软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(4):860−882 [doi: 10.13328/j.cnki.jos.005190]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

污点分析技术的原理和实践应用*

王 蕾[1,2], 李 丰[1], 李 炼[1], 冯晓兵[1]

[1](计算机系统结构国家重点实验室(中国科学院 计算技术研究所),北京 100190)

[2](中国科学院大学,北京 100190)

通讯作者: 王蕾, E-mail: wanglei2011@ict.ac.cn

摘 要: 信息流分析可以有效保证计算机系统中信息的保密性和完整性,污点分析作为其实践,被广泛用于软件系统的安全保障技术领域.对近些年来面向解决应用程序安全问题的污点分析技术进行综述:首先,总结了污点分析的基本原理以及在应用中的通用技术,即,使用动态和静态的方法解决污点传播;随后,分析该技术在移动终端、互联网平台上的应用过程中遇到的问题和解决方案,包括解决 Android 应用隐私泄露与检测 Web 系统安全漏洞的污点分析技术;最后,展望该技术的研究前景和发展趋势.

关键词: 污点分析;信息流分析;软件安全;静态分析与动态分析;Android;Web

中图法分类号: TP311

中文引用格式: 王蕾,李丰,李炼,冯晓兵.污点分析技术的原理和实践应用.软件学报,2017,28(4):860−882. http://www.jos.org.cn/1000-9825/5190.htm

英文引用格式: Wang L, Li F, Li L, Feng XB. Principle and practice of taint analysis. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):860−882 (in Chinese). http://www.jos.org.cn/1000-9825/5190.htm

Principle and Practice of Taint Analysis

WANG Lei[1,2], LI Feng[1], LI Lian[1], FENG Xiao-Bing[1]

[1](State Key Laboratory of Computer Architecture (Institute of Computing Technology, The Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Information flow analysis is a promising approach for protecting the confidentiality and integrity of information manipulated by computing systems. Taint analysis, as in practice, is widely used in the area of software security assurance. This survey summarizes the latest advances on taint analysis, especially the solutions applied in different platform applications. Firstly, the basic principle of taint analysis is introduced along with the general technology of taint propagation implemented by dynamic and static analyses. Then, the proposals applied in different platform frameworks, including techniques for protecting privacy leakage on Android and finding security vulnerabilities on Web, are analyzed. Lastly, further research directions and future work are discussed.

Key words: taint analysis; information flow analysis; software security; static and dynamic analyses; Android; Web

随着互联网+、云计算、移动智能终端等技术的发展,软件系统对信息安全的需求越来越高.软件系统信息安全的两个重要特性是信息的保密性(confidentiality)和完整性(integrity)[1].保密性是指系统的敏感数据在使用过程中不会被泄露到外界,完整性是指系统重要数据在使用过程中不会被恶意地篡改或删除.随着 Web 2.0 以及智能手机的普及,在线支付、社交网络等互联网活动中涉及到大量的用户隐私数据.同时,大量第三方应用程

* 基金项目: 国家自然科学基金(61303053, 61402303)
Foundation item: National Natural Science Foundation of China (61303053, 61402303)
收稿时间: 2016-06-18; 修改时间: 2016-09-08; 采用时间: 2016-11-26; jos 在线出版时间: 2017-01-24
CNKI 网络优先出版: 2017-02-20 13:51:07, http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1351.007.html

序的使用导致隐私数据难以被有效地保护.据调查分析[2],隐私数据泄露问题普遍存在.例如,印度的一家公司设计了一组智能手机应用开发工具包SilverPush,它可以嵌入到一个正常的手机应用中并在后台运行,在用户不知情的情况下收集用户的隐私数据(包括IMEI ID、位置信息、视频与音频信息、Web浏览记录等)并将其发送给广告推荐商[3].不良的软件设计也会导致软件系统漏洞的出现,攻击者可以利用这些漏洞窃取或篡改用户数据[2].例如,2011年,黑客利用Citigroup网站中存在的不安全对象引用漏洞,对200 000张信用卡的账户信息和交易记录进行盗取[4].类似的安全问题还在不断地增加,它们破坏了软件系统中信息的保密性和完整性.如果这些问题不被有效地抑制或者解决,将对个人或者组织产生极大的不便甚至财产损失.

保护软件系统数据的保密性和完整性机制有很多[5−8],例如访问控制、防火墙、加密机制、杀毒软件等.这些机制虽然能够保证数据发布的安全性,但是无法阻止数据在通过这些机制的检查之后又被进一步传播使用[9].譬如:访问控制通过赋予文件读写权限的方法保证只有授权用户才能对文件数据进行读取或修改,但是它无法保证授权用户读取数据之后不会将其传播给恶意程序或未授权用户使用;防火墙使用的隔离技术虽然能够限制计算机系统软件只能与受信任的网络进行通信,但是它无法保证信任网络不会被恶意攻击者利用,导致信息在交换过程中被危险操作窃取或篡改.可见,保护软件系统的数据安全,除了要确保数据发布的安全以外,还必须进一步分析数据在系统中是如何传播、使用的,以确保隐私数据既不会外泄也不会被外界操作篡改.

信息流分析技术(information-flow analysis)通过分析程序中数据传播的合法性以保证信息安全,是防止数据完整性和保密性被破坏的有效手段.污点分析技术(taint analysis,又被称作信息流跟踪技术)[10]是信息流分析技术的一种实践方法,该技术通过对系统中敏感数据进行标记,继而跟踪标记数据在程序中的传播,以检测系统安全问题.图1所示是一段Android应用程序代码,运行该段程序会导致用户的密码数据通过发送短信的方式泄露.污点分析可以有效地检测该问题.污点分析首先要识别引入敏感数据的接口(source,污点源)并进行污点标记,具体到图1所示的程序,即识别到第4行中引入密码数据的passwordText接口为污点源,并对pwd变量进行污点标记.如果被标记的变量又通过程序依赖关系传播给了其他变量,那么根据相关传播规则继续标记对应的变量.比如,图1中pwd变量的污点标记按照箭头进行传播,所以第5行的leakedPwd变量和第6行的leakedMessage变量都将会被标记.当被标记的变量到达信息泄露的位置(sink,污点汇聚点)时,则根据对应的安全策略进行检测.图1中第8行带污点标记的leakedMessage变量可以传播到发送信息的sendTextMessage接口,这就意味着密码数据会被该接口泄露,污点分析将报告此泄露问题.与之相反,在第9行,密码数据通过加密(调用Encrypt方法)转化后赋给sanitizedPwd变量,继续传播sanitizedPwd变量并不会产生泄露问题.也就是说,如果污点变量经过一个使数据不再携带隐私信息的接口处理(sanitizer,无害处理),那么就可以移除该污点数据的污点标记.

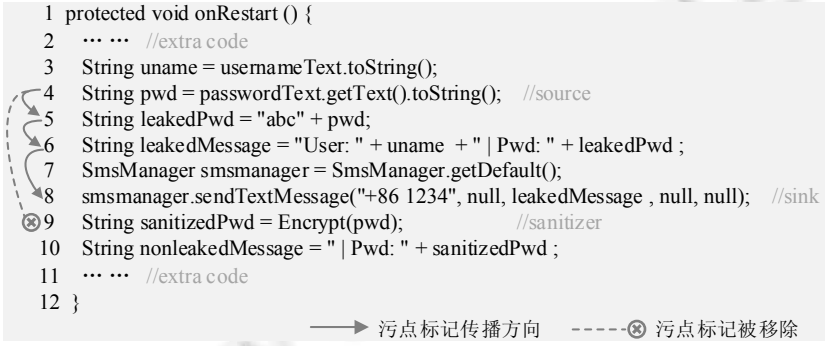


Fig.1 An example of taint analysis

图1 污点分析应用示例

近些年来,随着程序分析技术的发展及其在污点分析中的应用,污点分析可以针对实际应用程序(例如智能手机应用、Web应用等)中的信息安全问题实施更加精确、高效的分析检测[11−35].本文针对近年使用污点分析

技术解决实际应用程序中的完整性和保密性问题的工作进行分析综述,总结该技术的基本原理以及污点传播分析(污点分析中的一个重要步骤)的通用技术,并且针对不同平台上的应用程序(Android 应用程序和 Web 应用程序),分析该技术在实施过程中遇到的问题及解决方案.

本文第 1 节介绍污点分析技术的理论基础以及它在解决应用程序安全方面的意义.第 2 节介绍污点分析的基本原理.第 3 节介绍污点分析的通用技术,重点介绍如何使用动/静态程序分析技术进行污点传播分析.第 4 节介绍污点分析作用在不同平台上应用程序的信息安全检测过程中遇到的问题及解决方案,具体包括智能手机的隐私泄露检测和 Web 安全漏洞发掘两大类应用.第 5 节为总结与展望.

1 理论基础

污点分析是信息流分析的一种实践技术.在过去的 40 年里,信息流分析技术一直是信息安全领域的一个重要研究点.大量研究工作[36−48]尝试通过制定信息流策略(information-flow policies)来提供信息流安全保障:如果系统满足了用户定制的信息流策略,那么系统是信息流安全的.信息流分析就是一种分析信息流策略是否被有效实施的技术.

目前的信息流策略主要建立在安全类型模型上.1976 年,Denning 在文献[36]中提出为程序中的变量赋予安全类别(secure class),例如根据数据的保密级别将变量分成高级别(high)和低级别(low),并基于此提出了一套基于格(lattice)的理论模型.在这个模型中,一个格被定义为一个二元组$\langle SC,\sqsubseteq\rangle$.其中,$SC$ 是一个安全类别的集合;$\sqsubseteq$ 是一个建立在 SC 上的偏序关系(具有自反性、传递性、反对称性),这个偏序关系代表带有安全类型的变量之间是否可以进行赋值传递.例如:在 SC:{*Public*,*Secret*}集合上,$Public\sqsubseteq Secret$ 的偏序关系表示具有公共级别(*Public*)的变量可以赋值给具有保密级别(*Secret*)的变量.在此基础上,Goguen 和 Meseguer 在文献[38]中提出了一种适用于系统级别的信息流策略——无干扰性(non-interference).无干扰性的含义是指,如果一个系统满足无干扰性,当且仅当改变该系统输入中任意具有高级别标签的变量值都不会影响到系统输出中具有低级别标签的变量值.

后续的工作大多在上述理论基础上展开,其中一种解决方案尝试设计并实现满足无干扰性的安全类型系统(secure type system),并在安全类型系统的基础上构建或扩展具有安全类型的语言,例如 Jflow[39].基于安全类型语言的信息流策略为信息流分析技术在理论上进行了有效的探索,但在实践过程中,此类技术需要针对编程语言设计相关的类型扩展.例如,使用形如 int{*secret*} *passwd*;的语法为原语言的 int 类型变量 *passwd* 扩展一个代表信息安全级别的标签.随着移动终端、互联网络、云计算等技术的发展,当前,应用程序不再局限于传统的个人电脑应用.能够提供更为抽象、简洁的编程环境的新的编程框架和运行模式的频繁出现,导致基于安全类型语言的信息流分析的适应能力下降.例如,针对基于 Android 系统的应用程序进行信息流分析时,为新设计的类型系统而改变 Android 开发环境是不现实的.

污点分析作为信息流分析的一种实践技术,通过对带污点标记的数据的传播实施分析来达到保护数据完整性和保密性的目的.污点分析的理论模型是一个建立在 SC:{*Tainted*;*Untainted*}集合上的格:如果信息从 *Tainted* 类型的变量传播给 *Untainted* 类型的变量,那么需要 *Untainted* 类型的数据改成 *Tainted* 类型;如果 *Tainted* 类型的变量传递到重要数据区域或者信息泄露点,那就意味着信息流策略被违反.在解决应用程序安全问题的实践中,污点分析需要程序分析技术的支持.譬如:静态污点分析技术可以通过分析源码或字节码中语句或指令之间的静态依赖关系来判断污点标记所有可能的传播途径;动态污点分析技术可以借助程序插桩,结合定制的硬件来跟踪污点标记的传播.可见,污点分析既不必改变应用程序原有的编程模型或语言特性,又可以提供精确的数据流传播跟踪.

当前,污点分析被广泛地应用在系统隐私数据泄露检测、系统安全漏洞发掘等实际领域.然而,由于系统框架、编程模型、语言特性等方面的差异,污点分析应用在不同平台的不同应用程序上时可能面临不同的研究问题,需要根据上述问题对污点分析技术进行定制.接下来,我们将首先介绍污点分析的基本原理和污点传播分析

的通用技术,然后介绍污点分析在不同应用平台上实施时遇到的具体问题和解决方法.

