文章编号:1001-9081(2011)09-2367-06 doi:10.3724/SP.J.1087.2011.02367

基于虚拟化技术的动态污点分析

陈衍铃[1],赵 静[2]
(1.电子工程学院 网络系,合肥 230037; 2.北京理工大学 计算机学院,北京 100083)
(cylhot@126.com)

摘 要:在现有的污点分析技术基础上,针对当前污点分析工具的记录不准确等缺陷,研究并实现了基于虚拟化技术的动态污点分析。结合虚拟化技术设计了动态污点分析框架,针对内存污点数据和硬盘污点数据分别设计了基于Hook技术的污点标记模型和Hash遍历的污点标记模型,依据Intel&AMD的指令编码格式对指令进行分类并依据指令类型设计污点传播策略,为解决信息记录冗余问题设计了基于指令筛选的污点记录策略。实验证明,该技术是有效的动态污点分析方法,可以很好地运用于模糊测试中的测试用例生成与漏洞检测。

关键词:模糊测试;虚拟化;污点分析;漏洞挖掘;信息安全
中图分类号:TP309.2 **文献标志码**:A

Dynamic taint analysis based on virtual technology

CHEN Yan-ling[1], ZHAO Jing[2]
(1. *Department of Network, Electrical Engineering Institute, Hefei Anhui* 230037, *China*;
2. *School of Computer Science and Technology, Beijing Institute of Technology, Beijing* 100083, *China*)

Abstract: The record of the current taint analysis tool is not accurate. To solve this, dynamic taint analysis based on the virtual technology was studied and implemented. A virtualization based dynamic taint analysis framework was designed, and two kinds of taint signature models based on Hook technology and Hash-traversal technology were given respectively for memory taint and hard disk taint. A taint propagation strategy was put forward according to the instruction type which was classified by instruction encoding format of Inter&AMD, and a taint record strategy based on instruction filtering was given to solve the problem of redundant information records. The experimental results prove that the proposed method is effective, and can be well used in test case generation and vulnerability detection of fuzzy test.

Key words: fuzzy test; virtualization; taint analysis; vulnerability discovery; information security

0 引言

动态污点分析技术是指在程序运行中实时标记跟踪和分析特定的数据。当前的污点分析技术运用范围很广泛,主要用来防范各种类型的攻击,包括:缓冲区溢出攻击、格式字符串攻击、SQL注入攻击、跨站脚本攻击等。

文献[1]介绍了防止缓冲区溢出的污点分析技术,该技术是指将所有来自于网络的数据标识为污点数据,接着实时监控程序中传播的污点数据,时刻防止污点数据影响到jump等跳转指令、格式字符串和系统函数参数。文献[2-3]介绍了基于硬件的污点分析技术。文献[4]介绍了应用污点分析技术防止SQL注入攻击。文献[5]介绍了COMET系统,该系统利用污点分析技术构造污点数据图来表示输入的污点数据和影响分支条件数据的关联性,利用这种关联性,在生成测试用例时特定变化可以影响程序分支条件的输入数据,从而提高测试用例的代码覆盖率。

污点分析技术还被应用于安全领域的其他各个方面,例如Renove[6]用于智能脱壳,它实现了自动提取病毒关键代码的功能;Panorama[7]是信息流跟踪分析工具,主要用来检测恶意行为;HookFinder[8]主要用于检测恶意软件的HOOK操作;MineSweeper[9]用于检测程序的隐蔽恶意行为。

当前实现污点分析的一些工具有DynamoRIO、Pin、Valgrind,这些工具存在以下几个缺点:

1)工具运行在用户态,只能记录用户态下的进程信息;

2)这些工具和被记录的程序共同运行在同一个操作系统中,因此有可能会影响到被记录的程序原来的流程;

3)一些污点记录工具还有可能需要修改被记录程序的源代码。

采用虚拟化技术实现的动态污点分析可以很好地克服这些缺点。采用虚拟化技术实现动态污点分析有如下几个优点:

1)由于虚拟化技术可以使污点分析模块和被监测的程序处于不同的运行环境,其中污点分析模块处于虚拟机监控器(Virtual Machine Monitor, VMM)层,被监控程序处于Guest OS层,所以污点分析模块不会影响到被监控的程序,使得记录的污点信息更加精确;

2)因为污点分析模块运行在VMM层,VMM层具有比Guest OS更高的权限,使得污点分析模块可以监控到系统底层的信息,而不局限于用户态,这样的污点分析模块可以发现驱动级的漏洞;

3)虚拟化技术实现的污点分析模块无需对被监控的程序的代码和行为进行预先的研究,更不需要修改程序的代码。

目前功能较好的虚拟化污点分析工具主要有BitBlaze中的TraceCap插件,但经过分析发现该插件有如下一些缺点:

1)由于插件没有开源,只有一些固定接口,难于集成到漏洞挖掘系统中,无法和测试用例的生成算法相结合;

收稿日期:2011-02-14;修回日期:2011-03-22。

作者简介:陈衍铃(1985-),男,福建龙岩人,硕士研究生,主要研究方向:网络安全; 赵静(1985-),男,安徽太和人,硕士研究生,主要研究方向:网络安全。

2)污点记录的数据过于繁杂,数据量大,不利于分析;

