# Linux 系统中基于多路径的恶意行为规范挖掘①

刘琳爽　缪　力　(湖南大学 软件学院 湖南 长沙 410082)

**摘　要：** Linux 恶意代码检测是 Linux 安全框架的一个重要组成部分。大多数现存的依照特征进行检测的方法通常落后于恶意代码的发展，已经不能满足日益迫切的安全需求，而基于行为的检测器往往需要高质量的恶意行为规范。使用了一种基于系统调用的自动挖掘规范技术，并在此基础上开发恶意代码的多执行路径，使其规范更详细更全面，从而提高检测器的检测率。

**关键词：** Linux 恶意代码;行为规范;自动挖掘;多执行路径

## Mining Specifications of Malicious Behaviors Based on Multiple Paths in Linux

LIU Lin-Shuang, MIAO Li (Department of Software, Hunan University, Changsha 410082, China)

**Abstract:** The malware detection is one of the important parts in a secure Linux frame. Most existing malware detection methods are based on the signature and generally leave behind the development of the malware technology, which cannot meet the ever increaing needs of security. Behavior-based detectors require high-quality specifications of malicious behavior. This paper introduces an automatic technique to mine specifications of malicious behavior based on system calls and explores multiple execution paths for malware specifications, which helps the specifications to be more specific and more comprehensive, and improves the rate of detection of the behavior-based detectors.

**Keywords:** Linux malicious code; specifications of behavior; automatic mining; multiple paths

## 1　引言

随着互联网的蓬勃发展，Linux 操作系统使用的普及，针对 Linux 操作系统的病毒和攻击程序也随之不断出现，急需一种有效的方法来对各种恶意程序做出主动响应和防御。目前大多数防病毒软件采用的策略都是依据恶意代码的特征来检测和查杀，因而对大多数新产生或使用模糊变换技术[1]的恶意代码都无能为力，防御手段严重滞后于攻击手段。新一代的病毒防御策略就是要通过程序的行为来判断是否为恶意代码并主动做出响应，而这些基于行为的检测器都需要高质量的恶意行为规范，这些规范往往需要专业人员的手工构建，耗时又繁琐。文献[2]首先提出了一种自动的基于系统调用的恶意代码规范挖掘方法，它将恶意程序和良好程序的执行行为进行对比挖掘，这种方法快速、规范语言灵活、性能良好。本文将在此基础上进行恶意代码的多执行路径开发，能自动构造更全面的系统调用依赖图、生成更详细的规范，从而检测更多的恶意代码。

## 2　Linux系统调用机制和Strace工具

在 Linux 操作系统中，系统调用实际上就是用户程序和操作系统的一个接口。Linux 操作系统出于保护的目的，区分开了用户空间和内核空间，操作系统内核运行在内核空间，用户进程运行于用户空间。由于保护级别不同，用户进程不能直接访问内核的数据，也不能直接调用内核的函数。操作系统内核是为用户服务的，用户所有的操作最终都要通过内核里的相应函数来实现。为此，操作系统提供了一种机制，既能保护内核信息，也可以为用户提供必须的服务，这种机制就是系统调用。对于内核数据空间的访问主要使

① 收稿时间:2010-01-03;收到修改稿时间:2010-04-02

用软中断 int$0x80 实现的。一个系统调用实际上就是一个中断，系统调用的过程就是软中断的处理过程。系统调用运行于当前进程的地址空间。用户进程通过中断进入内核空间后，可以根据系统调用号在内核的 sys_call table 表中定位相应的系统调用服务例程并执行，并返回执行结果给用户进程。每一个进程在被执行的过程中，在内核中的操作实际上是由一个系统调用序列所组成的。因此，进程的行为最终可以定义为系统调用序列的执行过程。每一个进程的执行期间必定要调用不同的系统调用例程，而每一个系统调用执行次序和执行参数也不同，这是使用系统调用分析恶意代码行为的前提。

Strace 是一款 Linux 自带的分析和记录目标进程各种系统调用的工具。将它和可疑程序同时运行，Strace 可以启动并监控对应进程或者附加到正在运行的进程。除了拦截系统调用，Strace 还可以捕获信号量和进程间通信。Strace 收集的信息对分析可疑程序的实时行为、确定程序的性质和目的非常有用。

