基于路径驱动的多路径分析算法

张 平 李清宝 崔 晨
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 针对动态二进制程序分析方法存在路径覆盖不全的问题,提出了基于路径驱动的多路径分析算法。其核心思想是在可控的模拟调试环境中动态执行被分析的二进制程序,通过修改CPU程序计数器PC值,驱动程序执行在当前输入条件下无法访问的程序路径,实现对多条程序路径的访问。基于该算法,设计并实现了一个基于路径驱动的多路径二进制程序分析系统。测试结果表明,该算法能够较全面地发掘程序执行路径,有效提高了分析代码的覆盖率。

关键词 路径驱动,动态分析,二进制程序,模拟调试环境
中图法分类号 P309.5 **文献标识码** A

Exploring Multiple Execution Paths Based on Execution Path Driven

ZHANG Ping LI Qing-bao CUI Chen
(Department of Computer Science and Technology, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract To solve the problem in dynamic analysis for binary program that not all program execution paths can be explored, a algorithm based on execution path driven was presented. The main idea of the algorithm is to run the program in a controllable simulation instrument environment, and drive it to execute the program paths that can't be executed under current input set by modifying the value of program counter(PC), so that multiple execution paths can be explored. Based on this algorithm, a prototype system of dynamic analysis for binary based on execution path driven was designed and implemented. Experiments results illustrate that the algorithm is effective in exploring execution path from binary program.

Keywords Path driven, Dynamic analysis, Binary program, Simulation instrument environment

1 引言

商业软件或第三方软件通常以二进制可执行程序的形式发布,从软件工程或软件系统安全的角度出发,对这些二进制程序进行逆向分析非常重要。目前,针对二进制代码进行逆向分析主要有静态分析和动态分析两大类方法,两者的根本区别在于是否需要执行二进制代码。静态分析方法在不执行程序的情况下以反汇编技术为基础对程序进行分析,其优点是可以较为全面地分析程序代码,不会受到用户输入、系统进程调度等外部事件的影响。但由于静态分析很难解决间接转移等问题,因此还需要大量的人工干预。虽然有很多研究致力于解决间接指令问题,但目前仍有一些问题无法解决[1]。另外,对于一些经过变形和混淆等反逆向分析技术处理的代码也很难处理[2]。动态分析方法在真实的目标机器或者虚拟机上实际执行代码,利用程序运行时的数据提高分析的效率和准确性,可以有效解决间接转移问题,同时避免了反静态分析技术带来的困难。但是,动态分析由于一次只能执行程序的一个实例,无法发掘程序的所有执行路径,因此存在代码覆盖率和扩展性问题[3]。

为了解决动态分析路径执行不全的问题,人们开展了多路径分析技术的研究。为了提高动态分析的代码覆盖率,需要在动态执行过程中使程序尽可能遍历所有可能的执行路径,即进行多路径分析[4]。多路径分析是全面分析可执行程序的重要方法之一,采用的主要方法有:(1)手工调试:在可控环境中执行待分析程序,采用调试方法,手动修改控制转移语句的判断条件,使程序执行不同路径。这种方法工作量大、易出错且不具全面性。(2)Fuzz方法:通过自动构造不同的输入集,触发程序执行不同路径。这种方法在一定程度上提高了分析的效率和覆盖率,但是大多数情况下,仍不能穷举所有的输入,遍历所有程序路径。(3)转移条件求解:采用符号执行的方法,建立转移路径的条件约束表达式,通过对路径条件的求解,发掘程序的多条执行路径。这种方法避免了Fuzz方法的随机性,但是由于存在约束条件无法求解和路径爆炸等问题,仍不能很好地解决代码覆盖率问题[4]。

本文针对动态分析方法的路径覆盖率问题,借鉴手工调试和转移路径求解多路径分析的基本思想,提出了一种通过修改PC值实现路径驱动的多路径分析方法。该方法在可控的模拟调试环境中运行被分析程序,根据程序执行状态,通过

到稿日期:2012-04-05 返修日期:2012-07-27 本文受国防型号项目资助。
张 平(1969—),女,博士,副教授,主要研究方向为软件逆向分析、信息系统安全,E-mail:zhangping69@gmail.com;李清宝(1967—),男,博士,教授,主要研究方向为计算机安全;崔 晨(1986—),女,硕士生,主要研究方向为软件逆向分析。