3)没有对记录的污点数据进行筛选,难于得到可用信息;

4)缺少实时分析的功能,不能及时发现异常。

针对以上问题,本文研究并实现了适合漏洞挖掘和分析的动态污点分析模块,主要具有以下特点:

1)污点标记准确,可以直接精确到污点输入数据的每一个字节,更好地指导测试用例生成和漏洞检测;

2)采用了合理的污点传播策略,去除了相关冗余的污点信息;

3)对记录的污点指令采用了合适的筛选策略,及时记录敏感的污点指令;

4)结合了指令匹配模型,及时发现不易发现的漏洞。

1 基于虚拟化的动态污点分析设计

虚拟化技术在计算机中的应用是指计算机元件在虚拟的环境中运行而不是在真实的硬件基础上运行。一般的虚拟环境主要由三个部分组成:硬件、VMM 和虚拟机。

当前在 x86 下的主要虚拟化技术有硬件虚拟化、全虚拟化、半虚拟化、操作系统级的虚拟化。

动态污点分析模块处于 VMM 层,包括污点标记子模块、污点传播子模块、污点信息记录子模块、指令特征子模块、漏洞特征数据库 5 个部分,共同对客户操作系统中的被测试程序进行监控和记录特定信息。

动态污点分析模块架构如图 1 所示。

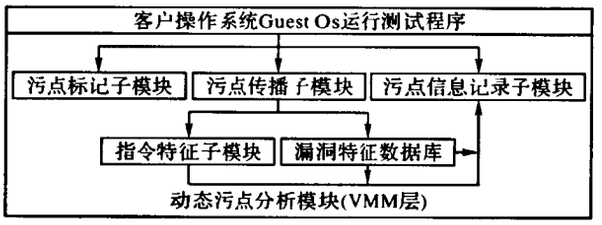

图 1 动态污点分析架构

污点标记子模块 该子模块负责对测试用例进行污点标记。模块实现了两种标记方法,分别是 HOOK 标记和磁盘数据标记。同时,污点标记子模块实现了合理的污点标记策略。

污点传播子模块 该子模块负责制定污点传播策略,并结合指令特征子模块和漏洞特征数据库,一起对传播指令进行匹配筛选并推送到污点信息记录子模块。

指令特征子模块 该子模块对汇编指令进行了一定的特征提取,为污点传播子模块提供特征数据。

漏洞特征数据库 该数据库提供常见漏洞的路径模型,结合污点传播子模块,对测试用例的污点执行路径进行比较评价,并将信息推送到污点信息记录子模块。

污点信息记录子模块 该子模块负责记录污点数据的各种信息,包括经过指令、执行路径、特定节点的执行环境等,同时接收指令特征子模块和漏洞特征子模块反馈的污点数据信息。

1.1 污点标记策略

为了进行污点传播,首先要对污点数据进行标记。污点数据是指来自外部设备的数据。本文定义的外部设备的数据主要有两种存在方式:第一种是被测试程序通过网络直接接收数据并将其保存在内存中;第二种是先将输入数据存放在硬盘上,然后由被测试程序进行读取。

通过网络接收到的污点数据,本文主要在内存中对其进行标记,设计了基于 HOOK 技术的污点标记模型。该污点标记模型主要根据操作系统的系统服务调用方式进行设计。

系统服务调用过程如图 2 所示。

污点传播主要涉及的接收污点数据的函数主要有 Receive 和 ReadFile 函数。以 ReadFile 函数为例,其系统服务调用过程为:

1)调用 ntdll.dll 中的 NtReadFile 函数;

2)Ntdll.dll 中的 NtReadFile 函数提供函数服务号;

3)系统根据函数服务号,遍历系统服务描述表(System Services Descriptor Table, SSDT),索引到 notskrnl.exe 中 NtReadFile 地址,并由 NtReadFile 实现 ReadFile 函数功能。

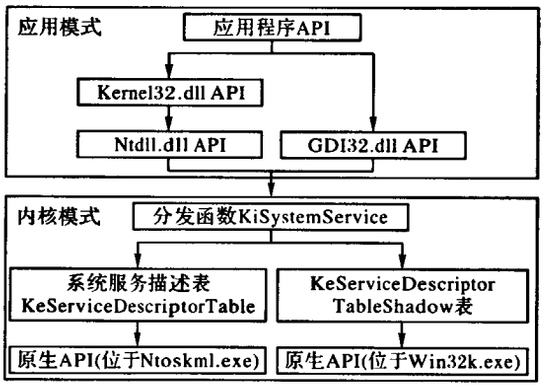

图 2 系统服务调用过程

依据系统服务调用过程,本文设计了内存污点标记模型。设计的主要思想是(以 ReadFile 函数为例):

1)利用 HOOK 模块,修改 SSDT 的 NtReadFile 的索引地址,将地址指向自定义的 MyReadFile 函数地址;

