TaintART: A Practical Multi-level Information-Flow Tracking

System for Android RunTime

Mobile operating systems like Android failed to provide suf-ficient protection on personal data, and privacy leakage be-comes a major concern. To understand the security risksand privacy leakage, analysts have to carry out data-flowanalysis. In 2014, Android upgraded with a fundamentallynew design known as Android RunTime (ART) environ-ment in Android 5.0. ART adopts ahead-of-time compi-lation strategy and replaces previous virtual-machine-basedDalvik. Unfortunately, many data-flow analysis systems likeTaintDroid [19] were designed for the legacy Dalvik environ-ment. This makes data-flow analysis of new apps and mal-ware infeasible. We design a multi-level information-flowtracking system for the new Android system called Taint-ART. TaintART employs a multi-level taint analysis tech-nique to minimize the taint tag storage. Therefore, tainttags can be stored in processor registers to provide efficienttaint propagation operations. We also customize the ARTcompiler to maximize performance gains of the ahead-of-time compilation optimizations. Based on the general de-sign of TaintART, we also implement a multi-level privacyenforcement to prevent sensitive data leakage. We demon-strate that TaintART only incurs less than 15% overheadson a CPU-bound microbenchmark and negligible overheadon built-in or third-party applications. Compared to legacyDalvik environment in Android 4.4, TaintART achievesabout 99.7% faster performance for Java runtime benchmark.

Mobile devices such as smartphones, tablets and wearabledevices are widely used for communication, photo taking,entertainment, and monitoring health status. Many appli-cations (apps for short) installed on the smartphones provideuseful services, but they may also privately send sensitive in-formation to remote servers for various data analytics [12].Worse yet, some of them can gain profit from these personaldata [38]. Furthermore, malware can secretly steal sensitivePermission to make digital or hard copies of all or part of this work for personal orclassroom use is granted without fee provided that copies are not made or distributedfor profit or commercial advantage and that copies bear this notice and the full citationon the first page. Copyrights for components of this work owned by others than theauthor(s) must be honored. Abstracting with credit is permitted. To copy otherwise, orrepublish, to post on servers or to redistribute to lists, requires prior specific permissionand/or a fee. Request permissions from permissions@acm.org.CCS’16, October 24 - 28, 2016, Vienna, Austriac ? 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.ISBN 978-1-4503-4139-4/16/10...$15.00DOI: http://dx.doi.org/10.1145/2976749.2978343information such as contact lists without users’ consent. Allthese indicate that privacy leakage is a serious threat to alarge community of mobile users.To understand the possibility of privacy leakage, research-ers seek solutions in two directions of data-flow analysis.Firstly, with the disassembled code of a given app, research-ers can perform static data-flow analysis techniques such asstatic taint analysis and symbolic execution. This type ofmethods can statically derive a set of possible data whichmay leave devices at runtime, and decide whether sensitivedata leaks to untrusted channels. The limitation of thismethod is that it cannot detect runtime information dis-closures when the app developers use techniques such ascode with Java reflection, code encryption, or dynamic codeloading techniques. Therefore, researchers proposed to usedynamic methodologies to monitor suspicious behaviors atruntime. The dynamic taint analysis technique [46] is one ofmany dynamic methodologies which can track the informa-tion flows within apps at runtime. The dynamic taint anal-ysis technique will label (taint) sensitive data from certainsources and handle label transitions (taint propagation) be-tween variables, files, and procedures at runtime. If a taintedlabel transmits out of the mobile device through some func-tions (sinks), one can then monitor the data leakage dynam-ically. This method can accurately track data flows at anapp’s execution time.TaintDroid [19] is a notable dynamic taint analysis systemfor Android apps. It customizes Android runtime (DalvikVirtual Machine) to achieve taint storage and taint prop-agation. Many systems [16, 63, 5, 43, 42, 54] are basedon TaintDroid to conduct further analysis. However, thereare several constraints which make TaintDroid can no longerfunction on the latest Android for privacy tracking and mal-ware analysis (and to a certain extent, data flow analysis).Firstly, TaintDroid was originally designed for virtual-machine-based system (i.e., Dalvik virtual machine), andimplemented on legacy Android systems 2.1, 2.3, 4.1, and4.3. TaintDroid utilizes the internal memory of Dalvik vir-tual machine for taint storage and propagation. To enhancethe performance of Android, Google recently changed to theahead-of-time (AOT) compilation strategy and introducedAndroid RunTime (ART) to replace Dalvik VM startingfrom Android 5 and onward. Instead of interpreting code(or using JIT [25, 62]) by virtual machine at runtime, theAOT compilation strategy will directly compile apps intonative code at the first installation time. Therefore, onecannot use TaintDroid for the newly-designed runtime andTaintDroid can at most only support legacy systems up toAndroid 4.4.Secondly, although the latest Android still provides a fall-back runtime interpreter for debugging, the performance isnot acceptable (as shown in our evaluation). Therefore,porting TaintDroid to this fallback runtime cannot take ad-vantage of compiler optimization and the performance issuehinders effective security and data flow analysis.Thirdly, because of the compatibility and performance is-sues, users cannot use TaintDroid for policy enforcement toprevent privacy leakage. As shown in the Android distribu-tion statistics [24], about half of Android users have alreadyupgraded to Android 5.0 or above, and this number contin-ues to grow.Last but not least, we discover that app developers tendto target newer Android versions so as to use latest featureswhich TaintDroid does not support. We measured SDK ver-sions of five hundreds apps in Google Play’s“Top Charts”onOctober 2015 and February 2016. As shown in Figure 1, theaverage target SDK version has changed from 19 (Android4.4) to 20 (Android 5.0). Again, this implies that manynew apps may not be analyzed by TaintDroid, and malwarecan exploit this incapability to bypass security or data flowanalysis.

