跟踪了 3711 个恶意应用的作者(共计 2265 名), 对他们的应用进行了为期 15 周的监控(从 2014 年 11 月到 2015 年 2 月)。在这期间, 我们发现这些开发者的 204 个应用消失了, 这些应用都被 MassVet 检测出来。有趣的是我们并没有通过 VirusTotal 去扫描这些应用, 这意味着谷歌也可能利用这些应用的恶意代码片段去定位其他恶意应用, 或者根据作者信息去查找相同作者的其他应用是否包含恶意代码。但显然谷歌没有在全谷歌市场进行相似恶意代码检测(我们发现拥有相同恶意代码的应用仍然在 Google Play 上)。如果这是由于扫描器的性能花费导致的漏扫, MassVet 可以很好的解决这个问题。

我们还发现一些恶意软件作者在他们的应用被移除后, 会再次上传相同或相似的应用到应用商店。在 2125 个重新出现的应用中, 有 604 个(28.4%)被确认为恶意应用, 这些应用没有做任何改动。此外, 这些恶意作者仅通过改名字的方式, 用同样的代码发布了另外共 829 个恶意应用。

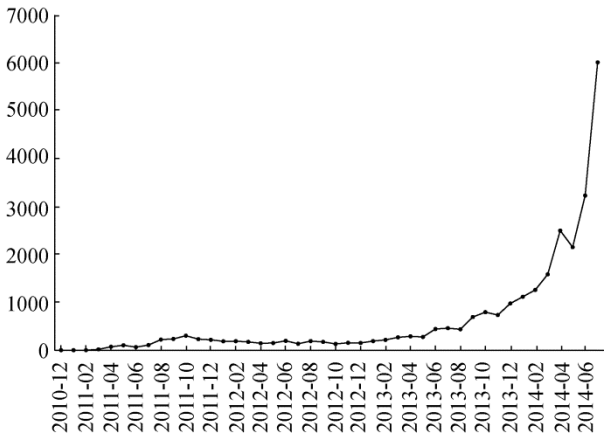


图 8 恶意软件数量随时间变化图

4.4 重打包应用分析

由差异分析模块得到的小部分重打包数据集中, 大多数是来自第三方市场的(92.35%)。

图 9 显示了公共分析过程中, 相同的恶意代码的分布情况。一小部分恶意函数被大量的恶意应用使用。比较热门的代码被 Google Play 上 9438 个恶意应用和第三方市场上 144 个应用使用, 这个函数可能是“com/startapp”这个库的一部分。在使用这个库的应用中, 有 98%被 VirusTotal 标记为恶意应用, 剩下部分我们通过人工分析的方式也推断为恶意应用。这个函数把用户的地理位置信息发送到一个可疑的网站上。其他类型的恶意函数包括在“guohead”、“purchasesdk”和“SDKUtils”等库中。

签名和认证 对于每一个确认的恶意应用, 我们观察其“签名”, 即用来验证应用完整性的 X.509 证书上的公钥。某些签名已经被超过 1000 个恶意软件使用, 显然, 一些恶意软件作者发布了大量的恶意应用, 并在不同的市场进行了传播。另外, 我们通过检查在 Google play 里的发现的恶意软件的 metadata, 发现有些签名已经与很多身份相关联(如 creator 字