2)在 MyReadFile 函数中,首先调用 NtReadFile 函数实现正常的 ReadFile 功能,然后写入一段 Magic 特定的指令,并将 NtReadFile 函数参数中的内存地址写入寄存器;

3)在虚拟系统的指令回调函数中,当检测到一段特殊的 Magic 指令时,取出当前寄存器中存放的内存地址,并进行标记。

具体的内存污点标记模型如图 3 所示。

图 3 内存污点标记模型

针对硬盘中的污点数据,本文设计了 Hash 遍历的污点标记模型。该模型主要是为了定位硬盘中污点数据的物理位置。

Hash 标识主要分为三步:

1)将磁盘按照 512 B 进行分块,并进行扇区编号;

2)对每个 512 B 的数据计算 Hash 值;

3)将 Hash 值和对应的扇区编号组成磁盘数据摘要。

Hash 标识算法如图 4 所示。

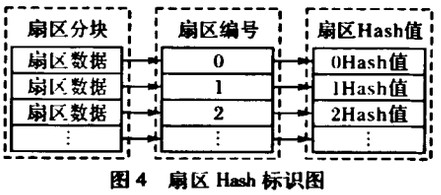

图 4 扇区 Hash 标识图

依据扇区的 Hash 数据表,对磁盘文件的标识采用如下算法:

1)读取要标识的文件,得到文件的全部数据,将数据按 512 B 进行分块。

2)取出所读取数据 512 B,计算 Hash 值。

3)依据计算的 Hash 值索引扇区 Hash 数据表,找到对应的扇区编号,确定该文件块的扇区位置,并记录位置信息。

4)查看是否已经读取数据到文件结尾,如果没有,则转到 2);如果已经读取到文件结尾,则转到 5)。

5)依据文件每 512 B 块的扇区位置进行污点标记。

文件污点标记算法如图 5 所示。

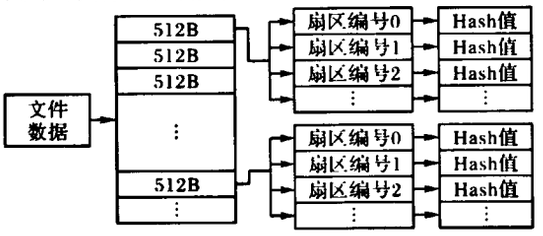

图 5　文件标记图

对污点数据的标记除了需要设计合理的标记算法以外,还需要定义合理的污点数据结构。内存、寄存器、网卡、磁盘等各种计算机存储设备的管理结构都不一样。

对内存数据的管理,采用位数组的方式。以一个 64 b 的无符号数来标记存储区域连续 64 B 的使用信息。例如,128 MB的物理内存,以 64 B 为单位划分,即可得到 128 × 1 024/64 = 2 048 个数组,数组的每个元素指向一个结构体,此结构体存放 64 B 的污染信息。

对磁盘数据的管理,也采用位数组结构来标记数据污点。由于磁盘存储容量较大,需要采用链表来动态管理。例如 4 GB大小的磁盘,以 64 B 为单位,可以分成 4 × 1 024 × 1 024/64 个块。对这么多块采用一个大小为 1 024 的数组来管理。数组的每一个元素指向一个链表,并动态增添位图结构。具体结构如图 6 所示。

硬盘数据每 64 B 分块,共同构成数据块链表(Disk Table)。链表中每个节点和位数组页(Entry Page)中的每一块相对应。物理内存(Physical Memory)中每 64 B 块同样对应于位数组页中每一个节点。

位数组页中每一个节点主要由两部分构成:位数组和污点结构体。位数组中每个字节由 0 和 1 标示是否是污点,同时每个字节对应一个污点结构体,该污点结构体包含要传播的污点信息。污点信息主要包括污点标记、偏移量(文件中的偏移量)、污点权重等信息。

1.2　污点传播策略

虚拟化的动态污点分析模块需要一个很好的虚拟化平台做支撑,在权衡多种虚拟化系统后,本文选择快速模拟器(Quick Emulator, QEMU)作为开发平台。

为了实现污点传播,本文在 QEMU 动态翻译指令时加入污点传播函数。以 JMP 指令为例,如果需要实现 JMP 指令的污点传播,则需要在 JMP 指令被切分成微指令的过程中加入污点传播的微指令。改进后的 JMP 指令微指令如下所示:

```
gen_op_movl_T0_im(selector);
gen_op_movl_T1_imu(offset);
  gen_op_movl_seg_T0_vm(offsetof(CPUX86State, segs[R_CS]));
gen_op_movl_T0_T1();
gen_op_jmp_T0();
gen_op_movl_T0_0();
gen_op_ taint();                    //添加的污点传播微指令
gen_op_exit_tb();
```

因为指令数量很多,分为多种类型,每一种指令污点传播的实现方式不一样,因此要实现精确的污点传播,就需要在指令被切分成微指令的过程中对指令进行分类,同一类的指令采用相同的污点传播策略。本文依据指令编码格式对指令进行分类。

