文章编号:1001 - 9081(2005)12 - 2739 - 03

对计算机系统中程序行为的分析和研究

朱国强,刘 真,李宗伯
(国防科学技术大学 计算机学院, 湖南 长沙 410073)
(qqrilxk@ newhua. com)

摘 要:对程序行为的三种提取方法进行了分析比较,并采用 LKM(Linux Kernel Module)方式对程序行为进行提取分析。从字符串参数长度分布,字符串参数字符特征分布及特殊系统调用参数三个方面来对系统调用参数进行分析,丰富了程序行为分析手段,提高了程序异常检测精度。

关键词:程序行为;系统调用参数;Linux 内核模块
中图分类号: TP309 **文献标识码**:A

Analyses and research of the program behavior in computer system

ZHU Guo-qiang, LIU Zhen, LI Zong-bo
(*School of Computer Science*, *National University of Defense Technology*, *Changsha Hunan* 410073, *China*)

Abstract: Three methods of distillation in the program behavior were introduced, and the program behavior was distilled and analyzed in LKM. The system call arguments was anlalyzed from the length distribution of character string, characteristic distribution of character string and special system call arguments, which rich the technique to analyze the program behavior and improve the exactness of detection of program anomalism.

Key words: program behavior; system call arguments; Linux Kernel Module(LKM)

0 引言

计算机安全问题很大程度上表现为软件的脆弱性,各种程序不可避免地存在设计上的缺陷,这常常被黑客所利用。然而,存储在硬盘上的代码只有当它运行并执行了系统调用时才可能破坏系统[4]。因此,若能约束应用程序所能执行的操作就能限制入侵者对系统所造成的影响,进而从根本上解决由于程序的不可靠性所带来的问题。因此对程序行为进行分析,从系统调用级对程序行为进行约束是一个十分有意义的选择。

本文将在系统调用级阐述怎样提取计算机系统中的关键程序行为,并对其进行分析,为入侵检测提供准确而详细的参考资料。

1 程序行为的提取

计算机系统中程序的行为集中表现为系统调用序列。无论对系统的攻击采用何种手段,最终造成的伤害都是通过某些系统调用来完成的。因此,如果能截获每个进程的系统调用,并对其进行控制(不执行该系统调用、改变调用参数等),就可以预先避免攻击的发生。

在 Linux 系统中对系统调用的截获可以有以下方法:

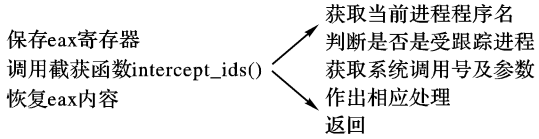
1)执行跟踪技术

执行跟踪是一个程序监视另一个程序执行的技术。被跟踪的程序一步步执行,直到接收到一个信号或调用一个系统调用。

在 Linux 中通过 ptrace( )系统调用执行跟踪。ptrace( )系统调用修改被跟踪进程描述符的 p_pptr 域以使它指向跟踪进程,因此,跟踪进程就变为了被跟踪进程的有效父进程。当一个监控事件发生时,被跟踪的程序停止,且将信号 SIGCHID 发送给它的父进程。父进程在此时可完成对被跟踪进程的信息采集,传递及修改工作,然后再发出命令恢复被跟踪进程的执行。执行跟踪往往在用户态下运行,系统开销较大,实时性不强,并且不能较好地提取系统调用参数等信息。因此不适合用在入侵检测这一类对实时性要求较高的场合。

2)直接修改并编译内核

直接修改并编译内核通过对内核中系统调用的关键点进行修改并重新编译内核来实现。所有的系统调用在经过 INT 80 后都会经过一个公共的入口点 system_call。修改内核方法就是当 system_call 函数进行寄存器值保存之后,在系统调用处理程序之前截获系统调用。在 entry. S 中如下修改:



修改并编译内核的方法能高效实现,实时性较强,处理灵活。但对用户要求较高,需要用户对内核知识有所了解并能自己编译内核。

2 编写 LKM

编写 LKM 方法能够在不修改内核的情况下在内核态截获系统调用,实时性较强,可根据要求动态加载卸载,这样方便了用户而又不用冒着编译内核的风险。

在 Linux 2.4.18-14 及其以后内核版本中 sys_call_table[ ]不再导出,因此我们也就无法通过截获系统调用函数来截获系统调用。在此,我们通过修改中断向量表的方法来实现截获功能。我们知道一条系统调用的发生在经过_syscallN宏后转化

收稿日期:2005 - 06 - 21;**修订日期**:2005 - 08 - 29

作者简介:朱国强(1979 -),男,湖南娄底人,硕士研究生,主要研究方向:计算机网络信息安全; 刘真(1964 -),男,湖南长沙人,教授,主要研究方向:计算机网络信息安全; 李宗伯(1972 -),男,湖南长沙人,副教授,主要研究方向:计算机体系结构.