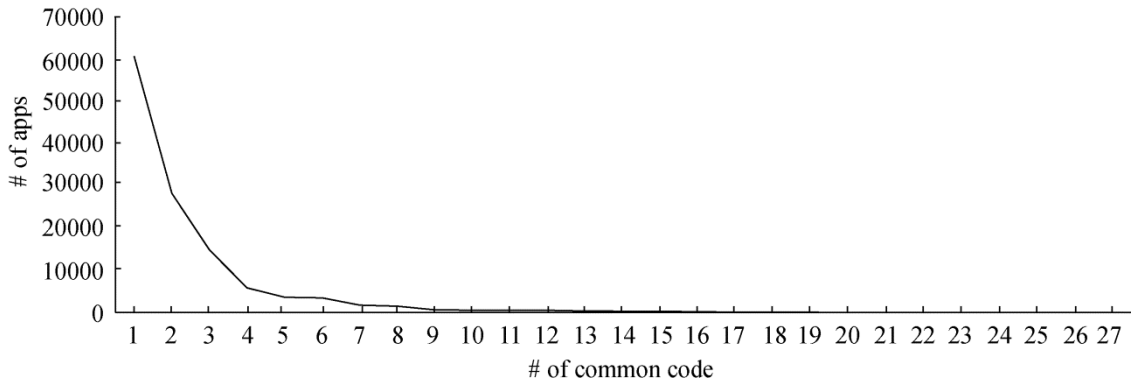


图 9　恶意软件中公共代码分布情况

段)。特别的, 有一个签名已经被 604 个身份关联, 也就意味着恶意作者可能已经创建了很多账户来发布应用。

实例研究　与 VirusTotal 发现的恶意类型不同, MassVet 识别出的可疑应用是一组 APK。我们从这个集合中随机抽样 40 个样本进行分析, 经过人工分析发现其中 20 个确实有问题, 很可能是零日(zero-day)恶意软件。这些恶意软件的行为包括在未经过用户许可的情况下安装应用、收集用户的隐私信息(例如, 获得其他应用的屏幕截图)以及加载和执行本地二进制代码进行控制等。部分恶意软件采用各种各样的技巧来躲避检测。例如, 有些应用在执行恶意代码之前, 先隐藏一段时间, “Durak card game”就是这样类型的一个游戏应用, 其下载量已超过 500 万次, 在 2015 年 2 月 4 日 BBC 报道该恶意软件之前一直存在于 Google play 上[24]。此外, 还有部分恶意应用利用 Java 反射和其他混淆技术来隐藏它们的恶意代码。

5　探讨

正如上述讨论, MassVet 旨在通过检测重打包恶意软件, 发现应用中潜在的恶意代码。这是因为恶意软件的作者很难为了传播其恶意软件, 而去花费大量的时间和金钱去开发一个受欢迎的应用, 所以选择了重打包的方式来传播恶意代码。我们的方法针对这种传播模式的特点, 即利用相同的恶意代码对热门应用进行重打包, 以至减少恶意软件传播的花费。考虑到问题的根本和针对这类恶意软件的有效性, 我们当前的方法是有限的, 尤其是针对一些逃避检测的方法。

特别的, 尽管在当前应用的视图结构中增加垃圾视图会影响用户体验, 甚至影响应用的功能, 但是一个更加有效地方法是混淆视图间的链接(如 StartActivity 调用)。这种处理方式得到的视图结构难以被分析器识别, 这就使得软件应用变得更为可疑。我们也可以对该应用进行动态分析, 基于像 Monkey 这样的工具去分析不同视图之间的连接关系。需要注意的是这样做总体的性能会受到影响, 因为大部分提交给应用商店的应用都是合法的, 且它们中大多数视图结构都是可以由静态分析得到的。

另外, 恶意作者往往通过混淆的方式来躲避相似性分析。正如 3.3 部分所述, 这种做法本身就是无意义的, 因为只有改变应用的控制流程图(CFG)才会导致 m-cores 的变化。只有通过在 CFG 中增加大量的垃圾代码才有可能实现。我们目前的实现还不能处理这种类型的对抗, 目前也没有出现这种类型的恶意代码。如何更好的去处理这种问题还需要进一步的研究。

移除合法的库和代码是 DiffCom 分析的关键。例如, 我们可以利用爬虫从网络上定期的收集共享库和代码模板, 来更新我们的白名单。另外, 可以分析合法的热门应用中的库和代码来弥补爬虫的不足。我们也可以利用一些应用市场的独特的资源, 例如, 虽然应用被不同的证书签名, 但上传应用的账号可能是相同的。同一个组织或机构的多个应用很有可能会用同一个账号去上传。根据上述信息, 应用商店可以通过判断两个应用是否相关, 从而得到他们的共享库。

