行为特征值序列匹配检测Android恶意应用

张 震,曹天杰
ZHANG Zhen, CAO Tianjie

中国矿业大学 计算机科学与技术学院,江苏 徐州 221116
School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China

ZHANG Zhen, CAO Tianjie. Detecting Android malware based on matching sequence of behavioral characteristic value. Computer Engineering and Applications, 2018, 54(24):97-102.

Abstract: Aiming at the problem that the existing methods can not detect obfuscated, hidden, and encrypted Android malicious code effectively, a dynamic detection method based on a malicious application behavior feature value sequence is proposed. Firstly, the dynamic detection module is injected into the Zygote process of the Android system through the remote injection technology, and then an inline hooking technology is executed to monitor important functions in the application. The important behavior of the detected Android application is obtained through the function. Further, the behavior is quantized into characteristic values according to the characteristic, and then the behavior characteristic values are arrayed as the sequence of behavior characteristic values in chronological order. Using support vector machines to train 5, 560 malicious samples, a sequence of behavioral eigenvalues of a malicious application family is obtained. Finally, this sequence is compared with the sequence of the detected application to determine whether the application is a malicious application. The rate of correct detection in malicious applications can reach 95.1%, and only 21.9 kilobytes of memory can be used for detected applications. Experimental results show that the proposed method can detect the obfuscated, encrypted and hidden malicious code, improve the accuracy of malicious application detection, reduce the memory space, and effectively improve the detection results.

Key words: Android malware; remote inline hook; dynamic detection; support vector machine; feature sequence

摘 要: 针对Android恶意代码的混淆、隐藏、加密情况以及现有方法的检测能力不足问题,提出了一种基于恶意应用行为特征值序列的动态检测方法。首先利用远程注入技术将动态检测的模块注入到Android系统的Zygote进程中,执行内联挂钩来监测应用中的重要函数。然后,通过函数监听得到Android应用的重要行为;进而,按照行为的特征将其量化为特征值,再按照时间顺序将行为特征值排为序列,得到行为特征值序列。通过利用支持向量机来训练5 560个恶意样本,得到恶意应用家族的行为特征值序列;最后利用此序列与被检测应用的序列进行相似度比较,判断应用是否为恶意应用。在恶意应用动态检测方面的正确率可达到95.1%,以及只增加被检测的应用21.9 KB内存。实验结果表明,所提方法能够正常检测经过代码混淆、代码加密、代码隐藏的恶意应用,提高了恶意应用检测的正确率,所占内存空间减少,有效提升检测效果。

关键词: Android恶意应用;远程内联挂钩;动态检测;支持向量机;特征值序列
文献标志码: A **中图分类号:** TP309.1 doi:10.3778/j.issn.1002-8331.1805-0241

1 引言

2017年全年,Android平台新增的恶意程序样本被截获的一共有757.3万个,平均每天新增2.1万个。因此,如何高效准确地检测出Android恶意应用,成为了一个亟需解决的问题。

针对Android恶意应用的检测,有静态检测的方法提出。其中,Arp[1]等人提出了一种基于静态的分析方法,它是从Android安装包解包得到的AndroidManifest.

基金项目: 国家自然科学基金(No.61303263)。
作者简介: 张震(1993—),男,硕士研究生,研究领域为移动信息安全、系统安全,E-mail:juanhaha@cumt.edu.cn;曹天杰(1967—),男,博士,教授,研究领域为密码学、信息安全。
收稿日期: 2018-05-15 **修回日期:** 2018-06-27 **文章编号:** 1002-8331(2018)24-0097-06

xml文件和经过反编译得到的代码文件中分析出特征，并将特征分为多个集合，再利用支持向量机将特征集中的分为良性和恶性，然后根据集合来对恶意软件进行特征分类。而Guo[2]等人使用了一种数字序列的方法来静态检测第三方广告包，利用apktool和dex2jar工具反编译得到Java源码，通过分析Java源码生成的第三方恶意包的特征序列。

