Journal of Computer Applications
计算机应用,2019,39(8):2302 - 2309
ISSN 1001-9081
CODEN JYIIDU
2019-08-10
http://www.joca.cn

文章编号:1001-9081(2019)08-2302-08
doi:10.11772/j.issn.1001-9081.2019020238

污点分析技术研究综述

任玉柱[1,2*],张有为[3],艾成炜[1,2]
(1.信息工程大学 网络空间安全学院,郑州 450001; 2.数学工程与先进计算国家重点实验室,郑州 450001;
3.郑州信大先进技术研究院,郑州 450001)
(* 通信作者电子邮箱 11732304@qq.com)

摘 要: 污点分析技术是保护隐私数据安全和实现漏洞检测的重要技术手段,也是信息安全研究的热点领域。对近年来污点分析技术的研究现状和发展情况进行综述,介绍了污点分析的理论基础以及静态污点分析和动态污点分析的基本概念、关键技术和研究进展,并从技术实现方式的角度出发,阐述了基于硬件、软件、虚拟环境和代码等四种污点分析技术的实现方式、核心思想以及优缺点;然后,从污点数据流向的角度出发,概述了污点分析技术在相关领域的两种典型应用,即隐私数据泄露检测和漏洞探测;最后,简要分析了污点分析的缺点和不足,并展望该技术的研究前景和发展趋势。

关键词: 信息流分析;静态污点分析;动态污点分析;隐私数据;漏洞挖掘
中图分类号: TP309 **文献标志码:** A

Survey on taint analysis technology

REN Yuzhu[1,2*], ZHANG Youwei[3], AI Chengwei[1,2]
(1. *School of Cyber Security, Information Engineering University, Zhengzhou Henan* 450001, *China*;
2. *State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou Henan* 450001, *China*;
3. *Advanced Technology Research Institute of Zhengzhou Xinda, Zhengzhou Henan* 450001, *China*)

Abstract: Taint analysis technology is an important method to protect private data security and realize vulnerability detection, and it is also a hot area of information security research. The research status and development of taint analysis technology in recent years were summarized. The theoretical basis of taint analysis, the basic concepts, key technologies and research progress of static taint analysis and dynamic taint analysis were introduced. From the perspective of the implementation, the implementation methods, core ideas, advantages and disadvantages of four taint analysis technologies based on hardware, software, virtual environment and code were expounded; from the perspective of the flow of taint data, two typical applications in related fields, privacy data leakage detection and vulnerability detection, were outlined. Finally, the shortcomings of taint analysis were briefly analyzed, and the research prospects and development trends of the technology were predicted.

Key words: information flow analysis; static taint analysis; dynamic taint analysis; privacy data; vulnerability discovery

0 引言

随着“互联网+”概念的流行和物联网技术的发展,世界已进入以信息数据为主要经济发展驱动力的全球化“3.0”时代,信息技术对人类社会的意义愈发重要,随之而来的是人类在信息安全领域面临着诸多难题。僵尸网络、蠕虫病毒、木马病毒及分布式拒绝服务攻击等计算机病毒和网络攻击行为屡屡发生。2017年,以智能手机为代表的手持移动设备已经超越计算机成为数量最多的上网终端,全球社交媒体和移动端活跃用户数量都已达到30亿左右,预计到2020年全球联网设备总数可能会超过500亿台[1],信息安全已经成为热点研究领域,特别是有关信息完整性和保密性的研究更受到越来越多的重视。完整性指信息在存储、传输和处理过程中不被修改、破坏和丢失的特性。保密性指信息不会被泄露给非授权用户的特性。随着物联网的发展和智能终端的普及,在社交网络和移动支付等活动中都会涉及到大量的用户隐私数据。作为维护数据完整性和保密性的有效手段之一,信息流分析技术得到了广泛的运用。

信息流分析技术[2]的核心思想是:首先给数据添加一个标签,此标签随数据在系统中传播,并根据标签来获得数据在系统或应用程序中传播的相关信息;然后制定信息流策略并建立相应的信息流模型,通过分析数据传播的相关信息以确定是否存在违反信息流策略的信息安全隐患,即若被分析对象的信息传播过程符合制定的信息流策略,则认为被分析对象是安全的,否则就是不安全的。根据信息流策略描述方法的不同,信息流模型可以分为基于格、基于安全类型、基于安全进程代数(Security Process Algebra, SPA)和基于自动机等四种主要形式。由于本文主要介绍污点分析技术,所以只对主流的信息流模型进行整理对比,如表1所示。

本文给出了污点分析定义和模型,介绍了静态污点分析和动态污点分析的基本原理、运用和优缺点;从技术实现方式的角度,分别阐述基于硬件、软件、虚拟环境和代码实现污点分析的核心原理、技术方法以及优缺点;从数据流向的角度,概述污点分析

收稿日期:2019-02-18;修回日期:2018-04-12;录用日期:2018-04-28。

作者简介:任玉柱(1985—),男,山东莱州人,硕士研究生,主要研究方向:逆向工程、漏洞挖掘; 张有为(1975—),男,湖北荆州人,讲师,硕士,主要研究方向:逆向工程、电子取证、数据恢复; 艾成炜(1990—),男,湖南益阳人,硕士研究生,主要研究方向:逆向工程、漏洞挖掘。

技术在隐私数据泄漏检测和漏洞探测这两个主要领域的应用情况; 分析污点分析技术的不足,并探讨未来的发展趋势。

表 1　信息流模型研究代表工作

Tab. 1　Representative works on information flow model research

模型分类	优点	缺点
基于格的信息流模型	要求数据只能从高安全级别向低安全级别流动,保证数据安全	1. 模型要求数据只能单向流动,影响了模型应用范围; 2. 不能对系统中存在的隐蔽通道进行准确描述
基于安全类型的信息流模型	可根据需要定制安全类型和规则	1. 安全类型与应用的编程语言相关,可移植性较差; 2. 基于安全类型的模型存在精确性差的问题
基于 SPA 的信息流模型	以代数形式描述系统行为,能够较好地解决系统概率行为引起的信息泄露和安全降密等问题	1. 存在安全性验证困难问题; 2. 在全系统实现模型比较困难,一般只用于局部性验证
基于自动机的信息流模型	用自动机模型来形式化描述系统安全状态转移规则,应用广泛,移植性好	实现难度随分析对象规模增大而增大