2 污点分析的基本原理

2.1 污点分析定义

污点分析可以抽象成一个三元组⟨*sources*,*sinks*,*sanitizers*⟩的形式,其中,*source* 即污点源,代表直接引入不受信任的数据或者机密数据到系统中;*sink* 即污点汇聚点,代表直接产生安全敏感操作(违反数据完整性)或者泄露隐私数据到外界(违反数据保密性);*sanitizer* 即无害处理,代表通过数据加密或者移除危害操作等手段使数据传播不再对软件系统的信息安全产生危害.污点分析就是分析程序中由污点源引入的数据是否能够不经无害处理,而直接传播到污点汇聚点.如果不能,说明系统是信息流安全的;否则,说明系统产生了隐私数据泄露或危险数据操作等安全问题.

污点分析的处理过程可以分成 3 个阶段(如图 2 所示):(1) 识别污点源和汇聚点;(2) 污点传播分析;(3) 无害处理.

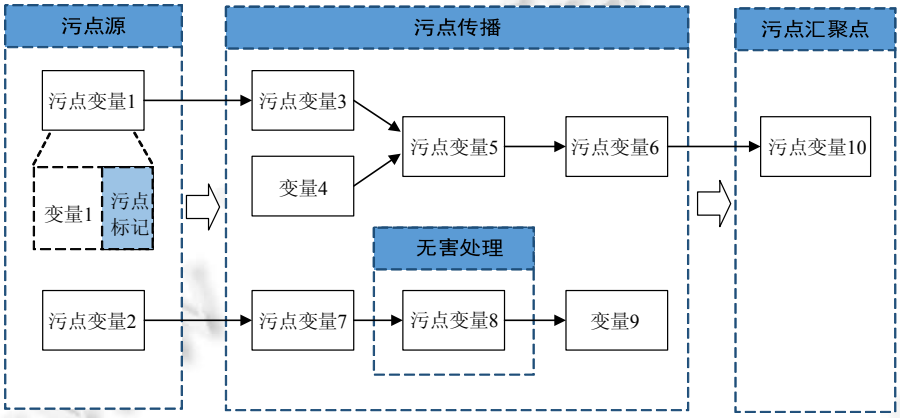


Fig.2 An intuitive example about the process of taint analysis

图 2 污点分析过程直观示例

2.2 识别污点源和汇聚点

识别污点源和污点汇聚点是污点分析的前提.目前,在不同的应用程序中识别污点源和汇聚点的方法各不相同.缺乏通用方法的原因一方面来自系统模型、编程语言之间的差异.另一方面,污点分析关注的安全漏洞类型不同,也会导致对污点源和污点汇聚点的收集方法迥异.表 1 所示为在 Web 应用程序漏洞检测中的污点源示例[29],它们是 Web 框架中关键对象的属性.

Table 1 An example of source proposed by Ref.[29]

表 1 文献[29]提出的污点源示例

对象	污点属性
Document	cookie, domain, forms, lastModified, links, referrer, title, URL
Form	Action
Any form input element	checked, defaultChecked, defaultValue, name, selectedIndex, toString, value
History	current, next, previous, toString
Select option	defaultSelected, selected, text, value
Location and Link	hash, host, hostname, href, pathname, port, protocol, search, toString
Window	defaultStatus, status

现有的识别污点源和汇聚点的方法可以大致分成 3 类:一类使用启发式的策略进行标记,例如把来自程序外部输入的数据统称为“污点”数据,保守地认为这些数据有可能包含恶意的攻击数据(如 PHP Aspis[26]);第 2 类

工具会根据具体应用程序调用的 API 或者重要的数据类型,手工标记源和汇聚点(如 DroidSafe[12]);第 3 类工具[11]使用统计或机器学习技术自动地识别和标记污点源及汇聚点.

2.3 污点传播分析

污点传播分析就是分析污点标记数据在程序中的传播途径.按照分析过程中关注的程序依赖关系的不同,可以将污点传播分析分为显式流分析和隐式流分析.

污点传播分析中的显式流分析就是分析污点标记如何随程序中变量之间的数据依赖关系传播.以图 3 所示的程序为例,变量 *a* 和 *b* 被预定义的污点源函数 *source* 标记为污点源.假设 *a* 和 *b* 被赋予的污点标记分别为 *taint_a* 和 *taint_b*.由于第 5 行的变量 *x* 直接数据依赖于变量 *a*,第 6 行的变量 *y* 直接数据依赖于变量 *b*,显式流分析会分别将污点标记 *taint_a* 和 *taint_b* 传播给第 5 行的变量 *x* 和第 6 行的变量 *y*.又由于 *x* 和 *y* 分别可以到达第 7 行和第 8 行的污点汇聚点(用预定义的污点汇聚点函数 *sink* 标识),图 3 所示的代码存在信息泄漏的问题.我们将在第 3.1 节具体介绍目前污点传播分析中显式流分析面临的主要挑战和解决方法.

污点传播分析中的隐式流分析是分析污点标记如何随程序中变量之间的控制依赖关系传播,也就是分析污点标记如何从条件指令传播到其所控制的语句.在图 4 所示的程序中,变量 *X* 是被污点标记的字符串类型变量,变量 *Y* 和变量 *X* 之间并没有直接或间接的数据依赖关系(显式流关系),但 *X* 上的污点标记可以经过控制依赖隐式地传播到 *Y*.具体来说,由第 4 行的循环条件控制的外层循环顺序地取出 *X* 中的每一个字符,转化成整型后赋给变量 *x*,再由第 7 行的循环条件控制的内层循环以累加的方式将 *x* 的值赋给 *y*,最后由外层循环将 *y* 逐一传给 *Y*.最终,第 12 行的 *Y* 值和 *X* 值相同,程序存在信息泄漏问题.但是,如果不进行隐式流污点传播分析,第 12 行的变量 *Y* 将不会被赋予污点标记,程序的信息泄漏问题被掩盖.隐式流污点传播一直以来都是一个重要的问题[49],和显式流一样,如果不被正确处理,会使污点分析的结果不精确.由于对隐式流污点传播处理不当导致本应被标记的变量没有被标记的问题称为欠污染(under-taint)问题.相反地,由于污点标记的数量过多而导致污点变量大量扩散的问题称为过污染(over-taint)问题.目前,针对隐式流问题的研究重点是尽量减少欠污染和过污染的情况.我们将在第 3.2 节具体介绍现有技术是如何解决上述问题的.

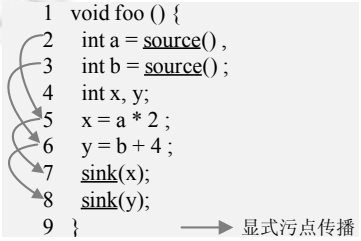


Fig.3 Example code for explicit taint analysis

图 3 显式污点分析示例代码

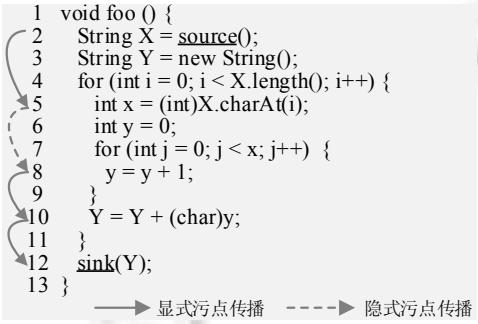


Fig.4 Example code for implicit taint analysis

图 4 隐式流污点分析示例代码

2.4 无害处理

污点数据在传播的过程中可能会经过无害处理模块,无害处理模块是指污点数据经过该模块的处理后,数据本身不再携带敏感信息或者针对该数据的操作不会再对系统产生危害.换言之,带污点标记的数据在经过无害处理模块后,污点标记可以被移除.正确地使用无害处理可以降低系统中污点标记的数量,提高污点分析的效率,并且避免由于污点扩散导致的分析结果不精确的问题.

在应用过程中,为了防止敏感数据被泄露(保护保密性),通常会对敏感数据进行加密处理.此时,加密库函数应该被识别成无害处理模块.这一方面是由于库函数中使用了大量的加密算法,导致攻击者很难有效地计算出密码的可能范围;另一方面是加密后的数据不再具有威胁性,继续传播污点标记没有意义.

此外,为了防止外界数据因为携带危险操作而对系统关键区域产生危害(保护完整性),通常会对输入的数据进行验证.此时,输入验证(input validation)模块应当被识别成无害处理模块.例如,为了防止代码注入漏洞,PHP 提供的 htmlentities 函数可以将特殊含义的 HTML 字符串转化成 HTML 实体(例如,将'<'转化成'<').输入字符串经过上述转化后不会再携带可能产生危害的代码,可以安全地发送给用户使用.除了语言自带的输入验证函数外,一些系统还提供了额外的输入验证工具,比如 ScriptGard[50],CSAS[51],XSS Auditor[52],Bek[53].这些工具也应被识别成无害处理模块.

综上,目前对污点源、污点汇聚点以及无害处理模块的识别通常根据系统或漏洞类型使用定制的方法.由于这些方法都比较直接,本文将不再进行更深入的探讨.下一节将重点介绍污点传播中的关键技术.

3 污点传播分析的关键技术

污点传播分析是当前污点分析领域的研究重点.如第 1 节所述,与程序分析技术相结合,可以获得更加高效、精确的污点分析结果.根据分析过程中是否需要运行程序,可以将污点传播分析分为静态污点分析和动态污点分析.本节主要介绍如何使用动/静态程序分析技术来解决污点传播中的显式流分析和隐式流分析问题.

3.1 污点传播中的显式流分析

3.1.1 静态分析技术

静态污点传播分析(简称静态污点分析)是指在不运行且不修改代码的前提下,通过分析程序变量间的数据依赖关系来检测数据能否从污点源传播到污点汇聚点.静态污点分析的对象一般是程序的源码或中间表示.可以将对污点传播中显式流的静态分析问题转化为对程序中静态数据依赖的分析:首先,根据程序中的函数调用关系构建调用图(call graph,简称 CG);然后,在函数内或者函数间根据不同的程序特性进行具体的数据流传播分析.常见的显式流污点传播方式包括直接赋值传播、通过函数(过程)调用传播以及通过别名(指针)传播.

以图 5 所示的 Java 程序为例:第 3 行的变量 *b* 为初始的污点标记变量,程序第 4 行将一个包含变量 *b* 的算术表达式的计算结果直接赋给变量 *c*.由于变量 *c* 和变量 *b* 之间具有直接的赋值关系,污点标记可直接从赋值语句右部的变量传播到左部,也就是上述 3 种显式流污点传播方式中的直接赋值传播.接下来,变量 *c* 被作为实参传递给程序第 5 行的函数 *foo*,*c* 上的污点标记也通过函数调用传播到 *foo* 的形参 *z*,*z* 的污点标记又通过直接赋值传播到程序第 8 行的 *x.f*.由于 *foo* 的另外两个参数对象 *x* 和 *y* 都是对对象 *a* 的引用,二者之间存在别名,因此,*x.f* 的污点标记可以通过别名传播到第 9 行的污点汇聚点,程序存在泄漏问题.

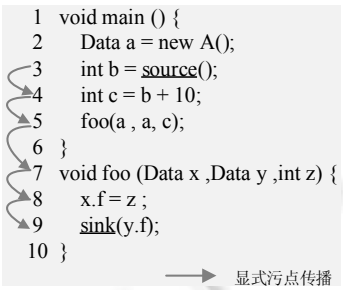


Fig.5 Example code for static taint propagation

图 5 静态污点传播示例代码

目前,利用数据流分析解决显式污点传播分析中的直接赋值传播和函数调用传播已经相当成熟[54−59],研究的重点是如何为别名传播的分析提供更精确、高效的解决方案.由于精确度越高(上下文敏感、流敏感、域敏感、对象敏感等)的程序静态分析技术往往伴随着越大的时空开销,追求全敏感且高效的别名分析难度较大[60].又由于静态污点传播分析关注的是从污点源到污点汇聚点之间的数据流关系,分析对象并非完整的程序,而是确定的入口和出口之间的程序片段.这就意味着可以尝试采用按需(on-demand)定制的别名分析方法来解决显

式流静态污点分析中的别名传播问题.文献[10]使用按需的上下文敏感的别名分析的污点分析方法来检测 Java 应用程序漏洞.TAJ[25]工具使用了混合切片结合对象敏感的别名分析来进行 Java Web 应用上的污点分析.Andromeda[61]工具使用了按需的对象敏感别名分析技术解决对象的访问路径(access path)问题,FlowDroid[13]工具提出一种按需的别名分析,从而提供上下文敏感、流敏感、域敏感、对象敏感的污点分析,用以解决 Android 的隐私泄露问题.

