文章编号:2095-6134(2017)05-0647-10

# 一种基于参数污点分析的软件行为模型*

尹芷仪[1],沈嘉荟[1†],郭晓博[1],查达仁[1,2]

(1 中国科学院信息工程研究所, 北京 100093; 2 信息安全国家重点实验室, 北京 100093)

(2016 年 7 月 3 日收稿; 2016 年 11 月 14 日收修改稿)

**Yin Z Y, Shen J H, Guo X B, et al. A software behavior model based on dynamic taint analysis[J]. Journal of University of Chinese Academy of Sciences, 2017,34(5):647-656.**

**摘 要** 基于细粒度二进制动态分析平台,提出通过系统调用参数的污点分析构建软件行为模型的方法。该方法主要在指令级别监控应用程序运行,跟踪系统调用参数的污点传播获取参数与参数、局部变量和外部数据之间的关联关系,进而抽取出参数的污点传播链。其次,基于参数污点传播链和系统调用序列构造能够同时反映控制流和数据流特性的软件动态行为模型。最后,分析和验证该模型具备检测隐秘的非控制流数据攻击的能力。

**关键词** 系统调用参数; 非控制数据; 虚拟机; 动态污点分析; 入侵检测

**中图分类号**:TP309 **文献标志码**:A **doi**:10.7523/j.issn.2095-6134.2017.05.016

# A software behavior model based on dynamic taint analysis

YIN Zhiyi[1], SHEN Jiahui[1], GUO Xiaobo[1], ZHA Daren[1,2]

(1 *Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093, *China*;
2 *State Key Laboratory of Information Security, Beijing* 100093, *China*)

**Abstract** Based on the fine-grained binary dynamic analysis platform, we propose a taint analysis method to construct the software behavior model using the system call arguments. Firstly, the method obtains the associations between the arguments, between an argument and a local variable, and between an argument and a foreign data through monitoring the applications running and tracking the taint propagation of system call arguments at the instruction level, and then the taint propagation chains between arguments are generated. Secondly, a software behavior model, which covers control-flow and data-flow, is built according to these chains and system call sequence. Finally, the experimental and analytical results demonstrate that this model can be used to detect stealthy non-control attacks.

**Keywords** system call arguments; non-control data; virtual machine; dynamic taint analysis; intrusion detection

系统调用是操作系统提供给应用程序的操作接口,是程序访问系统资源的关键途径,其调用情况在很大程度上可反映该程序的行为特征。因此,通过构造程序正常行为时的系统调

* 院部合作基金(AQ1703,AQ1708)资助

† 通信作者,E-mail:shenjiahui@iie.ac.cn

用表征模型,可以对软件行为进行有效地监控,并检测其可能产生的异常行为。目前,国内外基于系统调用的异常检测方法主要包括控制流分析和数据流分析。基于控制流的分析通过抽取程序运行时系统调用的实时信息进行监测,但忽略了对函数调用参数及其他有价值信息的分析,可能不能发现密钥泄露、浏览器恶意代码注入和服务器权限的改变等攻击行为。基于数据流的分析是在不改变程序控制流的前提下利用易被程序漏洞篡改的敏感数据,如配置参数、用户输入、认证数据、系统调用参数等,实施异常检测。但是简单地对系统调用参数进行建模[1-4]不能对同一参数的位置进行识别,同时基于系统调用参数而构造的异常检测器只能监控系统调用事件,不能对局部变量进行监控,这就割裂了系统调用参数与非系统调用代码和局部变量之间的联系(如图 1 所示),造成无法识别针对非控制流数据的攻击[5]。

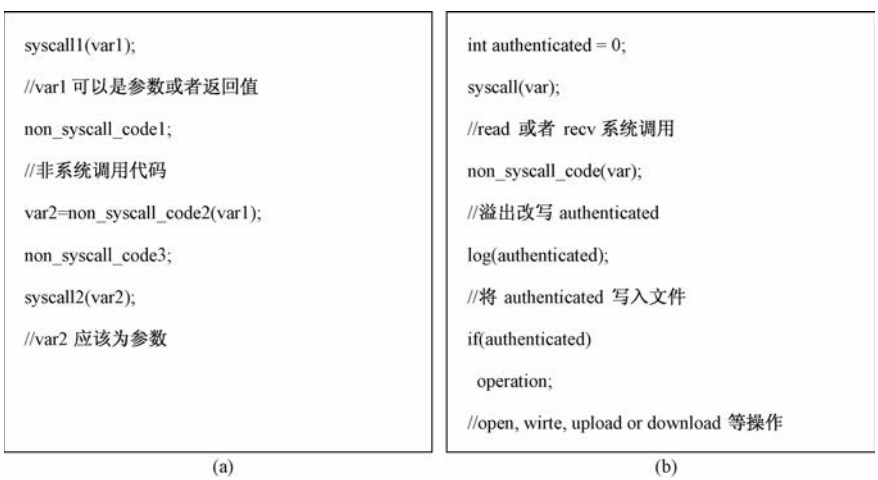


**图 1 系统调用参数分别与非系统调用、局部变量相关联的程序模式**

**Fig. 1 The associated program modes between system call arguments and the non-system call and between system call arguments with the local variable**