6　相关工作

恶意软件检测　应用审查大部分依赖于安卓恶意代码检测的方法。现有的大部分检测恶意应用的方法或者是基于代码特征的分析[20, 26, 21, 44, 50, 56, 19, 53, 17, 4], 或者是基于行为特征的分析[11, 30, 47, 46, 41, 18, 33]。这些方法通常依赖大规模的静态或者动态分析技术, 无法用来检测未知的恶意代码或行为。MassVet 利用重打包恶意软件的特点解决了上述问题。PiggyApp[53]

与我们的研究内容相关, 该技术利用一些特征(如权限, API 等)找出两个应用中公共部分, 并找到其他包含该公共部分的应用, 然后将这些应用的其余部分代码进行聚类, 并称之为 piggybacked payloads。从这个集合中人为抽样判断是否确实含有恶意代码。相比较而言, MassVet 通过检查具有相同视图结构应用的不同代码部分和不相关的应用中相同代码, 从而自动的检测出恶意软件。就研究规模而言, ANDRUBIS [25, 45] 在四年内动态的检测了超过 100 万个应用程序, 与 ANDRUBIS 这个离线分析工具不同, MassVet 是一个快速的在线扫描工具, 它可以检测出恶意软件而不需要知道其具体的恶意行为, 它在很短的时间内扫描超过 120 万个应用。

重打包与代码重用检测 重打包与代码重用检测与我们工作相关[54, 21,1, 9, 10, 40, 34, 5]。与 MassVet 最相关的是质心比较方法[7], 该技术可以用来检测出代码重用, 但没有用来检测恶意软件。我们的研究意义在于构建视图结构分析和代码分析模型, 从而实现精确的恶意软件的扫描。同时, 为了防止代码混淆, 利用视图结构的相似性来判断应用的相似关系[49]。然而, 这个方法效率较低, 检测一对应用需要 11 秒的时间。我们提出了一个更加高效的视图结构比较方法, 将应用的视图结构映射为一个几何中心。这大大提高了基于视图分析的方法的性能, 可以用来大规模的分析检测。

7 总结

我们提出了一个新的恶意软件检测方法——MassVet, 它通过比较上传的应用与市场上的现存的应用, 着重检测具有相同视图结构应用的不同代码以及不相关应用的公共代码部分。我们实现了对超过 120 万个应用的分析, 并发现了 127429 个恶意软件, 其中 20 个很可能是零日(zero-day)恶意软件。同时, 相比于市场上主流的反病毒产品, MassVet 实现了更高的覆盖率。

参考文献

[1] ANDROGUARD. Reverse engineering, malware and goodware analysis of android applications ... and more. http://code.google.com/p/androguard/, 2013.

[2] APPBRAIN. Ad networks - android library statistics — appbrain.com. http://www.appbrain.com/stats/libraries/ad. (Visited on 11/11/2014).

[3] APPCELERATOR. 6 steps to great mobile apps. http://www.appcelerator.com/. 2014.

[4] ARZT, S., RASTHOFER, S., FRITZ, C., BODDEN, E., BARTEL,A., KLEIN, J., LE TRAON, Y., OCTEAU, D., AND MCDANIEL, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (2014), ACM, p. 29.

[5] BAYER, U., COMPARETTI, P. M., HLAUSCHEK, C., KRUEGEL, C., AND KIRDA, E. Scalable, behavior-based malware clustering. In NDSS (2009), vol. 9, Citeseer, pp. 8–11.

[6] CHEN, K. A list of shared libraries and ad libraries used in android apps. http://sites.psu.edu/kaichen/2014/02/20/a-list-ofshared-libraries-and-ad-libraries-used-in-android-apps.

[7] CHEN, K., LIU, P., AND ZHANG, Y. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In ICSE (2014).

[8] CISCO. “cisco 2014 annual security report,”. http://www.cisco.com/web/offer/gist ty2 asset/ Cisco 2014 ASR.pdf, 2014.

[9] CRUSSELL, J., GIBLER, C., AND CHEN, H. Attack of the clones: Detecting cloned applications on android markets. ESORICS (2012), 37–54.

[10] CRUSSELL, J., GIBLER, C., AND CHEN, H. Scalable semantics based detection of similar android applications. In ESORICS (2013).

[11] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J.,MCDANIEL, P., AND SHETH, A. Taintdroid: An information flow tracking system for realtime privacy monitoring on smartphones.In OSDI (2010), vol. 10, pp. 1–6.