## 3 恶意规范挖掘算法

恶意规范挖掘算法是使用动态分析技术[3]，将运行的程序看作一个黑盒，只监测其与操作系统的交互，即程序的系统调用。该算法首先收集一个恶意程序和一个或更多个良好程序的系统调用序列，然后根据系统调用参数之间的依赖关系构造相应的依赖图，最后再将恶意程序和良好程序的依赖图进行一一对比，找出依赖图之间的差别，并将这有差别的依赖图子图作为恶意行为规范。定义恶意行为规范为一个系统调用依赖图。假设∑是系统调用集合，一个系统调用 $S\in\sum$ 是 N 个变量的函数，即 $S:T_1\times\cdots T_N\rightarrow T_R$，其中 Ti 是操作中第 i 个参数的类型，$T_R$ 是返回类型。类型系统由不透明指针类型(用来涉及操作系统各种对象)、整型类型、字符串类型以及类型构造器组成。

系统调用之间的依赖关系可以转换成用来约束系统调用参数值间的逻辑公式。假设 $L_{Dep}$ 是这样的一个逻辑，Vars 表明在这个逻辑公式中出现的变量集合，集合 $\sum\times 2^{Vars}$ 表示符号系统调用集合。

定义 1. 恶意规范是一个有向无环图(Directed Acyclic Graph, DAG)，其中表∑中的系统调用作为节点(node)标签，在逻辑 $L_{Dep}$ 中的逻辑公式作为边(edge)标签。一个恶意规范(malspec)M 可以定义为一个四元组，M=(V, E, γ, ρ)，其中：

(1) V 是顶点集，E 是边集，$E\subseteq V\times V$；

(2) $\gamma: V\rightarrow\sum\times 2^{Vars}$；

(3) $\rho: VE\rightarrow L_{Dep}$。

关于恶意规范挖掘算法的伪代码如下所示：

输入：一个恶意程序 M 和一系列良好程序

输出：一系列的恶意规范{…,$M_j$,…}

```
Begin
  /* 收集执行路径 */
  tM ← ExecutionTrace(M) ;
  for i=1 to K do ti ← ExecutionTrace(Bi) ;
  /* 创建依赖图，每次 trace 生成一个图 */
  foreach tx ∈ {tM , t1,… , tK } do
  V← Events(tx) ;
  E← InferDependences(tx) ;
  Gx← (V, E);
  /* 从 GM 中计算得出规范 */
  foreach Hj ∈ UniqueComponents (GM) do
  if IsTrivialComponent(Hj) then continue;
  Mj← (∅,∅) ;
  /*∪= 最大并集, Θ = 最小对比子图 */
  for i ← 1 to K do Mj ← Mj∪(Hj Θ Gi)
  return {. . . , Mj , . . . } ;
end
```

### 3.1 收集执行路径

如果恶意程序和良好程序的执行路径没有差异，那么就不能识别出恶意行为。事实上，让一个恶意程序表现出恶意行为并不难，因为大部分的恶意程序都是试图感染并尽可能在多环境中传播。笔者是在 Qemu[4]虚拟机中使用 Strace 工具来收集恶意程序和良好程序的执行路径。

### 3.2 构造依赖图

在依赖图中有三种反映系统调用彼此间依赖关系的类型。第一种是定义-使用依赖关系，它指的是一个系统调用的输出值在另一个系统调用中使用，类似于程序分析中的定义-使用依赖概念。次序依赖(ordering dependences)关系表示在两个系统调用之中第一个系统调用必须先于第二个系统调用，由此可见次序依赖关系应该基于 API 规范(例如在一个调用关闭之后就不能再读、写)或者协议规范，在协议规范

中一条指令通过通信通道进行发送和接收必须遵守一个指定的序列。第三种是值依赖(value dependences)关系，它是一个逻辑公式，用来描述系统调用之间任何非琐碎(non-trivial)数据在程序执行中的使用。