而在动态检测方面，Xu[3]等人使用Aurasium对原Android安装包插入监控代码并重打包，通过修改全局偏移表来重写APIs达到动态检测监控效果。Dash[4]等人则是提出了一种基于运行时行为的Android恶意应用的分类，它将Android运行时的行为分为4类，并利用支持向量机对运行时中的良性行为和恶意行为进行分类，并且通过CopperDorid[5]系统，分析Android应用执行过程中的系统调用。并重构出Android应用中的恶意行为[6]。Yuan[7]等人则是利用DroidBox与TaintDroid[8]进行污点动态跟踪，来获取Android应用行为，利用深度学习算法来分析Android应用的行为来判断是否为Android恶意应用。

但是，静态检测的方法拥有局限性。目前的应用市场上的Android应用都拥有代码保护措施，用于抵抗反编译分析[9]。不仅如此，静态分析对于恶意代码混淆的分析效率低下，甚至无法检测出经过隐藏和加密的恶意代码。因此，本文所提方法为了解决静态检测的局限性，使用了动态检测来对Android应用进行检测。

而在动态检测方面。Aurasium所使用对于动态检测的方法是利用反射机制来对Java层的敏感APIs进行监听。而对于Native层的系统APIs则是利用全局偏移变量表来对目标函数进行挂钩，但是，此方法不能监听不在全局偏移变量表中出现的函数。而在Dash等人使用的方法，需要将没有修改过的镜像体统运行在一个特别修改制定的Android模拟器中，对于系统环境的局限性过高，不能适应绝大多数系统。而Yuan等人使用的动态分析方法中的深度学习，还可以再进行优化。因此，对于以上问题，本文使用的方法在监听Native层是利用Inline Hook技术来对敏感APIs进行监听，此方法可以在Android应用中的任意一处进行Hook。而本文所使用的方法是在Zygote进程当中注入动态检测模块，既可以在Android模拟器上实现，也可以在Android的真机上实现，能够适应多种运行平台。而在检测的方法上，本文提出了一种基于行为特征值序列的匹配检测方法，通过行为特征值序列，可以更加直观高效地得到Android应用的行为，再配合支持向量机来进行深度学习，效率和准确率都有一定的提高。

2 动态获取应用行为

本文实现了一个能够对Android应用的系统调用进行动态监听的方法。该方法能够适应绝大多数以ARM的32位处理器为架构的Android系统。动态检测模块主要分为两个步骤：第一步是将能够实现动态检测模块的共享库注入到Android系统的Zygote进程当中；第二步是利用注入的共享库来对系统调用进行监听，并且重构能记录下Android应用的行为。

2.1 在Zygote中注入动态检测模块

Zygote[10]进程是Android系统运行的第一个Dalvik虚拟机进程，Android系统的所有其他的APP进程都由它“孵化”。那么Zygote“孵化”其他APP进程的方式实际上是使用了*fork*()函数创建一个子进程供APP运行。而Android系统是基于Linux内核[11]，因此*fork*()函数创建的子进程(运行Android应用的进程)复制了父进程(Zygote进程)的副本，当执行动态检测的共享库注入到了Zygote进程当中，Zygote进程即加载了动态检测的模块，当系统中的Android应用打开时，会从Zygote进程中复制一个拥有动态检测模块的副本。通过Zygote进程执行fork函数的注入，可以绕过被检测应用的反调试检测。

把动态模块加载到Zygote进程，需要远程注入技术把模块加载到Zygote进程当中。Linux中提供了*ptarce*()函数，可以使得一个函数观察和控制另外一个进程。Android系统是基于Linux内核的。注入的流程如图1所示，注入器(Tracer)远程调用*mmap*()函数为Zygote进程(Tracee)申请一个能够执行加载动态检测模块的shellcode的空间，然后控制Zygote进程执行加载动态检测模块的shellcode。加载之后，然后注入器只需要Detach，取消对Zygote进程的附加即完成了动态检测模块的注入。

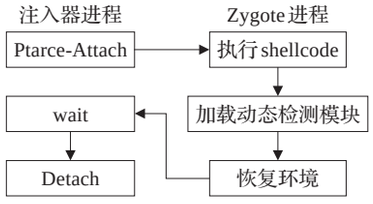

图1 远程注入动态模块到Zygote进程

2.2 对系统调用进行挂钩

动态检测模块实际上是一个共享库(也就是一个ELF文件)。当完成共享库的链接和装载之后，会首先调用ELF文件中的构造函数。因此，动态检测模块中对Zygote进程的系统调用的挂钩的操作就是在构造函数中完成。而且构造函数执行时间是比其他函数要早，因此可以避免在进行挂钩时与其他子进程或者子线程产生冲突而造成崩溃。

