Journal of Computer Applications
计算机应用,2013,33(9):2520 - 2524
ISSN 1001-9081
CODEN JYIIDU
2013-09-01
http://www.joca.cn

文章编号:1001-9081(2013)09-2520-05
doi:10.11772/j.issn.1001-9081.2013.09.2520

基于函数级控制流监控的软件防篡改

张贵民*,李清宝,王　炜,朱　毅
(数学工程与先进计算国家重点实验室(信息工程大学),郑州 450000)
(*通信作者电子邮箱 zh.guimin@163.com)

摘　要:软件防篡改是软件保护的重要手段。针对由缓冲区溢出等攻击导致的控制流篡改,提出一种基于函数级控制流监控的软件防篡改方法。以函数级控制流描述软件正常行为,利用二进制重写技术在软件函数入口处植入哨兵,由监控模块实时获取哨兵发送的软件运行状态,通过对比运行状态和预期值判断程序是否被篡改。实现了原型系统并对其进行了性能分析,实验结果表明,基于函数级控制流监控的软件防篡改方法能有效检测对控制流的篡改攻击,无误报且开销较低,其实现不依赖程序源码,无需修改底层硬件和操作系统,监控机制与被保护软件隔离,提高了安全性。

关键词:软件防篡改;函数级控制流;二进制重写;哨兵;可信平台模块
中图分类号:TP309.1;TP309.5　　**文献标志码**:A

Software tamper resistance based on function-level control-flow monitoring

ZHANG Guimin*, LI Qingbao, WANG Wei, ZHU Yi
(*State Key Laboratory of Mathematical Engineering and Advanced Computing*
(*Information Engineering University*), *Zhengzhou Henan* 450000, *China*)

Abstract: Software tamper resistance is an important method for software protection. Concerning the control-flow tampering invoked by buffer overflow as well as some other software attacks, a software tamper-proofing method based on Function-Level Control-Flow (FLCF) monitoring was proposed. This method described the software's normal behaviors by FLCF and instrumented one guard at every entrance of functions by binary rewriting technology. The monitoring module decided whether the software was tampered or not by comparing the running status received from the guards' reports with the expected condition. A prototype system was realized and its performance was analyzed. The experimental results show that this method can effectively detect the control-flow tampering with less overhead and no false positives. It can be easily deployed and transplanted as its implementation does not need source code or any modifications of underlying devices, and system security is strengthened by isolating the monitoring module with the software being protected.

Key words: software tamper resistance; Function-Level Control-Flow (FLCF); binary rewriting; guard; Trusted Platform Module(TPM)

0　引言

随着计算机和互联网的迅速发展,计算机软件得到广泛应用,同时软件安全性问题也日益突出。由于软件中存在各种漏洞,如缓冲区溢出(buffer overflow)漏洞[1]、格式化字符串(format string)漏洞等,使得攻击者能够利用这些漏洞实现对软件行为的篡改,达到获取用户个人信息、执行恶意代码等目的。如何保护软件不受非法篡改,保证其按照预定的方式正常运行,已成为软件安全保护领域的一个重要研究内容。

软件防篡改技术可分为两大类:静态防篡改技术,基于代码变换思想,通过代码变换增加软件被篡改的难度;动态防篡改技术,基于检测-响应的思想,通过增加软件或硬件措施使被非法篡改后的程序无法正常运行[2]。动态防篡改技术研究的一个重要方向是借助软件行为分析和实时监控技术,通过对比软件运行的实时信息和预期行为模型判断篡改是否发生。