QEMU 对指令的动态翻译是依据 Intel&AMD 的指令编码格式来对指令编码进行解析的。Intel&AMD 的指令编码格式如图 7 所示。

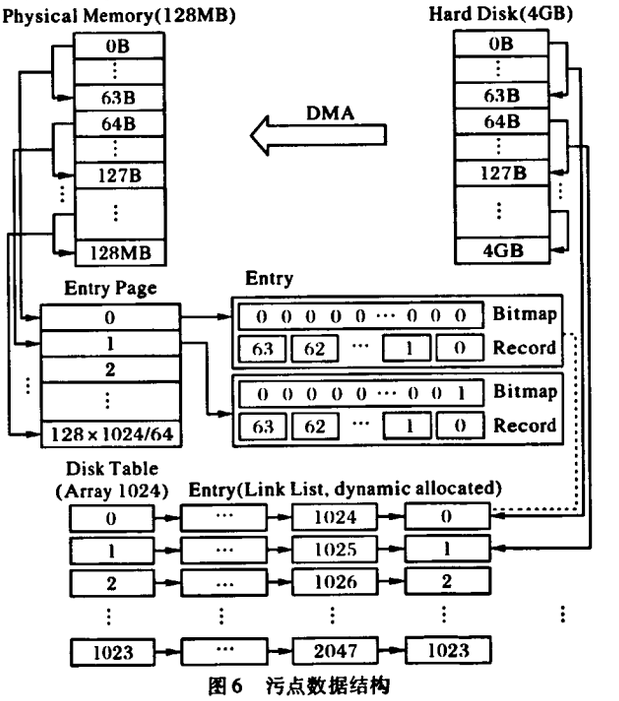

图 6　污点数据结构

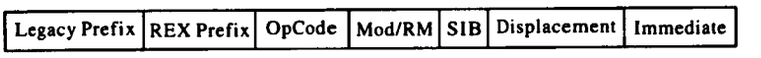

图 7　Intel&AMD 的指令编码格式

根据动态翻译的机制和指令编码,本文先将涉及到污点传播的指令,按指令功能划分为几类污点指令。每个污点指令通过指令编码中的操作码 Opcode 来标识。

1)将数据由寄存器传到内存中的指令。

Mov Ev,Gv:将 Gv 中的数据转移到 Ev 中。通过查询 OpCode 表得知该指令类型的 OpCode 是 0x89。目标操作数是 Ev,根据 OpCode 属性表得知 Ev 代表是寄存器或者是内存;源操作数是 Gv,根据 OpCode 属性表得知 Gv 代表是通用寄存器。

Push 寄存器的指令:Push 指令将寄存器数据压入堆栈,因此有可能进行由寄存器到内存的污点传播。通过查询 OpCode 表得知该类型指令的 OpCode 值是 0x50 到 0x57。当检测到 0x50 到 0x57 时,有可能进行寄存器到内存的污点数据传播。

2)将数据由内存传播到寄存器的指令。

Pop 指令:pop 指令将堆栈中的数据弹出到寄存器中,因此有可能将污点数据从内存中传播到寄存器中。该类型指令的 OpCode 值是 0x58 到 0x5f。当检测到 0x58 到 0x5f 值时,如果传播的是污点数据,就将数据由内存传播到寄存器中。

Mov Gv,Ev:该指令类型的 OpCode 是 0x8b 和 0x8a。源操作数是 Ev,根据 OpCode 属性表得知 Ev 代表是寄存器或者是内存;目的操作数是 Gv,根据 OpCode 属性表得知 Gv 代表是通用寄存器。因此在解析指令时,遇到 OpCode 是 0x8b,有可能进行污点数据由内存到寄存器的传播。

3)立即数传播到内存的指令。

Push ES 指令:将 ES 压入堆栈,OpCode 值为 0x06。

Push CS 指令:将 CS 压入堆栈,OpCode 值是 0x0e。

Push SS 指令:将 SS 压入堆栈,OpCode 值是 0x16。

Push DS 指令:将 DS 压入堆栈,OpCode 值是 0x1e。

Push FS 指令:将 FS 压入堆栈,OpCode 值是 0x1a0。

Push GS 指令:将 GS 压入堆栈,OpCode 值是 0x1a8。

Push IV 指令:将 IV 压入堆栈,OpCode 值是 0x68。

Mov Ev,Iv:该指令类型的 OpCode 是 0xc6 和 0xc7。源操作数是 Iv,根据 OpCode 属性表得知 Iv 代表是立即数;目的操作数是 Gv,根据 OpCode 属性表得知 Ev 代表是寄存器或者是内存。因此在解析指令时,遇到 OpCode 是 0xc6 或 0xc7,有可能进行污点数据由立即数到内存的传播。

Call Im:该指令类型的 OpCode 是 0xe8。调用一个地址也是将数据由立即数传播到内存中。

4)数据由寄存器传播到寄存器。

此类指令数量较多,这些指令污点的传播,可直接在虚拟系统中原有的微指令中实现。

5)立即数到寄存器的数据传播。

此类指令同样数目众多,需要在虚拟系统中原有的微指令中实现。

由于指令数量多,差异性大,较难把所有的指令的污点数据传播都通过同一种方式实现,因此指令的污点数据传播在虚拟系统中需要结合两种方式:第一种是在虚拟系统原有的微指令中实现污点传播;第二种需要在虚拟系统解析指令时加入污点传播的微指令。