TaintART：适用于Android RunTime的实用多级信息流跟踪系统

像Android这样的移动操作系统未能对个人数据提供足够的保护，隐私泄露成为一个主要问题。为了理解安全风险和隐私泄露，分析师必须进行数据流分析。 2014年，Android在Android 5.0中升级了基本全新的Android RunTime（ART）环境设计。 ART采用提前编译策略，并取代以前的基于虚拟机器的Dalvik。不幸的是，很多数据流分析系统，如ToddDroid [19]都是为Dalvik传统环境设计的。这使得新应用和恶意软件的数据流分析变得不可行。我们为新的Android系统设计了一个名为Taint-ART的多级信息流跟踪系统。 TaintART采用多层次污点分析技术来减少污染标签的存储。因此，可以将Taint标签存储在处理器寄存器中以提供有效的传播操作。我们还定制了ART编译器，以最大限度地提高提前编译优化的性能。基于TaintART的一般设计，我们还实施了多层次的隐私保护措施，以防止敏感数据泄漏。我们证明TaintART仅占用CPU限制的微基准的不到15％的开销，并且可以忽略内置或第三方应用程序的开销。与Android 4.4中的legacyDalvik环境相比，TaintART在Java运行时基准测试中的性能提高了99.7％。

智能手机，平板电脑和可穿戴设备等移动设备广泛用于通信，拍照，娱乐和监控健康状况。智能手机上安装的许多应用程序（简称应用程序）提供有用的服务，但它们也可能私下将敏感信息发送到远程服务器进行各种数据分析[12]。更糟糕的是，其中一些应用程序可从这些个人数据中获利[ 38。所有这些都表明，隐私泄露对移动用户群体是一个严重的威胁。为了理解隐私泄露的可能性，研究人员寻求两个方向的数据流分析解决方案。首先，通过给定应用程序的反汇编代码，研究可以执行静态数据流分析技术，如静态污点分析和符号执行。这种方法可以静态获得一组可能的数据，这些数据可能会在运行时离开设备，并决定敏感数据是否泄漏到不受信任的通道。此方法的局限性在于，当应用程序开发人员使用Java反射代码，代码加密或动态代码加载技术等技术时，它无法检测运行时信息泄漏。因此，研究人员提出使用动态方法来监控可疑行为。动态污点分析技术[46]是可以在运行时跟踪应用程序内部信息流的多种动态方法之一。动态污点分析技术将标记（污染）来自某些源的敏感数据，并在运行时处理变量，文件和过程之间的标签转换（污染传播）。如果污染标签通过某些功能（接收器）传输出移动设备，则可以动态监控数据泄漏。该方法可以准确地跟踪anapp执行时的数据流.TaintDroid [19]是Android应用程序的一个着名的动态污点分析系统。它定制Android运行时（DalvikVirtual Machine）以实现污点存储和污点支持。许多系统[16,63,5,43,42,54]基于TaintDroid进行进一步分析。但是，TaintDroid在最新的Android上不能再用于隐私跟踪和恶意软件分析（以及某种程度上的数据流分析），有几个限制。首先，TaintDroid最初是为虚拟机系统设计的，Dalvik虚拟机），并在传统Android系统2.1,2.3,4.1和4.3上实现。 TaintDroid利用Dalvik虚拟机的内部存储器进行污染存储和传播。为了提升Android的性能，谷歌最近更新了AOT编译策略，并引入了Android RunTime（ART），以取代从Android 5开始的Dalvik VM。 AOT编译策略不是在运行时通过虚拟机解释代码（或使用JIT [25,62]），而是在第一次安装时直接编译应用程序代码。因此，不能将TaintDroid用于新设计的运行时，而且TaintDroid最多只能支持高达Android4.4的旧版系统。其次，尽管最新的Android仍然提供了用于调试的后备运行时解释器，但性能不可接受（如我们的评估所示）。因此，将TaintDroid移植到此回退运行时不能优化编译器优化和性能问题，从而影响有效的安全性和数据流分析。再次，由于兼容性和性能方面的原因，用户不能使用TaintDroid进行策略执行以防止隐私泄露。如Android的分布统计[24]所示，大约一半的Android用户已经升级到Android 5.0或更高版本，并且这个数字还在不断增长。最后但并非最不重要的，我们发现应用开发者倾向于将更新的Android版本以便使用TaintDroid不支持的最新功能。我们在2015年10月和2016年2月的Google Play的“热门图表”中测量了500个应用程序的SDK版本。如图1所示，平均目标SDK版本从19（Android4.4）更改为20（Android 5.0）。再次