1　理论基础

污点分析技术是信息流分析技术的一种实践方法[3],在信息泄露检测、漏洞探测、逆向工程等方面有广泛的应用,并被移植到各种不同的环境和平台当中。污点分析技术将系统或应用程序中的数据标记为污点或非污点,当污点数据根据信息流传播策略可影响到非污点数据时,则将该非污点数据的标记修改为污点。当污点标签最终随数据传播到指定的存储区域或者信息泄露点时,则认定该系统违反了信息流策略。

1.1　污点分析模型

污点分析技术模型包括污点源、污点汇聚点和无害化处理等部分。其中,污点源表示将污点数据引入到系统中; 污点汇聚点表示系统将污点数据输出到敏感数据区或者外界,造成敏感数据区被非法改写或者隐私数据泄露; 无害化处理表示通过数据加密或重新赋值等操作使数据传播不再对系统的完整性和保密性产生危害。以下面的代码为例说明污点分析的一般过程。

```
1)  def rectangle ( ) :
2)      var1 = inpu( 'Enter the first number: ' )
3)      var3 = 5
4)      var4 = var1 *  var3
5)      var7 = int( var4)
6)      return var7
7)  def reset ( ) :
8)      var2 = inpu( 'Enter the second number: ' )
9)      var5 = var2 *  0
10)     return var5
11) print ( rectangle_area( ) )
12) print ( reset_switch( ) )
```

其中,rectangle 函数表示将输入数值乘 5 后返回结果,reset 函数表示将输入数值重置为 0,代码中各变量之间的关系如图 1 所示。

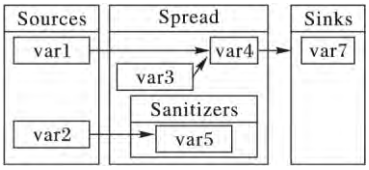

图 1　污点分析模型

Fig. 1　Taint analysis model

将代码中第 2) 和第 10) 行的输入标记为污点源后,var1 和 var2 均被标记为污点变量。根据数据依赖关系,var4 和 var7 也会被标记为污点变量。由于污点数据 var7 位于污点汇聚点,即可认为 rectangle 函数违反了信息安全的完整性或保密性。

污点数据主要通过数据间的依赖关系在系统或应用程序中进行传播,这种依赖关系又分为数据依赖和控制依赖两种。数据依赖主要包括程序中各变量间的直接赋值、数学计算等操作,如上述代码中 var1 含有的污点信息通过算术运算操作传递给 var4。控制依赖主要包括程序中各变量间的条件判断与指令跳转等情况,即 var1 含有的污点信息通过包含 var1 的条件指令间接传递给其他变量,如下面代码所示。

```
1)  def implicit_flow( ) :
2)      var1 = inpu( 'Enter your input: ' )
3)      ( var2, var3, var4) = ( 0,0,0)
4)      if ( var1 == 10) :
5)          var2 = 1
6)      elif ( var1  >  10) :
7)          var3 = 2
8)      elif ( var1  <  10) :
9)          var4 = 3
10) return print_answer( var2, var3, var4)
11) def print_answer( var2, var3, var4) :
12)     print( 'var2 = ', var2)
13)     print( 'var3 = ', var3)
14)     print( 'var4 = ', var4)
15) implicit_flow( )
```

将代码中第 2) 行的输入标记为污点源后,var1 被标记为污点变量,第 4) 、6) 、8) 行是包含 var1 的条件分支指令。若此时的污点传播策略为只要分支指令中包含污点数据就进行污点传播,那么 var2、var3 和 var4 均被标记为污点变量,并最终传播到第 10) 行的污点汇聚点,会出现污点标记数量过多的问题。但实际上,该程序根据 var1 数值不同,每次运行最多只有一条分支会被污染。若仅按照数据依赖关系进行污点传播,则各分支均不会被标记为污点,会造成污点标记数量过少。两种情况都会导致污点分析效率和准确性降低的问题。

污点分析一般存在两个问题,即过污染和欠污染。其中,过污染是指在污染分析过程中将与污点源没有数据和控制依赖关系的数据变量标记为污点变量,即产生误报; 欠污染则是在污染分析过程中将与污点源存在数据或控制依赖关系的程序变量标记为非污点变量,即产生漏报。对控制依赖关系的分析不足和不当可能会造成过污染和欠污染问题。动态污点分析工具 Dytan[4] 引入了控制流污染的概念,提升了污点分析的全面性。

无害化处理是指系统中的某些模块、函数或者数据处理操作,使得污点数据经过处理后不再携带有敏感数据或者不会对系统造成危害。如第一段代码中的 reset 函数可看作是

一个无害化处理操作(任何数值与 0 的乘积均为 0),所以 var5 不会被标记为污点变量。

在检测系统或应用程序是否存在隐私数据泄露时,应用程序调用了某个加密函数对隐私数据进行加密,这样即便此数据通过 API 向外发送,攻击者也难以通过解密来获得隐私数据,则可认为该加密函数完成了一个无害化处理操作。在探测系统或应用是否存在漏洞时,输入验证模块也可以被识别成无害化处理操作,比如某些 Web 网站会采用对输入的用户名和密码进行转义以防止 SQL 注入,此时,可以将该转义操作认为是一个无害化处理操作。污点数据经过无害化处理后,污点标记可被移除。正确地识别和利用无害化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,并避免由于污点扩散导致的过污染问题。

根据分析过程中是否需要运行目标程序,污点分析技术被分为静态污点分析和动态污点分析两种。

1.2 静态污点分析

静态污点分析主要通过词法和语法分析等方法离线分析变量间数据和控制依赖关系,以检测污点数据能否从污点源传播到污点汇聚点,在此过程中既不运行目标程序,也无需修改代码。静态污点分析的对象是程序代码或中间表示(Intermediate Representation, IR)。

在分析数据依赖关系时,首先需要恢复函数调用关系图,然后根据具体的程序特性进行函数内或函数间的污点传播分析。常见的数据依赖关系包括赋值、函数调用和别名,其中研究难点在于对别名的分析,为了降低静态污点分析技术的资源开销,研究者们设计了多种方法来进行别名分析。如 Livshits 等[5]采取按需的上下文敏感的别名分析方法,配合程序查询语言污点检查策略,来检测 Java 应用程序漏洞。ANDROMEDA[6]工具利用对象敏感别名分析方法解决分析对象的访问路径问题。