3.1.2 动态分析技术

动态污点传播分析(简称动态污点分析)是指在程序运行过程中,通过实时监控程序的污点数据在系统程序中的传播来检测数据能否从污点源传播到污点汇聚点.动态污点传播分析首先需要为污点数据扩展一个污点标记(tainted tag)的标签并将其存储在存储单元(内存、寄存器、缓存等)中,然后根据指令类型和指令操作数设计相应的传播逻辑传播污点标记.动态污点传播分析按照实现层次被分为基于硬件、基于软件以及混合型的污点传播分析这 3 类.基于硬件的污点传播分析需要定制的硬件支持,一般需要在原有体系结构上为寄存器或者内存扩展一个标记位,用来存储污点标记,代表的系统有 Minos[62],Raksha[63]等.基于软件的污点传播分析通过修改程序的二进制代码来进行污点标记位的存储与传播,代表的系统有 TaintEraser[64],TaintDroid[19]等.基于软件的污点传播的优点在于不必更改处理器等底层的硬件,并且可以支持更高的语义逻辑的安全策略(利用其更贴近源程序层次的特点),但缺点是使用插桩(instrumentation)或代码重写(code rewriting)修改程序往往会给分析系统带来巨大的开销.相反地,基于硬件的污点传播分析虽然可以利用定制硬件降低开销,但通常不能支持更高的语义逻辑的安全策略,并且需要对处理器结构进行重新设计.混合型的污点分析是对上述两类方法的折中,即,通过尽可能少的硬件结构改动以保证更高的语义逻辑的安全策略,代表的系统有 Flexitaint[65],PIFT[66]等.

目前,针对动态污点传播分析的研究工作关注的首要问题是如何设计有效的污点传播逻辑,以确保精确的污点传播分析.TaintCheck[67]利用插桩工具 Valgrind[68]对其中间表示 Ucode 插桩并提供移动指令、算术指令以及除移动和算术外其他指令的 3 类传播逻辑实现对 x86 程序的动态污点分析.Privacy Scope[69],Dytan[70]和 Libdft[71]以插桩工具 Pin[72]为基础,实现针对 x86 程序的动态污点分析,并解决了一系列 x86 指令污点传播逻辑的问题.TaintDroid[19]提供了一套基于 Android Dalvik 虚拟机的 DEX 格式[73]的污点传播分析方法.由于 DEX 的指令包含多数常用的指令和具有面向对象特性的指令,普适性高,这里以 TaintDroid 中污点传播方法为示例,介绍动态污点传播的逻辑.

DEX 支持的变量类型有 5 种:本地变量、方法参数、类静态域、类实例域和数组.TaintDroid 用 υ_X 代表本地变量和方法参数,f_X 代表类的静态域,$\upsilon_Y(f_X)$代表实例域,其中,υ_Y 是具体实例的变量引用.$\upsilon_X[\cdot]$代表数组,其中,υ_X 表示数组的对象引用.同时,TaintDroid 使用虚拟污点映射函数 $\tau(\cdot)$来辅助污点传播,对于变量 $\upsilon,\tau(\upsilon)$返回变量的污点标记 $t.\tau(\upsilon)$可以被赋值给其他的变量.符号←代表将位于符号右部的变量的污点标记传播给左部的变量.具体的污点传播逻辑规则见表 2.

- 对常数、移动(赋值)、一元算术逻辑指令的传播逻辑是直接将指令的右值的污点标记传递给指令的左值;
- 对于多元算术逻辑指令,需要将指令的右值的污点标记进行合并之后传播给指令的左值;
- 对于返回指令和异常处理指令,分别将变量标记传递给与返回、异常处理相关的变量;
- 对于数组指令,除了对数组变量的标记进行传播外,还需要将数组索引变量的标记合并传播.例如,对于数组赋值 $b=Z[a]$,索引变量 a 的污点标记也需要传播给 b 变量;
- 对于域操作相关指令,同样需要将对象的域变量污点标记以及域所属对象变量的标记进行合并传播.

动态污点传播分析的另一个研究重点是如何降低分析代价.如前所述,传统的基于硬件的动态污点传播分析技术需要定制硬件的支持,而基于软件的技术由于程序插桩或代码重写会带来额外的性能开销.为控制分析代价,一类研究工作采用的思路是有选择地对系统中的指令进行污点传播分析.例如,LIFT[74]提出的快速路径(fast-path)优化技术通过提前判断一个模块的输入和输出是否是具有威胁的(如果没有威胁,则无需进行污点传播)以降低需要重写的代码的数量;合并检查(merged check)优化技术将多个基本块合并成 1 个进行检查,以降低

检查次数;快速切换(fast switch)优化则利用活性分析消除一些之后不活跃的条件寄存器的 save 和 restore 操作,以减少需要分析的指令数量.PIFT[66]系统提出了基于预测的污点跟踪(传播)方式,他们设计统计实验观察到 CPU 指令流中 load 和 store 指令存在一些特殊性质(load 指令与其子序列中 store 指令距离接近、load 指令之后的 store 指令个数不多、连续的 load 指令之间的距离是均匀的).基于此,提出了基于预测的只跟踪 load 和 store 指令的策略,即:如果 load 指令的操作数是污点数据,那么将其一定距离内的 store 指令的目的地址标记成污点数据.该方法减少了跟踪其他复杂 CPU 指令的开销.另外一类降低分析代价的思路是,使用低开销的机制代替高开销机制.例如,SHIFT[75]将动态污点分析转化成延迟例外(deferred exceptions)处理的问题.延迟例外是指在例外发生后,将例外标记在相关的指令中,之后再通过检测指令检测出被标记指令中的例外并处理例外,这种机制与污点标记传播的机制类似.SHIFT 将污点传播分析实现在支持投机执行(speculative execution)的处理器中,既不需要改变计算机处理器本身,又可以充分利用该处理器高速处理延迟例外的优势,从而达到降低动态污点分析开销的效果.又比如,LIFT 的快速切换(fast switch)优化使用低开销的 lahf/sahf 指令代替高开销的 pushq/popq 指令,以提高插桩代码与原始二进制文件之间的切换效率.

Table 2 Dynamic taint propagation logic proposed by Ref.[19]

表 2 文献[19]提出动态污点传播逻辑

指令格式	指令语义	污点传播规则	解释
const-op υ_A C	$\upsilon_A \leftarrow C$	$\tau(\upsilon_A) \leftarrow \varnothing$	将υ_A的标记清空
move-op υ_A υ_B	$\upsilon_A \leftarrow \upsilon_B$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B)$	将υ_B的标记赋值给υ_A
move-op R υ_A	$\upsilon_A \leftarrow R$	$\tau(\upsilon_A) \leftarrow \tau(R)$	将返回值的标记赋值给υ_A
return-op υ_A	$R \leftarrow \upsilon_A$	$\tau(R) \leftarrow \tau(\upsilon_A)$	设置返回值标记
move-op-E υ_A	$\upsilon_A \leftarrow E$	$\tau(\upsilon_A) \leftarrow \tau(E)$	将异常处理变量的标记赋值给υ_A
throw-op υ_A	$E \leftarrow \upsilon_A$	$\tau(E) \leftarrow \tau(\upsilon_A)$	设置异常处理变量标记
unary-op υ_A υ_B	$\upsilon_A \leftarrow \otimes\upsilon_B$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B)$	将υ_B的标记赋值给υ_A
binary-op υ_A υ_B υ_C	$\upsilon_A \leftarrow \upsilon_B \otimes \upsilon_C$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B) \cup \tau(\upsilon_C)$	将$\upsilon_B \cup \upsilon_C$的标记赋值给$\upsilon_A$
binary-op υ_A υ_B	$\upsilon_A \leftarrow \upsilon_A \otimes \upsilon_B$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_A) \cup \tau(\upsilon_B)$	将$\upsilon_A \cup \upsilon_B$的标记赋值给$\upsilon_A$
binary-op υ_A υ_B C	$\upsilon_A \leftarrow \upsilon_B \otimes C$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B)$	将υ_B的标记赋值给υ_A
aput-op υ_A υ_B υ_C	$\upsilon_B[\upsilon_C] \leftarrow \upsilon_A$	$\tau(\upsilon_B[\cdot]) \leftarrow \tau(\upsilon_B[\cdot]) \cup \tau(\upsilon_A)$	将υ_A的标记赋值给数组υ_B
aget-op υ_A υ_B υ_C	$\upsilon_A \leftarrow \upsilon_B[\upsilon_C]$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B[\cdot]) \cup \tau(\upsilon_C)$	将数组υ_B和索引标记赋值给υ_A
sput-op υ_A f_B	$f_B \leftarrow \upsilon_A$	$\tau(f_B) \leftarrow \tau(\upsilon_A)$	将υ_A的标记赋值给域f_B
sget-op υ_A f_B	$\upsilon_A \leftarrow f_B$	$\tau(\upsilon_A) \leftarrow \tau(f_B)$	将域f_B的标记赋值给υ_A
iput-op υ_A υ_B f_C	$\upsilon_B(f_C) \leftarrow \upsilon_A$	$\tau(\upsilon_B(f_C)) \leftarrow \tau(\upsilon_A)$	将υ_A的标记赋值给域f_C
iget-op υ_A υ_B f_C	$\upsilon_A \leftarrow \upsilon_B(f_C)$	$\tau(\upsilon_A) \leftarrow \tau(\upsilon_B(f_C)) \cup \tau(\upsilon_B)$	将(域$f_C \cup$对象υ_B)的标记赋值给υ_A

3.2 污点传播中的隐式流分析

如第 2.3 节所述,污点传播分析中的隐式流分析就是分析污点数据如何通过控制依赖进行传播,如果忽略了对隐式流污点传播的分析,则会导致欠污染的情况;如果对隐式流分析不当,那么除了欠污染之外,还可能出现过污染的情况.与显式流分析类似,隐式流分析技术同样也可以分为静态分析和动态分析两类.

静态隐式流分析面临的核心问题是精度与效率不可兼得的问题.精确的隐式流污点传播分析需要分析每一个分支控制条件是否需要传播污点标记.路径敏感的数据流分析往往会产生路径爆炸问题,导致开销难以接受.为了降低开销,一种简单的静态传播(标记)分支语句的污点标记方法是将控制依赖于它的语句全部进行污点标记,但该方法会导致一些并不携带隐私数据的变量被标记,导致过污染情况的发生.过污染会引起污点的大量扩散,最终导致用户得到的报告中信息过多,难以使用.

动态隐式流分析关注的首要问题是如何确定污点控制条件下需要标记的语句的范围.由于动态执行轨迹并不能反映出被执行的指令之间的控制依赖关系,目前的研究多采用离线的静态分析辅助判断动态污点传播中的隐式流标记范围.Clause 等人[70]提出,利用离线静态分析得到的控制流图节点间的后支配(post-dominate)关系来解决动态污点传播中的隐式流标记问题.例如,如图 6(a)所示,程序第 3 行的分支语句被标记为污点源,当 document.cookie 的值为 abc 时,会发生污点数据泄露.根据基于后支配关系的标记算法,会对该示例第 4 行语句的指令目的地,即 x 的值进行污点标记.

动态分析面临的第2个问题是由于部分泄漏(partially leaked)导致的漏报.部分泄漏是指污点信息通过动态未执行部分进行传播并泄漏.Vogt 等人[29]发现,只动态地标记分支条件下的语句会发生这种情况.仍以图 6(a)中的程序为例:当第3行的控制条件被执行时,对应的 *x* 会被标记.此时,*x* 的值为 true,而 *y* 值没有变化,仍然为 false.在后续执行过程中,由于第9行的污点汇聚点不可达,而第12行的汇聚点可达,动态分析没有检测到污点数据泄漏.但攻击者由第11行 *y* 等于 false 的条件能够反推出程序执行了第3行的分支条件,程序实际上存在信息泄漏的问题.这个信息泄露是由第 6 行未被执行到的 *y* 的赋值语句所触发的.因此,*y* 应该被动态污点传播分析所标记.为了解决部分泄漏问题,Vogt 等人在传统的动态污点分析基础上增加了离线的静态分析,以跟踪动态执行过程中的控制依赖关系,对污点分支控制范围内的所有赋值语句中的变量都进行标记.具体到图 6(a)所示的例子,就是第4行和第6行中的变量均会被污点标记.但是,Vogt 等人的方法仍然会产生过污染的情况.