本文所使用的挂钩方法是使用了内联挂钩。内联挂钩的优点是能够对应用内的任何一处进行挂钩，而全局偏移表的挂钩则不能够对表外的函数进行挂钩。利

用内联挂钩技术修改重写内存中任意一处的指令,并且跳转到自定义的函数当中。此方法与系统平台可执行文件的汇编指令集密不可分,本实验使用的动态检测模块只针对ARM指令、Thumb32和Thumb16的指令的内联挂钩,流程如图2所示。首先是获得系统调用API的入口地址,将入口地址处的指令修改为跳转到执行监听并记录应用行为的函数,执行完毕之后恢复到正常的函数逻辑当中。

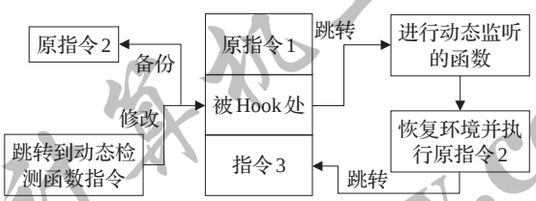

图2　Inline Hook实现原理流程图

对于Java层的APIs的挂钩,由于Java层的代码是基于DVM(DalvIk Virtual Machine)的,因此并不能直接找到函数入口地址来进行内联挂钩。而JNI提供了一种反射的方法,可以使得在原生(Native)层调用Java层的函数。而且JNI还提供了方法将Java层的方法转为Native函数,当Java层的函数成功转为Native函数后,即和Native层的系统调用一样可以使用内联挂钩来对其进行挂钩,从而实行监听。

3　基于行为的恶意应用检测

3.1　检测模块动态监听的行为

由于Android系统中的APIs数量庞大,若监听Android应用的所有APIs会极大地降低检测效率。因此本文只挑选其中重要的APIs进行监听。并将这些APIs的行为按照行为特征分为4类,并用4个集合表示。如表1所示,第一类特征是网络访问行为,第二类是文件访问行为,第三类是运行可执行文件的行为,第四类则是访问Java层敏感APIs[12]的行为。

表1　行为特征集的分类以及监听内容

集合符号	行为名称	监听内容
S_1	网络访问行为	访问的IP地址,端口号,发送的数据,接收的数据
S_2	文件访问行为	打开的文件的文件名,打开模式
S_3	运行可执行文件行为	执行的文件名,执行参数
S_4	调用敏感APIs行为	被调用的敏感函数的名称,调用时的参数

S_1,网络访问行为。网络访问行为是Android应用最普遍的行为,恶意应用会利用互联网下载恶意软件,下载广告软件并静默安装,并且将用户的隐私数据上传到收集数据的主机当中。因此,动态监听Android应用的网络访问行为,通过获得调用connect时的参数,可以得到应用访问的服务器的IP地址以及访问端口。通过监听*send*()系列函数和监听*recv*()系列函数,前者可以得到Android应用向目标主机发送的数据;而后者可以得到Android应用从目标主机处接收的数据。

S_2,文件访问行为是Android应用打开访问文件的行为,由于Android系统中,用户存在着很多其他的隐私文件,例如通讯录数据,照片数据。或者是将隐藏的恶意代码释放出来,或是打开恶意代码文件。因此,监听Android应用的文件访问行为的主要函数有*dlopen*(),*open*(),*fopen*()等函数。例如通过*dlopen*()函数的参数可以得知Android应用加载模块的名称。通过截取文件访问函数的参数,可以得到打开的文件的路径,从而得知Android应用访问了的文件名称。

S_3,运行可执行文件行为,指Android文件运行可执行文件的行为。例如APP可能会利用pm命令来进行静默安装广告软件甚至是恶意软件,或者利用可执行文件执行一些其他的窃取系统信息或者是提升用户权限的命令。对于此类的行为的访问的监听*popen*()函数,*popen*()函数会创建一个子进程执行运行文件,因此可以截获参数来得知Android应用执行命令。同理,截获*system*()函数以及是*exec*()系列函数中的参数,也可以得到Android应用执行的命令。

