研究内容简介：

1. 选题背景

近年来，手机已经成为人们生活中必不可少的工具之一，移动操作系统也备受人们的关注。安卓作为几大移动操作系统的佼佼者，在移动平台市场中占据不小的份额。据NET MARKETSHARE的数据显示，安卓在2014年7月首次超越IOS跃居市场份额榜首，直至2017年7月，安卓市场份额占比64.38%。然而由于安卓的开放性，安装在安卓系统上的诸多app在为用户带来便捷和多种服务的同时，也引入了新的问题 - 个人隐私的泄露。研究表明，乐观估计高达90%的手机应用会访问获取用户的隐私信息。这意味着不仅是恶意软件，合法的手机应用也会借助各种途径获取用户的隐私信息，比如位置信息等。因此，如何有效地检测及防御安卓应用对用户个人隐私信息的窃取，已成为日益关注的热点.

常用的检测隐私信息泄露的方法分为静态分析和动态分析两种，由于静态分析方法不能检测运行时信息，例如当app开发人员使用例如JAVA反射机制，代码加密或者动态加载等技术的代码时静态分析方法就不能很好地起到检测作用。然而由于Google对安卓系统进行了大规模的改动，以往的依赖于Dalvik虚拟机的动态污点分析已经不能满足现在的需求。

除运行时从DVM到ART的改进外，安卓的手机架构升级也同样带来了影响。随着2013年苹果公司推出首个64位架构的A7处理器，64位移动芯片的时代正式开启。ARM公司随后在2014年推出了64位架构的ARMv8-A。与之相应的是，2014年10月安卓在新推出的5.0版本中加入了对64位的支持，成为IOS 8之后第二个支持64位的移动操作系统，64位手机芯片成为越来越多的手机厂商的选择。

因此需要结合安卓系统的诸多特性对动态污点技术进行改进完善，以实现对安卓应用窃取用户隐私信息的检测及防御。

1. 研究内容

本次研究是针对日益严重的安卓应用窃取用户隐私信息的问题，拟采用动态污点分析技术，选定安卓 6.0.1系统版本，在AArch64的架构上，结合安卓系统新特性，以实现对安卓系统中app的动态监测和防御。

2.1 安卓系统新特性

2.1.1 arm与arm64

安卓系统支持arm/arm64/x86/x86\_64/mips/mips64等架构，相比于其他架构相比，安卓最早支持ARM，对ARM的支持也最为完善，且主要应用于手机市场；x86/x86\_64架构目前已经支持的比较完善，主要应用于上网本；mips/mips64目前正在移植完善中，主要应用于智能家电/上网本领域。

arm即ARMv7,是32位架构，支持32位指令集A32和16位指令集T16，含有15个32位的通用寄存器（r0-r14），以及32个64位浮点寄存器；相比于arm而言，arm64即ARMv8，是ARM公司推出的64位的全新架构，使用全新的A64指令集，同时兼容32位指令集AArch32，含有31个64位通用寄存器（x0-x31，只使用寄存器的低32位的寄存器命名为w0-w30），以及32个128位浮点寄存器。

其他arm和arm64在安卓中的不同之处在下表中体现：

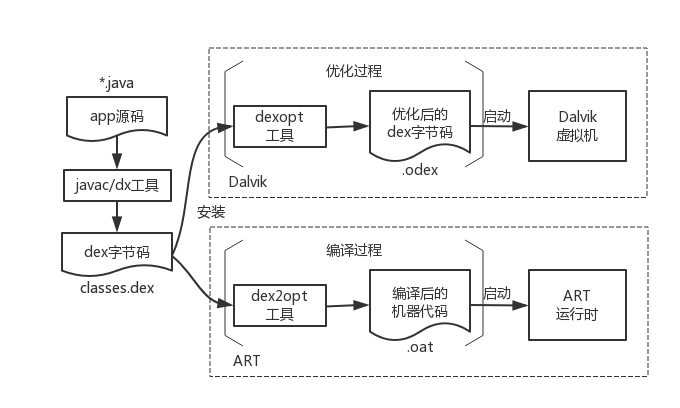
|  |  |  |
| --- | --- | --- |
|  | arm | arm64(包含AArch32和AArch64) |
| 通用寄存器数量 | 16(r0...r15) | 31(其中32位寄存器为w0...w30,64位寄存器为x0...x30) |
| 浮点寄存器数量 | 32(s0...s31) | 32(其中32位寄存器为s0...s30,64位寄存器为d0...d30,128位寄存器为q0...q30) |
| 程序计数器(pc) | 某些T16数据处理指令可以直接访问pc，或者某些跳转指令可以间接访问pc。在运行过程中，pc并不包含当前正在运行的指令的地址，当前指令地址为pc-8(A32)或者pc-4(T16)。  r15 | 不能直接通过寄存器名或者寄存器号来直接访问，pc中包含的是当前正在执行的指令的地址。 |
| 栈指针寄存器(sp) | r13 | AArch64中命名为：sp  AArch32中命名为：wsp  寄存器编号：63(in vixl)或者31(in compiler)  不属于通用寄存器。 |
| 函数参数(整型)使用的寄存器(调用规定) | r1...r3 | w/x1...w/x7 |
| 函数参数(浮点型)使用的寄存器(调用规定) | s0-s15 | s/d0...s/d7 |
| Callee saves | r5-r8,r10-r11,lr(r14) | x19...x30  d8...d15 |
| Temporary registers | r4...r11 | X9...x15,IP0(x16,vixl使用),IP1(x17，ART JNI Assembler使用)  d16...d31 |