在虚拟系统原有的微指令中实现污点数据传播主要是针对数据由寄存器传播到寄存器和数据由立即数传播到寄存器两种类型的指令。

实现的原理就是在微指令的 C 语言实现函数中直接加入污点传播的函数。以寄存器污点指令 Op_update2_cc 为例,改进后的 Op_update2_cc 函数如下:

```
Void Op_update2_cc()
{ CC_SRC = T1;
  CC_DST = T0;
  taint_reg();                                    //污点传播函数
}
```

针对数据由寄存器传播到寄存器的指令,实现的污点传播策略主要有以下几步:

1)检测源寄存器对应的污点位图是否被标记成污点;

2)如果源寄存器对应的污点位图被标记成污点,则将源寄存器中的污点数据结构中的污点信息复制到目标寄存器污点数据结构中,并把目标寄存器对应的位图标记成污点;

针对数据由立即数传播到寄存器的指令,如果立即数是污点数据则直接将寄存器对应的污点位图标记成污点。实现的原理是在对应的原虚拟机系统的微指令中加入污点传播的微指令。

在虚拟系统解析指令时加入污点传播的微指令,这种实现方式主要针对数据由寄存器传到内存中的指令、将数据由内存传播到寄存器的指令和立即数传播到内存的指令。

数据由寄存器传播到内存中的指令污点数据传播原理是:在虚拟系统解析指令时检查指令操作数 OpCode 值,在对应的指令解析代码中加入污点传播指令。其实现伪算法如下:

在指令解析函数中检查指令的 Opcode 值:

情况 1 寄存器到内存的污点传播。当操作数 OpCode 值是 0x89、0x50 到 0x57 中任意一个时,加入 Taint_Resgister_To_Memory()函数。

情况 2 内存传播到寄存器的污点传播。当操作数 OpCode 值是 0x8b、0x8a、0x58 到 0x5f 中任意一个时,加入 Taint_ Memory _To_ Resgister()函数。

情况 3 立即数传播到内存污点传播。当操作数 OpCode 值是 0x06、0x0e、0x16、0x1e、0x1a0、0x1a8、0x68、0xc6、0xc7、0xe8 中任意一个时,加入 Taint_ IM _To_ Memory()函数。

当检测到寄存器到内存的污点传播时,采用如下算法进行污点传播:

1)取出当前寄存器的位图组,查看当前的寄存器是否被标记成污点,如果被标记成污点进行 2),否则退出;

2)根据物理内存地址索引,依次对传播到的物理内存对应的位图组进行污点标记。

当检测到内存到寄存器的污点传播时采用的算法如下:

1)根据内存地址,查找对应的内存位图表中,判断该内存地址是否是污点,如果是污点数据则进行 2),否则退出;

2)对对应的寄存器污点位图组进行污点标记。

当检测到立即数到内存的污点传播时采用的算法如下:

1)查看立即数是否是污点数据,如果是则进行 2);

2)对对应的内存污点位图组进行污点标记。

1.3 结合指令筛选的污点信息记录策略

污点信息记录模块和指令筛选模块紧密结合,指令筛选贯穿于污点信息记录的全过程。污点信息记录要为后期的分析提供详细的程序执行上下文信息,因此需要定义合理的数据结构。污点信息记录模块定义的数据结构中主要包含以下信息:

1)传播污点数据的指令:这是污点信息记录核心部分,构成污点数据的指令流。

2)当前操作的污点数据信息:主要是指指令操作的污点数据结构里的信息,为更好地指导测试用例生成提供节点数据。

3)当前程序执行上下文信息:主要包括 EIP 等寄存器信息、内存信息等,为漏洞分析提供依据。

为了能够实时记录污点数据,需要将记录函数放入程序中每条指令的执行过程当中。由于虚拟系统是通过微指令来解析每一条汇编指令,因此在虚拟系统解析成微指令时,在真正的微指令前加入函数,即实现每条指令的回调函数,通过回调函数记录污点信息。

以 JMP 指令为例说明污点信息记录实现原理。加入回调函数的 JMP 微指令如下:

```
gen_insn_record_begin();                          //回调函数开始
  gen_op_movl_T0_im(selector);
  gen_op_movl_T1_imu(offset);
  gen_op_movl_seg_T0_vm(offsetof(CPUX86State, segs[R_CS]));
  gen_op_movl_T0_T1();
  gen_op_jmp_T0();
  gen_op_movl_T0_0();
  gen_op_exit_tb();
gen_insn_record_end();                            //回调函数结束
```

代码中 gen_insn_record_begin()负责信息记录,该微指令将在每条指令调用前被调用,因此可以在指令执行时就记录相关信息。

指令筛选策略主要为了在污点信息记录过程中,对指令进行筛选分析。首先根据 Intel&AMD 的指令编码格式设计了适合于大部分汇编指令的指令数据结构,主要是对数量众多的指令提取相应的指令特征,用于指令匹配。