S_4,调用敏感APIs行为。Android系统提供了APIs来获得手机信息,这些信息涉及用户隐私,因此对于此类行为也需要动态监听。例如*getDeviceId*()此函数是用于获得设备ID,而*getLastKnownLocation*()函数则可以获取手机的位置信息,又或者是*sendTextMessage*()函数可以利用手机设备发送信息。虽然这些APIs涉及的数据都非常敏感,但是部分良性应用的功能也会利用这些敏感信息,有可能会造成错误地将良性应用判断为恶意应用。因此需要分析函数调用顺序甚至结合集合S_1的网络访问行为来监听这些敏感数据的去向。

3.2　行为特征值序列

本文实验会从来自Arp[1]等人收集的5 560个恶意应用样本中将以上S_1~S_4共4种特征集中进行行为监听和收集,并且将收集的行为进行编号。编号的方式为$X_1X_2X_3$。编号的第一个数X_1是指行为特征集合所属类型。编号的第二个数字X_2则是特征行为集合中所调用的函数。而编号最后的数字X_3就是函数中的参数值。从表2中举个例子,当监测到Android应用调用了*connect*()函数访问了一个黑名单上的IP地址以及其端口,则编号为111。编号111中的第一个数字1所代表的是该行为是来自集合S_1的,而第二个数字1则代表了应用是从集合S_1调用了被监听的*connect*()函数,而最后一个数字1代表了调用*connect*()函数所要访问的IP地址是来自黑名单的。

表2 行为编号列表的部分展示

行为特征所属集合	监听的函数名	监听的参数	编号
S_1	*connect*()	黑名单中的IP地址	111
	connect()	白名单中的IP地址	112
	connect()	不在黑名单和白名单中的IP地址	113
	sendto()	发送内容有敏感信息	121
	sendto()	发送的内容没有敏感信息	122
	…	…	…
S_2	*fopen*()	打开了敏感文件	211
	fopen()	打开了正常文件	212
…	…	…	…

而要构造由行为编号组成的数字序列,则需要训练集来构造行为的编号以及每个恶意应用家族特定的恶意行为的数字序列。首先要将行为集合量化为编号集合,假设每个行为的判断为 s ,而 s 是来自5 560个用于数据训练的恶意样本,构成的恶意样本行为集合 $S_m=\{s_1,s_2,\cdots,s_n\}$, n 表示当前所有行为的集合。而 s 行为对应的编号为 x ,而集合 X 是由 x 组成的集合 $\{x_1,x_2,\cdots,x_n\}$ 。然后从集合 X 中的所有编号中组成一个数字序列 A 。

A 是由样本中的应用中的所有的行为的编号组成的序列。当动态检测模块中检测到应用的行为 s 时,则判断 s 是否存属于集合 S 当中,如果 s 属于集合 S ,则将 S 中的 s_i 所对应的 x_i 加入到序列 A 当中;若 s 不属于集合 S ,则将行为 s 以 s_{n+1} 并入到集合 S_m 当中,同时将所对应的 x_{n+1} 加入到 X 集合当中, A 序列加入行为编号 x_{n+1} 。所有被监听到的行为的编号组成序列 $A=a_1,a_2,\cdots,a_n$ 。

3.3 最长公共子序列特征行为

假设训练集上一共有 n 个恶意应用家族分类,每个家族的被检测的样本组成集合 $y,\{y_1,y_2,\cdots,y_n\}$ 。假设第 i 个家族集合 y_i 的数量 $|y_i|$ 所拥有的数量为 m_i 个,即恶意应用对应的行为特征序列为 $a_{ij}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$,即计算第 i 个家族中的所有的 m_i 个应用的所有行为特征序列的最长公共子序列。这里使用了迭代的思想,如公式(1)所示:

$$a=\begin{cases}a_i, i=1\\ f(a,a_i), 1<i\leqslant m\end{cases} \tag{1}$$

a 表示某个家族的特征行为序列, m 是样本中该家族的总数。而 $a_i(1\leqslant i\leqslant m)$ 则是家族中第 i 个应用的行为特征数列。而函数 f 则是求两个数字序列的最长公共子序列。假设输入的数列是 b 序列与 d 序列, f 函数的内容如公式(2)所示:

$$c[i,j]=\begin{cases}0, \text{if } i=0 \text{ or } j=0\\ c[i-1][j-1]+1, \text{if } i,j>0 \text{ and } b_i\neq d_i\\ \max(c[i,j-1],c[i-1,j]), \text{if } i,j>0 \text{ and } b_i\neq d_j\end{cases} \tag{2}$$