动态分析需要解决的第 3 个问题是如何选择合适的污点标记分支进行污点传播.鉴于单纯地将所有包含污点标记的分支进行传播会导致过污染的情况,可以根据信息泄漏范围的不同,定量地设计污点标记分支的选择策略.以图 6(b)所示的程序为例,第 2 行的变量 *a* 为初始的污点标记变量.第 5 行、第 7 行、第 9 行均为以 *a* 作为源操作数的污点标记的分支.如果传播策略为只要分支指令中包含污点标记就对其进行传播,那么第 5 行、第 7 行、第 9 行将分别被传播给第 6 行、第 8 行、第 10 行,并最终传播到第 12 行的污点汇聚点.如果对这段程序进行深入分析会发现,3 个分支条件所提供的信息值(所能泄露的信息范围)并不相同,分别是 *a* 等于 10、*a* 大于 10 且小于或等于 13(将 *w* 值代入计算)以及 *a* 小于 10.对于 *a* 等于 10 的情况,攻击者可以根据第 12 行泄漏的 *x* 的值直接还原出污点源处 *a* 的值(这类分支也被称为能够保存完整信息的分支);对于 *a* 大于 10 且小于或等于 13 的情况,攻击者也只需要尝试 3 次就可以还原信息;而对于 *a* 小于 10 的情况,攻击者所获得的不确定性较大,成功还原信息的几率显著低于前两种,对该分支进行污点传播的实际意义不大.Bao 等人[76]只将严格控制依赖(strict control dependence)识别成需要污点传播的分支,其中,严格控制依赖即分支条件表达式的两端具有常数差异的分支.但是,Bao 的方法只适用于能够在编译阶段计算出常数差异的分支.Kang 等人[77]提出的 DTA++ 工具使用基于离线执行踪迹(trace)的符号执行的方法来寻找进行污点传播的分支,但该方法只关注信息被完整保存的分支,即图 6(b)中第 5 行的 *a*==10 会被选择污点传播,但是信息仍然能够通过另一个范围(第 7 行的分支)而泄露.Cox 等人[78]提出的 SpanDex 的主要思想是:动态地获得控制分支中污点数据的范围,根据数据的更改以及数据间的依赖关系构建一个基于操作的有向无环图(OP-DAG),再结合一个在线的约束求解器(CSP solver)确定隐式流中传播的隐私数据值的范围,通过预先设定的阈值,选择是否对数据进行污点传播.该方法会对图 6(b)所示例子中的第 5 行、第 7 行的分支进行污点传播.但是目前,该方法只能求解密码字符的范围,暂不支持对复杂操作(位、除法、数组等操作)的求解.

```
1  x = false;
2  y = false;
3  if (document.cookie == "abc") {   //source
4     x = true;
5  } else {
6     y = true;
7  }
8  if (x == false) {
9     sink(x);
10 }
11 if (y == false) {
12    sink(y);
13 }
```

(a)

```
1  void foo () {
2     int a = source();
3     int x, y, z;
4     int w = a + 10;
5     if (a ==10) {
6        x = 1;
7     } else if (a > 10 & w <=23) {
8        y = 2;
9     } else if (a < 10) {
10       z = 3;
11    }
12    sink(x ,y ,z);
13 }
```

(b)

Fig.6 Example code for implicit taint analysis

图 6 隐式流污点分析示例代码

3.3 小 结

本节介绍了污点传播分析通用技术方法,主要包括利用静态或者动态的方法解决显式流和隐式流的污点传播.表 3 概述了目前通用污点传播分析技术面临的主要问题和代表性技术.

Table 3 Research points and solutions in the basic principle of taint propagation

表 3 通用污点传播分析中的关键技术概览

静态污点分析		
研究难点	代表工作	特点
精确、高效的别名分析	Livshits 等人[10]	上下文敏感的别名分析
	TAJ[25]	混合切片,对象敏感的别名分析
	Andromeda[61]	对象敏感别名分析(解决访问路径问题)
	FlowDroid [13]	上下文敏感、流敏感、域敏感、对象敏感的分析
动态污点分析		
研究难点	代表工作	特点
传播逻辑	TaintCheck[67]/Dytan[70] Libdft[71]/Privacy Scope[69]	解决 x86 指令传播逻辑问题
	TaintDroid[19]	解决 Android 的 DEX 指令的传播逻辑
效率优化	LIFT[74]	快速路径、合并检查、快速切换
	PIFT[66]	基于预测且只跟踪 load 和 store 指令
	SHIFT[75]	利用处理器延迟例外机制
欠污染	Clause 等人[70]	静态分析得到后支配关系确定标记范围
	Vogt 等人[29]	解决部分泄漏问题
过污染	Bao 等人[76]	只适用于能够在编译阶段计算出常数差异的分支
	DTA++[77]	基于离线踪迹进行符号执行,能够识别完整信息的分支
	SpanDex[78]	在线约束求解,提供更精确的污点传播范围

其中,静态的污点分析的对象是程序的源码或中间表示,可以离线地分析程序中存在的安全问题.针对静态污点分析技术的研究重点是提高静态别名分析的精度和效率.鉴于各种敏感度高的分析策略的出现,目前的研究倾向正逐渐转化为如何按需提供满足敏感度需要的高效分析策略.与静态污点分析技术相比,动态污点分析的优势在于,可以通过确定执行得到精确的程序运行时信息,但是由于动态污点分析的一次动态运行只能执行单一的程序路径,如果多次执行后仍然存在没有执行到的路径,那么这些路径上代码的安全问题就会被掩盖.因此,动态污点分析往往与测试技术(如测试用例生成技术、覆盖率评估技术等)结合使用以提高路径的覆盖率.目前,针对动态污点分析技术的研究重点包括传播逻辑的设计、分析效率的优化等.

隐式流分析是动/静态污点分析共同面临的一个重要问题,目前的方法还不能完全避免状态爆炸(静态分析)、部分泄漏以及如何合理选择污点传播分支(动态分析)的问题,需要研究者们进一步探索通用的解决方案.

4 污点分析在实际应用中的关键技术

污点分析被广泛地应用在系统隐私数据泄露、安全漏洞等问题的检测中.在实际应用过程中,由于系统框架、语言特性等方面的差异,通用的污点分析技术往往难以适用.比如:系统框架的高度模块化以及各模块之间复杂的调用关系导致污点源到汇聚点的传播路径变得复杂、庞大,采用通用的污点分析技术可能面临开销难以接受的问题;通用的污点分析技术对新的语言特性支持有限等.为此,需要针对不同的应用场景,对通用的污点分析技术进行扩展或定制.

本节以两个代表性的应用场景——智能手机的隐私泄漏检测和 Web 应用安全漏洞检测为切入点,总结近 10 年来污点分析技术在上述领域的应用实践过程中所面临的问题和关键解决技术.

4.1 检测智能手机隐私泄露

尽管智能手机的供应商提供了权限控制、沙箱等安全策略来保证手机系统内部的安全性[79],但智能手机隐私泄露问题仍然普遍存在[19,80,81].造成这一现象的原因是用户难以控制敏感数据不会被第三方应用程序所泄露.例如,Hornyack 等人[23]收集了 1 100 个 Android 应用程序,发现其中有 605 个程序(占被分析程序的 55%)至少

将其所申请的资源数据中的一项发送给互联网络.污点分析技术正被广泛地应用于检测此类问题:在手机 APP 发布到应用市场之前,检测人员可以对 APP 的字节码文件进行静态污点分析,或者在 APP 测试过程中进行动态污点分析.

由于目前 Android 智能手机几乎占全球智能手机 80%的市场份额[82],本节将重点介绍污点分析在 Android 系统上的隐私泄露检测关键技术.整个 Android 的框架包括 4 层[83]:最上层是应用层,包括了使用 Java 开发的各种类型的应用程序;第 2 层是应用程序框架层,提供开发人员访问应用程序的 API;第 3 层是系统运行库,其中包括用 C/C++编写的本地库和为 Android 平台定制的虚拟机 Dalvik;最底层是 Linux 内核,提供基本的操作系统支持.为了更好地实现资源的独立管理,Android 采用基于组件(component)的编程模型:将应用程序划分成以组件为基本单元执行.应用程序组件类型包括活动(activity)、服务(service)、广播接收器(broadcast receiver)和内容提供器(content provider)这 4 种.组件间的信息传递由组件间通信(inter-component communication,简称 ICC)提供.组件间的调用初始化和连接通过 android.content.Intent 对象实现.同时,Android 还支持利用 JNI 机制调用本地代码库.

针对 Android 的污点传播分析也围绕组件展开,按照传播可能通过的模块的不同,分为组件内污点传播、组件间污点传播、组件与库函数之间的污点传播这 3 类(如图 7 所示).接下来将分别介绍针对这 3 类传播问题的静态和动态污点传播分析技术.

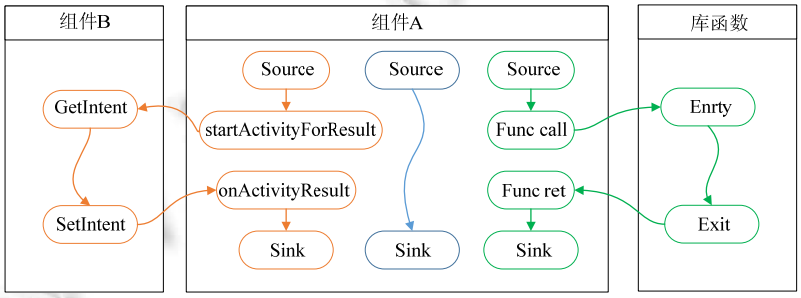


Fig.7 Classification of taint porpagation within Android

图 7 Android 上的污点传播分类

4.1.1 静态污点分析

(1) 组件内污点传播分析

组件内部污点分析面临的主要问题是如何构建完整的分析模型.不同于传统的 C/C++程序(有唯一的 Main 函数入口),Android 应用程序存在有多个入口函数的情况.这个情况源于 Android 应用程序复杂的运行生命周期(例如 onCreate,onStart,onResume,onPause 等)以及程序中大量存在的回调函数和异步函数调用.由于任何的程序入口都有可能是隐私数据的来源,在静态的污点分析开始之前必须构建完整的应用程序模型,以确保程序中每一种可能的执行路径都会被静态污点传播分析覆盖到.

LeakMiner[14]和 CHEX[15]尝试使用增量的方法构建系统调用图.Arzt 等人设计的 FlowDroid[13]提出了一种更系统的构建 Android 程序完整分析模型的方法:首先,通过 XML 配置文件提取与 Android 生命周期相关的入口函数,将这些方法作为节点,并根据 Android 生命周期构建调用图(如图 8 所示);其次,对于生命周期内的回调函数,在该调用图的基础上增加不透明谓词节点(即图 8 中菱形的 P 节点);然后,增量式地将回调函数加入这个函数调用图;最后,将调用图上所有的执行入口连接到一个虚假的 Main 函数上.FlowDroid 中的一次合法的执行,就是对调用图进行的一次遍历.Gordon 等人[12]提出的 DroidSafe 使用 Android 设备实现(Android device implementation)来构建 Android 的完整分析模型.Android 设备实现是对 Android 运行环境的一个简单模拟,它使用 Java 语言,结合 Android Open Source Porject(AOSP),实现了与原 Android 接口语义等价的模型,并使用精确分析存根(accurate analysis stub)将 AOSP 代码之外的函数加入到模型中.

(2) 组件间污点传播分析

即使正确分析了组件内的数据流关系,污点数据仍然可能通过组件间的数据流来传递,从而造成信息泄露.如图 7 左侧所示,即使保证了对组件 *A* 内部污点传播的精确分析,组件 *A* 仍然可能通过调用方法 startActivityforResult()将信息传递给组件 *B*,再通过组件 *B* 产生泄露.因此,针对 Android 应用的污点分析还需要分析出组件间所有可能的数据流信息.组件间通信是通过组件发送 Intent 消息对象完成的.Intent 按照参数字段是否包含目标组件名称分为显式 Intent 和隐式 Intent.如图 9 所示:显式 Intent 对象使用一个包含目标组件名称的参数显式地指定通信的下一个组件;隐式 Intent 使用 action,category 等域隐式地让 Android 系统通过 Intent Filter 自动选择一个组件调用.目前,解决该问题的主要思想是利用 Intent 参数信息分析组件间的数据流.

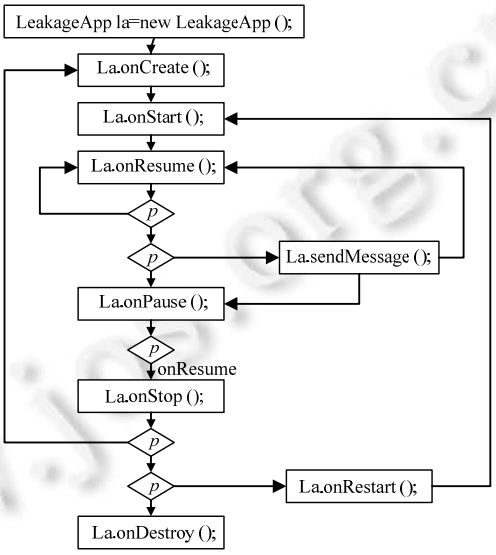


Fig.8 An example of CFG used in FlowDroid[13]

图 8 FlowDroid[13]使用控制流图示例

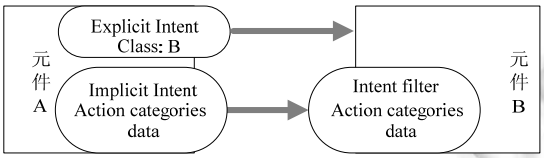


Fig.9 Explicit and implicit Intent ICC

图 9 显式和隐式 Intent 组件间通信

解决组件间数据流的前提是解析 Intent 的目的地,解析 Intent 目的地包括解析显式 Intent 的目的地和隐式 Intent 的目的地.由于显式 Intent 的目的地可以直接通过初始化 Intent 的地址字符串参数获得,目前,解析显式 Intent 目的地的常用方法是使用字符串分析工具(例如 JSA[84])提取 Intent 中字符串参数的信息.解析隐式 Intent 目的地的主要方法是分析配置文件信息与 Intent Filter 注册器之间的映射关系,建立发送 Intent 组件和接受 Intent 组件之间的配对关系.在解析出 Intent 目的地之后,问题的重点转移到如何提高组件间数据流分析的精度上.Klieber 等人[17]尝试在已经建立好的组件内污点分析的基础上,结合推导规则来分析组件间数据流.在分析之前,需要收集组件内部的污点源和汇聚点以及组件内 Intent 的发送目的地标签等信息.表 4 和表 5 给出了推导规则的前提定义和具体的推导规则,其中,一次完整的分析是指根据已知组件内部的信息 $src \rightarrow sink$ 以及推导规则识别所有 $src' \rightarrow sink'$ 的流集合.Octeau 等人[18]尝试使用现有的程序分析方法提高组件间数据流分析的精度,

他们将组件间数据流分析问题转化成 IDE(interprocedural distributive environment)问题[58]进行求解.DroidSafe 设计了一种对象敏感的别名分析技术,在此基础上提供的精度优化方法包括:提取 Intent 的目的地的字符串参数、将 Intent 目的地的初始化函数嵌入到目的组件当中以提高别名分析的精度,同时,增加处理 Android Service 的支持.