在分析控制依赖关系时,需要恢复控制流图。恢复控制流图通常采用递归算法:首先寻找并分析基本块 Ba,识别其后续基本块 Bb 和 Bc 并添加到控制流图中;然后重复对 Bb 和 Bc 进行相同的递归分析,直到没有新的后续基本块被识别且所有基本块均被添加到控制流图中。恢复控制流图的研究难点在于间接跳转,即当二进制文件将控制流传输到由寄存器或存储器位置中的值表示的目标时,发生间接跳转。与直接跳转的目标被编码到指令本身以正常解析不同,间接跳转的目标受多个因素的影响而变化。

常用的静态污点分析工具 BinCAT 通过抽象解释,实现了控制流图恢复和污点分析,并能集成到反汇编工具 IDA 中使用。静态二进制分析平台 Jakstab[7]在线性扫描和递归扫描的基础上,借助数据流分析进行多次迭代并恢复控制流图,能获得更准确的结果。

静态污点分析的优点是考虑了程序所有可能的执行路径,但由于不运行目标程序,无法得到程序运行时的额外信息,所以存在分析结果不够准确的问题。由于程序源代码和 IR 能够提供清晰的污点传播规则,因此在源代码和 IR 的基础上进行静态污点分析易于实现。

1.3 动态污点分析

动态污点分析是在目标程序运行过程中,通过实时跟踪监控并记录程序变量、寄存器和内存等的值,确定污点数据能否从污点源传播到污点汇聚点。动态污点分析能够比较准确地获得程序执行过程中各变量和存储单元的状态,有效提高污点分析的精确度。

动态污点分析首先采用插桩的方法,在不破坏目标程序原有逻辑的基础上插入一些采集信息的代码,从而获得程序运行的相关信息,目前常用的插桩平台有 Pin 和 Valgrind。接着,在原数据基础上增加一个污点标记来表示寄存器及内存的污染情况,通常被称为影子内存。Valgrind 实现了可保存四种污点信息状态的污点信息存储。然后,还要根据程序所运行环境的指令类型和指令操作数设计相应的污点传播规则传播污点标记。最后,通过分析程序运行相关信息获得目标程序的数据流和控制流信息。

动态污点分析工具 TaintCheck 基于插桩平台 Valgrind,由 TaintSeed、TaintTracker、TaintAssert 和 ExploitAnalyzer 等四个模块组成。TaintCheck 以二进制文件为分析对象,借助 Valgrind 在程序执行中动态地将执行指令翻译成 Ucode 形式的中间表示,并将污点信息存储在 Taint Structure 数据结构,最后通过 TaintAssert 检查是否存在危险数据操作。

动态污点分析的优点是通过在程序中插桩以获得程序执行中的具体信息,分析精度高,但频繁的插桩操作和影子内存的设计会占用系统资源,污点分析执行效率比静态分析低。此外,动态污点分析一般只能分析程序执行过程中所覆盖到的路径,可能会产生漏报。

在实际应用中,为了克服动态污点分析与静态污点分析的缺点,也有研究者将二者相结合,其方法通常是:先执行静态污点分析获得初步信息,提高路径覆盖率,节约分析时间和系统开销;然后再执行动态污点分析获得安全漏洞或者危险数据操作的具体信息,提高分析的准确度。除了提高路径覆盖率,静态污点分析还可以筛选出供后续动态污点分析的路径,减小动态污点分析的范围,提高分析执行效率。例如,用于检测 Android 应用隐私数据泄露的 Appaudit[8]首先利用轻量级静态分析技术筛选出可疑函数,再通过动态执行技术检测这些函数是否存在隐私数据泄露问题。黄强等[9]提出了一种检测软件脆弱性的方案,先利用静态分析技术排除不可达执行,搜索程序中导致软件潜在脆弱性的所有路径;然后使用污点检查策略识别脆弱性类型;最后,在潜在的问题点插桩数据验证和处理函数,随程序源代码一同编译生成可执行文件,增强程序运行时的监控能力。

2 污点分析实现方式

自污点分析技术被提出后,研究者们进行了深入的研究,并实现了许多框架和工具。根据其实现方式的不同,这些工具大致可分为基于硬件、基于软件、基于虚拟环境和基于代码等四个层次。

2.1 基于硬件的污点分析

基于硬件的污点分析的核心思想是对寄存器、缓存和内存等硬件结构进行重新设计,增加相应的污点信息标志位以实现污点信息的存储、传播和检测等功能。

例如,动态污点分析工具 Raksha[10]在硬件方面进行了诸多改动:

1)将包括寄存器、缓存和主存在内的所有存储单元都做了标记位扩展,以实现污点的标记,这也是所有基于硬件的污点分析工具的共同特点。

2)将指令集架构也进行了扩展,在实现常规功能的同时

支持污点标记的传播与检查操作,为了在硬件层面支持这一功能,Raksha 还设计了两类专用寄存器: 用于维护标记传播策略的标记传播寄存器和用于策略实施与检查的标记检测寄存器。

3) 为进行异常处理,系统还设计了引用监控寄存器、客户操作寄存器和异常处理寄存器,其中,异常处理寄存器主要用于保存污点标记传播过程中所发生的各类异常信息。

4) 为解决了不同型号处理器的兼容问题,系统专门设计了一个协处理器进行动态污点分析,而主处理器进行正常数据处理工作。

基于硬件的污点分析可以通过定制硬件的方式降低系统开销,提高污点分析执行效率,但由于运行在硬件层,不支持上层语义的逻辑安全策略,且需要对硬件结构进行重新设计,移植性较差。

2.2　基于软件的污点分析

基于软件的污点分析根据分析对象的不同还可细分为基于操作系统和基于应用程序的污点分析,其核心思想是将操作系统(如进程与线程、内存、文件等) 或应用程序(如数据库中的文件等) 中的相关资源进行标记,并在此基础上进行污点分析。基于软件的污点分析无需改变底层硬件,主要通过修改程序代码的方式实现污点信息的存储与传播,例如 TaintEraser [11] 、TaintDroid [12] 等。

Asbestos [13] 系统实现了基于操作系统的污点分析,主要基于最小特权原则和能够在系统进程通信方面提供了污点分析工具。Asbestos 系统的实现基于以下原则:

1) 最小特征原则,即向每一个用户或进程赋予其完成某种业务所需的最少特权;