图 1(a)为系统调用参数与非系统调用代码相关联的程序模式。若 syscall1 的参数或者返回值 var1 被后续非系统调用代码(non_syscall_code)实施诸如 hash 变换、加密等操作之后,再值赋给 syscall2 的参数 var2,如此一来 var1 和 var2 的关系就被模糊化了,文献[6-7]中所述方法就无法获取 var1 和 var2 之间的数据流特性,从而无法检测篡改 var1 或者 var2 的攻击。

图 1(b)为系统调用参数与局部关键数据相关联的程序模式。若非系统调用代码(non_syscall_code)对系统调用 syscall 的参数或返回结果 var 进行复制或者字符串连接等操作,就会引起溢出,并篡改授权变量 authenticated,然后基于此操作通过验证,进行非授权的文件操作。文献[6-7]中的异常检测方法无法实现对上述程序片段中局部敏感变量被篡改的检测,因为忽略了系统调用参数与局部重要变量之间的关联。

鉴于上述问题,我们需要一种新的异常检测方法,该方法应该具备跟踪系统调用参数在程序内部运行的能力。动态污点分析正是这样一种技术,它能够给应用程序的输入数据添加一个新的类型修饰变量 tainted,然后在程序运行时追踪特定数据传播路径[8],目前已广泛应用于软件测试,调试和漏洞检测。TaintCheck[9] 利用动态污点方法自动检测和分析商品软件,追踪程序执行时污点数据的传播,及时发现污点数据的危险使用并对攻击自动标记。Chen 等[10] 使用动态污点分析有效地获取变量、检验点、路径约束和输入数据之间的关系。Ma 等[11] 基于动态污点分析提出的多标签污点数据方法突破了现有漏洞监测工具的效率瓶颈。除此之外,它还成功用于检测多种攻击,包括缓冲区溢出[12-13],格式化字符串攻击[14],SQL 注入攻击[15-16],跨站式脚本攻击[17]以及信息泄露攻击[18]等。现有动态污点分析主要对程序外部数据进行染色处理,如将网络、键盘输入作为污点源,继而根据外部数据在程序中的传播而反推出恶意代码的行为特征。

本文将动态污点技术用于分析程序内部的数据流(如系统调用参数等),目的是推导出系统调用的底层数据流特征(如参数间的关联关系、参

数与局部变量间的关联关系等),然后结合程序的控制流构造更准确、更严格、更具有针对性的安全策略规则下的软件行为模型,用于检测对针对常控制流数据和非控制流数据的攻击。

## 1　系统架构

本文基于 QEMU 建立二进制动态分析平台[19],在此基础上实现参数污点分析和软件的行为建模,系统框架如图 2 所示。从拦截系统调用到建立行为模型并用于异常检测需要经历 5 个步骤:

1)在指令级别实现系统调用中断和捕获系统调用细粒度信息;

2)对参数实施污点操作:一是为清白的参数设置唯一的污点标签;二是跟踪污点标签在不同内存区域或者寄存器之间的传播,观察与其他参数之间的交互,根据污点源传播规则决定目的参数的污点标签;

3)生成污点日志文件,在污点跟踪过程中记录下系统调用的基本信息(PC、系统调用号、系统调用名、参数个数、参数的类型等),参数信息(参数值、虚拟地址、类型、长度),以及污点信息(污点源、污点标签和污点传播方式);

4)根据污点日志文件构造 FSA 模型,该模型既能体现系统调用的控制流信息,又能体现系统调用参数之间、参数与局部变量间的污点传播关系,前 4 步为离线学习阶段;

5)实时异常检测,主要是根据实际的控制流和数据流与训练得到的 FSA 模型是否匹配来判断异常与否。

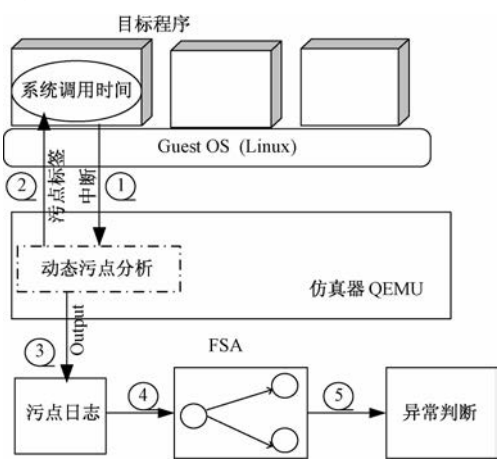


**图 2　系统框架图**

**Fig. 2　System framework**

## 2　系统调用参数动态污点分析

本节将具体阐述系统调用参数动态污点分析的实现过程。污点分析常见的步骤有:1)污点源的确定;2)污点传播规则的确定;3)跟踪污点传播。由于本文着重分析系统调用参数的传播轨迹,因此在确定污染源与污点传播跟踪时与以往的污点分析有所不同。