2.1.2安卓运行时新特性

Google在安卓4.4时引入ART运行时作为一个备选项，在安卓5.0时使用ART彻底替代了Dalvik虚拟机，并且在随后推出的安卓6.0中将默认的编译器后端从Quick模式变为了Optimizing模式，并且支持64位处理器。

Dalvik虚拟机曾经是安卓系统的核心组成部分之一，自Android 2.2开始，Dalvik便一直采用JIT编译方法（just-in-time，即时编译技术）。在Dalvik环境下，应用每次运行的时候，字节码都需要通过即时编译器转换为机器码，这会拖慢应用的运行效率，而在ART环境中，应用在第一次安装的时候，字节码就会预先编译成机器码，使其成为真正的本地应用。这个过程叫做预编译（AOT,Ahead-Of-Time），这样的话，应用的启动（首次）和执行都会变得更加快速。但是由于以往应用在安卓上的动态污点分析技术大部分都依赖Dalvik虚拟机的解释器来进行污点传播和检测，因此当AOT替换JIT成为安卓默认的编译方式时，曾经的根据DVM进行改进整合的动态污点分析技术对于ART环境便不再适用了。

下图为DVM与ART的对比图，app的源代码经过javac或者dx转变为dex字节码即classes.dex文件，Dalvik和ART的输入均为dex文件，但是Dalvik环境中，在app安装时仅进行了dex字节码的优化，生成odex文件，在app运行时将odex文件加载到Dalvik虚拟机中进行解释运行。而在ART环境中，在app安装时通过dex2oat编译器将dex字节码转换为编译好的机器代码 – oat文件，在app运行时无需进行字节码到机器码的解释过程。OAT文件是一种安卓私有的ELF文件格式，它不仅包含从dex文件翻译而来的本地机器指令，还包含有原来的dex文件内容。

图1 DVM与ART对比图

2.2 动态污点技术

动态污点分析技术是一个常见的用来检测隐私信息泄露的动态分析技术。动态污点分析技术的几个关键要素是污点源（敏感函数中的敏感数据（用户隐私数据）），污点标签（追踪敏感数据而添加的标签），污点标签存储（对数据的污点标签状态的存储），污点槽（污点数据会从某些特定的函数离开系统，这些函数被称为taint sink）和污点逻辑（污点的传播和消除）。

2010年 Enck 等人在Android2.1操作系统上实现了动态污点跟踪系统TaintDroid，2011年一系列基于TaintDroid的系统也被提出，包括AppFence[4]和Mock Droid等。它们都对TaintDroid进行了不同程度的扩展，但仍有一些缺陷并没有得到解决：①能够检测的隐私数据类型不够全面，包括了IMEI、电话号码、地理位置信息、相册、录音，而通信录、短信息等重要信息并没有得到检测；②检测的隐私泄露点不够多，仅仅包含一般网络通信(socket)，而蓝牙、短消息等通信方式没有被监测；③字符串跟踪粒度不够细，可能导致污点跟踪过程中可能会产生污点的过度扩散而产生误报；④不能提供程序测试过程中所执行的路径信息；⑤测试时需要人工的参与，不利于自动化测试；⑥极度依赖Dalvik虚拟机进行动态污点分析。针对上述部分缺陷，2012年杨广亮等实现了TaintChaser系统，对隐私信息进行细粒度的跟踪，能产生被测试程序在测试过程中所执行的路径信息，并基于TaintChaser实现了自动化检测系统。

2016年，针对最新的ART环境，Michael Backes等提出了ARTist，实现了对ART运行时和dex2oat编译器的插桩，并且成功进行了动态污点分析，但是该系统无法对拥有多个dex文件的app进行处理，并且无法实现binder IPC和文件级别的污点追踪。Sun Mingshen等提出了TaintART，修改了后端的转换，优化和代码生成器部分，实现了动态污点分析，但是也存在只能手动触发app行为以进行实验，以及只在arm处理器上实现的不足之处。