2) 强制访问控制原则,即将系统中的信息区分密级进行管理,以确保用户只能访问被明确允许访问的信息。

基于 Asbestos 系统的 Web 服务器工作流程架构如图 2 所示,这是一个实现多个用户动态登录的网站,其中 ok-demux 模块分析传入的连接请求并将其发送到相关的工作进程,idd 模块用于检查用户密码,ok-dbproxy 模块提供数据库接口。在网站运行时,系统给每个用户提供专用权限,不同用户之间通过标签和事件进程进行隔离。Asbestos 系统通过对标签的动态调整实现污点分析的功能,但未考虑对系统关键资源的控制,且对进程间的系统资源共享和进程标记隐蔽通道等问题也没有提出解决方案。

基于软件的污点分析支持更高层级的语义逻辑安全策略,但使用插桩或代码重写会导致系统资源的巨大开销。

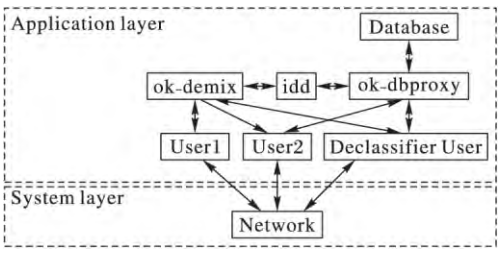

图 2　Asbestos 服务器工作流程示意图

Fig. 2　Schematic diagram of Asbestos server workflow

2.3　基于虚拟环境的污点分析

基于虚拟环境的污点分析的核心思想是在虚拟环境中增加污点分析模块,使污点分析模块和被分析的目标程序运行在不同环境中。其中,分析模块运行于虚拟机监控层,目标程序处于目标机操作系统层,分析模块不影响目标程序的运行,记录的污点信息更准确,如图 3 所示。

在基于虚拟环境执行污点分析的研究方面,研究者们实现了很多工具。如 Phosoher [14] 就是基于 Java 虚拟机环境的污点分析系统。Argos 通过 x86 模拟器在程序执行过程中跟踪网络数据,获取程序运行信息,使用位图映射标识内存及寄存器污点状态,保存当前环境为快照并提取特征码,能有效检测出针对服务器的攻击行为。动态分析平台 TEMU 在 QEMU 虚拟机的基础上进行了扩展,能够以较细的粒度分析内核与多进程间的交互情况。

图 3　基于虚拟环境的污点分析架构

Fig. 3　Taint analysis architecture based on virtual environment

基于虚拟环境的污点分析不需要修改程序的代码,由于虚拟机监控层的权限比目标机操作系统层更高,因此分析模块能够记录系统调用甚至内核执行的指令信息,可以发现系统底层漏洞,基于 Android 平台的污点分析工具一般采用基于虚拟环境的方式实现。但基于虚拟环境的污点分析需要搭设应用程序所需的工作环境,特别是在分析工控系统和物联网设备等与硬件环境联系紧密的固件程序时,虚拟环境的搭设工作十分复杂,且仅依靠基于虚拟环境的污点分析并不能保证操作系统的安全性,不能取代基于操作系统的污点分析的功能和作用。最后,基于虚拟环境的污点分析一般需要虚拟机解释执行的过程中执行污点分析,对系统运行效率影响较大。

2.4　基于代码的污点分析

基于代码的污点分析根据分析对象的不同也可以被细分为对程序源码进行污点分析和对二进制代码进行污点分析两种。基于代码的污点分析工具基本都采用了程序切片技术,使其对程序的理解和分析更为精确。程序切片是指能够影响程序某行的某个变量值的部分语句和判定表达式的集合。自 1979 年被提出后,程序切片技术已经发展出静态切片与动态切片、前向切片与后向切片以及一些变种技术等多种方法。Tripp 等 [15] 实现了 TAJ 工具,通过混合切片的方式对 Java Web 应用进行污点分析。

对程序源码进行污点分析是实现一个带有安全标签的类型系统,在编译过程中读取程序源码的安全标签,以检查是否存在违反信息流策略的情况。在第 1 章中已有所介绍,静态污点分析常采用这种方式。对程序源码进行污点分析能够在程序编译时发现程序中违反信息流策略的情况,对系统资源的开销较小,分析执行效率高,缺点是需要获得源代码,在商业环境下应用难度较高。

对二进制代码进行污点分析是在二进制代码中插入污点跟踪代码,以获取程序执行时的污点传播情况。动态污点分析常采用这种方式。对二进制代码进行污点分析可以在指令级别进行污点跟踪检测,精确度高,但频繁的插桩会占用系统资源,降低系统运行效率。此外,计算机的指令类型和规模使得二进制代码的精确建模极为困难,且二进制代码缺乏高层语义支持,一般需要依靠插桩平台或二进制反汇编工具以降低工作量。

此外，还有一些针对中间表示进行污点分析的研究，如梁彬等[16]实现了一种针对 JavaScript 优化编译执行模式的动态污点分析方法，在中间代码层面进行插桩操作，能够有效地在优化编译执行模式下进行污点跟踪。

综合本章内容，从实现基础、分析对象、可移植性、系统效率和静态/动态等五个方面对四种污点分析实现方式进行对比如表 2 所示。

对工程应用而言，研究者也会在一个工具中综合使用两种以上的实现方式，以便在不同层面进行污点分析，提高污点分析的准确性。如 FlexiTaint、PIFT[17]等工具就综合了基于硬件和基于软件两种污点分析思想，既尽可能少地改动硬件结构，又支持更高层级的语义逻辑安全策略。

表 2 污点分析技术实现方式对比

Tab. 2 Comparison of taint analysis′ implementing methods

实现方式		实现基础	分析对象	可移植性	系统效率	静态/动态
基于硬件		系统硬件	内存、寄存器等	需要定制硬件	高	动态
基于软件	操作系统	操作系统及内核	进程、文件、套接字等	中	中	动态
	应用程序	应用程序及支撑软件	应用程序数据结构或文件	高、中	中、低	动态
基于虚拟环境		虚拟机及支撑环境	虚拟机系统	需要配置虚拟环境	低	动态
基于代码	程序源码	编译器及支撑环境	程序变量、对象	高	中	静态/动态
	二进制代码	插桩工具	寄存器、内存、缓存等	高	低	静态/动态