[12] ENCK, W., OCTEAU, D., MCDANIEL, P., AND CHAUDHURI, S. A study of android application security. In USENIX security symposium (2011), vol. 2, p. 2.

[13] ENVATOMARKET. Android notification templates library. http://codecanyon.net/item/android-notification-templateslibrary/5292884. 2014.

[14] ERNST, M. D., JUST, R., MILLSTEIN, S., DIETL, W. M., PERNSTEINER, S., ROESNER, F., KOSCHER, K., BARROS, P., BHORASKAR, R., HAN, S., ET AL. Collaborative verification of information flow for a high-assurance app store.

[15] F-SECURE. F-secure: Internet security for all devices. http://fsecure.com, 2014.

[16] F-SECURE. Threat report h2 2013. Tech. rep., f-secure, http://www.f-secure.com/documents/996508/1030743/Threat Report H2 2013.pdf, 2014.

[17] FENG, Y., ANAND, S., DILLIG, I., AND AIKEN, A. Apposcopy: Semantics-based detection of android malware through static analysis. In SIGSOFT FSE (2014).

[18] GILBERT, P., CHUN, B.-G., COX, L. P., AND JUNG, J. Vision: automated security validation of mobile apps at app markets. In Proceedings of the second international workshop on Mobile cloud computing and services (2011), ACM, pp. 21–26.

[19] GRACE, M., ZHOU, Y., ZHANG, Q., ZOU, S., AND JIANG, X. Riskranker: scalable and accurate zero-day android malware detec-

tion. In Proceedings of the 10th international conference on Mobile systems, applications, and services (2012), ACM,pp. 281–294.

[20] GRIFFIN, K., SCHNEIDER, S., HU, X., AND CHIUEH, T.-C.Automatic generation of string signatures for malware detection. In Recent Advances in Intrusion Detection (2009), Springer, pp. 101–120.

[21] HANNA, S., HUANG, L., WU, E., LI, S., CHEN, C., AND SONG, D. Juxtapp: A scalable system for detecting code reuse among android applications. In DIMVA (2012).

[22] JING, Y., ZHAO, Z., AHN, G.-J., AND HU, H. Morpheus: automatically generating heuristics to detect android emulators. In Proceedings of the 30th Annual Computer Security Applications Conference (2014), ACM, pp. 216–225.

[23] KASSNER, M. Google play: Android's bouncer can be pwned. http://www.techrepublic.com/blog/it-security/-googleplay-androids-bouncer-can-be-pwned/, 2012.

[24] KELION, L. Android adware 'infects millions' of phones and tablets. http://www.bbc.com/news/technology-31129797, 2015.

[25] LINDORFER, M., NEUGSCHWANDTNER, M., WEICHSELBAUM, L., FRATANTONIO, Y., VAN DER VEEN, V., AND PLATZER, C. Andrubis-1000000 apps later: A view on current android malware behaviors. In Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS) (2014).

[26] LINDORFER, M., VOLANIS, S., SISTO, A., NEUGSCHWANDTNER, M., ATHANASOPOULOS, E., MAGGI, F., PLATZER, C., ZANERO, S., AND IOANNIDIS, S. Andradar: Fast discovery of android applications in alternative markets. In DIMVA (2014).

[27] LU, L., LI, Z., WU, Z., LEE, W., AND JIANG, G. Chex: statically vetting android apps for component hijacking vulnerabilities. In Proceedings of the 2012 ACM conference on Computer and communications security (2012), ACM, pp. 229–240.

[28] NETWORKX. Python package for creating and manipulating graphs and networks. https://pypi.python.org/pypi/networkx/1.9.1, 2015.

[29] OBERHEIDE, J., AND MILLER, C. Dissecting the android bouncer. SummerCon2012, New York (2012)

[30] RASTOGI, V., CHEN, Y., AND ENCK, W. Appsplayground: Automatic security analysis of smartphone applications. In Proceedings of the third ACM conference on Data and application security and privacy (2013), ACM, pp. 209–220.