### 3.3 对比子图生成

前两个步骤将恶意程序和良好程序的执行路径转化成相应的依赖图。恶意规范应该是不出现在任何良好依赖图中的恶意依赖图子图。

定义 2. 一个标记图(labeled graph)G 是一个四元组(V, E, α, β)，V 是顶点集，$E \subseteq V \times V$ 是一个边集，α 是分配标签给顶点的函数，β 是分配标签给边的函数。

定义 3. S=(W, F, α, β)是 G=(V, E, α, β)的子图，当且仅当：

(1)$W \subseteq V$；

(2)$F \subseteq E \cap (W \times W)$。

同样地，G 被称为 S 的超图。

定义 4. 一个边集是一个标记图并且图中不存在孤立的顶点。

定义 5. 假设 G´=(V´, E´, α´, β´)和 G=(V, E, α, β)，一个子图同构是一个单射函数 f:(V´)→V 满足：

(1)$\forall e' = (u, v) \in E'$，这里存在 e=(f(u), f(v)) ∈ E；

(2)$\forall u \in V'$，$\alpha'(u) = \alpha(f(u))$；

(3)$\forall e' \in E'$，$\beta'(e') = \beta(f(e'))$。

如果存在这样一个函数，那么 G?就是 G 的子图同构。如果 f:(V´) →V 是双射的，那么 G´同构于 G。

定义 6. $G_p$ 和 $G_n$，$C \subseteq G_p$ 是一个对比子图当且仅当 C 不是 $G_n$ 的子图同构。它是最小的如果它的任意子集都不是对比子图。

形式上，两个图 $G_1$ 和 $G_2$ 的一个对比子图指的是 $G_1$ 的一个子图且不是 $G_2$ 的子图同构。当一个对比子图的任意子图都不是对比子图时，那么这个对比子图是最小的。

恶意规范是使用恶意程序和一系列良好程序依赖图的最小对比子图来构建。Ting 和 Bailey 提出了一种计算最小对比子图的算法[5]。该算法首先使用递归回溯树(backtracking tree)来计算最大公共边集，然后运用最小遍历(minimal transversals)算法来得到最小对比子图集合。

## 4 开发多执行路径

Strace 工具是收集代码样本的单一执行路径。然而，对于某些只在特定环境下触发的恶意程序来说，就有可能收集不到具有恶意性质的执行路径。例如，某些恶意程序要求与 Internet 连接才运行恶意代码块；有的恶意行为是在某个时期或一天中的某个时间发生；有的恶意行为需要得到某条指令来触发。在这些情况下，如果程序行为只被一次执行结果所确定，那么必然会产生错误的分析报告。为了解决这个问题，需要对 Strace 工具进行扩展，使 Strace 工具具有收集程序多执行路径的功能。

程序多执行路径开发的基本思想是由监测某种输入来驱动，具体的说就是动态追踪程序读入的某些输入值(比如操作系统的当前时间，文件的内容)，这些输入值将被标上标签，对程序中有标签的分支点(即标记的输入值在分支语句中被使用)进行识别并创建快照，当该路径正常执行完后再返回到快照处，改变带标签的输入值，即选取了分支语句中的另一条路径，从而实现多路径的开发。当有多个分支点嵌套或者排列时，采用深度优先的策略选取。图 1 是一段 C 程序多路径开发的示意图，在该程序开始执行时它接收某个输入值并将之存储到 a 变量中，因此这里应对 a 标上标签。

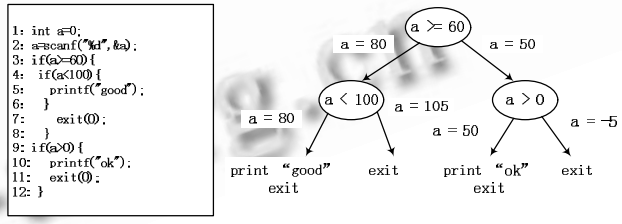


图 1　多路径开发示意图

### 4.1 追踪输入

在传统的基于污染的(taint-based)系统中，能够充分知道某一个内存单元依赖于一个或更多个输入值。为了获得这个信息，这种系统一般由三个部分组成：污染源集合、一个阴影内存和传播污染信息的机器指令扩展。污染源被用来最初分配标签给某个内存单元，比如 Vigilante[6]是用来检测在网络中传播的计算机蠕虫，在该系统中，网络就被认为是一个污染源。阴影内存需要及时追踪在某个点上哪个标签被分配给了哪个内存单元。当某一个操作处理或者移动了有标签的数据时，扩展的机器指令被要求传播污染信息。