Table 4 Prerequisite definition of inference rules proposed by Ref.[17]

表 4 文献[17]提出的推导规则前提定义

定义	解释
Src	污点源
Sink	污点汇聚点
S	污点源和汇聚点的总集合
C_{id}/*null*	标号为 *id* 的组件,使用 *null* 表示不能确定的组件
$I(C_1;C_2;id)$	从组件 C_1 发送到组件 C_2 带有源码位置 *id* 的 Intent
$I_{TX}/I(C_{TX},null,id)$	从组件 C_{TX} 发送的 Intent
$I_{RX}/I(null,C_{RX},null)$	从组件 C_{RX} 接收的 Intent
$R(I_1)$	Intent I_1 通过 setResult 方法的返回值

Table 5 Inference rules of taint analysis proposed by Ref.[17]

表 5 文献[17]提出的污点分析推导规则

编号	规则
1	如果 *src* 是一个组件内污点源,那么 $src'=src$
2	如果 *sink* 是一个组件内汇聚点,那么 $sink'=sink$
3	如果 *src* 具有 $I(null,C_{RX},null)$的形式,对于组件 C_{RX} 匹配所有 $I(C_{TX},null,id)\in S$,那么 src' 是 $I(C_{TX},C_{RX},id)$
4	如果 *sink* 具有 $I(C_{TX},null,id)$的形式,如果存在组件 C_{RX} 的 Intent filter 匹配 *sink*,那么 $sink'$ 是 $I(C_{TX},C_{RX},id)$
5	如果 *src* 具有 $R(I(null,C_{RX},null))$的形式,那么如果存在组件 C_{RX} 的 Intent filter 匹配 $I(C_{TX},null,id)\in S$,或者存在一个 Intent 的 $R(I(null,C_{RX},null))\in S$ 时,src' 是 $R(I(C_{TX},C_{RX},id))$
6	如果 *sink* 具有 $R(I(null,C_{RX},null))$的形式,那么如果存在组件 C_{RX} 的 Intent filter 匹配 $I(C_{TX},null,id)\in S$,或者存在一个 Intent 的 $R(I(C_{TX},null,null))\in S$ 时,$sink'$ 是 $R(I(C_{TX},C_{RX},id))$
7	如果 *src* 具有 $I(null,C_{RX},null)$的形式且 *sink* 具有 $R(I(null,C_{RX},null)$的形式,那么 $sink'$ 必须是 $R(src')$

(3) 组件与库函数之间的污点传播分析

组件与库函数之间的污点传播分析面临的主要问题包括对 Android 库函数自身庞大的代码量的分析以及组件和某些库函数使用的实现语言不同(Android 组件通常用 Java 实现,而本地库则采用 C/C++代码编写)这两方面.

目前的一类方法是使用手动定制来解决上述问题.比如,FlowDroid[13]尝试手动地分析库函数的语义,根据参数与返回值之间的关系为其提供显式传播规则.对于没有制定规则的库函数,FlowDroid 使用保守的策略,即:只要库函数的参数中包含污点数据参数,就对函数的返回值进行污点标记.DroidSafe[12]使用精确分析存根实现了 3 176 个本地代码库,这些存根是使用 Java 代码手动实现的,且与原本地代码的语义等价.对本地代码的调用被传递到对应的存根代码上进行分析.此类工作需要在人工理解程序语义的基础上加以实现.

与之相对的是自动推导的方法.StubDroid[85]首次提出了用自动推导解决库函数传播的问题,该方法将产生供分析的摘要文件(summary file),将推导的污点数据流存储到摘要文件中,并提供了相应的部署策略.StubDroid 的分析单元是一个 API 方法,将该方法内部的所有访问路径(包括方法参数、方法 this 引用和所有静态可见域)标记成源,将方法的返回值作为汇聚点.基于此,问题被转化成方法内部的污点分析问题.StubDroid 利用现有的静态污点分析工具 FlowDroid 自动地处理库内部的污点数据流,最终得到的摘要文件包含 API 方法中的访问路径源是否能够传播到返回值的信息.另外,StubDroid 还能够解决由于别名和访问路径带导致库函数传播中的精确度下降的问题,最后的摘要文件进一步被应用到 FlowDroid 框架,完成对整个 app 的分析.与 FlowDroid 类似,StudDroid 同样无法对本地库函数进行自动分析.

4.1.2 动态污点分析

Android 系统中的动态污点同样需要分析组件内污点传播、组件间污点传播以及组件代码与本地库之间

的污点传播.动态污点分析面临的主要挑战是系统信息除了在系统内部通过 DEX 指令传播以外,还会经过其他的通道,如本地库、网络、文件等.

TaintDroid[19]首先提出了面向 Android 平台上的动态污点分析工具,之后的工具大多是基于它的优化或者应用扩展(Appsplaygroud[20],VetDroid[21],BayesDroid[22],AppFence[23]).本节重点介绍 TaintDoid 上的 3 种污点传播处理.组件内的污点传播主要是在 Dalvik 虚拟机 DEX 指令的变量级别传播,详见第 3.1.2 节中的介绍.组件间的污点传播利用了 Binder 机制.Android 底层使用 Binder 机制完成 IPC 调用,数据被存储在包(parcels)对象结构中.TaintDroid 的污点传播方法是对包对象的结构进行重新设计,使之附带与污点相关的信息.TaintDroid 提供了两种包结构扩展方法:(a) 使用一个标签变量将污点信息存储到包中,然后通过网络进行传输.当接受者接收到这个结构时,将其中包含的标签变量提取出来并继续传输;(b) 在方法(a)的基础上进行改进,提出了基于污点标签向量的传播技术.组件与本地库间的污点传播包括污点数据通过本地库代码或文件进行传播.TaintDroid 通过设计后置条件对本地代码的函数进行污点传播.后置条件为:(a) 所有本地代码访问的外部变量都会被标记上污点标签;(b) 根据预定义规则将被赋值的函数返回值也标记上污点标签.TaintDroid 使用了人工插桩与启发式相结合的方法来提供这些后置条件.TaintDroid 解决污点数据通过文件进行传播的方法是以文件为单位添加污点标签,当文件被写入或者被读出到缓冲区时进行污点传播.如果一个被标记的数据被写入一个文件,就需要这个文件进行标记.该文件在后续的操作中将作为污点源,如果有对该文件的读取操作,那么读出的数据也需要被标记.

4.2 检测Web应用程序安全漏洞

目前,Web 应用程序中存在大量的安全漏洞,如跨站脚本攻击[24]、SQL 注入[26,27]等.污点分析可以有效地检测这些安全漏洞.基于 HTTP 协议的 Web 应用框架在总体上分成两大部分[86]:服务器端应用程序和客户端应用程序.用户浏览 Web 网页时,浏览器通过 HTTP 协议与 Web 服务器交换信息.浏览器端上的应用程序被称为客户端应用程序.客户端应用程序使用 HTML 结合脚本语言(比如 JavaScript)处理和交换数据,再将数据显示在网页上.Web 服务器端应用程序接收客户端脚本语言的请求(request),根据请求执行对应的逻辑计算或者数据库查询操作,然后返回一个响应(response)给客户端.如此往复地进行信息交换,完成网页浏览.

Web 应用程序中的安全漏洞可能发生在客户端的脚本程序中,也可能发生在服务器端的应用程序中,还可能是由服务器端和客户端程序合作触发的.无论是客户端的应用程序还是服务器端的应用程序,一旦安全漏洞被攻击者利用,都可能会给用户带来财产损失.目前的 Web 框架有数百种之多[87],本节选取一个服务器端 Web 应用框架(Java EE)和一个客户端 Web 框架(JavaScript)为代表进行论述.

4.2.1 Java Web 框架上的污点分析技术

Java EE(Java platform,enterprise edition)框架是一种典型的服务器端 Web 应用程序框架,目前被广泛应用于企业级 Web 的构建.Servlets 和 Java Server Pages(JSP)是 Java EE 中两个最重要的部分.Servlet 类主要提供逻辑处理功能:通过处理接口与配置文件交互来处理客户端请求;JSP 通过在传统的 HTML 页面中插入 Java 程序段和 JSP 标记的方式提供页面显示.

在 Java EE 的基础上又扩展了很多其他框架,这些框架可以提供更高级别的 Web 开发抽象,比如基于 MVC 模型的 Struts 框架.Struts 使用控制器(controller)处理主要业务逻辑,使用视图(view)渲染响应页面,使用模型(model)来存储数据模型.如图 10 所示是 Struts 处理 HTTP 请求的流程:当控制器 ActionServlet 收到一个 HTTP 请求时,它会解析得到其 URL 并根据配置文件决定处理该请求的 Action 子类.同时,添加一个 ActionForm 对象用以存储请求的数据.Struts 会自动完成对 ActionForm 对象的填充并启动 Action 子类的对象,再通过 execute 方法读入 ActionForm 对象,以进一步处理相关业务逻辑.处理后的 Action 子类会返回一个 ActionForward 类,Struts 会根据配置文件得出下一个跳转页面并转发给 JSP,由 JSP 将其渲染后发送到客户端进行显示.

如图 10 中虚线箭头所示为 Java EE 上的污点传播的分析过程:一般而言,Java EE 框架的污点来源于客户端的浏览器输入,但是 Java EE 框架不能直接分析浏览器中的客户端代码.考虑到浏览器中的内容直接来源于 JSP,污点分析首先需要获得 JSP 上相关的污点源信息;其次,Web 页面间的跳转关系决定了污点的传播逻辑,其

中,ActionServlet 通过 Java 的反射机制与配置文件交互决定由哪个 Action 类处理对应的页面跳转;最后,污点的汇聚点存在于具体的 Action 类的重要数据区域.上述分析过程表明:针对 Java EE 框架的污点传播分析仅仅实现在 Java 层是不够的,需要考虑到 Java 代码、配置文件、JSP 代码三者之间的交互处理问题.

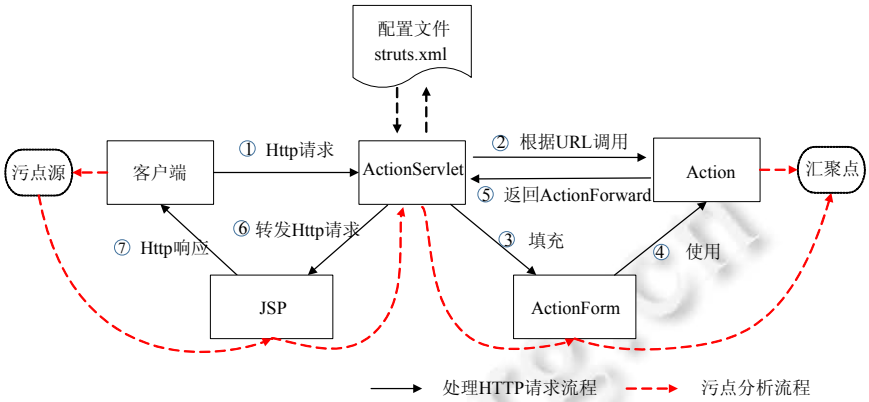


Fig.10 Request processing and taint analysis flow of Java EE (Struts)

图 10 Java EE(Struts)处理请求和污点分析流程

为了将这三者的代码集中分析,Møller 等人[30]将它们对应到 Java 代码中,提出了基于跟踪 Web 程序状态的污点分析方法.该方法包括 3 个步骤:识别客户端状态、识别共享程序状态和污点传播分析,其中,识别客户端状态就是识别 JSP 中能够直接引用服务器端的内部对象(比如数据库记录)的参数.具体方法是,首先分析配置文件中页面跳转关系信息,形成一张能够反映页面之间访问关系的图.图中的节点代表页面,边代表页面之间可能的跳转关系以及跳转过程中触发的 Action 和 From.对于一个 JSP 页面 p,识别引用参数的方法是将图中指向页面 p 的边上的 Action 类的参数和页面 p 中的文档参数(一般是隐藏字段、选择框、链接等)进行匹配,将能够被匹配的参数识别为客户端状态,也就是污点源.识别共享应用程序状态就是识别 Java 代码中重要的数据区域接口,也就是污点汇聚点.共享应用程序状态包括两个部分:(a) 内部应用程序状态,包括 HTTPServlet 对象、ServletContext 对象以及所有的静态域;(b) 外部应用程序状态,即,存储在文件和数据库中的状态代码.污点传播分析以识别出的客户端状态为污点源,以共享应用程序状态为污点汇聚点,判断客户端程序状态值是否会被写入到内部程序状态对象或者能否调用外部程序状态.