指令特征结构包含以下信息:指令前四个字节的掩码;指令操作码码字节;指令中主操作数的长度;指令的特别位,主

要提供一些特殊指令的特性；指令中参数的类型（三个参数位）；指令的其他信息；指令的助记符字符串。

指令特征结构适用于几乎所有的指令。常见指令的指令特征如下所示。

```
{0 x0000FF, 0x000090, 1,00, NNN, NNN, NNN,
  C_CMD +0, "NOP" }
{0 x0000FE, 0x00008A, 1, WW, REG, MRG, NNN,
  C_CMD +0, "MOV" }
{0 x0000F8, 0x000050, 1,00, RCM, NNN, NNN,
  C_PSH +0, "PUSH" }
{0 x0000FE, 0x000088, 1, WW, MRG, REG, NNN,
  C_CMD +0, "MOV" }
{0 x0000FF, 0x0000E8, 1,00, JOW, NNN, NNN,
  C_CAL +0, "CALL" }
{0 x0000FF, 0x000075, 1, CC, JOB, NNN, NNN,
  C_JMC +0, "JNZ, JNE" }
{0 x0000FF, 0x0000EB, 1,00, JOB, NNN, NNN,
  C_JMP +0, "JMP" }
{0 x0000FF, 0x0000E9, 1,00, JOW, NNN, NNN,
  C_JMP +0, "JMP" }
```

基于上述指令的几个特征，模块首先需要多个包含特殊指令的特征库，该指令数据库只包含污点信息记录模块需要记录的指令特征，例如跳转指令特征库中只有 JMP、JE、JZ 等跳转指令的特征。如果污点信息记录模块只需要记录跳转指令的污点传播，则把当前匹配的指令数据库替换成跳转指令的特征数据库。

模块采用的指令匹配算法如下：

输入：i 表示输入指令，i_j 表示输入指令结构的第 j 个参数，c 表示特殊指令特征库，c_i 表示指令特征库中第 i 条记录，c_{ij} 表示指令特征库中第 i 个指令特征信息的第 j 个参数。

输出：如果特征数据库中有输入指令则程序返回匹配到的指令 c_i，否则返回 NULL。

初始化：c_i 从第一条记录开始。

1）检查 c_i 是否为空，如果为空则退出，不为空进入 2）；

2）检测 i_1 和 c_{i1} 异或的是否和 c_{i2} 相同，如果相同，程序返回 c_i，不同则转到 3）；

3）c_i 取下一条记录，返回 1）。

通过指令筛选模块，依据指令特征结构，可以定义多种指令特征数据库，使得模块只记录特定指令数据库中的指令，具有以下用途：

1）去除了大量不必要的污点指令。污点数据经过的指令很多，如果所有指令都记录将难于分析。

2）可以及时关注一些关键指令，及时发现漏洞。例如 CALL 指令，如果污点数据影响到了 CALL 指令，很有可能就能导致漏洞发生。

3）可以及时得到指令经过路径的关键信息。例如在污点数据传播过程中，如果想知道污点数据经过了哪些分支节点，并想做些针对分支节点的变化，就可以定义只含跳转指令的特征数据库，及时定位到污点数据经过的分支节点，并采取相应策略。

2　实验分析

用虚拟化技术实现动态污点分析，可以很好地记录输入的污点数据在程序运行中的传播情况，从而既可以为漏洞挖掘提供详细的测试用例的运行信息，也可以为漏洞分析与利用提供漏洞样本的执行路径等信息，提高漏洞挖掘和利用的效率。

本文以一个 VC 程序为例，验证本文实现的基于虚拟化的污点分析功能。运行的程序代码如下：

```
#include < stdio. h >
void main()
{ getchar();
  _asm
  { mov bl, al
    cmp bl, 0x35
    jg L1
    jmp L2
  L1:
      sub bl, 1
      sub bl, 1
      sub bl, 1
      sub bl, 1
  L2:
      mov al, 0
      mov bl, 0}
```