自动修改 CPU 程序计数器(PC)的值驱动程序执行在当前输入下无法执行的程序路径。这种方法可以有效发掘程序的多条执行路径,提高动态分析方法的代码覆盖率。

本文的主要贡献在于:(1)分析了多路径分析过程中所要解决的主要问题,从逆向分析的目标出发,提出了基于自动路径驱动的多路径分析方法,通过系统地修改 PC 值,触发程序执行多条执行路径;(2)设计实现了一个基于路径驱动的多路径二进制程序分析原型系统,实际测试表明,其可以有效提高动态分析的覆盖率。

2 问题描述

2.1 指令分类

在动态执行过程中,程序会根据输入的数据执行部分程序代码,即访问部分程序路径。不同类型的指令对程序的控制流有不同的影响。为了进行多路径分析,将处理器指令分为两大类[5]:

(1)转移指令(Transfer Instructions):能够将程序的控制流转移到在内存中不同于下一条指令地址的某个地址的指令,包括:

- 无条件转移:把控制流转移到目标地址。
- 条件转移:如果条件为真,把控制流转移到目标地址,否则把控制转移到在指令序列中的下一条指令。
- 变址转移:把控制流转移到一些目标地址中的某一个。
- 子程序调用:把控制流转移到被调用的子程序。
- 子程序返回:用返回指令把控制流转移到调用子程序的子程序。
- 程序终点:程序结束。

(2)非转移指令(Non Transfer Instructions):把控制转移到指令序列中下一条指令的指令集合,即所有不属于 TI 集合的指令。

可见,只有转移类指令才会改变程序的执行路径,产生多条分支路径,因此也称其为分支指令。程序在分支指令处有多条分支路径。通常,在一次动态执行过程中,只有部分分支路径被执行,而分支指令的每次执行只有一条分支被触发。因此,分支指令是多路径分析所关注的对象,即要解决如何发掘分支指令的多条分支路径的问题。

2.2 处理器编程模型

二进制程序 P 的逆向分析要依据相应的处理器信息,包含指令集、寄存器组、存储模型、中断机制等。寄存器和存储器反映程序执行状态,指令集定义了处理器的输入和状态转换关系,我们将这些信息定义为处理器编程模型。

定义 1[6] 处理器的编程模型为 $M=(I,Q,F,q_0)$,其中,

I 为 M 的有穷输入集合,即指令集;

$Q=$(pc,reg,mem)为处理器的有穷状态集合,由程序计数器(pc)、通用寄存器(reg)、存储器(mem)这 3 个部分组成;

F 为 $I\times Q\to Q$,M 的状态转移函数,表示计算机在输入 $inst(inst\in I)$ 时发生的状态迁移;

q_0 为 M 的初始状态。

所以,二进制程序 P 的逆向分析就是给定一个处理器模型 M,对 P 进行分析的过程。其中,控制转移指令决定程序的执行流程,其目标地址决定控制流的去向。

2.3 路径驱动思想

在目前通用的冯·诺依曼体系结构下,二进制代码中既包含指令,也包含数据,因此逆向分析的目标是区分指令和数据,识别代码中所有指令,遍历所有有效执行路径。基于路径驱动的多路径分析方法的核心思想是在程序动态执行过程中,针对每一条分支指令通过修改程序计数器 PC 的值,强制驱动程序执行不同的分支路径,以达到全面分析程序的目的。

路径驱动是手工对可执行程序进行跟踪调试分析常采用的一种方法,在被分析程序中不存在反跟踪代码的情况下,可以达到良好的分析效果。在程序正常执行过程中,修改 PC 值,改变程序的执行路径,可能会改变程序的执行状态,使程序的执行结果不正确。但是因为多路径分析关心的并不是程序执行的结果,而是程序的控制结构和代码,所以希望尽可能发掘程序的执行路径,全面分析程序代码。而通常情况下(不存在反跟踪代码),程序控制流信息是程序的固有的内在属性,即程序路径结构不依赖于程序的特定输入,所以路径驱动不会影响逆向分析的正确性。

在基于路径驱动的多路径分析过程中,被分析的二进制代码被置于一个可控的执行环境中,在某一有效输入集下启动程序执行;监控每一条指令,识别分支指令,记录程序的当前执行状态,保存未被执行的分支路径,为该路径的驱动执行保存信息;在以后发掘程序未执行路径时,返回分支点,恢复程序状态,通过修改程序计数器 PC 的值,触发程序执行在当前输入集下未执行的路径。这种方法可以发掘分支指令的多条执行路径,提高多路径分析的覆盖率。

3 基于路径驱动的多路径分析算法

为了实现多路径驱动,将被分析程序置于一个可控的执行环境中。该环境为程序提供执行环境,并调度、监控程序的执行,这里采用课题组设计和实现的二进制代码模拟调试环境。

在程序执行过程中,转移指令改变程序的执行路径;而条件转移指令产生新的路径,每个条件转移形成两个分支,产生两条路径。在程序正常执行的情况下,每个分支处只有一个分支会被执行,因此路径驱动重点关注的是条件转移指令的处理。