控制流完整性(Control-Flow Integrity, CFI)方法[3]是一种通过保证控制流的完整性来实现软件防篡改的方法。该方法利用二进制重写技术向软件函数入口及调用返回处分别插入标识符ID和ID_check,通过对比ID和ID_check的值是否一致判断软件的函数执行过程是否符合预期,从而判断软件是否被篡改。该方法虽能防范多种类型攻击,但实现复杂,且由于判断软件是否被篡改的全部逻辑都在该软件自身中存在,故该防护机制易于被攻击者发现和利用。强制数据流完整性(Data-Flow Integrity, DFI)是另一种通过保证数据流完整性来防范软件被篡改的方法[4]。该方法首先对程序进行变量到达定义分析生成每个变量的到达定义集合,即静态数据流图(static data-flow graph),然后根据数据流图向程序中插入检测代码,通过该功能代码在每一个变量使用时判断该变量的最近定义位置是否在预先获取的到达定义集中,并由此判断软件是否被篡改。该方法可防范部分非控制数据和控制数据攻击,但由于采用相对于控制流分析更复杂的数据流分

收稿日期:2013-03-27;**修回日期**:2013-05-12。　**基金项目**:国家核高基项目(2013JH00103)。
作者简介:张贵民(1987 -),男,山东济南人,硕士研究生,主要研究方向:信息安全、可信计算;　李清宝(1967 -),男,四川乐山人,教授,博士生导师,博士,CCF会员,主要研究方向:计算机系统结构、信息安全、可信计算;　王炜(1975 -),男,湖北武汉人,讲师,博士,CCF会员,主要研究方向:计算机系统结构、信息安全、可信计算;　朱毅(1986 -),男,河北石家庄人,助理工程师,硕士研究生,主要研究方向:计算机系统结构、信息安全、可信计算。

析、分析过分依赖程序源码、时间和空间开销较大等问题限制了其应用。另外,它同样存在整个保护机制都设置于程序本身这一安全隐患。

1996 年 Forrest[5] 提出利用系统调用序列对软件行为建模的方法,并于 1998 年实现了一个基于系统调用序列模型 N-gram 的软件入侵检测系统[6]。此后,利用系统调用建模检测攻击和篡改[7-9]的方式不断得到发展和完善。该方式的实现要对系统内核进行修改以实时获取系统调用信息,且大多数实现只是单纯获取系统调用本身,检测能力有限,难以抵抗复杂攻击,如模仿攻击、数据流攻击等[10]。为提高利用系统调用检测攻击的精确性,一些研究者不再仅仅关注系统调用本身,而是将它与系统调用的相关参数等数据信息结合起来进行研究[8-9],但此类模型实现过于复杂且系统开销很大,实用性不高。如何在保证精度的情况下尽可能降低监控开销,是目前基于系统调用建模防篡改面临的主要问题。

本文提出一种基于函数级控制流(Function-Level Control-Flow, FLCF)监控的软件防篡改方法。该方法以二进制可执行文件作为研究对象分析获得软件函数级控制流图(Control-Flow Graph, CFG),利用二进制重写技术向二进制可执行文件的每个函数入口植入监控代码(以下称哨兵),哨兵在软件运行时向监控模块实时报告当前软件运行状态。监控模块将哨兵报告的运行状态与软件控制流图中函数预期运行状态进行对比,如对比发现当前运行状态不满足预期,监控模块即向用户发出警告,告知用户该软件已被篡改。

该方法主要有以下几个特点:1)直接对二进制可执行文件进行操作,不依赖源码;2)利用二进制重写植入哨兵获取软件运行状态信息,不需对底层硬件和操作系统进行任何修改;3)无误报,如果用户接收到一个警告,那么软件的控制流必然遭到了篡改;4)监控机制与被保护软件隔离,增强了该保护机制的安全性。

1　函数级控制流监控防篡改

不论攻击者以何种方式对软件进行篡改,最终篡改成功都体现于控制流的改变上[11],所以监控软件控制流是实现软件防篡改的一种有效手段。该监控方式需解决两个核心问题:一是如何获取软件实时运行状态;二是如何根据所得状态信息判断软件是否被篡改。