记录的污点数据如下所示：

```
7c871606:  rep movsl % ds: ( % esi), % es: ( % edi)  M@ 0x0025069c[ 0x000a0d39][ 4]( CR)  T1 {1 (9001, 0) ()()()}
    R@ ecx[ 0x00000000][ 4]( RCW)  T0  M@ 0x00422ae0[ 0x00000000][ 4]( CW)  T0
7c87160d:  rep movsb % ds: ( % esi), % es: ( % edi)  M@ 0x0025069c[ 0x00000039][ 1]( CR)  T1 {1 (9001, 0) ()()()}
    R@ ecx[ 0x00000003][ 4]( RCW)  T0  M@ 0x00422ae0[ 0x00000000][ 1]( CW)  T0
7c8018da:  cmpb  MYM0x1a, ( % eax)  I@ 0x00000000[ 0x0000001a][ 1]( R)  T0  M@ 0x00422ae0[ 0x00000039][ 1]( R)
    T1 {1 (9001, 0) ()()()}
40b670:  movsbl ( % ecx), % edx  M@ 0x00422ae0[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  R@ edx[ 0x00430ef0][ 4]( W)  T0
40b673:  cmp  MYM0xa, % edx  I@ 0x00000000[ 0x0000000a][ 1]( R)  T0  R@ edx[ 0x00000039][ 4]( R)  T1 {1 (9001, 0) ()()()}
40b6aa:  mov  0x8( % ebp), % edx  M@ 0x0012ff04[ 0x00000000][ 4]( R)  T0  R@ edx[ 0x00000039][ 4]( W)  T1 {1 (9001, 0) ()()()}
40b6f9:  movsbl ( % ecx), % edx  M@ 0x00422ae0[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  R@ edx[ 0x00422ae0][ 4]( W)  T0
40b6fc:  cmp  MYM0x1a, % edx  I@ 0x00000000[ 0x0000001a][ 1]( R)  T0  R@ edx[ 0x00000039][ 4]( R)  T1 {1 (9001, 0) ()()()}
40b759:  movsbl ( % eax), % ecx  M@ 0x00422ae0[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  R@ ecx[ 0x00422ae0][ 4]( W)  T0
40b75c:  cmp  MYM0xd, % ecx  I@ 0x00000000[ 0x0000000d][ 1]( R)  T0  R@ ecx[ 0x00000039][ 4]( R)  T1 {1 (9001, 0) ()()()}
40b761:  mov  -0xc( % ebp), % edx  M@ 0x0012fef0[ 0x00422ae0][ 4]( R)  T0  R@ edx[ 0x00000039][ 4]( W)
    T1 {1 (9001, 0) ()()()}
40b767:  mov  ( % eax), % cl  M@ 0x00422ae0[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  R@ cl[ 0x00000039][ 1]( W)
    T1 {1 (9001, 0) ()()()}
40b769:  mov  % cl, ( % edx)  R@ cl[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  M@ 0x00422ae0[ 0x00000039][ 1]( W)
    T1 {1 (9001, 0) ()()()}
40b6f6:  mov  -0x4( % ebp), % ecx  M@ 0x0012fef8[ 0x00422ae1][ 4]( R)  T0  R@ ecx[ 0x00000039][ 4]( W)  T1 {1 (9001, 0) ()()()}
40bc14:  movsbl ( % edx), % eax  M@ 0x00422ae0[ 0x00000039][ 1]( R)  T1 {1 (9001, 0) ()()()}  R@ eax[ 0x00421f00][ 4]( W)  T0
40bc17:  and  MYM0xff, % eax  I@ 0x00000000[ 0x000000ff][ 4]( R)  T0  R@ eax[ 0x00000039][ 4]( RW)  T1 {1 (9001, 0) ()()()}
40d75c:  mov  % eax, -0x4( % ebp)  R@ eax[ 0x00000039][ 4]( R)  T1 {1 (9001, 0) ()()()}  M@ 0x0012ff7c[ 0xcccccccc][ 4]( W)  T0
```

```
40d75f:   mov    %al, %bl   R@al[0x00000039][1](R)   T1 {1 (9001, 0) ()()()}   R@bl[0x00000000][1](W)   T0
40d761:   cmp    MYM0x35, %bl   I@0x00000000[0x00000035][1](R)   T0   R@bl[0x00000039][1](R)   T1 {1 (9001, 0) ()()()}
40d768:   sub    MYM0x1, %bl   I@0x00000000[0x00000001][1](R)   T0   R@bl[0x00000039][1](RW)   T1 {1 (9001, 0) ()()()}
40d76b:   sub    MYM0x1, %bl   I@0x00000000[0x00000001][1](R)   T0   R@bl[0x00000038][1](RW)   T1 {1 (9001, 0) ()()()}
40d76e:   sub    MYM0x1, %bl   I@0x00000000[0x00000001][1](R)   T0   R@bl[0x00000037][1](RW)   T1 {1 (9001, 0) ()()()}
40d771:   sub    MYM0x1, %bl   I@0x00000000[0x00000001][1](R)   T0   R@bl[0x00000036][1](RW)   T1 {1 (9001, 0) ()()()}
40d774:   mov    MYM0x0, %al   I@0x00000000[0x00000000][1](R)   T0   R@al[0x00000039][1](W)   T1 {1 (9001, 0) ()()()}
40d776:   mov    MYM0x0, %bl   I@0x00000000[0x00000000][1](R)   T0   R@bl[0x00000035][1](W)   T1 {1 (9001, 0) ()()()}
40d780:   call   0x0000000000401060   J@0x00000000[0xffff38e0][4](R)   T0   M@0x0012ff7c[0x00000039][4](W)
    T1 {1 (9001, 0) ()()()}
```

3 结语

本文实现了基于虚拟化的动态污点分析，在污点标记、污点传播策略和污点记录策略方面对现有的污点分析技术进行了改进。基于虚拟化的污点分析技术可以很好地运用于模糊测试中的测试用例生成和漏洞检测。在一般的漏洞挖掘工具中，经常用调试工具来判断系统是否异常或者直接查看系统是否出现错误。这种判定方式有一定的局限性，比如一个漏洞的触发，需要某个函数参数是个特定的值，而有时这个特定的数值出现具有很大的偶然性，因此可以通过动态污点分析对污点数据经过的路径进行分析，来判断该路径的危险性，从而提取有可能引发漏洞的路径。对危险程度高的污点执行路径可以结合人工分析，发现不易发现的漏洞。

今后，将在虚拟化的动态污点分析平台基础上，对污点路径分析和如何通过污点记录的信息指导测试用例生成两个方面进行深入研究，以提高模糊测试的测试效率。

参考文献：

[1] NEWSOME J, SONG D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software [D]. Pittsburgh: Carnegie Mellon University, School of Computer Science, 2005.