使用基于污染系统的原理可以追踪程序如何处理输入值。这里，污染源是能够返回与恶意代码行为相关的信息的系统调用。这些系统调用包括访问文件系统和网络的系统调用，另外也会考虑返回当前时间和网络连接状态的系统调用。当一个相关的系统调用被程序触发时，系统会自动的分配一个新的标签给每一个接收系统调用结果的内存单元。

除了阴影内存外，还需要增加一个逆向映射。阴影内存是将内存单元映射到标签，而逆向映射存储的是对每一个标签目前拥有该标签的所有内存单元地址。逆向映射保证了当某一个带标签的内存单元修改时能够同时改变所有拥有同样标签的其他单元，即进程状态保持一致。

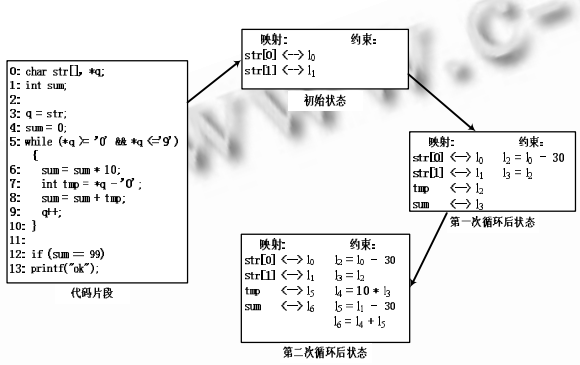


图 2 程序执行中约束产生示意图

在对带有标签的输入值进行重写时，不能简单的对具有同样标签的值一致重写，还应考虑在运算后所得到结果与带标签值的依赖关系。可以对任意包含带有标签参数的操作分配一个新的标签给其结果，并记录下带有新标签的值如何依赖于旧标签的值。这就需要新创建一个约束，根据操作的语义来得到旧标签和新标签之间的关系，这个约束随后加入到用来维持部分进程执行状态的约束系统中。目前，笔者只对输入变量值之间的线性依赖关系进行建模。也就是，约束系统是一个以$\{c_n*l_n+c_{n-1}*l_{n-1}+...+c_1*l_1+c_0\}$的形式来存储条件的线性约束系统，其中 $c_i$ 为常量，$l_i$ 为标签。这些条件通过等式或不等式操作连接。假设一个分支语句中使用的输入值其标签为 $l_n$，为了开发多执行路径，当条件的结果取反时就不得不重写带标签的值。为了保证一致性，应该首先使用线性约束系统来识别所有与 $l_n$ 相关的标签，然后通过线性约束求解器(solver)来决定这些内存单元的具体值。图 2 显示了当一个简单的 atoi 函数执行时约束系统中的情形。该函数是把字符串转换成整型数，假设函数在执行时需要转换的字符串由 3 个字符组成(其中第三个字符是字符串结束符)。

### 4.2 保存和恢复程序状态

程序执行过程中，当一个条件操作使用了一个或两个带标签的参数时，就需要对此操作进行识别，并创建当前进程的快照。当前执行状态的快照包括了正在使用的全部虚拟地址空间、映射和约束系统。在进程被允许继续之前，还需要考虑条件操作本身，这个约束被称为路径约束。当被标记的值重写时需要考虑路径约束，否则就有可能创造不一致的状态或者到达不可能的路径。