为一条 INT 80 指令,在执行 INT 80 指令时完成了以下几条操作:1)由于 INT 指令发生了不同优先级之间的控制转移,所以首先从 TSS(任务状态段)中获取高优先级的核心堆栈信息(SS 和 ESP);2)把低优先级堆栈信息(SS 和 ESP)保留到高优先级堆栈(即核心栈)中;3)把 EFLAGS,外层 CS,EIP 推入高优先级堆栈(核心栈)中;4)通过 IDT 加载 CS,EIP(控制转移至中断处理函数),然后就进入了中断 0x80 的处理函数 system_call。

从上面的过程可以看出,我们若能修改 IDT 中相应中断描述符的偏移字段值就可以使控制转移到我们的截获函数中,在截获函数中进行适当的操作处理后再进入中断 0x80 的处理函数 system_call。

每一个中断描述符都包含一个中断服务程序的地址,CPU 通过将中断号作为索引值从 IDT 中取得的就是这样一个“中断描述符”,通过“中断描述符”,CPU 就可以得到中断服务程序的地址。中断描述符的结构如图 1 所示。

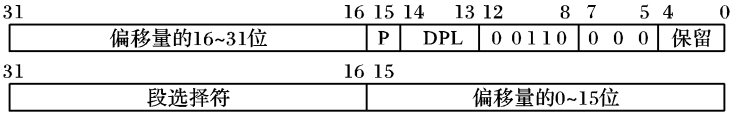


图 1 中断描述符的结构

偏移量总共是 32 位,它表示一个中断服务程序在内存中的位置。段选择符是指向全局描述符表(GDT)的一个指针。由于保护模式下,内存的寻址是由段选择符与偏移量指定的,所以在中断描述符中也分别设定了段选择符与偏移量位,它们共同决定了一个中断服务程序在内存中的位置。在这里只需要修改偏移量的值就够了,这是因为我们的截获代码是作为模块编译,也是放置在内核中的。

要修改中断描述符中的偏移字段值先得获得中断向量表在内存中的位置。在保护模式中,中断向量表可放在内存中的任何地方。但在 CPU 中有一个寄存器 IDTR 指向当前中断向量表 IDT[5]。

因此我们可先取得 IDTR 寄存器的内容,将其保存在结构变量 idtr 中:

```
__asm__ volatile ("sidt %0": "=m" (idtr));
```

idtr 为结构变量,声明如下:

```
struct
 { unsigned short limit;
   unsigned long base;
 }__attribute__ ((packed)) idtr;
```

结构变量 idtr 的成员 base 即为中断向量表(IDT)在内存中的地址。从而可根据偏移量 0x80 8 得到 INT 80 中断描述符地址 idte。

先保存源中断服务程序地址 orig_function。假设截获函数地址为 intercept_function,在截获函数中设有跳转语句转入源中断服务程序。则可用如下语句修改中断描述符地址 idte:

```
idte -> offset_high = (unsigned short) (intercept_function > >16);
idte -> offset_low = (unsigned short) (intercept_function &
 0x0000FFFF);
```

这样,当模块加载后就修改了 INT 80 中断描述符地址,使其指向 intercept_function,截获模块开始进行系统调用截获工作。这时系统调用流程图如图 2 所示。

在截获模块中先获得当前进程的进程名 current -> comm.[6],判断若为需监控的程序,就将系统调用号及参数写入数据库中,否则直接转入源中断服务程序。

图 2 系统调用流程

这样一个设计思想具有以下几个显著优点:

1)实时性强。由于是在系统调用刚进入内核时截获系统调用,因此当处于检测模式时可在某项活动发生但还未对系统造成实际影响时就能分析并预知它的动作,这为监控程序行为提供了一个很好的平台。