Sridharan 等人[24]提出的 F4F 使用了一种语言规范格式来解决不同语言代码间的交互问题.主要思想是:使用 Web 应用框架语言(Web application framework language,简称 WAFL)统一整个 Java EE 框架上的不同过程的代码,最后在 WAFL 的基础上进行污点分析.如图 11 所示是 WAFL 的文法规范,包括 3 个部分:全局变量(global)、合成方法(synthetic methods)和调用替换(call replacement).

location	l	::=	v | e.f
assignable	a	::=	l | **argToOrigCall**(-1)
expression	e	::=	l | a := e′ | f(e1,e2,...)
			| **taint** | **nondet**(e1,e2,...)
			| **argToOrigCall**(i)
global declaration	g	::=	**global**(**request**|**session**) v
			[**properties** v1,v2,...]
synthetic method	m	::=	**fun** [**entrypoint**] f(v1,v2,...)
			{(**var** v)*(e)*}
call replacement	r	::=	**replaceCall** callSiteId e
specification	p	::=	(g|m|r)*

Fig.11 Grammar of WAFL proposed by Ref.[24]

图 11 文献[24]提出 WAFL 语言文法

全局变量用来表示那些在不同模块之间传递的变量;合成方法对框架中不同模块的代码以及配置文件进行合成,将不同的代码放在一个方法中进行分析;调用替换表示一个调用点需要替换成一个具体的代码进行分析.F4F 还提供了一种有效的文法生成方法,生成的文法将作为污点分析的输入代码.

4.2.2 解决 JavaScript 上的污点分析

Web 脚本直接运行在客户端浏览器中,具有简单易学、开发速度快、可移植性强、便于集成成熟技术等优势.目前最流行的 Web 脚本语言是 JavaScript,它是一个灵活的动态脚本语言,提供面向对象特性、DOM 模型调用、与网络库动态交互信息等特性,可以提供更高的用户体验.例如:在使用 Gmail 时,需要对用户名和密码进行验证,这个验证过程是静态 HTML 无法完成的,但可以利用 JavaScript 实现.

脚本语言同样面临严重的安全性问题.在使用 JavaScript 的过程中,重要数据(如定位信息、RSS 等)可能会通过第三方应用脚本进行处理.恶意的第三方应用可能导致恶意的软件行为,比如窃取用户的隐私数据、破坏用户的重要数据或者将页面重定位到具有恶意行为的页面等.污点分析技术可以被用于检测上述问题,但是由于 JavaScript 语言中包含的大量有别于传统的 C/C++/Java 语言的新特性[88],需要对传统污点分析技术进行扩展.

JavaScript 上的污点分析首先需要解决 JavaScript 的特殊语法特性带来的问题,目前的方法多使用代码重写的方式解决该问题.文献[31]提出了解决 JavaScript 中函数、域、原型(prototypes)等特性带来的污点传播问题;ACTARUS[33]在抽象语法树级别进行代码重写,主要解决由于原型系统、对象创建(object creations)、反射属性访问(reflective property accesses)、词法作用域(lexical scoping)等语言特性导致的难以建立完整的调用图的问题.

JavaScript 语言大量地使用动态特性来提高用户的体验,例如在运行时动态地加载第三方库、使用 eval 方法动态地产生执行代码等.由于动态特性代码只有在运行时才能获得具体信息,传统的静态污点分析无法精确地分析出其中可能存在的安全问题.针对 JavaScript 语言的动态特性问题,Chugh 等人[31]提出了分阶段的污点分析方法.分阶段污点分析的主要思想是尽可能地在当前已知的代码上进行静态污点分析并提取定制的检查规则,然后在动态加载过程中进行规则检查.静态污点分析阶段使用一种基于约束求解的方法分析两个集合:必须不能写入集合(must not write set)和必须不能读取集合(must not read set).制定的检查规则是动态代码中的变量不能流入或流出到上述集合中.在动态阶段实施快速的规则检查,如果代码满足了规则检查,那么整个系统就被认为是安全的.Wei 等人[32]提出了利用混合的污点分析技术来解决 JavaScript 漏洞检测中的动态语言特性问题.混合分析的主要思想是利用动态分析生成执行踪迹,在执行踪迹的基础上利用静态方法进行污点分析.在动态分析阶段中,利用已有的测试集合动态执行并收集执行踪迹.每个网页的执行踪迹包括执行过的函数调用、创建对象的类型和静态不可见的动态加载执行代码,选择程序行为覆盖度较高的执行踪迹页面的子集.在静态污点分析阶段,通过分析执行踪迹子集建立调用图,并在调用图上实施静态的污点分析.

基于 DOM 的 XSS 安全问题[89]使得 JavaScript 的污点分析还需要考虑 JavaScript 与文档对象模型(DOM)的交互问题.Vogt 等人[29]提出了一种简单的解决方案,它的实现思想是:如果一个 DOM 节点被污点标记,则需要为当前节点存储一个污点数据;如果当前节点在此之后被访问,那么它的返回值也会被污点标记.Lekies 等人[34]提出了一种更为系统的解决方案,他们尝试通过修改浏览器源码来支持对基于 DOM 的 XSS 安全问题的污点分析,也称为动态的字符串级别的污点分析方法.该方法将 14 个污点源(例如 location.href,location.hash,document.referrer 等)压缩成单字节,并将其标记成污点字符串.污点标记的传播规则是基于字符串实现的.该方法改变了 JavaScript 引擎 V8 的底层字符串类型实现,扩展其字符串创建和分配内存方法.同时,改变支持 DOM 的 WebKit 库中字符串类的实现,增加一个数组存储成员变量的污点标记.

4.3 小 结

本节主要介绍了污点分析技术在实际应用中的关键技术,具体针对 Android 框架和 Web 框架(Java EE 和 JavaScript)上的关键技术进行分析.表 6 和表 7 给出了相关的技术总结以及各技术之间的对比,从表中可以看出,目前研究工作的首要任务是解决模型完整性的问题.产生这类问题的原因来源于目前框架本身的特性超出了

传统的线性代码特性的范围.例如:Android 框架中存在如 Java 反射调用、事件驱动、异步调用、RPC 调用、多线程等机制;Java EE 或 Struts 中大量的 Java 反射调用与配置文件进行交互;JavaScript 中的异步调用、特殊语言特性等问题.本节分别给出了不同框架中解决这些模型完整性问题的方案.从实现的自动化程度进行分析,初期的研究工作尽量尝试采用手工的方式进行模型处理(如手工添加相关回调函数、手工指定 Web 应用的入口函数等),此类工作需要大量的人工操作并且可扩展性较差.进一步的研究尝试提供半自动化的方式(例如提供规则匹配方案)以及实现更自动化的方法或工具.其次,目前的研究工作也尝试使用传统的优化方法提高模型分析的精度.例如,Octeau 等人[18]尝试将组件间的污点传播问题转化成 IDE 问题进行求解.最后,一些研究工作尝试使用混合的方法(动/静结合)解决静态无法获得待分析信息等问题(解决 JavaScript 动态特性问题)或者解决静态分析结果不精确等问题(Appaudit[90]尝试使用近似执行进一步提高检查精度).

Table 6 Research points and solutions in the practice of taint analysis in Android application

表 6 污点分析在 Android 应用中的关键技术概览

研究难点	代表工作		特点
静态	组件内传播分析	LeakMiner[14]/CHEX[15] FlowDroid[13] DroidSafe[12]	增量添加回调函数 增加处理回调函数虚拟接入点 实现 Android 语义等价模型
	组件间传播分析	Klieber 等人[17]	利用定制的推导规则匹配
		Octeau 等人[18]/DroidSafe[12]	转化数据流分析问题(IDE、别名分析)提高精度
	组件与库函数传播分析	FlowDroid[13]/DroidSafe[12] StubDroid[85]	手动分析、实现 利用 FlowDroid 自动化推导
动态	多级别传播分析	TaintDroid[19]	解决多种级别的传播策略
		Appsplaygroud[20], VetDroid[21], BayesDroid[22], AppFence[23]	基于 TaintDroid 的优化或应用扩展

Table 7 Research points and solutions in the practice of taint analysis in Web application

表 7 污点分析在 Web 应用中的关键技术概览

Web 框架(Java EE)		
研究难点	代表工作	特点
解决 Java 代码、配置文件 JSP 代码交互处理问题	TAJ[25] Møller 等人[30] F4F[24]	在 Java 代码中手动指定入口 利用页面间信息映射关系以及模式匹配识别统一在 Java 端进行分析 映射到统一语言进行分析
脚本语言(Javascript)		
研究难点	代表工作	特点
特殊语法特性	Chugh 等人[31]/ACTARUS[33]	抽象语法树级别代码重写
动态特性	Chugh 等人[31] Wei 等人[32]	静态制定约束规则,动态进行检查 多次动态执行形成执行踪迹集合,在执行踪迹集合上进行静态分析
DOM 模型交互	Vogt 等人[29] Lekies 等人[34]	简单规则 改变了 JavaScript 引擎 V8 的底层字符串类型

5 总结与展望

污点分析作为信息流分析的一种实践技术,被广泛应用于互联网及移动终端平台上应用程序的信息安全保障中.本文介绍了污点分析的基本原理和通用技术,并针对近年来污点分析在解决实际应用程序安全问题时遇到的问题和关键解决技术进行了分析综述.不同于基于安全类型系统的信息流分析技术,污点分析可以不改变程序现有的编程模型或语言特性,并提供精确信息流传播跟踪.在实际应用过程中,污点分析还需要借助传统的程序分析技术的支持,例如静态分析中的数据流分析、动态分析中的代码重写等技术.另外,结合测试用例生成技术、符号执行技术以及虚拟机技术,也会给污点分析带来更多行之有效的解决方案.

随着计算机系统的发展以及软件规模的扩大,污点分析技术将在应用实践中发挥更大的作用.我们认为,污点分析技术未来的研究趋势包括以下 3 个方面.

1) 首先,进一步提高污点分析通用技术的精度和效率.包括:

(1) 设计与应用别名分析技术.

别名分析技术是程序分析领域的重要研究方向,也是静态污点分析中的关键基础性技术.近年来,该领域的研究工作取得了进一步的发展,别名分析在精度和效率上有所提升.例如,Li 等人[91]使用了值流图(value-flow graph)将传统的流敏感指向分析简化为图可达问题,提升了流敏感别名分析的效率.相关的工作还有文献[91–96]等.研究者可以进一步探索高敏感度且低开销的别名分析技术,并将其应用到污点分析实践中.另外,针对污点分析这一特定需求,一些研究工作开始尝试按需设计的别名分析技术.这一方面可以缩小别名分析需要关注的程序规模(比如只关注从污点源到污点汇聚点之间的程序片段),降低分析开销;另一方面,可以根据被分析程序的语言特性和系统需求,选择合适的分析精度(比如对于 Android 程序中 Intent 目的地参数,使用对象敏感的别名分析可以提高建模精度,从而获得更高精度的污点分析结果).

(2) 优化污点分析(尤其是动态污点分析)的效率.

动态污点分析可以在线运行,运行效率直接影响到用户的体验(例如智能手机应用).因此,无论是基于硬件、基于软件还是混合型的污点分析,都需要尽可能地降低分析代价.目前的解决思路之一是选择性地控制需要进行污点传播的指令数量,之后的工作可以继续遵循这一思想,同时结合其他手段达到更好的优化效果.例如:利用无害处理消除污点数据中的标记,减少不必要的跟踪;利用语义分析或者借助测试方法去除不必要的分析范围等.另外,探索低开销的传播机制也是效率优化的研究趋势(例如 SHIFT 系统将动态污点分析转化为等价的延迟例外的处理问题).

(3) 解决污点传播中的隐藏通道问题.

隐藏通道包含隐式流、停机通道、计时通道、概率通道等[1].第 3.2 节介绍了目前针对隐式流问题的代表性解决方案,但这些方案仍然面临不同程度的欠污染或过污染的情况[97],需要进一步探索如何提供路径敏感且低开销的静态分析技术以及如何解决动态分析中部分泄漏和如何合理选择污点传播分支的问题.其他隐藏通道问题,例如 Android 组件与本地代码之间的污点传播以及程序之间通过文件的污点传播等,目前只能通过一些保守的启发式策略来加以解决.因此,还需要探索更通用的解决方案.