序列 c 是输出的 b 序列和 d 序列的最大公共子序列,可以通过动态规划的方式求出最长公共子序列的值。

3.4 利用SVM提高检测正确率

通过公式(1)和公式(2),可以计算出每个家族的特征行为序列。以这个序列为标准,和之后输入的Android应用的序列进行匹配,若与恶意家族标准序列中的其中一个相同,则判输入的Android应用为此家族的恶意应用。但是,这种方法需要解决两个重要的问题。

问题1 假若序列符合两种或以上的恶意家族序列,则需要继续从其他的行为中判断出其究竟属于哪一种恶意应用家族。

问题2 假若某种行为可以用功能相似的行为所替代,则会使得序列匹对失败,降低成功率。

为解决以上两个问题,本实验使用了SVM支持向量机。利用SVM支持向量机来对训练集上的学习进行优化。SVM能够自动创造一个基准线来将某个家族的相似的恶意行为、恶意应用进行划分,将其归类到同一个序列当中。这样就能够有效解决问题2的行为相似的难以判断的情况。不仅如此,还有非一致性特征行为的基准线,由于存在某些行为是恶意家族中个别Android应用中没有存在的行为,但是能够标识家族的特点行为,也需要划分基准线,如表3。

此类基准的划分就是按照该行为的编号出现的次数,倘若次数超过了SVM划分的基准线,入到家族的特征行为序列当中。

3.5 检测流程小结

总体来说,检测流程可以归纳为两点。分别是应用行为监听与行为特征值序列匹配检测。

在行为监听流程中,将动态检测模块远程注入到Zygote进程当中,然后利用Inline Hook将监听收集应用的行为。这样,就可以在Android系统不允许跨进程访问的条件下进行动态检测模块的注入并执行。同时,Inline Hook也可以在进程的任意一处进行修改,具有良好的扩展性,便于增添或者是减少需要监听的API。

通过上述的动态检测模块监听到了Android应用程序的行为,利用特征序列匹配来检测应用。利用每个家族的行为特征值序列与样本应用的序列进行匹配,其最长公共子序列达到阈值则判断样本是否为恶意应用软件。从行为特征值序列中找出恶意行为,能够将应用抽象的行为以数字序列的方式表达,从而能够得到应用行为的内容以及行为的时序,极大地提高了检测效率以及准确率。这里利用是SVM来优化家族的行为特征值向量以及判断的阈值,能够有效地区分相似性和可替代行为,提高检测的准确率。

表3　恶意家族的行为特征值列表

恶意应用家族	行为特征值序列
Plankton[13]	413 432 472 315 111 222 221 413 411 122 … 311 122 131
DroidKungFu[13]	422 413 412 221 312 211 111 122 121 421 … 113 123 133
GinMaster[13]	142 155 311 322 313 245 221 132 432 122 … 451 463 462
FakeDoc[13]	311 332 451 472 111 113 241 352 343 232 113 413
FakeInstaller[13]	341 491 441 223 311 322 312 111 113 233 … 211 234 431
Opfake[13]	412 311 324 213 221 333 471 312 223 112 … 133 112 222
Adrd[13]	321 222 111 321 122 132 112 221 322 411 223 471 421 132
Geinimi[13]	471 213 211 222 312 421 441 111 123 132 … 431 462 212
Golddream[13]	461 232 411 314 221 111 112 324 311 223 … 411 231 341
…	…

4　实验结果分析

4.1　实验样本

为了得到本文的训练集,这里使用了 Arp[1]等人的收集的5 560个恶意软件集合。而对于本实验的方法进行验证,从 VirusShare[14]网站,以及国内的应用宝、豌豆荚、百度手机助手、360手机助手等应用市场中获得1 000个样本。其中有700个恶意样本以及300个良性样本。这个1 000个样本已经在 VirusTota[14]网站上进行检测过,确定是700个恶意样本以及300个良性样本。