3 相关应用

污点分析技术自提出后就在隐私数据泄露检测、逆向分析、漏洞探测等方面得到广泛应用，本章重点介绍隐私数据泄露检测和漏洞探测。

3.1 隐私数据泄露检测

污点分析技术用于隐私数据泄露检测的核心思想是：首先将系统或应用程序中的隐私数据标记为污点源，将对外输出数据的 API 标记为污点汇聚点，然后在程序运行过程中根据污点传播规则跟踪被标记为污点的隐私数据的传播路径，当系统调用 API 向外发送数据时检测其中是否包含污点标记，从而判定是否发生隐私数据泄露。比如污点分析工具 AndroidLeaks[18]通过自动化分析 Android 框架代码来获取 API 与权限的对应关系并确定污点源列表，从而大规模检测应用程序中存在的隐私数据泄露行为。

当前，移动设备用户数量已经超过了传统互联网用户，其中 Android 平台设备占据了主要份额，因此近年来，针对 Android 平台隐私数据泄露检测技术的研究已经成为热点，本文重点介绍污点分析技术在 Android 平台上的应用。Android 平台架构分为四层，自下而上分别为 Linux 内核层、系统运行层、应用程序框架层和应用程序层。由于 Android 平台采用基于组件的编程模型，根据污点传播涉及的范围不同，可分为组件内、组件间以及组件与库函数之间三类污点传播方式，如图 4 所示。

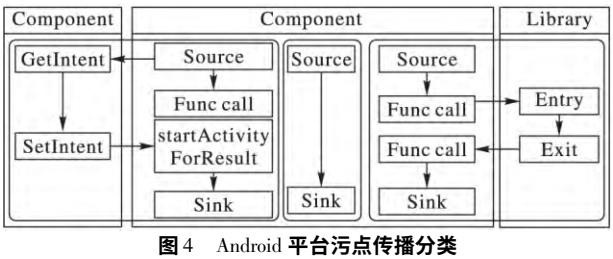

图 4 Android 平台污点传播分类

Fig. 4 Classification of taint spread on Android platform

静态污点分析和动态污点分析对这三类传播的处理方式也有所不同。采用静态污点分析技术进行隐私数据泄露检测时，首先需要构建完整的分析模型，以确保可以覆盖所有的可执行路径。静态分析框架 Dalysis[19]和自动化检测工具 CHEX 采用增量方法构建系统调用图，对目标 App 进行全局数据流分析。另一种常用污点分析工具 FlowDroid[20]则针对 Android 系统的异步工作机制对分析模型进行了大量优化。其次，静态污点分析通过分析 Intent 参数信息来完成组件间的污点传播分析。Octeau 等[21]提出了一个组件间通信机制寻址方案，将组件间污点传播转化为 IDE 问题，再进行求解。最后，静态污点分析还需要考虑组件与库函数之间的污点传播问题，由于库函数数量庞大且本地库多为 C/C++ 语言编写，因此针对这一问题没有统一的解决方案。如 FlowDroid 采用保守策略，即若调用库函数的参数中含有污点数据，则将其函数返回值标记为污点。

采用动态污点分析技术进行隐私数据泄露时最常用的工具是 TaintDroid，通过分析 Dalvik 虚拟机 DEX 指令类型并根据制定的污点传播规则来跟踪组件内的污点传播情况，并且 TaintDroid 重新设计了用于存储数据的包对象结构，使之能够存储与污点相关的信息，这样污点信息就可以在组件间进行传播。最后，TaintDroid 也采用了类似于保守策略的方法来处理组件与库函数之间的污点传播。与 TaintDroid 相似的是，刘阳等[22]通过对框架层源码进行插桩实现添加和提取污点标记，并在此基础上实现了检测隐私数据泄露的方法。

表 3 中列举了几种基于 Android 平台的常用污点分析工具的优缺点及分析性能的对比分析。从表中可以看到：FlowDroid 可以对组件内和组件间的污点传播进行分析，但在组件与库函数之间默认采用了保守策略，因此误报率较高；DroidMark 在其基础上引入了深度学习技术，大幅降低了误报率和漏报率，但实际性能与深度学习模型训练效果有关；TaintDroid[23]只对数据流进行污点分析，漏报率相对较高，但其系统开销增加不大，且能实现多个粒度级别的污点分析，实用性较强。

3.2 漏洞探测

除了隐私数据泄露检测外，污点分析技术还被应用于检测缓冲区溢出、SQL 注入、XSS 攻击、格式字符串攻击等方面。这类攻击行为的共同特点是用户的输入利用了系统中的某个漏洞，可以不经合法授权就改写系统数据敏感区的值，从而达到攻击的目的。污点分析技术被用于漏洞探测的核心思想是：将外部输入数据标记为污点源，将敏感数据区标记为污点汇聚点，然后在程序运行过程中根据污点传播规则跟踪污点数据的传播路径，检查敏感数据区是否被污点数据污染，从而判定系统或应用程序是否存在漏洞。

在污点分析提出以来，该领域一直都是研究者们重点关注的对象。达小文等[24]通过分析大量缓冲区溢出实例，归纳

了六种缓冲区溢出的漏洞定位模型,并结合补丁比对和污点传播分析生成污点传播路径图,最后将补丁源码的污点传播路径图与定位模型相匹配以精确定位漏洞所在位置; InterTracer [25] 利用静态污点分析方法来检测 Android 应用中的 Intent 注入漏洞; DexterJS [26] 平台可在字符粒度级别检测 Web 应用中基于 DOM 的 XSS 漏洞。

本节主要从污点数据流向的角度介绍了污点分析技术在隐私数据泄露检测和漏洞探测领域的应用。除了上述两种典型应用,污点分析技术在其他领域也得到广泛的应用,例如污点分析工具 Renove [27] 能够实现病毒关键代码的自动化提取,可用于病毒的智能脱壳; Eudaemon [28] 结合了动态污点分析技术和蜜罐与终端入侵检测技术检测 0day 漏洞。

表 3　基于 Andoid 平台的常用污点分析工具对比

Tab. 3　Comparison of common taint analysis tools on Android platform

名称	静态/动态	优点	缺点	误报率/%	漏报率/%
FlowDroid	静态	1. 针对 Android 的应用生命周期和异步工作机制进行了大量优化; 2. 采用具有上下文敏感和对象敏感的按需算法,提高分析准确性	1. 需要手工设置组件与库函数之间的污点传播规则; 2. 没有手工设定的组件与库函数之间的污点传播采用保守策略,会产生误报	14.26	7.08
DroidMark	静态	在 FlowDroid 的基础上引入深度学习,提高了分析精确度	需要训练贝叶斯网络,并收集样本集	3.23	6.20
TaintDroid	动态	1. 同时跟踪多个敏感数据源; 2. 支持 4 种粒度(变量级、消息级、方法级、文件级)的污点分析	1. 只分析数据流,不分析控制流; 2. 系统总开销增加不超过 14%	11.31	15.95
TaintART	动态	1. 针对新的 Java 运行时环境进行污点分析; 2. 可实现多级污点标记分析	1. 只分析数据流,不分析控制流; 2. 系统总开销不超过 15%	11.14	15.06