针对第一个问题,控制流完整性防篡改和数据流完整性防篡改都是向软件中植入监控代码获取软件实时运行信息;系统调用建模防篡改则修改系统内核监控软件运行时的系统调用信息。监控代码植入方式能避免对底层操作系统的修改,降低系统实现复杂度,提高系统可移植性,但同时该方式也需考虑多个因素的影响,包括操作对象(源文件或二进制文件)选择、代码植入位置和数量的选择以及如何降低监控代码对软件性能的影响。对于第二个问题,如对软件控制流进行监控,选择将控制流图作为判断标准是合理的,关键问题在于如何对实时信息和控制流图进行对比以提高检测精度和速度。

1.1　函数级控制流图

控制流图是描述软件行为的一种重要方法。它可通过源文件或二进制文件静态分析获取,也可通过动态训练方式获取。为摆脱源码依赖,选择二进制文件分析方式。

从二进制可执行文件获取控制流图[12-13]的研究成果较多,且不是本文研究重点,不再赘述。软件控制流图的描述可分为细粒度和粗粒度两种:细粒度(指令级)控制流图虽能更全面更细致地描述软件运行过程,但获取难度和监控开销都比较大;粗粒度控制流图虽降低了精度,但监控开销明显减少,且函数作为程序的基本功能模块,对函数级控制流图的监控即可基本满足防篡改需求。基于上述原因,选择函数级控制流图描述软件正常行为。图 1 所示为含缓冲区溢出漏洞的程序 example 的函数级控制流图。

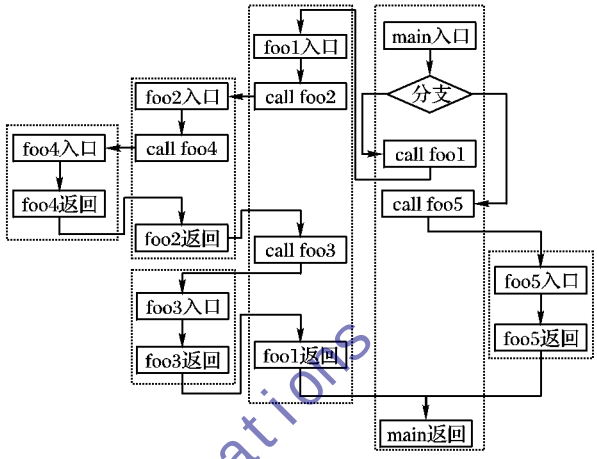

图 1　程序 example 的函数级控制流图

1.2　标准信息存储和保护

获取程序的函数级控制流图,是为了将该图所含信息作为软件是否被篡改的判断标准,所以对该类信息的存储和保护是保证整个监控系统正常运作的基础。

采用树双亲表示法存储程序函数级控制流图中的信息,该存储方式可以快速地随机获取某个节点的直接前驱。首先将函数级控制流图转化为信息树的形式,例如图 1 所示函数级控制流图转化后如图 2 所示。

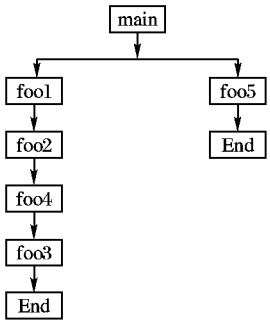

图 2　程序 example 的信息树

双亲表示法节点结构定义如下:

```
typedef struct node
{
  string nodeData;                          //存放函数名
  int parent;                    //本节点双亲节点在数组中的位置
}TreeNode;
```

标准值在存储后通过采用可信平台模块(Trusted Platform Module, TPM)的可信验证机制防止其被非法篡改。TPM 是可信计算的核心部件,是一种具有密码运算和存储能力的安全芯片。在将某个程序的标准信息按照双亲表示法存储之后,首先调用 TPM 的 hash 函数对标准信息进行处理获取 hash 值,并将其摘要值存放于某个平台配置寄存器(Platform Configuration Register, PCR)中。当要获取标准信息与动态信息比对时,首先重新计算其摘要值,并与存放于 PCR 中的原值比较,如果两者一致,说明标准信息没有被篡改,然后即可读取进行比较;否则,标准信息被破坏并发送错误报告。