2) 其次,针对新型编程模型,提供定制的污点分析技术.随着互联网+、云计算、机器人等技术的应用普及,不仅应用程序的规模和复杂度进一步提高,新的编程模型也陆续出现.面对新的编程模型的冲击,污点分析仍然是解决其中安全问题的有效手段.污点分析需要针对新的编程模型提供定制的技术,以确保对新编程模型分析的完整性、对新语言特性的适应能力以及与新程序语言模型的协作能力.我们认为,面对新的编程模型的挑战,可以探索使用以下定制策略.

(1) 提供自动化的污点分析工具或方法.

手工解决特定的编程模型问题不仅需要消耗大量的人力成本,而且可移植性较差,目前的研究趋势更趋向于提供自动或半自动的方法解决模型的完整性.第 4 节介绍的研究工作提供了许多有益的技术参考,例如:可以利用添加虚拟接入点的方法解决回调函数特性问题;对于基础性语言类问题,可以尝试提供语法树级别的代码重写;解决不同模块语言差异问题,可以借鉴提供统一映射语言分析(F4F[24])等方案,用户可以根据自己的具体需求(精确度)定制针对性的解决方案.

(2) 充分应用现有的分析方法提高分析精度和效率.

传统的分析框架大多适用于分析独立的模块,新型的编程框架往往导致不同的模块产生分离.研究者需要发掘模型与模型之间的联系将其转化成已有的分析问题进行求解或提高精度.例如:文献[18]将 Android 组件间的污点传播问题转化成 IDE 问题进行求解;在解决组件与库函数的污点传播分析问题时,StubDroid[85]尝试复用 FlowDroid 的数据流分析框架以提供分析支持.

(3) 探索混合的污点分析方法.

静态污点分析存在过于保守的问题,可能带来一定程度的误报.对于某些特定的应用(如 JavaScript 中的动态特性代码),只有在运行时才能获得具体信息,传统的静态污点分析无法精确地分析出其中可能存在的安全问题.而动态污点分析由于只关注程序执行过程中覆盖到的路径,可能忽略了未执行路径上隐藏的安全问题.探索

动静结合的污点分析方法,既可以弥补动态分析由于路径覆盖不足引起的信息丢失的问题,又能够提供某些静态分析无法获得的精确运行信息,提高污点分析的精度.例如:针对 Android 隐私泄露检测工具 Appaudit[90],首先利用轻量级的静态分析缩小了可能的分析范围(指出一些可疑的函数);之后,结合近似执行技术动态验证这些可疑的函数是否具有威胁.

3) 最后,除了解决应用程序中的安全问题这一传统职责外,污点分析技术也逐渐被应用在程序分析领域其他问题(如性能分析、软件缺陷定位、错误检测等)的研究和实践中.RETracer[98]提出了一种针对二进制代码的动态后向污点分析技术来定位引起软件系统崩溃的浅层原因.ConfAid[99]基于通用污点分析技术提出了一种专门用于解决由于配置文件中的错误而导致性能下降的错误定位技术.X-ray[100]在 ConfAid 的基础上,利用污点分析技术逆向追溯导致性能瓶颈的根本原因.如何将污点分析技术与程序分析领域的其他技术相结合、扩展污点分析技术的应用范畴,也是一个值得探索的研究方向.

References:

[1] Sabelfeld A, Myers AC. Language-Based information-flow security. IEEE Journal on Selected Areas in Communications, 2003, 21(1):5−19. [doi: 10.1109/JSAC.2002.806121]

[2] Foundation TO. Top ten most critical Web application vulnerabilities. 2013. https://www.owasp.org/index.php/Top_10_2013-Top_10

[3] McAfee. Mobile Threat Report. 2016. http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf

[4] Citi credit card data breached for 200000 customers. 2011. https://www.wired.com/2011/06/citi-credit-card-breach/

[5] Dennis JB, Van Horn EC. Programming semantics for multiprogrammed computations. Communications of the ACM, 1966,9(3): 143−155. [doi: 10.1145/365230.365252]

[6] Sandhu RS, Samarati P. Access control: Principles and practice. Communications Magazine, 1994,32(9):40−48. [doi: 10.1109/35.312842]

[7] Oppliger R. Internet security: Firewalls and beyond. Communications of the ACM, 1997,40(5):92−102. [doi: 10.1145/253769.253802]

[8] Bellare M, Boldyreva A, Micali S. Public-Key encryption in a multi-user setting: Security proofs and improvements. In: Proc. of the Int'l Conf. on the Theory and Applications of Cryptographic Techniques. Berlin, Heidelberg: Springer-Verlag, 2000. 259−274. [doi: 10.1007/3-540-45539-6_18]

[9] Myers AC, Liskov B. A decentralized model for information flow control. ACM SIGOPS Operating Systems Review, 1997,31(5): 129−142. [doi: 10.1145/268998.266669]

[10] Livshits VB, Lam MS. Finding security vulnerabilities in Java applications with static analysis. In: Proc. of the Conf. on Usenix Security Symp. USENIX Association, 2005. 262−266. https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits_html/

[11] Rasthofer S, Arzt S, Bodden E. A machine-learning approach for classifying and categorizing android sources and sinks. In: Proc. of the Network and Distributed System Security Symp. (NDSS). 2014. [doi: 10.14722/ndss.2014.23039]

[12] Gordon MI, Kim D, Perkins JH, Gilham L, Nguyen N, Rinard MC. Information flow analysis of Android applications in DroidSafe. In: Proc. of the NDSS 2015. 2015. [doi: 10.14722/ndss.2015.23089]

[13] Arzt S, Rasthofer S, Fritz C, Bodden E, Bartel A, Klein J, Le Traon Y, Octeau D, McDaniel P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. ACM SIGPLAN Notices, 2014,49(6):259−269. [doi: 10.1145/2594291.2594299]

[14] Yang Z, Yang M. Leakminer: Detect information leakage on Android with static taint analysis. In: Proc. of the Software Engineering. IEEE, 2012. 101−104. [doi: 10.1109/WCSE.2012.26]

[15] Lu L, Li Z, Wu Z, Lee W, Jiang G. Chex: Statically vetting Android apps for component hijacking vulnerabilities. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. ACM Press, 2012. 229−240. [doi: 10.1145/2382196.2382223]

[16] Zhao Z, Osono FC. "TrustDroid™": Preventing the use of SmartPhones for information leaking in corporate networks through the used of static analysis taint tracking. In: Proc. of the 7th Int'l Conf. on Malicious and Unwanted Software (MALWARE). IEEE, 2012. 135−143. [doi: 10.1109/MALWARE.2012.6461017]

[17] Klieber W, Flynn L, Bhosale A, Jia L, Bauer L. Android taint flow analysis for app sets. In: Proc. of the ACM Sigplan Int'l Workshop on the State of the Art in Java Program Analysis. ACM Press, 2014. 1−6. [doi: 10.1145/2614628.2614633]

[18] Octeau D, McDaniel P, Jha S, Bartel A, Bodden E, Klein J, Le Traon Y. Effective inter-component communication mapping in android with EPICC: An essential step towards holistic security analysis. In: Proc. of the Usenix Conf. on Security. 2013. 543−558. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/octeau

[19] Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. on Computer Systems, 2014,32(2):393−407. [doi: 10.1145/2619091]

[20] Rastogi V, Chen Y, Enck W. AppsPlayground: Automatic security analysis of smartphone applications. In: Proc. of the ACM Conf. on Data and Application Security and Privacy. 2013. 209−220. [doi: 10.1145/2435349.2435379]

[21] Zhang Y, Yang M, Xu B, Yang Z, Gu G, Ning P, Wang XS, Zang B. Vetting undesirable behaviors in Android apps with permission use analysis. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. 2013. 611−622. [doi: 10.1145/2508859.2516689]

[22] Tripp O, Rubin J. A Bayesian approach to privacy enforcement in smartphones. In: Proc. of the Usenix Conf. on Security Symp. USENIX Association, 2014. 175−190. https://www.usenix.org/node/184428

[23] Hornyack P, Han S, Jung J, Schechter S, Wetherall D. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In: Proc. of the ACM Conf. on Computer and Communications Security (CCS 2011). Chicago, 2011. 639−652. [doi: 10.1145/2046707.2046780]

[24] Sridharan M, Artzi S, Pistoia M, Guarnieri S, Tripp O, Berg R. F4F: Taint analysis of framework-based Web applications. ACM SIGPLAN Notices, 2011,46(10):1053−1068. [doi: 10.1145/2048066.2048145]

[25] Tripp O, Pistoia M, Fink SJ, Sridharan M, Weisman O. TAJ: Effective taint analysis of Web applications. ACM SIGPLAN Notices, 2009,44(6):87−97. [doi: 10.1145/1542476.1542486]

[26] Papagiannis I, Migliavacca M, Pietzuch P. PHP ASPIS: Using partial taint tracking to protect against injection attacks. In: Proc. of the Usenix Conf. on Web Application Development. USENIX Association, 2011. 2. https://www.usenix.org/conference/webapps11/php-aspis-using-partial-taint-tracking-protect-against-injection-attacks

[27] Balzarotti D, Cova M, Felmetsger V, Jovanovic N, Kirda E, Kruegel C, Vigna G. Saner: Composing static and dynamic analysis to validate sanitization in Web applications. In: Proc. of the 2012 IEEE Symp. on Security and Privacy. IEEE, 2008. 387−401. [doi: 10.1109/SP.2008.22]

[28] Halfond WG, Orso A, Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In: Proc. of the ACM Sigsoft Int'l Symp. on Foundations of Software Engineering (FSE 2006). Oregon, 2006. 175−185. [doi: 10.1145/1181775.1181797]

[29] Vogt P, Nentwich F, Jovanovic N, Kirda E, Kruegel C, Vigna G. Cross site scripting prevention with dynamic data tainting and static analysis. In: Proc. of the NDSS 2007. 2007. 12. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.4505

[30] Møller A, Schwarz M. Automated detection of client-state manipulation vulnerabilities. ACM Trans. on Software Engineering and Methodology (TOSEM), 2014,23(4):29. [doi: 10.1145/2531921]

[31] Chugh R, Meister JA, Jhala R, Lerner S. Staged information flow for JavaScript. ACM SIGPLAN Notices, 2009,44(6):50−62. [doi: 10.1145/1542476.1542483]

[32] Wei S, Ryder BG. Practical blended taint analysis for JavaScript. In: Proc. of the 2013 Int'l Symp. on Software Testing and Analysis. ACM Press, 2013. 336−346. [doi: 10.1145/2483760.2483788]

[33] Guarnieri S, Pistoia M, Tripp O, Dolby J, Teilhet S, Berg R. Saving the world wide Web from vulnerable JavaScript. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. ACM Press, 2011. 177−187. [doi: 10.1145/2001420.2001442]

[34] Lekies S, Stock B, Johns M. 25 million flows later: Large-Scale detection of DOM-based XSS. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. ACM Press, 2013. 1193−1204. [doi: 10.1145/2508859.2516703]

[35] Stock B, Lekies S, Mueller T, Spiegel P, Johns M. Precise client-side protection against dom-based cross-site scripting. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 2014). 2014. 655−670. https://www.usenix.org/node/184492

[36] Denning DE. A lattice model of secure information flow. Communications of the ACM, 1976,19(5):236−243. [doi: 10.1145/360051.360056]

[37] Denning DE, Denning PJ. Certification of programs for secure information flow. Communications of the ACM, 1977,20(7):504−13. [doi: 10.1145/359636.359712]

[38] Goguen JA, Meseguer J. Security policies and security models. In: Proc. of the '82 IEEE Symp. on Security and Privacy. IEEE, 1982. 11−11. [doi: 10.1109/SP.1982.10014]

[39] Myers AC. JFlow: Practical mostly-static information flow control. In: Proc. of the 26th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM Press, 1999. 228−241. [doi: 10.1145/292540.292561]

[40] Heintze N, Riecke JG. The SLam calculus: Programming with secrecy and integrity. In: Proc. of the 25th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM Press, 1998. 365−377. [doi: 10.1145/268946.268976]

[41] Myers AC, Liskov B. Protecting privacy using the decentralized label model. ACM Trans. on Software Engineering and Methodology (TOSEM), 2000,9(4):410−442. [doi: 10.1145/363516.363526]

[42] Sabelfeld A, Sands D. Probabilistic noninterference for multi-threaded programs. In: Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW-13). IEEE, 2000. 200−214. [doi: 10.1109/CSFW.2000.856937]

[43] Pottier F, Simonet V. Information flow inference for ML. ACM Trans. on Programming Languages and Systems (TOPLAS), 2003, 25(1):117−158. [doi: 10.1145/596980.596983]

[44] Abadi M, Banerjee A, Heintze N, Riecke JG. A core calculus of dependency. In: Proc. of the 26th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM Press, 1999. 147−160. [doi: 10.1145/292540.292555]