4　研究趋势

污点分析技术虽然取得了丰硕的研究成果,但也有很多问题并没有得到完全解决。一是现有工具仅基于某一层次进行污点分析。比如基于应用程序的污点分析虽然可以检测程序运行中存在的隐私数据泄露问题,但隐私数据在数据库或系统进程中同样存在泄露隐患。二是动态污点分析精度与效率之间的矛盾不可调和。动态污点分析是在原系统基础上扩展了污点跟踪机制,需要占用系统资源,导致系统效率下降,分析精度要求越高,需要获得的污点分析数据就要越详细,系统效率下降就越严重。所有的动态污点分析工具都需要根据应用环境的要求在精度与效率之间进行平衡。

因此,污点分析技术未来研究的出发点就是如何提高污点分析技术的性能。总的来说,主要包括以下两个方面。

4.1　优化污点分析技术方法

继续优化污点分析技术的算法,通过设计更科学的污点识别方法、归纳更准确的污点传播策略以及引入新算法等方法来提高污点分析技术的准确性和分析效率。

4.1.1　提高污点数据识别的精度

污点源的识别不当很容易造成污点标签在程序中的错误扩散,继而影响污点分析技术的准确性和分析效率。为了解决这一问题,有研究者引入机器学习的概念来识别污点源和污点汇聚点,如 Tripp 等 [29] 使用 Bayesian 模型对污点源进行分类,从中排除正常的数据发送请求,降低了对污点分析造成的影响; 而 DroidMark 则采用了深度学习技术,将对 Android 设备中恶意软件的检测精度提升至 96.87%。

4.1.2　制定更科学的污点传播策略

污点传播策略的选择与污点分析的精度和效率关系密切,比如有的工具(如 IPSSA)在处理污点传播时采用了保守策略,将所有与污染数据有关的操作均认定为符合污染传播规则,扩大了污点属性的标记范围,降低了模型的精确度; 有的细粒度污点分析工具在执行粗粒度分析时,占用了过多的系统资源,导致分析效率不高。

因此在工程实现时研究者们往往结合分析需要制定科学的污点传播方法。例如在函数级污点的传播过程中采用函数摘要思想,使污点在函数外层快速传播,避免了函数内部指令插桩等系统开销较大的操作,既保证了分析准确性又提高了系统效率。马金鑫等 [30] 还提出了基于执行踪迹离线索引的方法,在污点分析过程中,筛选出与污点数据相关的操作的同时忽略与污点数据无关的指令,从而缩短污点分析时间。

4.1.3　引入新算法提高污点分析性能

一方面是将污点分析问题转换为其他问题,如: 静态污点分析框架 Parfait,在程序预处理阶段将污点分析问题转换为图可达性问题,实现了针对 C 代码的静态污点分析; MalScope 将污点分析问题转换为加权下推自动机的广义下推后继问题,利用污点数据在程序中的可达性,减少后续分析中需要精确执行的路径数,可有效解决静态污点分析方法在进行路径敏感分析时面临的路径爆炸问题; DROIDPF [31] 应用软件模型检查技术验证 Android 应用,并采用了一种 Android 特有的还原技术来缓解状态空间爆炸。另一方面是采用并行处理技术进行污点分析,提高污点分析执行速度,如: 动态污点分析框架 DTAM [32] 可并行执行多个线程单元的污点分析,再将分析结果进行聚合,提高了对可并行执行的多线程程序进行动态污点分析的效率; PAGDTA [33] 采用虚拟化技术将各测试用例并行执行动态污点分析,大幅度提高了路径覆盖率和执行效率。

4.2　结合其他研究方法

为提高污点分析技术的精度和效率,还有研究者将不同的技术方法结合使用,以综合各方法的优点,其中关于静态污点分析和动态污点分析结合的研究工作在第 1 章已经进行过介绍,本节主要介绍以下两种方法。

4.2.1　与符号执行结合使用

符号执行(Symbolic Execution, SE)的核心思想是使用符号值替代具体数据作为程序输入,并在程序执行过程中以符号值进行计算,用包含符号的表达式表示程序变量的值和最终的输出结果。将污点分析与符号执行结合使用的分析工具有 Angr [34] 和 DsVD [35] 等。符号执行的精度比污点分析技术高,可以在程序运行过程中精确记录路径上所有的约束条件,有助于提高路径可达性判定的准确性; 但缺点是用于二进制程序分析时识别变量较为困难,会产生空间爆炸问题,需要强大的计

算能力支持。为了解决这一问题,研究者们提出了混合符号执行技术,这种技术结合了符号执行和具体执行,当符号执行无法判定或计算取值时,用具体值来代替部分符号值,以使符号执行能够继续进行。

此外,陈力波等[36]提出了基于类型的动态污点分析技术,其核心思想是将类型信息和符号值作为污点信息,在程序执行时同时进行动态污点分析和符号执行,程序运行结束后获得污点传播信息和程序执行路径条件,最后通过求解得到输入变量满足的约束条件。这种技术结合了动态污点分析和符号执行的优点,对程序有较为全面的理解。朱正欣等[37]基于动态插桩平台Pin以及libdft提出了动态符号化污点分析方法,其核心思想是将污点信息及污点检查策略符号化,并分析污点数据在传播过程中是否违反污点检查策略,进而判断程序在动态执行过程中是否存在非法数据操作。还有BitBlaze[38]将动态分析与静态分析、混合符号执行与具体执行、系统仿真与二进制重写等多种分析方法相结合,对二进制代码进行分析。