基于路径驱动的多路径分析算法如算法 1 所示。

算法 1 基于路径驱动的多路径分析算法

```
Input:二进制代码 P,输入集 I
Load(p);
  PC=Start(P,I);
  S=NULL;//S 为栈结构,保存分支指令和程序状态
  While(true)
  {
    While(i=GetInsreuction(PC)是结束指令)
    {
      If(i! 是结束指令)
      {
        If(i 是条件分支指令)
        {
          If(i 的两条分支都未被执行)
          {
            Push(PC 和程序执行状态);
            标记将被执行分支;
          }
```

```
        If(i 的两条分支都已被执行)
          Break;
        }
      }
      执行指令 i;
      PC=next(PC);
    }
    If(S! =NULL)
    {
      PC=POP(S);
      修改 PC 值;
      恢复程序执行状态;
    }
    Else break;
}
```

多路径分析的目标是发掘程序的所有执行路径，从而可以分析程序的全部代码，构建控制流图，进行数据流分析等，进而可以实现如安全分析、设计思想分析等各种程序分析工作。算法 1 只给出了路径驱动的多路径分析过程，省略了其它的分析过程。

3.1 算法相关问题

基于路径驱动的多路径分析在模拟调试环境中运行程序，通过修改 CPU 程序计数器，驱动程序执行所有分支路径。分析的过程实际上是深度优先搜索遍历程序的过程，在实现过程中必须解决以下几个问题。

3.1.1 执行状态管理

基于路径驱动多路径分析算法的核心就是在分支指令处通过修改 PC 值触发程序执行在特定输入下未能执行的路径。因为在正常程序中，大部分指令的目标地址独立于中间程序状态，所以通过路径驱动，可以发掘程序的所有执行路径。路径驱动需要正确保存程序的执行状态，因为间接目标地址通常存储在寄存器或内存单元中。如果不保存，当执行一条分支路径时，这些内容将被覆盖。当驱动执行另一条路径时，获取的目标地址可能不正确。所以，在分支指令处需要保存程序执行状态，当第一次遇到分支指令时，保存程序的执行状态；当需要驱动程序执行目标路径时，恢复程序执行状态，修改 PC 值，触发目标路径执行。实现有效的目标路径驱动需要保存和恢复的程序执行状态包括：CPU 寄存器值和程序使用的内存单元内容。在模拟调试环境中提供了对程序状态数据的保存机制，保存的状态数据称为快照，并提供了相应的机制对快照内容进行存取。

3.1.2 搜索路径记录

基于多路径分析的目标，所有指令须被执行且仅须被执行一次，但由于路径驱动的需要，有些指令可能被访问多次。为了保证所有指令仅被执行一次，对在动态执行过程中起关键作用的条件转移指令设置了访问标志 *visited*，*visited* 被初始化为 0，表示转移指令的两条分支路径都未被执行；在特定输入条件下，满足条件的分支已被执行，则令 *visited*=1；若分支路径都已被执行，则令 *visited*=2。

3.1.3 搜索策略

基于路径驱动的多路径分析算法采用深度优先搜索策略遍历程序路径，在某一输入条件下，程序在模拟调试环境下执行，这是一种自然的深度优先顺序；当自然路径执行结束，通过修改 PC 值驱动执行未访问路径时，同样采用深度优先策略。为实现深度优先策略，定义了一个栈结构 S 来保存程序状态信息。在程序执行的每一个路径分支点，如果 *visited*=0，则保存程序状态信息，将状态信息入栈；如果程序执行结束，而状态栈非空，则从栈顶取出并恢复程序状态，驱动程序执行未执行的路径。

4 基于路径驱动的多路径分析框架

基于模拟调试环境，设计了基于路径驱动的多路径分析框架，其结构如图 1 所示。

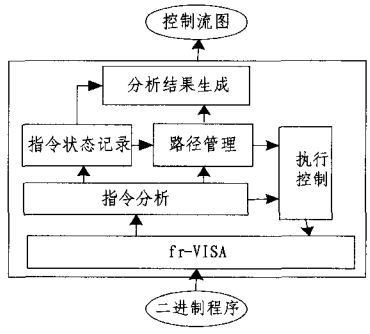

图 1 基于路径驱动的多路径分析框架

基于路径驱动的多路径分析框架建立在动态模拟调试环境 fr-VISA 之上，以二进制程序为输入，遍历程序执行路径。这里我们输出程序控制流图和汇编语言代码，主要包括：指令分析、指令状态记录、路径管理、执行控制、分析结果生成几个模块。其中模拟调试环境 fr-VISA 负责二进制程序的加载，模拟程序指令的执行和程序执行状态的获取与设置，向用户提供了加载、启动、运行、暂停、设置断点等命令[7]。