例如，图 1 中的程序使用了三个检测(checks)来保证 a>=60、a<100 和 a>0。当到达第一个 if 语句时，将会创建一个当前状态的快照。因为 a 有初始值 80，程序会顺着 if-branch 语句往下执行。在程序继续之前，还必须记下只有当带标签的值大于或等于 60 时 if-branch 语句才能发生，将此路径约束 $l_0>=60$ 加入到约束系统中。随后在第 4 行的分支语句上创建另一个快照，此时，if-branch 语句执行并将路径约束 $l_0<100$ 加入约束系统。程序继续执行并调用 printf 函数，当执行完后，返回到前一个存储状态，这次第四行的 else-branch 语句执行，之后将 $l_0>=100$ 加入约束系统。此时，约束系统有两个 entries，一个是刚刚加入的约束 $l_0>=100$，另一个是最开始加入的约束 $l_0>=60$，约束系统求解器分析之后得出一个解 $l_0=105$，这样 a 被重写为 105，程序执行完后再返回到最开始存储的状态，进行新一轮的值更新。

当程序状态需要恢复时，首要的任务就是将之前保存的程序地址空间内容进行装载并重写其当前值。然后将已保存的约束系统装入，对已使用过的路径约束取反，将这新的路径约束加入到约束系统中并使用约束系统求解器求解。当找到解时，使用新值对所有相关标签的变量重写其相应的内存单元。如果没有找到解，那么就不能开发另外一条执行路径。

## 5 实验结果分析

笔者选取 Linux.Psybot 作为恶意代码样本，将 Mozilla Firefox 3.5.4、QQ for Linux 1.0、Open

Office for Linux 3.1.0 作为良好程序集合。每一个良好程序运行 1 到 2 分钟,并给某些程序提供手工输入,例如，在 Firefox 中指定一个网址并点击链接。

在未开发程序多路径之前，自动生成的 Linux.Psybot 恶意行为规范为空，说明其恶意行为在通常的单一执行路径中没有发生。在采用了收集程序多执行路径的恶意规范挖掘算法后，找到 Linux.Psybot 的三种恶意行为。经分析发现该蠕虫需要接收 wget、ftpget 命令才能试图从远程端下载和运行自己的复本，然后通过 TCP 端口 80、22、23 来锁定管理员的访问，接着在被感染的机器上打开一个后门来允许远程攻击者执行一些恶意行为。

如表 1 所示，自动生成的 Linux.Psybot 恶意行为规范与赛门铁克(Symantec)反病毒研究中心由专家分析得出的报告匹配了三条描述，说明该算法具有高度的正确性和可行性。

表 1　专家提供的行为描述与挖掘规范 malspecs 的对比

| 专家特征描述 | Malspecs 描　述 |
|---|---|
| 分布式拒绝服务攻击 | 一致 |
| 端口扫描 | / |
| 打开和关闭攻击 | 一致 |
| 下载和运行文件，其中某些文件可能包含升级版本的蠕虫 | 一致 |

## 6　结论

本文使用了一种基于系统调用的自动挖掘规范技术，并在此基础上研究了如何收集程序多路径的系统调用序列，该技术不仅可以发现使用单一执行路径很难发现的恶意行为，还可以使得用于高级检测器的恶意规范更加全面。下一步的工作的重点是增加系统调用参数依赖关系，结合动态和静态分析技术生成更好的依赖图。

## 参考文献


1 庞立会,胡华平.恶意代码模糊变换技术研究.计算机工程, 2007,33(12):154－156.

2 Christodorescu M, Jha S, Kruegel C. Mining specifications of malicious behavior. Proc. of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), September 2007,5－14.

3 Bayer U, Moser A, Kruegel C, Kirda E. Dynamic analysis of malicious code. Computer Virology, 2006, 2:67－77.

4 Bellard F. Qemu, a Fast and Portable Dynamic Translator. Proc. of Usenix Annual Technical Conference, 2005.

5 Ting RMH, Bailey J. Mining minimal contrast subgraph patterns. Proc. of the 6th SIAM International Conference on Data Mining, 2006, 639－643.

6 Costa M, Crowcroft J, Castro M, Rowstron A, Zhou L, Zhang L, Barham P. Vigilante: End-to-End Containment of Internet Worms. Proc. of 20th ACM Symposium on Operating Systems Principles (SOSP), 2005.