[31] RASTOGI, V., CHEN, Y., AND JIANG, X. Catch me if you can: Evaluating android anti-malware against transformation attacks. Information Forensics and Security, IEEE Transactions on 9, 1(2014), 99–108.

[32] READING, I. . D. Google play exploits bypass malware checks. http://www.darkreading.com/riskmanagement/google-play-exploits-bypass-malware-checks/d/d-id/1104730?, 6 2012.

[33] REINA, A., FATTORI, A., AND CAVALLARO, L. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. EuroSec, April (2013).

[34] REN, C., CHEN, K., AND LIU, P. Droidmarking: resilient software watermarking for impeding android application repackaging. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering (2014), ACM, pp. 635–646.

[35] SMALI. An assembler/disassembler for android's dex format. http://code.google.com/p/smali/, 2013.

[36] SQUARE, G. Dexguard. https://www.saikoa.com/dexguard,2015.

[37] SQUARE, G. Proguard. https://www.saikoa.com/proguard, 2015.

[38] STATISTA. Statista : The statistics portal. http://www.statista.com/, 2014.

[39] STORM, A. Storm, distributed and fault-tolerant realtime computation. https://storm.apache.org/.

[40] VIDAS, T., AND CHRISTIN, N. Sweetening android lemon markets: measuring and combating malware in application marketplaces. In Proceedings of the third ACM conference on Data and application security and privacy (2013), ACM, pp. 197–208.

[41] VIDAS, T., TAN, J., NAHATA, J., TAN, C. L., CHRISTIN, N., AND TAGUE, P. A5: Automated analysis of adversarial android applications. In Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices (2014), ACM,pp. 39–50.

[42] VIRUSTOTAL. Virustotal - free online virus, malware and url scanner. https://www.virustotal.com/, 2014.

[43] VIRUSTOTAL. Virustotal for android. https://www.virustotal.com/en/documentation/mobile-applications/, 2015.

[44] WALENSTEIN, A., AND LAKHOTIA, A. The software similarity problem in malware analysis. Internat. Begegnungs-und Forschungszentrum f¨ur Informatik, 2007.

[45] WEICHSELBAUM, L., NEUGSCHWANDTNER, M., LINDORFER, M., FRATANTONIO, Y., VAN DER VEEN, V., AND PLATZER, C. Andrubis: Android malware under the magnifying glass. Vienna University of Technology, Tech. Rep. TRISECLAB-0414-001(2014).

[46] WU, C., ZHOU, Y., PATEL, K., LIANG, Z., AND JIANG, X.Airbag: Boosting smartphone resistance to malware infection. In NDSS (2014).

[47] YAN, L. K., AND YIN, H. Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In In USENIX rSecurity 12'.

[48] YAN, P. A look at repackaged apps and their effect on the mobile threat landscape. http://blog.trendmicro.com/trendlabssecurity-intelligence/a-look-into-repackaged-apps-and-its-role-in-the-mobile-threat-landscape/, 7 2014. Visited on 11/10/2014.

[49] ZHANG, F., HUANG, H., ZHU, S., WU, D., AND LIU, P. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec2014). ACM (2014).

[50] ZHANG, Q., AND REEVES, D. S. Metaaware: Identifying me-

ta-morphic malware. In Computer Security Applications Con-fer-ence, 2007. ACSAC 2007. Twenty-Third Annual (2007), IEEE,pp. 411–420.

[51] ZHANG, Y., YANG, M., XU, B., YANG, Z., GU, G., NING, P.,WANG, X. S., AND ZANG, B. Vetting undesirable behaviors in android apps with permission use analysis. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013), ACM, pp. 611–622.

[52] ZHENG, M., LEE, P. P., AND LUI, J. C. Adam: An automatic and extensible platform to stress test android anti-virus systems. In Detection of Intrusions and Malware, and Vulnerability Assess-ment (2013), pp. 82–101.

[53] ZHOU, W., ZHOU, Y., GRACE, M., JIANG, X., AND ZOU, S. Fast, scalable detection of piggybacked mobile applications. In CODASPY (2013).

[54] ZHOU, W., ZHOU, Y., JIANG, X., AND NING, P. Detecting re-packaged smartphone applications in third-party android market-places. In Proceedings of the second ACM conference on Data and Application Security and Privacy (2012), ACM, pp. 317–326.