### 2.1　确定污点源

以往的污点分析[13,20]将外部输入数据作为污点源,而本文主要是利用污点分析技术捕获系统参数的传播轨迹,因此本文将系统调用参数作为一种新的污点源。由于程序运行时会产生多个系统调用事件,不同系统调用之间会发生直接或者间接的值传递关系,因此参数的污点标签也存在传播关系。参数是否为新的污点源需要视下面的情况而定:1)首个系统调用的全部参数均为新的污点源;2)首个系统调用的返回值(与参数无关)为新的污点源;3)后续所有系统调用的参数和返回值均要检查其所在内存或者寄存器位图的污点标记情况,若被标记则拷贝已有污点记录,否则视为新的污点源并打上标签。

### 2.2　确定污点传播规则

根据不同类型的系统调用函数定义污点传播规则如下:

1)若移数指令如 MOVE、PUSH、LOAD、STORE、POP 等的源数据带有污点标签,则目的数据即为污点数据;

2)若算术指令如 MUL、DIV 等的源数据带有污点标签,则运算结果即为污点数据;

3)常数指令如 xor、eax 等的运算结果均为清白数据;

4)若表查询指令需要访问的内存中已存在带有污点标签的数据,则该区域所有数据均为污点数据。

### 2.3　系统调用参数的污点跟踪

系统调用参数的动态污点跟踪分析流程为:

1)跟踪并分析 CPU 正在处理的指令。若 sysenter 为当前执行的指令,则针对来宾系统进行如下操作:中断被监测程序、捕获系统调用并解析相关信息;若当前指令的地址范围已经超出内核模式,为系统调用返回时刻,需要对返回值信息进行解析。

2)解析系统调用参数和返回值。当发生系

统调用时,解析按照读取系统调用号 - 系统调用基本信息 - 参数信息的顺序进行。系统调用号保存在寄存器 EAX 中,系统调用基本信息是根据系统调用号在系统调用初始化列表中获取。当系统调用退出时,需要从寄存器 EAX 获取返回值的相关信息。

3)参数污点标注。解析完参数的相关信息,然后根据参数的类型检测它对应的位图是否带有污点标记,并结合 2.1 和 2.2 节所说的情况标注参数污点。在完成解析和标注两步之后要生成污点日志,写入当前系统调用的基本信息和参数的污染信息。

4)污点传播跟踪。根据前面定义的污点传播规则,对 QEMU 的代码做修改,加入一些回调函数来实现在指令级别跟踪污点传播。触发这些回调函数的时刻包括:①基本块的进入和退出;②每条指令的进入和退出;③污点传播;④内存的读、写;⑤寄存器的读、写;⑥硬件或网络输入、输出。当系统调用的参数带有污点标签并参与 2.2 节中的指令时,则调用相应的回调函数进行处理,获取程序的参数与参数之间,参数与其他数据之间的污点传播轨迹。

## 3 行为模型的构造

有限状态自动机(finite state automaton, FSA)是一种可以用来描述程序控制流程的计算模型,具有捕获所有分支和循环结构、记录无穷多个任意长度序列、脱离库函数而进行程序行为轨迹学习的优势[20]。因此,本章将创新性提出构建基于自动机的软件行为模型。

**定义 1** 系统调用参数总体表示为 $argument[0,1,\cdots,parameter\_number]$,其中 $parameter\_number$ 表示参数个数,$argument[0]$ 表示返回值。每个参数所包含的信息如式(1)所示,从左到右依次为参数类型、虚拟地址、长度、取值和污点标记。

$$argument[i] = \{type, address, length, value, tainted\} \quad (0 \leqslant i \leqslant parameter\_number), \tag{1}$$

**定义 2** 程序 $P$ 的一个系统调用事件表示为 $S$,主要包括系统调用信息和定义 1 中的参数信息(如式(2)所示),$PC$ 和 $sys\_call\_number$、$sys\_call\_name$ 分别代表程序计数器值、系统调用号和系统调用名,其余为参数信息。

$$S(PC_k, sys\_call\_number) = \{PC_k, sys\_call\_number, sys\_call\_name, \{argument[0], \cdots, argument[parameter\_number]\}\} \quad (0 \leqslant k \leqslant sys\_call\_total\_number - 1). \tag{2}$$

**定义 3** 程序 $P$ 的系统调用集合称之为序列链,表示为 $SL$,如式(3)所示。

$$SL = \{ < (PC_i, sys\_call\_number), (PC_j, sys\_call\_number) > \} \quad 0 \leqslant i,j \leqslant sys\_call\_total\_number - 1. \tag{3}$$