指令分析模块从模拟调试环境 fr-VISA 中获取当前执行指令和程序执行状态，分析指令类型，根据指令类型调用路径管理模块生成路径驱动信息。如果是条件控制转移指令，则调用路径管理模块对路径分析进行处理，保存或恢复程序执行状态，通知执行控制模块，调度指令在模拟调试环境中执行。

指令状态记录模块动态维护一个已分析指令列表，列表中每一条记录对应一条指令，保存的信息包括指令地址、指令汇编代码、指令执行状态标志 *visited*（主要针对条件转移指令）。

visited=0 表示条件转移指令的两条路径都未被发掘；

visited=1 表示条件转移指令的一条路径未被发掘；

visited=2 表示条件分支指令的两条路径都已被发掘。

该模块提供访问指令列表的接口，可以方便插入记录和读取记录，并为汇编程序的生成提供支持。

路径管理模块负责记录和维护路径驱动信息。该模块中维护一个栈结构，用来保存用于路径驱动的分支指令和状态信息。在下列情况下路径管理模块被调用：

① 遇到条件转移指令时，如果该指令第一次被执行，即其两条分支路径都未被执行时（*visited*=0），在栈结构中记录路径驱动信息。

② 当程序执行结束时，判断路径栈是否为空，如果栈不为空，出栈栈顶元素，将路径驱动信息提交执行控制模块，恢复相应条件转移指令的程序执行状态，触发该路径的执行。

（下转第 185 页）

neering,1996,22(2):158-159
[4] Matsumoto M, Futatsugi K. Highly Reliable Component-Based Software Development by Using Algebraic Behavioral Specification[C]//Proceedings of 3rd IEEE International Conference on Formal Engineering Methods,2000. IEEE Press,2000:35-43
[5] Eriksson H-E,Penker M. UML Toolkit [Z]. Publisher by John Wiley & Sons,Inc. 1997
[6] Küster J. Filipe:A logic-based formalization for component specification[J]. Journal of Object Technology,2002,1(3):231-248
[7] 梅宏,陈锋,冯耀东,等. ABC:基于体系结构、面向构件的软件开发方法[J]. 软件学报,2003,14(4):721-732
[8] 周晓峰. 基于语义的软件构件匹配方法及在水利领域中应用的研究[D]. 南京:河海大学,2006
[9] Cervantes H, Hall R S. Automating Service Dependency Management in a Service-Oriented Component Model[C]// Proceedings of the 6th Workshop of the European Software Engineering Conference/Foundations of Software Engineering Component Based Software Engineering. September 2003:379-382
[10] Cervantes H,Hall R S. Autonomous Adaptation to Dynamic Availability Using a Service Oriented Component Model[C]// Proceedings of the 26th International Conference on Software Engineering. ICSE2004:614-623
[11] 孟凡超,初佃辉,战德臣,等. 基于服务规约匹配的业务构件获取[J]. 哈尔滨工业大学学报,2010,42(7):1112-1116

(上接第 147 页)

栈中所保存的路径驱动信息包括以下内容:

① 条件转移指令,包括指令地址和代码;

② 指令执行状态,包括 CPU 寄存器值和程序使用的内存单元内容。

执行控制模块根据指令分析的结果和路径管理模块保存的信息调度模拟调试环境执行相应的指令,向模拟调试环境提供待执行的指令信息和程序状态信息,触发模拟仿真环境执行指令。

分析结果生成模块综合记录和保存的程序信息生成分析结果报告,这里给出被分析程序的控制流图和汇编程序。

5 测试与分析

为了测试本文所提算法的有效性和实用性,实现了基于路径驱动的多路径二进制程序分析原型系统 D-CFG,建立了测试环境。硬件环境为 Dell 计算机系统,CPU:Intel Pentium Dual E2200 2.19GHz;内存:2 GB,硬盘:160 GB;OS 系统为 Windows XP SP3,D-CFG 由 C++语言实现。

为了说明算法的效果,以生成的控制流图作为比较的对象,测试和比较的内容包括生成控制流图中的基本块数和边数。测试结果与采用传统的动态执行方法(不包含路径驱动)的结果以及 IDA Pro 的执行结果进行对比。IDA Pro 是目前逆向分析领域较为成熟和流行的反汇编器,它支持不同类型处理器和不同格式的可执行文件,采用递归遍历反汇编策略对可执行文件进行反汇编处理,从而提取目标代码的控制流图,并且它还提供了可扩展接口。插件(plug-in)机制可以提供应用分析、动态调试、文件格式支持、处理器支持的插件扩展功能[8]。