2)监控高效,准确。由于是通过修改中断向量表的方式来实现,对信息的提取工作是用汇编码进行寄存器相关操作来完成,运行速度快,且能快速准确识别需要监控的程序。

3)监控具有透明性,易用性。采用 LKM 方式来实现,易于加载。对程序员及用户具有透明性。

4)实现方便,可利用信息量大,能很好的满足要求。

3 程序行为的分析

由于程序功能及运行环境的复杂性,系统调用序列串表现出一定的变化性,但其系统调用序列具有局部一致性[1]。因此 Forrest 等人提出了利用滑动窗口机制来分割系统调用序列对程序行为进行分析,通过大量的训练活动来提取程序在正常运行情况下所发出的系统调用短序列,并将这些系统调用短序列以一定算法存入数据库中作为程序行为的正常行为库[2]。实验表明这种方式能较好地刻画程序的正常行为,但它也具有一定的不精确性。原因在于程序行为不仅体现在不同系统调用间的先后次序上,即使同一个系统调用由于其所带参数的不同也会让程序表现大不一样。因此准确分析程序行为必须充分考虑系统调用参数(系统调用序列串分析不是本文讨论范围)。本文提出的 LKM 方法能很方便地获得系统调用参数,这为参数的分析提供了前提条件。然而系统调用参数常常很多,并且各类型参数混杂在一块,无一定规律可循。既使同一个系统调用的不同类型参数也随系统调用的出现而发生变化。这使得从某一个或几个特定参数来分析程序行为不太可行。因此本文将从参数的各种特性分布规律上来对程序行为进行分析。

3.1 对系统调用中字符串参数长度分布的分析

通常系统调用参数由可读字符组成,并且其很少超过一定的长度[3](比如说 100)。当某一个恶意输入传递给程序时,这个恶意输入也常常会出现在系统调用参数中。恶意的程序输入通过堆栈溢出获得超级用户身份,控制程序运行。因此用户传给程序的恶意参数较长(大于预先分配的缓冲区长度),根据这一常识,采用如下算法来检测参数长度异常:

1)通过在大量的训练活动中获得用户输入的正常参数长度作为样本,然后计算这些样本长度的均值 u 和方差 σ,并将它们作为参数长度的均值和方差。

2)在检测中计算用户某一输入的参数长度 l,若 $l > u$,则认为可能存在长度异常,计算参数长度异常值 p:

$$p = \frac{(l-u)^2}{\sigma^2}$$

当 p 的值大于某一给定值(例如取 1.2)时即视为可能存在参数异常。

3)结合对参数的字符特征分布的分析(下面提到)来判断程序是否异常。

3.2　对字符串参数的字符特征分布的分析

经观察知在程序运行过程中同一系统调用所带参数在长度及内容上往往具有一定的相类性。因此为了能够正确识别正常的系统调用参数,防止诸如在调用参数中隐藏恶意代码引发堆栈溢出之类的情形,对字符串参数的字符特征分布进行分析,分别计算系统调用参数中每个字符 r 在本参数中出现的频率值 p_r。算法如下:

1) 通过大量实验,可以得到各参数中字符 r 的一系列频率值 $p_{r1}, p_{r2}, \cdots, p_{rm}$($p_{ri}$ 即第 i 个参数中字符 r 出现的频率值,当 r 在参数中不出现时不统计在内)。实验发现频率值 p_r 通常较小且较接近。

$$pr = \frac{\text{字符串中含 } r \text{ 的个数}}{\text{字符串的长度}}$$

2) 由此可计算每个字符 r 频率值 p 的样本均值 u_p 和样本方差 σ_p 作为频率 p_r 的均值 u 和方差 σ。

$$u_p = \frac{1}{n}\sum_{i=1}^{n} p_{ri}$$

$$\sigma_p^2 = \frac{1}{n-1}\sum_{i=1}^{n} (p_{ri} - u_p)^2$$

3) 在检测阶段中,若某个参数中的字符 r 出现的频率过大或过小都将视为参数异常,即不在区间 $[u_p - \alpha_p(1+a), u_p + \sigma_p(1+a)]$ 内时($0 \leqslant a < 1$,a 根据在实际应用中的灵敏度情况来调整其大小),系统将认为它可能是恶意参数,这时进一步结合参数长度异常值来决定程序行为是否异常。这样,在实验中输入只含字母 a 的字符串"aaaaaa",尽管频率值 p_a 为1,但由于参数长度($l = 6$)无异常(假定参数长度均值 $u = 7$),系统不会报警。但在一串正常字符串后紧跟一串恶意代码,这样就肯定会引起大量字符的特征分布异常并常伴随产生字符串长度异常,系统产生报警。

3.3　对特殊系统调用参数的分析

在程序中存在有一些对入侵至关重要的系统调用,其中一些系统调用的参数是 int, mode_t 型等非字符串形式,用上述分析方法无法对其进行监控。因此对这些重要系统调用的参数分开对待就很有必要。下面为本文认为应单独列出对参数作特殊分析的系统调用: getuid; setuid; sys_chmod; sys_chown; sys_fchmod; sys_fchown; sys_chroot; sys_execve; sys_fork; sys_fcntl; sys_open; sys_delete_module; sys_rename。