**定义 4** 系统调用 $S$ 中参数的污点信息表示为 $TA$。当该参数为污点源时,其污点标记 $tainted = 0$,并含有 $origin$ 的唯一标签;当污点标记不为0时,视为被污染的参数,将复制污点源的污点传播方式和污染字节记录数,分别用 $taintPropag$ 和 $numRecords$ 表示,其中污点传播方式有读、修改、查询、修改查询、查询已修改污点源数据 5 种,代表的数字为0、10、20、11 和21。另外复制内容还涉及到污染字节记录,包括污染源 $source$、污染参数源 $origin$、污染数据缓冲偏移量 $offset$ 等污点记录信息。式(4)表示的是第 $k$ 个系统调用的第 $argNum$ 个参数的污染属性信息。

$$TA(PC_k, sys\_call\_number, argNum) = \{taintPropag, numRecords, \{source, origin, offset\}\} \quad (0 \leqslant argNum \leqslant parameter\_number, 0 \leqslant k \leqslant sys\_call\_total\_number - 1). \tag{4}$$

**定义 5** 参数污点传播链,即系统调用的集合,特点是该集合内的系统调用均具有相同污点标签,表示为 $TL$,公式(5)所示。

$$\begin{aligned} TL = \{&(sys\_call\_name, sys\_call\_number, PC_k, offset, argNum) \\ &= (sys\_call\_name, sys\_call\_number, PC_j, offset, argNum) \\ &\mid TA(PC_k, sys\_call\_number, argNum).origin \\ &= TA(PC_j, sys\_call\_number, argNum).origin\}. \end{aligned} \tag{5}$$

**定义 6** 污染边,由 $TL$ 中提取污染源参数和被污染参数的信息构成,即:源 $PC$,目标 $PC$,源参数号,目的参数号,污点传播方式,表示为 $TE$,如式(6)所示。

$TE = (source\_PC, target\_PC, source\_argNum, target\_argNum, taintPropag)$. (6)

**定义 7**　程序 $P$ 的执行轨迹 $T(P)$,是与其相关的所有系统调用事件和参数污染属性的集合,即 $T(P) = \{S \cup \{TA\}\}$。

# 4　行为模型构造和异常检测算法

## 4.1　行为模型构造算法

本文基于有限自动机构造出能反映程序控制流和数据流的算法如下。

算法:模型构造算法

输入: $Q$——$PC$ 有序集合 {S}——系统调用事件集合
$SL$—— 系统调用序列集合 $TL$—— 污染传播链 $\{TE\}$ ——污染边集合
输出:XML 格式的 FSA

```
void createFSA( )
{
    State_FSA ⇐ {PC in Q};   /* PC 作为节点 */
    Transition_FSA ⇐ Φ ;   /* 状态转移初始为空集 */
  for( < (PC_i,sys_call_number),(PC_j,sys_call_number) > ∈ SL )
    {   /* 添加系统调用序列状态转换边 */
        Transition_FSA ←Transition_FSA ∪
{S(PC_i,sys_call_number),S(PC_j,sys_call_number) >} ;
    }
for( < (sys_call_number,pc_i,argNum),(sys_call_number,PC_j,argNum) > ∈ TL )
    { /* 添加参数污点传播状态转换边 */
        Transition_FSA ←Transition_FSA ∪
{(PC_i,PC_j,argNum,argNum,taintPropag)} ;
    }
showFSA( ); /* 输出 XML 格式的 FSA 模型 */
}
```

该行为模型构造算法的关键步骤为:一是利用程序序列链生成能体现该程序控制流的有限自动机;二是利用污染传播链为其增加污染边。

## 4.2　模型异常检测算法

实际异常检测时,将生成的行为模型与实际截获的系统调用行为进行匹配,算法如下:

从模型检测算法可知,当程序行为同时满足以下 3 个条件时才视为处于正常:1)程序的系统调用序列与行为模型一致;2)系统调用的基本信息与参数信息与行为模型一致;3)污染边满足行为模型中参数污点传播链的约束。

# 5　模型实例

本节主要用实际的程序来展示污点跟踪分析的中间结果,以及解释第 4 节定义的概念和最终生成的行为模型。

图 3(a)实现的是文件的打开、读、写和关闭等操作。经 QEMU 在指令级别捕获到示例程序的系统调用序列为(open, write, fstat64, mmap2, write, close, mumap, rt_sigprocmask, rt_sigaction, rt_sigprocmask, nanosleep)。图 3(b)给出的是其中一个系统调用 close 其参数与返回值的基本信息和污点属性。close 的第 1 个返回值的污点标志 tainted 不为 0,表示不是污点源,它的 origin 和 offset 与 open 的返回值一致,并且污点传播方式 taintPropag = 0,这表示它读取了 open 的返回值,复制了其污点记录。图 3(c)是示例程序第 1 个系统调用 open 返回值的污点传播链。该传播链