4.2.2 与模糊测试结合使用

模糊测试的核心思想是生成大量测试用例来测试程序鲁棒性,以保证较高的代码覆盖率,但缺点是对程序漏洞作用机理缺乏基本的理解。为了解决这个问题,研究者们提出了定向模糊测试[39]方法,将模糊测试和污点分析相结合,在异常测试样本集的基础上进行污点分析,深入了解漏洞作用机理,并在此基础上实现了SmartDroid[40]、Brahmastra[41]等工具。此外,Ganesh等[42]实现了自动化白盒模糊测试工具BuzzFuzz,其核心思想是使用动态污点分析技术追踪输入数据中影响程序关键数据区的范围,能够发现隐藏在程序中深层错误,适合复杂结构的输入文件测试。Wang等[43]实现了自动化模糊测试系统TaintScope,将动态污点分析和符号执行相结合,提升了对具有校验机制的程序进行模糊测试的效率。

除了污点分析技术性能以外,如何扩大污点分析技术的应用范围也是研究的热点领域。特别是随着近年来工控系统安全受到重视,污点分析技术也被引入工控安全的研究领域,例如LUCON[44]在工业物联网环境下采用动态污点分析,检测是否存在信息泄露隐患。

5 结语

本文比较系统地介绍了污点分析的理论基础、实现方法和典型应用,并分析了其中的优点和不足,但对具体的工具设计和功能实现并没有作深入阐述,还需要参阅相关文献以详加了解。污点分析技术已经被广泛应用于信息安全的各个领域,但其研究趋势仍是提高分析精度和效率以及扩大应用领域两个方面。本文对目前提高分析精度和效率的主要做法进行了介绍,在扩大应用领域方面,还有研究者正将污点分析技术应用于云计算、物联网、工业控制领域的安全研究中,这也是污点分析技术研究的下一个热点领域。

参考文献(References)

[1] EVANS D. The Internet of things: how the next evolution of the Internet is changing everything [EB/OL]. [2018-12-03]. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[2] 吴泽智,陈性元,杨智,等. 信息流控制研究进展[J]. 软件学报,2017,28(1):135-159. (WU Z Z, CHEN X Y, YANG Z, et al. Survey on information flow control[J]. Journal of Software, 2017, 28(1): 135-159.)

[3] 王蕾,李丰,李炼,等. 污点分析技术的原理和实践应用[J]. 软件学报,2017,28(4):860-882. (WANG L, LI F, LI L, et al. Principle and practice of taint analysis[J]. Journal of Software, 2017, 28(4): 860-882.)

[4] CLAUSE J, LI W, ORSO A. Dytan: a generic dynamic taint analysis framework [C]// Proceedings of the 2007 International Symposium on Software Testing and Analysis. New York: ACM, 2007: 196-206.

[5] LIVSHITS V B, LAM M S. Finding security vulnerabilities in Java applications with static analysis [C]// Proceedings of the 14th Conference on USENIX Security Symposium. Berkeley, CA: USENIX Association, 2005, 14: 271-286.

[6] TRIPP O, PISTOIA M, COUSOT P, et al. ANDROMEDA: accurate and scalable security analysis of Web applications [C]// Proceedings of the 2013 International Conference on Fundamental Approaches to Software Engineering, LNCS 7793. Berlin: Springer, 2013: 210-225.

[7] KINDER J, VEITH H. Jakstab: a static analysis platform for binaries [C]// Proceedings of the 2008 International Conference on Computer Aided Verification, LNCS 5123. Berlin: Springer, 2008: 423-427.

[8] XIA M, GONG L, LYU Y, et al. Effective real-time Android application auditing [C]// Proceedings of the 2015 IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Society, 2015: 899-914.

[9] 黄强,曾庆凯. 基于信息流策略的污点传播分析及动态验证[J]. 软件学报,2011,22(9):2036-2048. (HUANG Q, ZENG Q K. Taint propagation analysis and dynamic verification with information flow policy [J]. Journal of Software, 2011, 22(9): 2036-2048.)

[10] DALTON M, KANNAN H, KOZYRAKIS C. Raksha: a flexible information flow architecture for software security [C]// Proceedings of the 34th International Symposium on Computer Architecture. New York: ACM, 2007: 482-493.

[11] ZHU D (Y), JUNG J, SONG D, et al. TaintEraser: protecting sensitive data leaks using application-level taint tracking [J]. ACM SIDOPS Operating Systems Review, 2011, 45(1): 142-154.

[12] ENCK W, GILBERT P, CHUN B-G, et al. TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones [J]. Communications of the ACM, 2014, 57(3): 99-106.

[13] EFSTATHOPOULOS P, KROHN E, VANDEBOGART S, et al. Labels and event processes in the Asbestos operating system [J]. ACM SIGOPS Operating Systems Review, 2005, 39(5): 17-30.

[14] BELL J, KAISER G. Phosphor: illuminating dynamic data flow in commodity JVMs [J]. ACM SIGPLAN Notices, 2014, 49(10): 83-101.

[15] TRIPP O, PISTOIA M, FINK S J, et al. TAJ: effective taint analysis of Web applications [J]. ACM SIGPLAN Notices, 2009, 44(6): 87-97.

[16] 梁彬,龚伟刚,游伟,等. JavaScript优化编译执行模式下的动态污点分析技术[J]. 清华大学学报:自然科学版,2017(9):932-938. (LIANG B, HONG W G, YOU W, et al. DTA technique for JavaScript optimizing compilation mode [J]. Journal of Tsinghua University (Science and Technology), 2017, 57(9): 932-938.)

[17] YOON M, SALAJEGHEH N, CHEN Y, et al. PIFT: predictive Information-flow tracking [J]. ACM SIGARCH Computer Architecture News, 2016, 44(2): 713-725.

[18] GIBLER C, CRUSSELL J, ERICKSON J, et al. AndroidLeaks: automatically detecting potential privacy leaks in Android applications on a large scale [C]// Proceedings of the 2012 International Conference on Trust & Trustworthy Computing, LNCS 7344. Ber-

lin: Springer, 2012: 291 – 307.

[19] LU L, LI Z C, WU Z Y, et al. CHEX: statically vetting Android apps for component hijacking vulnerabilities [C]// Proceedings of the 2012 ACM Conference on Computer & Communications Security. New York: ACM, 2012: 229 – 240.

[20] ARZT S, RASTHOFER S, FRITZ C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps [J]. ACM SIGPLAN Notices, 2014, 49(6): 259 – 269.

[21] OCTEAU D, MCDANIEL P, JHA S, et al. Effective inter-component communication mapping in Android with Epicc: an essential step towards holistic security analysis [C]// Proceedings of the 22nd USENIX Conference on Security. Berkeley, CA: USENIX Association, 2013: 543 – 558.