[2] SUH G E, LEE J W, ZHANG D, *et al.* Secure program execution via dynamic information flow tracking [C]// ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2004: 85 - 96.

[3] KONG JINGFEI, ZOU C C, ZHOU HUIYANG. Improving software security via runtime instruction-level taint checking [C]// ASID '06: Proceedings of the 1st Workshop on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2006: 18 - 24.

[4] NGUYEN-TUONG A, GUARNIERI S, GREENE D, *et al.* Automatically hardening Web applications using precise tainting [J]. IFIP Advances in Information and Communication Technology, 2005, 181(10): 295 - 307.

[5] LEEK T R, BAKER G Z, BROWN R E, *et al.* Coverage maximization using dynamic taint tracing, TR-1112 [R]. Lexington, Massachusetts, US: MIT Lincoln Laboratory, 2007.

[6] KANG M G, POOSANKAM P, YIN H. Renovo: a hidden code extractor for packed executables [C]// WORM 07: Proceedings of the 2007 ACM Workshop on Recurring Malcode. New York: ACM, 2007: 46 - 53.

[7] YIN H, SONG D, EGELE M, *et al.* Panorama: capturing system-wide information flow for malware detection and analysis [C]// CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security. New York: ACM, 2007: 116 - 127.

[8] YIN H, LIANG Z, SONG D. HookFinder: identifying and understanding malware hooking behaviors [C]// NDSS 2008: Proceedings of the Network & Distributed System Security Symposium. San Diego, California: [s. n], 2008: 16 - 23.

[9] BRUMLEY D, HARTWIG C, LIANG Z, *et al.* Automatically identifying trigger-based behavior in malware [J]. Botnet Detection, 2008, 36(5): 65 - 88.

(上接第2351页)

本文在消息携带车辆选择过程中忽略了具体的消息传递方式选择算法，针对车辆到车辆和车辆到AP的通信相关算法进行改进是下一步研究的重点。

参考文献：

[1] ZHAO JING, CAO GUOHONG. VADD: vehicle-assisted data delivery in vehicular Ad Hoc networks [J]. IEEE Transactions on Vehicular Technology, 2008, 57(3): 1910 - 1922.

[2] JEONG J, GUO SHUO, GU YU, *et al.* TBD: trajectory-based data forwarding for light-traffic vehicular networks [C]// ICDCS '09: 29th International Conference on Distributed Computing Systems. Piscataway, NJ: IEEE, 2009: 231 - 238.

[3] SKORDYLIS A, TRIGONI N. Delay-bounded routing in vehicular Ad-Hoc networks [C]// MobiHoc 2008: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing. New York: ACM, 2008: 341 - 350.

[4] ZHANG YANG, ZHAO JING, CAO GUOHONG. On scheduling vehicle-roadside data access [C]// VANET '07: Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks. New York: ACM, 2007: 9 - 18.

[5] DING YONG, WANG CHEN, XIAO LI. A static-node assisted adaptive routing protocol in vehicular networks [C]// VANET '07: Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks. New York: ACM, 2007: 59 - 68.

[6] NANDAN A, DAS S, PAU G, *et al.* Co-operative downloading in vehicular Ad-Hoc wireless networks [C]// WONS 2005: The 2nd Annual Conference on Wireless Ondemand Network Systems and Services. Piscataway, NJ: IEEE, 2005: 32 - 41.

[7] 陈立家，江昊，吴静，等. 车用自组织网络传输控制研究[J]. 软件学报，2007，18(6)：1477 - 1490.

[8] 苏金树，胡乔林，赵宝康，等. 容延容断网络路由技术[J]. 软件学报，2010，21(1)：119 - 132.

[9] JEONG J, GUO SHUO, GU YU, *et al.* TSF: trajectory-based statistical forwarding for infrastructure-to-vehicle data delivery in vehicular networks [C]// ICDCS 2010: 2010 International Conference on Distributed Computing Systems. Piscataway, NJ: IEEE, 2010: 557 - 566.

[10] FIORE M, BARCELO-ORDINAS J M. Cooperative download in urban vehicular networks [C]// MASS'09. IEEE 6th International Conference on Mobile Ad Hoc and Sensor Systems. Piscataway, NJ: IEEE, 2009: 20 - 29.

[11] 于斌，孙斌，温暖，等. NS2与网络模拟[M]. 北京：人民邮电出版社，2007.

[12] JOHNSON D, MALTZ D. Dynamic source routing in Ad Hoc wireless networks [J]. Engineering and Computer Science, 1996, 353(10): 153 - 181.