算法:模型异常检测算法

输入:fsa——训练得到的自动机模型

$\{SysCall_0, \cdots, SysCall_n\}$——系统调用事件,包括序列、参数及污点传播信息

输出: Anomaly——0 表示正常,1 表示异常

```
void ModelCheck( )
{   SysCallEvent * preSysCall = NULL;
  /* 判断第一个系统调用是否与 FSA 相匹配 */
  if( SysCall0. PC ∈ {q0} ∩ SysCall0. S ∈ {S} )
       Anomaly = 0;
  else
         Anomaly = 1;
    preSysCall = SysCall0;
    for( i = 1; i < = n; i + + ) /* 第二个系统调用到最后一个系统调用 */
  {/* 检验两个系统调用是否符合序列的状态转移条件,发现异常则退出循环体 */
       bool checkTrans_base( SysCallEvent
                * preSysCall, SysCallEvent * SysCalli, FSA * fsa);
/* 检验两个系统调用是否符合足污点传播条件,发现异常则退出循环体 */
       bool checkTrans_taint( SysCallEvent * preSysCall, SysCallEvent
                * SysCalli, FSA * fsa);
       preSysCall = SysCalli ; } }
```

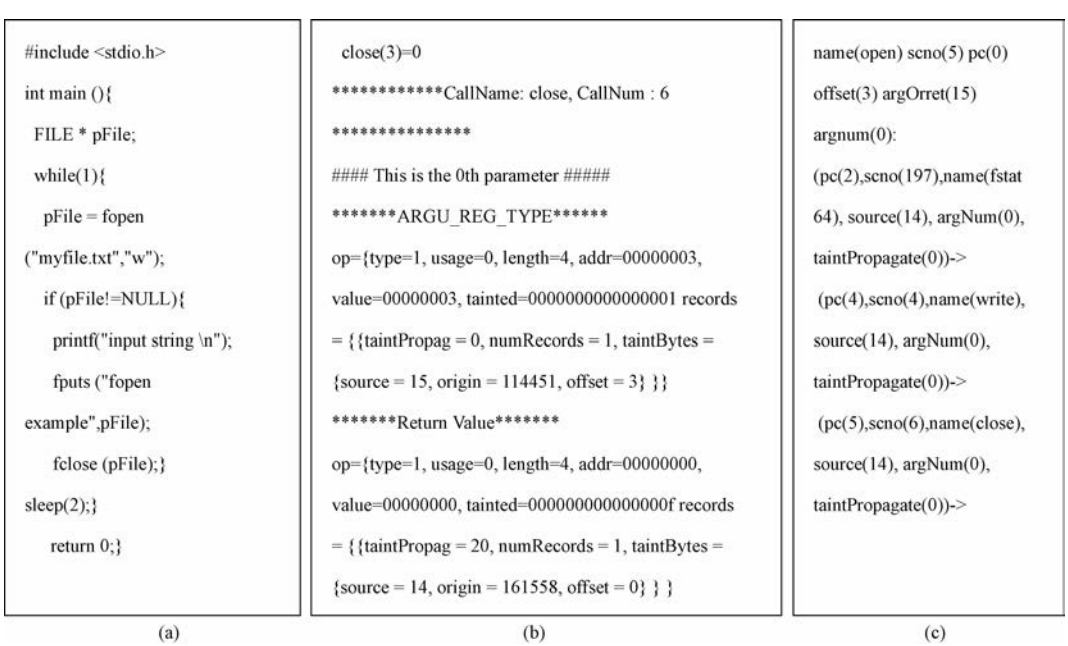


**图 3 示例程序、系统调用 close 的信息和参数污点传播链**

**Fig. 3 The sample program, the information of the system call "close", and the taint propagation chains of parameters**

显示出 fstagt64( )的第 1 个参数,write( )的第 1 个参数和 close( )的第 1 个参数都复用了 open( )的返回值。故本文中提出的污点传播链的相关信息既可表示程序系统调用的底层数据流特征,还能被抽象形成安全规则用于入侵检测。

利用 QEMU 底层捕获的系统调用控制流和

参数污点传播链,本文构造出的基于有限自动机的软件行为模型如图 4 所示。图中 $P_i(i = 0,1,\cdots,n)$ 代表的是动态变化的系统调用 $PC$ 值的相对位置,实线代表系统调用控制流,如"$R_0 = open(P_{(0,0)},P_{(0,1)},P_{(0,2)})$",虚线箭头代表的是系统参数的全局数据流特性,如$(P_0,R_0,P_5,P_{(5,0)},0)$,这二者组成 FSA 的两种状态转换边。其中,$P_{(i,j)}$ 表示第 $i$ 个系统调用的第 $j$ 个参数(从 0 开始计数),$R_i$ 表示第 $i$ 个系统调用的返回值,故$(P_0,R_0,P_5,P_{(5,0)},0)$ 表示第 5 个系统调用的第 0 个参数是被第 0 个系统调用的返回值所污染的,污染传播方式是 0。需要注意的是,系统调用之间的污染边个数与存在污染传播的参数个数成正比。