1.3 程序运行状态获取

本文中,程序运行状态即程序的函数执行序列。为实时获取软件运行状态、监控软件行为,采取向软件中植入哨兵的方式。

为保证每个函数在真正运行之前都经过正确性验证,选择在每个函数入口处(如图 1 中所示)进行哨兵植入,哨兵执行完成之后函数才能继续执行。下面分别展示了程序 example 中函数 foo1 在哨兵植入前后的反汇编代码。其中哨兵植入前的反汇编代码为:

```
0804849c <_Z4foo1v>:
804849c:   55                    push   %ebp
804849d:   89 e5                 mov    %esp,%ebp
804849f:   83 ec 18              sub    $0x18,%esp
80484a2:   c7 04 24 0c 87 04 08  movl   $0x804870c,(%esp,1)
80484a9:   e8 46 ff ff ff        call   80483f4 <_init +0x80>
80484ae:   e8 1d 00 00 00        call   80484d0 <_Z4foo2v>
80484b3:   89 45 fc              mov    %eax,0xfffffffc(%ebp)
80484b6:   e8 2d 00 00 00        call   80484e8 <_Z4foo3v>
80484bb:   8b 45 fc              mov    0xfffffffc(%ebp),%eax
80484be:   89 44 24 04           mov    %eax,0x4(%esp,1)
80484c2:   c7 04 24 1e 87 04 08  movl   $0x804871e,(%esp,1)
80484c9:   e8 26 ff ff ff        call   80483f4 <_init +0x80>
80484ce:   c9                    leave
80484cf:   c3                    ret
```

哨兵植入后的反汇编代码为:

```
0804849c <_Z4foo1v>:
804849c    e9 bb 83 00 00        jmp    805085c <foo1_dyninst>
80484a1    cc                    int3
...
80484ad    cc                    int3
80484ae    e8 1d 00 00 00        call   80484d0 <_Z4foo2v>
80484b3    89 45 fc              mov    %eax,0xfffffffc(%ebp)
80484b6    e8 2d 00 00 00        call   80484e8 <_Z4foo3v>
80484bb    8b 45 fc              mov    0xfffffffc(%ebp),%eax
80484be    89 44 24 04           mov    %eax,0x4(%esp,1)
80484c2    c7 04 24 1e 87 04 08  movl   $0x804871e,(%esp,1)
80484c9    e8 26 ff ff ff        call   80483f4 <targ80483f4>
80484ce    c9                    leave
80484cf    c3                    ret
```

通过对比函数 foo1 在哨兵植入前后的反汇编代码可知,哨兵的实现方式可简单描述为在函数入口处植入一条跳转指令 jmp,由该指令将软件控制流引向统一的哨兵代码起始位置并执行,在哨兵执行完成后再跳回函数入口处继续执行。

当一个刚刚调度到的函数开始执行时,首先要运行哨兵代码,哨兵代码中维护一个长度为 2 的字符串数组 func,每执行一个函数,哨兵就将函数名赋给 func[i]($0 \leqslant i \leqslant 1$),当 $i=1$ 时,哨兵即向监控模块发送一个报告 R = < currentFunc, formerFunc >,其中 currentFunc 是当前正要开始运行的新函数,即 func[1], formerFunc 则记录了在 currentFunc 执行之前所执行的函数(两者之间可能是调用关系也可能不是),即 func[0]。报告发送后,i 置零,并令 func[0]等于原 func[1],以实现报告内容的连续性。

由此可知,哨兵发送的状态报告就是对当前软件运行情况的一种简单描述。通过获取并分析报告内容,监控模块即可掌握软件函数的实时运行信息,对于如何利用该信息判断篡改是否发生,详细过程将在下一节中论述。