4.2　度量指标

本实验使用了召回率、准确率、正确率这3种检测指标进行检测。并且使用了正确分类的恶意软件数量(*TP*),正确分类良性应用的数量(*TN*),错误分类恶意应用数量(*FP*),错误分类良性应用的数量(*FN*)。由此4个变量来计算3种检测指标。

(1)召回率(recall rate)。恶意应用软件被正确地检测的数量在所有的恶意软件中的比例:$R_{\text{recall}}=TP/(TP+FN)$。

(2)准确率(precision rate)。良性应用被正确检测的数量在所有良性软件中的比例:$R_{\text{precision}}=TN/(TN+FN)$。

(3)正确率(accuracy)。被正确检测的软件的数量在所有软件的数量中的比例:$R_{\text{accuracy}}=(TP+TN)/(TP+TN+FP+FN)$。

4.3　实验结果

本实验的环境是 Inter@Core™ i3-2100 CPU,内存是2 GB,系统是 Ubuntu Kylin 14.04。在此基础上创建的 Android 虚拟机,内存为2 GB,为 Android5.1.1 系统。本文所使用名为 Monkey 的 Android 程序,随机模拟出用户在 Android 系统上的各种操作,并且模拟 Android 的各种消息,来模拟动态测试应用软件。

本文的实验结果和静态分析工具 drebin[1]和 Androguard[15]相比较。这两个对于5 560个的样本检测的准确率虽然高。但是对于从应用下载的1 000个应用的检测的准确率却非常低。成功率几乎为0,原因是因为现在的 Android 应用都添加了代码保护壳,代码混淆等对抗反编译的措施。因此直接反编译 APK 文件中的 .dex 文件不能得到应用真正的 Java 代码。但是本文实验的方法却没有收到代码保护壳的丝毫影响。

表3是通过训练集得到的每个家族的行为特征值序列。通过每个家族的特征值来对输入的 Android 应用进行动态检测,得到输入的 Android 应用额度行为特征值序列,再与这些家族额度特征值序列进行比较。

由于输入的应用的序列可能与训练出来的恶意应用的家族的行为特征值序列有一定的偏差值。这个偏差值最大限度则设置为阈值。通过阈值的调整来优化实验的动态检测的效果。这里使用阈值的是从0.5~1.0选取,精度为0.1,如图3所示。

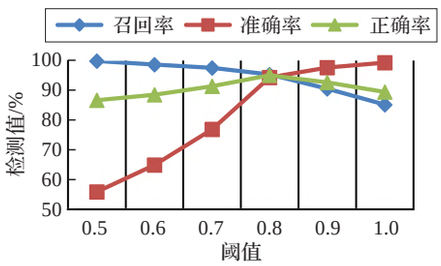

图3　不同阈值下的动态检测的召回率、准确率、正确率

由图3中的数据得到,当阈值为0.8时的正确率是最高的。此时的正确率为95.1%。DroidScribe[4]与本文所提方法使用均是来自 Arp[1]的同一个训练集。而 DroidScribe 对于从应用市场等收集的1 000个样本的正确率最高只能够达到94%左右。IntelliDroid[16]虽然与本文使用的不是同一个训练集,用其对本文收集的1 000个样本进行检测,成功率最高只能达到90%。因此,本文所使用的方法最高的正确率比 DroidScribe 的正确率高1%左右,而比 IntelliDroid 的正确率高5%左右。表明本文方法检测恶意应用的正确率更高。而 Aurasium[3]的正确率最高能够达到95%,而本实验的正确率只能高0.1%左右,因此从正确率上与 Aurasium 相比并不能体现出优势。但是,Aurasium 与本实验都是将动态检测的代码注入到被检测应用的进程当中去检测,注入动态检测代码需要占据一定的内存。

表4 动态检测模块注入所占空间大小的比较

动态检测方法	平均额外占用空间大小/KB
本文方法	21.9
Aurasium[3]	52.2
DroidScribe[4]	2 200.0
IntelliDroid[16]	12 300.0