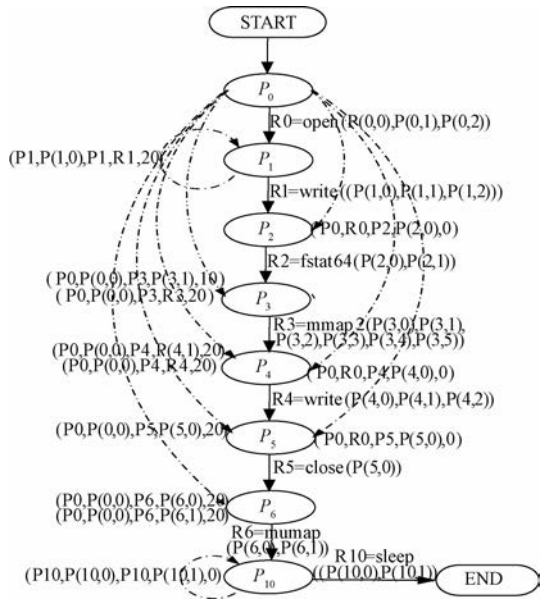


**图 4　示例程序的 FSA 模型图**

**Fig. 4　The FSA model of sample program**

## 6　模型检测能力评估

这节主要是评估本文所构造的行为模型的检测能力。本文模型充分融合了系统调用的控制流和数据流两方面特性,能够有效地检测出隐蔽的攻击,该类攻击的手法一般为:文献[6]和[21]中提出的非控制流劫持攻击(代码执行、竞争条件)或劫持控制流后模拟正常调用序列(Mimicry);或者针对现有学习系统调用参数数据流特性所存在的缺陷而提出的攻击,将在 6.2 中详细描述。

### 6.1　非控制流数据攻击的检测

本文模型可以检测出不同类型的非控制流数据的攻击,如表 1 所示。

**表 1　检测非控制流数据攻击的原因**

**Table 1　Detection reasons for non-control attacks**

| 程序 | 攻击描述 | 检测原因 |
|---|---|---|
| WU-FTPD | 通过格式字符串改写 userid 数据 | 破坏 pass( )函数中的系统调用 setuid( )参数和 getdatasock( )中的 setuid( )参数之间的污点传播; |
| Netkit Telnetd | 堆溢出破坏 execve 程序的名字 | 破坏命令行与 execve( )参数间的污点传播; |
| Fingerd | 符号链接漏洞 | 破坏命令行与 open( )参数间的污点传播; |
| GNU rm | 条件竞争攻击 | 破坏命令行与 unlink( )及 rmdir( )参数间的污点传播; |
| Synthetic | 攻击文件描述符 | stderr 与 Open( )的返回值之间原本没有污点传播关系; |
| Null httpd | 攻击服务器配置数据 | 破坏 HTTP POST 请求中与 CGI-BIN 配置间污点传播; |
| Ghttpd | 攻击用户输入数据 | log( )处理超长 GET 请求改写 serveconnection( )中的 ptr 局部变量时的污点传播; |
| Sudo CVE-2012-0809 | 格式化字符串漏洞 | sudo_debug( )中通过 getprogname( )获取 argv[0]与 fmt2 之间存在污点传播关系; |
| Nginx CVE-2013-2028 | 栈缓冲区溢出漏洞 | 破坏命令行与 ngx_http_parse_chunked( )中参数间的污点传播关系; |
| Nginx CVE-2013-4547 | 注入式攻击 | 破坏 ngx_http_parse_request_line ( )中 url 参数与 fastcgi 的污点传播关系; |
| Heartbleed[22] | 缓存区溢出漏洞 | 破坏 memcpy( )中数据地址指针与 payload 之间的污点传播关系; |
| Shellshock[23] | 注入式攻击 | 攻击环境变量函数,向 bash 注入命令时的污点传播。 |