[22] 刘阳,俞研. 基于动态污点分析的 Android 隐私泄露检测方法[J]. 计算机应用与软件,2017(9): 142 – 146. (LIU Y, YU Y. Detection of Android privacy leak based on dynamic taint analysis [J]. Computer Applications and Software, 2017, 34(9): 142 – 146.)

[23] RATHI D, JINDAL R. DroidMark: a tool for android malware detection using taint analysis and Bayesian network [J]. International Journal on Recent and Innovation Trends in Computing and Communication. 2018, 6(5): 71 – 76.

[24] 达小文,毛俐旻,吴明杰,等. 一种基于补丁比对和静态污点分析的漏洞定位技术研究[J]. 信息网络安全,2017(9): 5 – 9. (DA X W, MAO L M, WU M J, et al. Research on a vulnerability locaction technology based on patch matching and static taint analysis [J]. Netinfo Security, 2017(9): 5 – 9.)

[25] 王允超,魏强,武泽慧. 基于静态污点分析的 Android 应用 Intent 注入漏洞检测方法[J]. 计算机科学,2016,43(9): 192 – 196. (WANG Y C, WEI Q, WU Z H. Approach of Android applications Intent injection vulnerability detection based on static taint analysis [J]. Computer Science, 2016, 43(9): 192 – 196.)

[26] PARAMESHWARAN I, BUDIANTO E, SHINDE S, et al. DexterJS: robust testing platform for DOM-based XSS vulnerabilities [C]// Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2015: 946 – 949.

[27] KANG M G, POOSANKAM P, YIN H. Renovo: a hidden code extractor for packed executables [C]// Proceedings of the 2007 ACM Workshop on Recuring Malcode. New York: ACM. 2007: 46 – 53.

[28] PORTOKALIDIS G, BOS H. Eudaemon: involuntary and on-demand emulation against zero-day exploits [C]// Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems. New York: ACM, 2008: 287 – 299.

[29] TRIPP O, RUBIN J. A Bayesian approach to privacy enforcement in smartphones [C]// Proceedings of the 23rd USENIX Conference on Security Symposium. Berkeley, CA: USENIX Association, 2014: 175 – 190.

[30] 马金鑫,李舟军,张涛,等. 基于执行踪迹离线索引的污点分析方法研究[J]. 软件学报,2017,28(9): 2388 – 2401. (MA J X, LI Z J, ZHAO T, et al. Taint analysis method based on offline indices of instruction trace [J]. Journal of Software, 2017, 28(9): 2388 – 2401.)

[31] BAI G, YE Q, WU Y, et al. Towards model checking Android applications [J]. IEEE Transactions on Software Engineering, 2018, 44(6): 595 – 612.

[32] GANAI M, LEE D, GUPTA A. DTAM: dynamic taint analysis of multi-threaded programs for relevancy [C]// Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. New York: ACM, 2012: No. 46.

[33] 董国良,臧洌,李航,等. 一种应用于动态污点分析的路径自动生成方法[J]. 计算机与现代化,2017(7): 32 – 37,41. (DONG G L, ZANG L, LI H, et al. A path automatic generation method for dynamic taint analysis [J]. Computer and Modernization, 2017(7): 32 – 37, 41.)

[34] SHOSHITAISHVILI Y, WANG R, SALLS C, et al. SOK: (state of) the art of war: offensive techniques in binary analysis [C]// Proceeddings of the 2016 IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Society, 2016: 138 – 157.

[35] WANG Z, TANG Z, ZHOU K, et al. DsVD: an effective low-overhead dynamic software vulnerability discoverer [C]// Proceedings of the 10th International Symposium on Autonomous Decentralized Systems. Washington, DC: IEEE Computer Society, 2011: 372 – 377.

[36] 诸葛建伟,陈力波,田繁,等. 基于类型的动态污点分析技术研究[J]. 清华大学学报(自然科学版),2012,52(10): 1320 – 1334. (ZHUGE J W, CHEN L B, TIAN F, et al. Research of technology for type-based dynamic taint analysis [J]. Journal of Tsinghua University (Science and Technology), 2012, 52(10): 1320 – 1334.)

[37] 朱正欣,曾凡平,黄心依. 二进制程序的动态符号化污点分析[J]. 计算机科学,2016,43(2): 155 – 158,187. (ZHU Z X, ZENG F P, HUANG X Y. Dynamic symbolic taint analysis of binary programs [J]. Computer Science, 2016, 43(2): 155 – 158, 187.)

[38] SONG D, BRUMLEY D, YIN H, et al. BitBlaze: a new approach to computer security via binary analysis [C]// Proceedings of the 2008 International Conference on Information Systems Security, LNCS 5352. Berlin: Springer, 2008: 1 – 25.

[39] 张玉清,方喆君,王凯,等. Android 安全漏洞挖掘技术综述[J]. 计算机研究与发展,2015,52(10): 2167 – 2177. (ZHANG Y Q, FANG Z J, WANG K, et al. Survey of Android vulnerability detection [J]. Journal of Computer Research and Development, 2015, 52(10): 2167 – 2177.)

[40] ZHENG C, ZHU S, DAI S, et al. SmartDroid: an automatic system for revealing UI-based trigger conditions in Android applications[C]// Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. New York: ACM, 2012: 93 – 104.

[41] BHORASKAR R, HAN S, JEON J, et al. Brahmastra: driving Apps to test the security of third-party components [C]// Proceedings of the 23nd USENIX Security Symposium. Berkeley, CA: USENIX Association 2014: 1021 – 1036.

[42] GANESH V, LEEK T, RINARD M. Taint-based directed whitebox fuzzing [C]// Proceedings of the 31st IEEE International Conference on Software Engineering. Washington, DC: IEEE Computer Society, 2009: 474 – 484.

[43] WANG T, WEI T, GU G, et al. Checksum-aware fuzzing combined with dynamic taint analysis and symbolic execution [J]. ACM Transactions on Information & System Security, 2011, 14(2): No. 15.

[44] SCHÜTTE J, BROST G S. LUCON: data flow control for message-based IoT systems [J]. arXiv E-print, 2018: arXiv: 1805.05887.

REN Yuzhu, born in 1985, M. S. candidate. His research interests include reverse engineering, vulnerability mining.

ZHANG Youwei, born in 1975, M. S., professor. His research interests include reverse engineering, electronic forensics, data restoration.

AI Chengwei, born in 1990, M. S. candidate. His research interests include reverse engineering, vulnerability mining.