这里使用的检测方法利用查看动态模块的so文件的大小以及是利用Android系统的shell命令“cat/prop/[pid]/status”[17]查看增加的实际内存占用。发现实际增加的内存占用和要加载的动态检测模块的so文件相似,平均值大概就是21.9 KB。远远少于DroidScribe和IntelliDroid方法所消耗的内润,与占用内存最少的Aurasium相比,比其占用的52.2 KB将近少了一半以上。由此可知,在正确率相近的情况下,本文使用的方法占用更少的系统资源,更加轻量级。

5 结束语

本文通过以时间为顺序的Android应用行为构成一个行为特征值序列。以动态检测的行为时间顺序的序列作为判断恶意应用以及恶意家族分类的重要依据。以数字序列作为描述恶意家族的特征,检测正确率更高;再结合支持向量机的优化,更有效地将动态检测的行为量化作为特征值进行恶意应用检测。而且实验用的动态检测模块适用范围更宽广,占用系统资源也更少,属于高效轻量级的检测方法。

本文的正确率最高值只有95.1%,与主流的恶意检测软件的正确率相比较是要低的,因此本方法还拥有提升的空间。提升的空间主要有支持向量机的参数优化,得到更加符合恶意应用行为特征值序列,使得实验的正确率能够进一步地提高。并且还能够再对动态检测的时机进行优化,本实验只有对API,进行挂钩;假若能够动态检测更加底层的kernel函数,则能够更加准确地捕获到应用的行为。

参考文献:

[1] Arp D,Spreitzenbarth M,Hübner M,et al.DREBIN:effective and explainable detection of Android malware in your pocket[C]//Network and Distributed System Security Symposium,2014.

[2] Guo Y,Hou J,Chen W,et al.A method to detect Android ad plugins based on decompiled digital sequence[C]//International Conference on Cloud Computing and Intelligence Systems,2016:104-108.

[3] Xu R,Anderson R.Aurasium:practical policy enforcement for Android applications[C]//Usenix Conference on Security Symposium,2012:27.

[4] Dash S K,Suarez-Tangil G,Khan S,et al.DroidScribe:classifying Android malware based on runtime behavior[C]//Security and Privacy Workshops,2016:252-261.

[5] Yan L K,Yin H.DroidScope:seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis[C]//Proceedings of the 21st USENIX Conference on Security Symposium,2013:29.

[6] Reina A,Fattori A,Cavallaro L.A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors[C]//ACM European Workshop on Systems Security,2013:1-6.

[7] Yuan Zhenlong,Lu Yongqiang,Xue Yibo.Droid detector:Android malware characterization and detection using deep learning[J].Tsinghua Science and Technology,2016,21(1):114-123.

[8] Enck W,Gilbert P,Chun B G,et al.TaintDroid:an information flow tracking system for real-time privacy monitoring on smartphones[J].ACM Transactions on Computer Systems,2010,32(2):1-29.

[9] Maiorca D,Ariu D,Corona I,et al.Stealth attacks:an extended insight into the obfuscation effects on Android malware[J].Computers & Security,2015,51:16-31.

[10] Ding Y,Peng Z,Zhou Y,et al.Android low entropy demystified[C]//2014 IEEE International Conference on Communications,2014:659-664.

[11] 尹锡训.ARM Linux内核源码剖析[M].崔范松,译.北京:人民邮电出版社,2014:408-418.

[12] 罗文塽,曹天杰.基于非用户操作序列的恶意软件检测方法[J].计算机应用,2018,38(1):56-60.

[13] Zhou Y,Jiang X.Dissecting android malware:characterization and evolution[C]//2012 IEEE Symposium on Security and Privacy(SP),2012:95-109.

[14] 侯勤胜,曹天杰.基于网络行为分析的Android恶意软件动态检测[D].江苏 徐州:中国矿业大学,2017.

[15] Drake J J,Mulliner C,Fora P O,et al.Android安全攻防权威指南[M].诸葛建伟,杨坤,肖梓航,译.北京:人民邮电出版社,2015:72-97.

[16] Wong M Y,Lie D.IntelliDroid:a targeted input generator for the dynamic analysis of android malware[C]//Network and Distributed System Security Symposium,2016.

[17] 吴文刚,张志文.信息安全等级保护Linux服务器shell脚本测评方法[J].现代工业经济和信息化,2017,7(13):59-61.