### 6.2　更隐秘的非控制流数据攻击的检测

本节通过分析 2 个针对非控制流数据的较为隐秘的攻击实例,详细说明文献[6-7]中描述的攻击检测方法无法检测此类攻击的原因,同时突出本文中提出模型的优点。

图 5 中程序的正常功能是:从文件中读取文件名作为 MakeHashFile 的参数,对其进行 hash 变换,然后与真实密码进行对比,如果验证通过则在以参数 hash 值命名的文档中保存重要文件的名称。程序原有逻辑的目的是防止关键文件名被随意访问以及文件名被篡改。该程序的第 6 行存在漏洞,通过 password 的溢出可以改写输出文件名。因为程序中变量 var1 和 var2 之间存在依赖关系并且 var2 作为函数的局部变量具有一定的随机

```
1：#define PASSWORD "12345678"
void MakeHashFile(char * password,FILE* fp){
 char var1[256];
2： char var2[20];
3： char passbuf[8];
4： fscanf(fp,"%s",var1);
5： sprintf(var2,"%x",MakeHash(var1));
//MakeHash() 基于字符串生成一个整数
6： strcpy(passbuf,password); //溢出,覆盖 var2
7： if(strncmp(passbuf,PASSWORD,
strlen(PASSWORD))=={
8： FILE* fp2= fopen(var2,"w");
9： fwrite(var1,1,strlen(var1),fp2);
            fclose(fp2);}}
void main(){ char password[1024];
 FILE * fp;
          FILE * fp2;
10：fp=fopen("password","r")))
11： fscanf(fp,"%s",password);
//password 值为 12345678a.exe, 作为攻击变量
12：fp2 = fopen("filename","r");
13：MakeHashFile(password,fp2); }
```

**图 5 针对缺失系统调用参数与非系统调用代码关联的攻击**

**Fig. 5 Attack to the program lacking of association between system call arguments and non-system call arguments**

性,那么对于图 5 程序中第 6 行存在漏洞,仅仅依赖文献[6-7]中阐述的传统方法是无法检测出篡改 var2 而实施的攻击。而本文提出的基于污点传播的检测模型可以通过污点链的变化有效检测出这种攻击,因为在攻击成功后,password 将污点传播给了 var2 导致新的传播链出现。

如图 6 所示的 SSH 程序片段,存在整数溢出漏洞[24],可篡改决策变量实现非法 root 登录。该攻击的主要过程是:首先基于 SSH 客户端向服务器端发送一个数值很大的 packet;然后,服务器端使用 packet_read( )进行数据接收和检测,并引发整数溢出,导致权限标志值 authenticated 被改为非零,尽管 if 语句中的密码认证环节失败,仍然可以跳出 while 循环并执行函数剩余部分。因为服务器端接收到的数据包与权限标志量之间没有任何依赖关系,正常运行时二者之间是不存在污点链的,但是由于攻击时溢出 packet 覆盖了 authenticated,导致一条新的 packet 和 authenticated 之间的污染传播链出现。该攻击是通过控制系统调用参数来篡改局部决策变量的,因为本文提出的模型充分考虑了参数与局部变量之间的依赖关系,所以比对本文训练生成的行为模型和被攻击后的执行情况就可以检测出来,而以往忽略这种多重关系的行为模型是检测不出的。

```
void do_authentication(char * user, …){
1: int authenticated = 0;
2: while(!authenticated){
        /*Get a packet from the client */
3:  type = packet_read();
   // calls detect_attack() internally
4:  switch(type) {
5:  case SSH_CMSG_AUTH_PASSWORD:
6:    if ( auth_password(user,password))
7:          authenticated = 1;
    case …}
8:  if(authenticate) break;}
        /* Perform session preparation */
9：log(authenticated);  //write authenticated to file
10： do _authenticated (pw);}
```

**图 6 针对缺失系统调用参数与局部变量关联的攻击**

**Fig. 6 Attack to the program lacking of association between system call arguments and local variable**

## 7 总结

本文提出一种基于系统调用参数动态分析的软件行为建模系统,该系统在指令级别监控应用程序运行,拦截系统调用事件并解析相关参数的详细信息,借助基于 QEMU 的动态污点分析平台,增加关键功能,从而捕获参数在整个程序内部的污点传播信息,最后根据系统调用的基本信息和参数污点传播链建立自动机模型。通过比对实时监控进程执行得到系统调用控制流和参数关系与自动机模型是否匹配来判断异常。具有以下特点:

1)利用虚拟技术,实现指令级程序跟踪,获取系统调用细粒度信息;

2)利用动态污点分析技术,捕获系统调用参数在程序全局的传播轨迹,从而将不同系统调用

的参数,非系统调用代码以及局部变量三者有机联系起来;

3)利用系统调用的控制流和数据流构建 FSA 模型,抵抗模仿攻击能力更强;

4)该系统能够检测出更为隐秘的非控制流数据攻击且漏报率更低。

目前,本文中提出的模型使用基于前提,即软件在指令级的行为异常能导致结果的变化,为突破模型通用性和改善模型性能,未来工作将从如下方面展开:1) 建立入侵检测系统的通用性能指标,评估行为模型精度;2) 拓展动态污点分析数据源,加入外部输入数据和重要的局部变量等;3) 优化系统的时间损耗,如将动态污点分析平台移植到系统内核中,从而脱离虚拟平台;4) 动态污点分析在判断异常时是着眼于局部行为,所以考虑如何将局部行为和结果相结合进行最终判断。

## 参考文献


[1] Tandon G, Chan P K. On the learning of system call attributes for host-based anomaly detection[J]. International Journal on Artificial Intelligence Tools, 2006, 15(6): 875-892.