[45] Ashcraft K, Engler D. Using programmer-written compiler extensions to catch security holes. In: Proc. of the 2002 IEEE Symp. on Security and Privacy. IEEE, 2002. 143−159. [doi: 10.1109/SECPRI.2002.1004368]

[46] Volpano D, Irvine C, Smith G. A sound type system for secure flow analysis. Journal of Computer Security, 1996,4(2-3):167−87. [doi: 10.3233/JCS-1996-42-304]

[47] Foster JS, Fähndrich M, Aiken A. A theory of type qualifiers. ACM SIGPLAN Notices, 1999,34(5):192−203. [doi: 10.1145/301631.301665]

[48] Shankar U, Talwar K, Foster JS, Wagner D. Detecting format string vulnerabilities with type qualifiers. In: Proc. of the USENIX Security Symp. 2001. 201−220. https://www.usenix.org/conference/10th-usenix-security-symposium/detecting-format-string-vulnerabilities-type-qualifiers

[49] King D, Hicks B, Hicks M, Jaeger T. Implicit flows: Can't live with 'em, can't live without 'em. In: Proc. of the Int'l Conf. on Information Systems Security. Berlin, Heidelberg: Springer-Verlah, 2008. 56−70. [doi: 10.1007/978-3-540-89862-7_4]

[50] Saxena P, Molnar D, Livshits B. SCRIPTGARD: Automatic context-sensitive sanitization for large-scale legacy Web applications. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 601−614. [doi: 10.1145/2046707.2046776]

[51] Samuel M, Saxena P, Song D. Context-Sensitive auto-sanitization in Web templating languages using type qualifiers. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 587−600. [doi: 10.1145/2046707.2046775]

[52] Bates D, Barth A, Jackson C. Regular expressions considered harmful in client-side XSS filters. In: Proc. of the 19th Int'l Conf. on World Wide Web. ACM Press, 2010. 91−100. [doi: 10.1145/1772690.1772701]

[53] Hooimeijer P, Livshits B, Molnar D, Saxena P, Veanes M. Fast and precise sanitizer analysis with BEK. In: Proc. of the 20th USENIX Conf. on Security. USENIX Association, 2011. 1−1. http://dl.acm.org/citation.cfm?id=2028068

[54] Aho AV, Sethi R, Ullman JD. Compilers, Principles, Techniques. Boston: Addison Wesley, 1986.

[55] Nielson F, Nielson HR, Hankin C. Principles of Program Analysis. New York: Springer-Verlag, 2015.

[56] Andersen LO. Program analysis and specialization for the C programming language. Addison-Wesley Series in Computer Science, 1994,2(1):37−77.

[57] Reps T, Horwitz S, Sagiv M. Precise interprocedural dataflow analysis via graph reachability. In: Proc. of the 22nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM Press, 1995. 49−61. [doi: 10.1145/199448.199462]

[58] Sagiv M, Reps T, Horwitz S. Precise interprocedural dataflow analysis with applications to constant propagation. Theoretical Computer Science, 1996,167(1):131−70. [doi: 10.1016/0304-3975(96)00072-2]

[59] Li Y, Tan T, Zhang Y, Xue J. Program tailoring: Slicing by sequential criteria. In: Proc. of the ECOOP. 2016. http://drops.dagstuhl.de/opus/volltexte/2016/6109/

[60] Ming CC, Huo W, Yu HT. A survey of optimization technology of inclusion-based pointer analysis. Chinese Journal of Computers, 2011,34(7):1224−1238 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01224]

[61] Tripp O, Pistoia M, Cousot P, Cousot R, Guarnieri S. Andromeda: Accurate and scalable security analysis of Web applications. In: Proc. of the Int'l Conf. on Fundamental Approaches to Software Engineering. Berlin, Heidelberg: Springer-Verlag, 2013. 210−225. [doi: 10.1007/978-3-642-37057-1_15]

[62] Crandall JR, Chong FT. Minos: Control data attack prevention orthogonal to memory model. In: Proc. of the 37th Int'l Symp. on Microarchitecture (MICRO-37). IEEE, 2004. 221−232. [doi: 10.1109/MICRO.2004.26]

[63] Dalton M, Kannan H, Kozyrakis C. Raksha: A flexible information flow architecture for software security. ACM SIGARCH Computer Architecture News, 2007,35(2):482−493. [doi: 10.1145/1273440.1250722]

[64] Zhu DY, Jung J, Song D, Kohno T, Wetherall D. Tainteraser: Protecting sensitive data leaks using applicationlevel taint tracking. ACM SIGOPS Operating Systems Review, 2011,45(1):142−154. [doi: 10.1145/1945023.1945039]

[65] Venkataramani G, Doudalis I, Solihin Y, Prvulovic M. Flexitaint: A programmable accelerator for dynamic taint propagation. In: Proc. of the 2008 IEEE 14th Int'l Symp. on High Performance Computer Architecture. IEEE, 2008. 173−184. [doi: 10.1109/HPCA.2008.4658637]

[66] Yoon MK, Salajegheh N, Chen Y, Christodorescu M. PIFT: Predictive information-flow tracking. In: Proc. of the 21st Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM Press, 2016. 713−725. [doi: 10.1145/2872362.2872403]

[67] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. of the Network and Distributed System Security Symp. 2005. 720−724.

[68] Nethercote N, Seward J. Valgrind: A program supervision framework. Electronic Notes in Theoretical Computer Science, 2003, 89(2):44−66. [doi: 10.1016/S1571-0661(04)81042-9]

[69] Zhu Y, Jung J, Song D, Kohno T, Wetherall D. Privacy scope: A precise information flow tracking system for finding application leaks. Technical Report, EECS-2009-145, Berkeley: University of California, 2009.

[70] Clause J, Li W, Orso A. DYTAN: A generic dynamic taint analysis framework. In: Proc. of the 2007 Int'l Symp. on Software Testing and Analysis. ACM Press, 2007. 196−206. [doi: 10.1145/1273463.1273490]

[71] Kemerlis VP, Portokalidis G, Jee K, Keromytis AD. libdft: Practical dynamic data flow tracking for commodity systems. ACM SIGPLAN Notices, 2012,47(7):121−132. [doi: 10.1145/2365864.2151042]

[72] Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K. Pin: Building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005,40(6):190−200. [doi: 10.1145/1064978.1065034]

[73] Dalvik executable format. 2016. https://source.android.com/devices/tech/dalvik/dex-format.html

[74] Qin F, Wang C, Li Z, Kim HS, Zhou Y, Wu Y. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In: Proc. of the 39th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO 2006). IEEE, 2006. 135−148. [doi: 10.1109/MICRO.2006.29]

[75] Chen H, Wu X, Yuan L, Zang B, Yew PC, Chong FT. From speculation to security: Practical and efficient information flow tracking using speculative hardware. In: Proc. of the Int'l Symp. on Computer Architecture. IEEE Computer Society, 2008. 401−412. [doi: 10.1109/ISCA.2008.18]

[76] Bao T, Zheng Y, Lin Z, Zhang X, Xu D. Strict control dependence and its effect on dynamic information flow analyses. In: Proc. of the 19th Int'l Symp. on Software Testing and Analysis. ACM Press, 2010. 13−24. [doi: 10.1145/1831708.1831711]

[77] Kang MG, McCamant S, Poosankam P, Song D. DTA++: Dynamic taint analysis with targeted control-flow propagation. In: Proc. of the Network and Distributed System Security Symp. (NDSS 2011). San Diego, 2011.

[78] Cox LP, Gilbert P, Lawler G, Pistol V, Razeen A, Wu B, Cheemalapati S. Spandex: Secure password tracking for Android. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 2014). 2014. 481−494. https://www.usenix.org/node/184402

[79] Qing SH. Research progress on Android security. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):45−71 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4914.htm [doi: 10.13328/j.cnki.jos.004914]

[80] Zhang YQ, Wang K, Yang H, Fang ZJ, Wang ZQ, Cao C. Survey of Android OS security. Journal of Computer Research and Development, 2014,51(7):1385−1396 (in Chinese with English abstract). [doi: 10.7544/issnl1000-1239.2014.20140098]

[81] Felt AP, Chin E, Hanna S, Song D, Wagner D. Android permissions demystified. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 627−638. [doi: 10.1145/2046707.2046779]

[82] Corporation ID. Smartphone OS market share. 2015. http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[83] The Android software stack. 2016. https://developer.android.com/guide/platform/index.html

[84] Christensen AS, Møller A, Schwartzbach MI. Precise analysis of string expressions. In: Proc. of the Int'l Static Analysis Symp. Berlin, Heidelberg: Springer-Verlag, 2003. 1−18. [doi: 10.1007/3-540-44898-5_1]

[85] Arzt S, Bodden E. StubDroid: Automatic inference of precise data-flow summaries for the Android framework. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 725−735. [doi: 10.1145/2884781.2884816]

[86] Wikipedia. Comparison of Web application frameworks. 2016. https://en.wikipedia.org/wiki/Comparison_of_web_frameworks

[87] Web framework. 2016. https://en.wikipedia.org/wiki/Web_framework

[88] ECMA Int'l. ECMAScript language specification. Version 5.1. 2011. http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf

[89] Klein A. Dom based cross site scripting or XSS of the third kind. Web Application Security Consortium, 2005,4:365−372.

[90] Xia M, Gong L, Lyu Y, Qi Z, Liu X. Effective real-time Android application auditing. In: Proc. of the 2015 IEEE Symp. on Security and Privacy. IEEE, 2015. 899−914. [doi: 10.1109/SP.2015.60]

[91] Li L, Cifuentes C, Keynes N. Boosting the performance of flow-sensitive points-to analysis using value flow. In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering. ACM Press, 2011. 343−353. [doi: 10.1145/2025113.2025160]

[92] Li L, Cifuentes C, Keynes N. Precise and scalable context-sensitive pointer analysis via value flow graph. ACM SIGPLAN Notices, 2013,48(11):85−96. [doi: 10.1145/2555670.2466483]

[93] Yu H, Xue J, Huo W, Feng X, Zhang Z. Level by level: Making flow-and context-sensitive pointer analysis scalable for millions of lines of code. In: Proc. of the 8th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization. ACM Press, 2010. 218−229. [doi: 10.1145/1772954.1772985]

[94] Sui Y, Xue J. On-Demand strong update analysis via value-flow refinement. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016. 460−473. [doi: 10.1145/2950290.2950296]

[95] Tan T, Li Y, Xue J. Making *k*-object-sensitive pointer analysis more precise with still *k*-limiting. In: Proc. of the Int'l Static Analysis Symp. Berlin, Heidelberg: Springer-Verlag, 2016. 489−510. [doi: 10.1007/978-3-662-53413-7_24]

[96] Sui Y, Ye S, Xue J, Zhang J. Making context-sensitive inclusion-based pointer analysis practical for compilers using parameterised summarisation. Software: Practice and Experience, 2014,44(12):1485−1510. [doi: 10.1002/spe.2214]

[97] Sarwar G, Mehani O, Boreli R, Kaafar MA. On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices. In: Proc. of the 2013 Int'l Conf. on Security and Cryptography (SECRYPT). IEEE, 2013. 1−8. http://ieeexplore.ieee.org/document/7223198/

[98] Cui W, Peinado M, Cha SK, Fratantonio Y, Kemerlis VP. RETracer: Triaging crashes by reverse execution from partial memory dumps. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 820−831. [doi: 10.1145/2884781.2884844]

[99] Attariyan M, Flinn J. Automating configuration troubleshooting with dynamic information flow analysis. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation. USENIX Association, 2010. 1−11. https://www.usenix.org/conference/osdi10/automating-configuration-troubleshooting-dynamic-information-flow-analysis

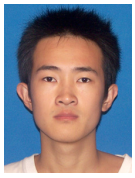
[100] Attariyan M, Chow M, Flinn J. *X*-Ray: Automating root-cause diagnosis of performance anomalies in production software. In: Proc. of the Presented as Part of the 10th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2012). 2012. 307−320. https://www.usenix.org/node/170861

附中文参考文献:

[60] 陈聪明,霍玮,于洪涛,冯晓兵.基于包含的指针分析优化技术综述.计算机学报,2011,34(7):1224−1238. [doi: 10.3724/SP.J.1016.2011.01224]

[79] 卿斯汉.Android 安全研究进展.软件学报,2016,27(1):45−71. http://www.jos.org.cn/1000-9825/4914.htm [doi: 10.13328/j.cnki.jos.004914]

[80] 张玉清,王凯,杨欢,方喆君,王志强,曹琛.Android 安全综述.计算机研究与发展,2014,51(7):1385−1396. [doi: 10.7544/issnl1000-1239.2014.20140098]



王蕾(1989－),男,吉林白山人,博士生,主要研究领域为程序分析,软件安全.

李炼(1977－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为程序分析,编译,自动测试,软件安全.

李丰(1985－),女,博士,助理研究员,CCF 专业会员,主要研究领域为程序分析,故障定位.

冯晓兵(1969－),男,博士,研究员,博士生导师,CCF 杰出会员,主要研究领域为编程模型,编译优化.