选取 SPEC2000 的 CINT2000 程序集中的几个典型程序,将编译后的二进制程序代码作为测试集。令 b_{D_CFG} 表示 D-CFG 识别的基本块,b_{D_OLD} 表示传统动态执行识别的基本块,$cb_{D\text{-}OLD}=\frac{b_{D\text{-}CFG}\cap b_{D\text{-}OLD}}{b_{D\text{-}OLD}}\times 100\%$ 表示 D-CFG 识别的基本块对传统动态执行识别基本块的包含度,b_{IDA} 表示 IDA 识别的基本块,$cb_{IDA}=\frac{b_{D\text{-}CFG}\cap b_{IDA}}{b_{IDA}}\times 100\%$ 表示 D-CFG 识别的基本块对 IDA 识别基本块的包含度;$e_{D\text{-}CFG}$ 表示 D-CFG 识别的边数,$e_{D\text{-}OLD}$ 表示传统动态执行识别的边数,$ce_{D\text{-}OLD}=\frac{e_{D\text{-}CFG}\cap e_{D\text{-}OLD}}{e_{D\text{-}OLD}}\times 100\%$ 表示 D-CFG 识别的边对传统动态执行识别边的包含度,e_{IDA} 表示 IDA 识别的边数,$ce_{IDA}=\frac{e_{D\text{-}CFG}\cap e_{IDA}}{e_{IDA}}\times 100\%$ 表示 D-CFG 识别的边对 IDA 识别边的包含度。表 1 给出 D-CFG、传统动态执行和 IDA 的测试结果。

表 1 测试结果

程序名称	$b_{D\text{-}CFG}$	$b_{D\text{-}OLD}$	cb_{D_OLD}	b_{IDA}	cb_{IDA}	$e_{D\text{-}CFG}$	$e_{D\text{-}OLD}$	ce_{D_OLD}	e_{IDA}	ce_{IDA}
mcf	1672	1367	100%	1583	100%	2061	1526	100%	1738	100%
bzip	1489	1245	100%	1435	100%	2166	1629	100%	1840	100%
gzip	1580	1250	100%	1557	100%	2283	1835	100%	2007	100%
twolf	6453	5800	100%	5128	100%	7528	6538	100%	7106	100%
parser	6328	5783	100%	6015	100%	7513	5001	100%	5968	100%
crafty	7137	6779	100%	6538	100%	12584	8932	100%	11175	100%

从测试结果可看出,D-CFG 采用路径驱动的多路径方法较好地解决了动态执行的覆盖率问题,识别路径的覆盖率远远高于传统的动态分析方法,也优于目前广泛应用的 IDA。

结束语 动态多路径分析方法的目标是提高分析的路径覆盖率。本文提出的基于自动路径驱动的多路径分析算法,通过在转移分支指令处修改程序计数器 PC 值,驱动程序执行不同路径,可以有效提高分析路径的覆盖率。但是,转移指令有多种情况,目前的方法还不能完全发掘所有的程序路径,完善间接调用和变址转移的处理是我们下一步的主要任务。

参 考 文 献

[1] Bai Li-li, Pang Jian-min, Zhang Ping. Analysing Indirect Table Based on Critical Semantic Subtree[C]// 2010 International Conference on Compuer Design and Application, ICCDA2010. Volume 1,2010:9-13
[2] Linn C,Debray S. Obfuscation of executable code to improve resistance to static disassembly[C]// Proceedings of the 10th ACM Conference Oll Computer and Communications Sacufity. Washington Dc,USA,2003:290-299
[3] Song D, Brumley D, Yin H, et al. BitBlaze:a new approach to computer security via binary analysis[C]// Proceedings of the 4th International Conference on Information Systems Security. Berlin:Springer,2008:1-25
[4] 王祥根,司端锋,冯登国,等. 基于代码覆盖的恶意代码多路径分析方法[J]. 电子学报,2009,37(4):701-705
[5] Cifuentes C. Reverse Compilation Techniques[D]. Queensland: Queensland University of Technology,1994
[6] 胡刚,张平,李清宝. 基于静态模拟的二进制控制流恢复算法[J]. 计算机工程,2011,37:276-281
[7] 胡刚. 固件程序代码逆向分析关键技术研究[D]. 郑州:解放军信息工程大学信息工程学院,2010
[8] Micallef S. IDA Plug-In Writing In C/C++ [EB/OL]. http://www.binarypool.com/idapluginwriting/,2009