[2] Kruegel C, Mutz D, Valeur F, et al. On the detection of anomalous system call arguments [C] // Snekkenes E, Gollmann D. Computer Security-Esorics 2003, Proceedings. Berlin: Springer-Verlag Berlin, 2003: 326-343.

[3] Oyama Y, Yonezawa A. Prevention of code-injection attacks by encrypting system call arguments[R/OL]. University of Tokyo, 2006[2016-06-20]. https://www.researchgate.net/profile/Akinori_Yonezawa2/publication/228576079_Prevention_of_code-injection_attacks_by_encrypting_system_call_arguments/links/53eaf9ee0cf2fb1b9b6ad0fd.pdf.

[4] Sufatrio, Yap R H C. Improving host-based IDS with argument abstraction to prevent mimicry attacks[C]//Valdes A, Zamboni D. Recent Advances in Intrusion Detection. Berlin: Springer-Verlag Berlin, 2006: 146-164.

[5] Demay J C, Totel E, Tronel F. SIDAN: a tool dedicated to software instrumentation for detecting attacks on non-control-data [C] // Risks and Security of Internet and Systems (CRiSIS), 2009 Fourth International Conference on. IEEE, 2009: 51-58.

[6] Bhatkar S, Chaturvedi A, Sekar R, et al. Dataflow anomaly detection[C] // 2006 IEEE Symposium on Security and Privacy, Proceedings. Los Alamitos: Ieee Computer Soc, 2006: 48-62.

[7] Li P, Park H, Gao D, et al. Bridging the gap between data-flow and control-flow analysis for anomaly detection [C] // 24th Annual Computer Security Applications Conference, Proceedings. Los Alamitos: Ieee Computer Soc, 2008: 392-401.

[8] 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. ACM, 2007: 196-206.

[9] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software [C] // Proceedings of NDSS' 05. San Diego, California, USA: 2005.

[10] Chen K, Feng D, Su P, et al. Black-box testing based on colorful taint analysis [J]. Science China Information Sciences, 2012, 55(1): 171-183.

[11] Ma J X, Zhang P H, Dong G W, et al. TWalker: an efficient taint analysis tool[C]//2014 10th International Conference on Information Assurance and Security. New York: Ieee, 2014: 18-22.

[12] Haller I, Slowinska A, Neugschwandtner M, et al. Dowsing for overflows: a guided fuzzer to find buffer boundary violations[C]//22nd USENIX Security Symposium (USENIX Security 13). 2013: 49-64.

[13] Kong J, Zou CC, Zhou H. Improving software security via runtime instruction-level taint checking[C]//Proceedings of the 1st workshop on Architectural and system support for improving software dependability. ACM, 2006: 18-24.

[14] Qin F, Wang C, Li Z, et al. LIFT: a low-overhead practical information flow tracking system for detecting security attacks [C] // 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06). 2006: 135-148.

[15] Halfond W G J, Orso A, Manolios P. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks [C]//Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. Portland, Oregon, USA: ACM, 2006: 175-185.

[16] Kiezun A, Guo PJ, Jayaraman K, et al. Automatic creation of SQL injection and cross-site scripting attacks[C]//2009 31st International Conference on Software Engineering, Proceedings. New York: Ieee, 2009: 199-209.

[17] Vogt P, Nentwich F, Jovanovic N, et al. Cross site scripting prevention with dynamic data tainting and static analysis[C]//NDSS. 2007: 12.

[18] Yin H, Song D, Egele M, et al. Panorama: capturing system-wide information flow for malware detection and analysis[C]//Proceedings of the 14th ACM conference on Computer and communications security. Alexandria, Virginia, USA: ACM, 2007: 116-127.

[19] Song D, Brumley D, Yin H, et al. Information systems security: 4th International Conference, ICISS 2008, Hyderabad, India, December 16-20, 2008 Proceedings[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 1-25.

[20] Sekar R, Bendre M, Dhurjati D, et al. A fast automaton-

based method for detecting anomalous program behaviors[C]// 2001 Ieee Symposium on Security and Privacy, Proceedings. Los Alamitos: Ieee Computer Soc, 2001: 144-155.

[21] Chen S, Xu J, Sezer E C, et al. Non-control-data attacks are realistic threats[C]//USENIX Association Proceedings of the 14th USENIX Security Symposium. Berkeley: Usenix Assoc, 2005: 177-191.

[22] Hu H, Shinde S, Adrian S, et al. Data-oriented programming: on the expressiveness of non-control data attacks[C]//2016 IEEE Symposium on Security and Privacy (SP). San Jose, USA: 2016: 969-986.

[23] Delamore B, Ko R K L. A global, empirical analysis of the shellshock vulnerability in web applications[C]//Trustcom/BigDataSE/ISPA, 2015 IEEE. 2015: 1 129-1 135.

[24] Zalewski M. SSH1 CRC-32 compensation attack detector vulnerability[DB/OL]. (2001)[2016-06-20]. http://www.securityfocus.com/advisories/3088.