在本文中，基于TaintART，我们提出了在arm64架构上实现的与安卓新特性相结合的动态污点分析系统，并且在对app进行实际检测时，可以有针对性的对app的某些行为进行触发。从而在相同的测试时间内可以提高检测的准确度和app行为的检测覆盖率。

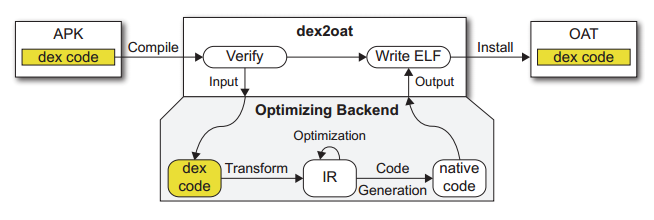
2.3 有针对性的触发app行为

3关键技术

本文使用的关键技术包括：与安卓系统新特性相结合的动态污点技术，

在安装app需要经过的编译过程中的代码生成阶段，arm64架构采用了vixl – ARMv8动态代码生成工具，主要用到了vixl中的宏汇编程序(MacroAssembler)来生成A64代码，而arm架构则同其他架构类似，使用的是安卓编译器内部自己实现的ArmAssembler来生成A32代码。

图2所示，为ART环境中的dex2oat编译器的工作流程图，dex2oat的输入仍旧是dex文件，它首先检验dex文件的有效性，然后dex2oat将整个的编译过程委托给了后端，它只负责将编译的结果和原始的dex代码写入oat文件。安卓系统中允许的后端有两种模式，一种是QUICK模式一种是Optimizing模式，但是在安卓6.0中，Google将安卓5.0中默认的Quick后端模式替换成了Optimizing模式。后端需要完成的工作分为三个步骤：①转换：将dex代码转换为中间表示形式IR；②优化：对中间形式IR进行代码优化，比如去除冗余指令等；③代码生成：使用代码生成器将IR转化成本地代码。为了实现动态污点分析，需要对ART的dex2oat进行改进，使其允许污点的传播和检测。



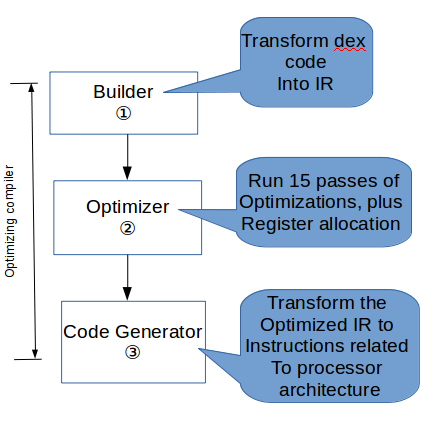


图3 optimizing compiler工作流程

图3中展示的optimizing compiler主要进行了以上三步工作。

4.论文计划

计划在2017年12月完成实验和

5.论文进度与目标

论文进展情况：

工作成果：

1. 已完成论文工作内容
2. 阶段性成果
3. 主要创新点

计划及进度安排：

问题及整改方案：

参考文献：

[1]NET MARKETSHARE.Mobile/Tablet Operating System Market Share.http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1. 2017

[2]Ge Y, Deng B, Sun Y, et al. A Comprehensive Investigation of User Privacy Leakage to 安卓 Applications[C]//Computer Communication and Networks (ICCCN), 2016 25th International Conference on. IEEE, 2016: 1-6.

[3]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.

[4]HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S., AND WETHERALL, D. These Aren’t the Droids You’re Looking For: Retrofitting Android to Protect Data from Imperious Applications. In Proceedings of ACM CCS (2011).

[5]Beresford A R, Rice A, Skehin N, et al. Mockdroid: trading privacy for application functionality on smartphones[C]//Proceedings of the 12th workshop on mobile computing systems and applications. ACM, 2011: 49-54.

[6]杨广亮, 龚晓锐, 姚刚, 等. 一个面向 Android 的隐私泄露检测系统[J]. 计算机工程, 2012, 38(23): 1-6.

[7]Backes M, Bugiel S, Schranz O, et al. ARTist: The Android Runtime Instrumentation and Security Toolkit[J]. arXiv preprint arXiv:1607.06619, 2016.

[8]Sun M, Wei T, Lui J. TaintART: A Practical Multi-level Information-Flow Tracking System for Android RunTime[C]//Proceedings of the 2016 ACM SIGSAC Conference  on Computer and Communications Security. ACM, 2016: 331-342.