[55] ZHOU, Y., AND JIANG, X. Dissecting android malware: Charac-terization and evolution. In Security and Privacy (SP), 2012 IEEE Symposium on (2012), IEEE, pp. 95–109.

[56] ZHOU, Y., WANG, Z., ZHOU, W., AND JIANG, X. Hey, you, get off of my market: Detecting malicious apps in official and alterna-tive android markets. In NDSS (2012).

[57] ZORZ, Z. 1.2info. http://www.netsecurity.org/secworld.php?id= 15976, 11 2013. (Visited on 11/10/2014).

附录

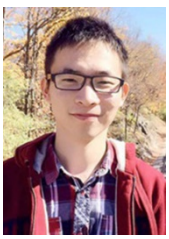
不同市场收集的应用和恶意软件

市场	恶意应用	参与分析应用	百分比	国家
Anzhi	17921	46055	38.91	China
Yidong	1088	3026	35.96	China
yy138	828	2950	28.07	China
Anfen	365	1572	23.22	China
Slideme	3285	15367	21.38	US
AndroidLeyuan	997	6053	16.47	China
gfun	17779	108736	16.35	China
16apk	4008	25714	15.59	China
Pandaapp	1577	10679	14.77	US
Lenovo	9799	68839	14.23	China
Haozhuo	1100	8052	13.66	China
Dangle	2992	22183	13.49	China
3533 world	1331	9886	13.46	China
Appchina	8396	62449	13.44	China
Wangyi	85	663	12.82	China
Youyi	408	3628	11.25	China
Nduo	20	190	10.53	China
Sogou	2414	23774	10.15	China
Huawei	148	1466	10.1	China
Yingyongbao	272	2812	9.67	China
AndroidRuan-jian	198	2308	8.58	China
Anji	3467	41607	8.33	China
AndroidMarket	1997	24332	8.21	China
Opera	4852	61866	7.84	Europe
Mumayi	6129	79594	7.7	China

陈恺　于 2010 年在中国科学院大学获得博士学位，现在中国科学院信息工程研究所担任副研究员。他的研究兴趣包括软件分析与测试，智能手机、隐私安全。
Email: chenkai@iie.ac.cn

Yeonjoon Lee　现在印第安纳大学计算机专业攻读博士学位。研究领域为信息安全。研究兴趣包括：移动系统安全、病毒检测。
Email: yl52@indiana.edu

张楠　于 2012 年在东北大学计算机应用技术专业获得硕士学位，现在美国印第安纳大学计算机科学专业攻读博士学位。研究领域为系统安全、移动设备安全。研究兴趣包括: Android 系统安全, IOS 系统安全和物联网设备安全。
Email: nz3@indiana.edu



王鹏　于 2014 年在中国人民大学计算机专业获得硕士学位。现在印第安纳大学计算机专业攻读博士学位。研究领域为信息安全。研究兴趣包括：移动系统安全、病毒检测。Email: pw7@indiana.edu

王晓峰　于 2004 年在卡内基梅隆大学(CMU)获得计算机工程博士学位，现在印第安纳大学伯明顿分校计算机科学系教授，研究兴趣包括：系统安全、隐私保护、人类基因组隐私、应用密码学等。
Email: xw7@indiana.edu

黄鹤清　于 2010 年在华中科技大学专业获得计算机学士学位。现在宾州州立学校手机系统安全专业攻读博士学位。研究兴趣包括：手机系统安全，手机程序安全，大数据分析。
Email: hnh5043@cse.psu.edu

邹维 于 1988 年在中国科学院计算技术研究所计算机软件专业获得硕士学位。现任中国科学院信息工程研究所研究员。研究领域为软件安全分析。研究兴趣包括: 规模化软件漏洞挖掘技术。
Email: zouwei@iie.ac.cn

刘鹏 于 1999 年在美国 George Mason University 获得博士学位。现为美国宾夕法尼亚州立大学信息科学与技术学院教授, 信息安全实验室主任。研究领域为系统安全。
Email: pliu@ist.pst.edu