对每一条关键系统调用的参数应分开建立规则库。规则库的建立视不同系统调用而不同。比如 sys_setuid 系统调用常被黑客利用来获得超级用户身份。通过对 sys_setuid 进行单独管理使我们在大量训练中可获知程序正常运行中哪些 uid 能获得超级用户身份,任何通过非法手段来改变其他 uid 用户的身份都会被及时发现,作为程序异常而被阻断。规则如下:

```
setuid(uid) |(uid not in {...})
  exit(-1)
```

有些系统调用各参数间具有较大的相关性,将这些参数结合起来进行检验是十分必要的。比如对于系统调用 sys_open,若在大量的训练活动中发现被打开的文件 myfile 都以只读方式打开,在检查参数时我们就可认为只读方式打开 myfile 为正常程序行为,以读写方式打开 myfile 将被视为程序异常而被阻断。甚至可以将在程序运行过程中涉及到的用于保存有十分重要信息的文件(比如说 passwd),通过在大量训练中学习到有哪些用户可对其进行读写操作而用单独的规则保护起来。

通常一个程序从开始运行到结束所涉及到的不同系统调用只有几十个,并且以上这些重要的系统调用不会大量重复出现。因此对这些重要的系统调用进行分开管理并不会严重影响系统性能,还极大提高了程序运行安全性。

当然,对于系统调用序列串的分析和对系统调用参数的分析并不是分开进行的,它们应是一个统一的整体。用系统调用序列串来对程序行为进行分析能识别大多数的异常程序行为,但有较大的误报率。本文通过引入系统调用参数的分析丰富了检测手段,以较小的性能开销极大的降低了误报率,为入侵检测提供了很好的理论和实践基础。

4　实验及结果分析

对 wuftpd, Apache, sendmail, linuxconf 四个程序分别在有参数检测和无参数检测两种状态下进行实验。两次检测都用本实验系统生成的同一系统调用序列串数据库,滑动窗口大小取为 7。在实验环境中对这四个程序分别进行数小时的系统调用及参数提取分析工作后,再分别进行缓冲区溢出攻击,结果如表 1 所示。

表 1　实验结果

	攻击数	检测到攻击数(无参数检测)	误报率(无参数检测)	检测到攻击数(带参数检测)	误报率(带参数检测)
Apache	3	3	3	3	0
sendmail	5	4	6	5	1
linuxconf	3	2	5	3	1
wuftpd	3	3	7	3	2

从表 1 可以看出,在增加对系统调用参数的分析检测工作后不仅提高了检测到的攻击数,还大大降低了误报率。

因此本文提出的应在系统调用序列串分析的基础上进一步分析其参数,建立多个参数模型的方法加强了对程序异常的检测,设计思想具有如下特点:

1) 程序行为提取具有可靠性,高效性。在系统调用的入口处拦截系统调用,进行行为提取分析,这保证了高效性和可靠性。

2) 系统具有易使用性。以模块实现,可方便地加载和卸载,整个行为提取分析过程不需人工干预。

3) 对程序异常的检测精确度高。在对系统调用序列串分析的基础上进一步分析其参数,以多个模型匹配降低了误报率。

参考文献:

[1] FORREST S, HOFMEYR SA, SOMAYAJI A, *et al.* A Sense of Self for UNIX Processes[EB/OL]. http://www.cc.gatech.edu/~wenke/ids-readings/unix_process_self.pdf, 1996.

[2] HOFMEYR SA, FORREST S, SOMAYAJI A. Intrusion Detection using Sequences of System Calls[J]. Journal of Computer Security, 1998, 6(3): 151-180.

[3] KRUEGEL C, MUTZ D, VALEUR F, *et al.* On the Detection of Anomalous System Call Arguments[A]. Proceedings of ESORICS 2003[C]. Gjovik, Norway, 2003.

[4] 资武成, 徐鹏飞. 一种基于系统调用的主机入侵检测及实现[J]. 计算机与现代化, 2003, (1): 44-45.

[5] 陈莉君, 冯锐, 牛欣源, 译. 深入理解 Linux 内核[M]. 北京: 中国电力出版社, 2001. 122-123.

[6] 毛德操, 胡希明. Linux 内核源代码情景分析[M]. 浙江: 浙江大学出版社, 2000. 270-271.