1.4 篡改检测

获取控制流图是为明确软件正常运行过程,哨兵植入是为获取软件实时运行状态,而如何根据状态信息准确判断软件是否被篡改,即如何解决上文提到的第二个核心问题则是监控模块要实现的。

不同于 CFI 和 DFI 的监控方式,监控模块是独立于被监控软件的另一个程序,实现了监控机制和被监控软件的分离,提高了监控模块的安全性。它的功能就是实时获取哨兵所发送的报告 R,然后通过 TPM 模块验证标准值信息库是否完整可信,如可信再从库中提取对应的标准值,即获取 currentFunc 的双亲节点,如果结果和 formerFunc 一致则说明软件运行正常,否则软件控制流被篡改。

所采用报告中不含检测点之前整个函数运行序列[6,9],只有当前函数及其前函数,这不仅能降低监控开销,且可证明仍满足监控需求。对于任意一个程序,假设其所含函数用 f_1, f_2, …, f_{n-1}, f_n(n 为非零自然数,f_1 为软件运行的第一个函数)表示,可得出以下结论:如在软件运行过程中检测到报告 R 异常,则软件被篡改。在证明这个结论之前首先要明确如何判断一个报告 R 是否异常。

评判标准 对于获得的任意报告 R = < currentFunc, formerFunc >,如经验证后发现 currentFunc 的直接前驱为 formerFunc,那么报告正常;否则报告异常。

证明 当检测到报告 $R = < f_{x+1}, f_x >$ ($1 \leqslant x \leqslant n-1$) 异常时,可知 f_x 不是 f_{x+1} 的直接前驱,那么 f_x 和 f_{x+1} 在控制流图中只存在两种关系:一是 f_x 到 f_{x+1} 不存在路径相连通;二是 f_x 到 f_{x+1} 有路径连通,但软件运行时在两者之间还会执行其他函数。由哨兵的报告机制可知,在这两种情况下都不会产生报告 $R = < f_{x+1}, f_x >$,故软件的控制流已经发生了篡改。综上,结论成立。

监控模块接收到报告后,即检查报告是否异常。如果没有发现报告异常,监控模块不作任何响应,程序正常运行;否则,监控模块向用户发送警告,告知用户该程序已经被非法篡改,然后由用户对该程序进行相应处理。且由上文已证明的结论可知,只要监控模块发出警告,软件就一定发生了控制流篡改,即误报率为零。

2 原型系统实现

在 Linux 环境下对基于函数级控制流监控的软件防篡改方法进行了原型实现,其总体架构如图 3 所示。

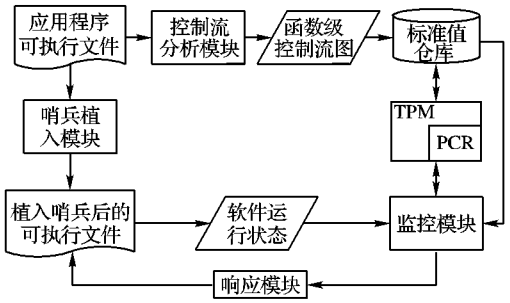

图 3 系统总体架构

对需要保护的软件,首先通过控制流分析模块获取其函数级控制流图,将函数级控制流图中的信息转化为信息树后存储在标准值仓库中。标准值仓库的内容通过 TPM 提供的 hash 函数处理获取 hash 值,并将其摘要存放于 PCR 中。植入哨兵的软件在执行时,监控模块获取哨兵的报告内容,当要获取标准值与动态信息对比时,首先要验证标准值仓库的完整性。监控模块调用 TPM 再次对标准值仓库进行可信性验证,

比较再次度量的摘要值和 PCR 保存的原值是否一致。如果不一致,响应模块即发出错误报告,提示用户标准值仓库被篡改;如果一致,监控模块再从标准值仓库中提取出对应的标准值与动态报告内容对比,根据对比结果判断软件是否被篡改。如发现软件控制流被篡改,响应模块发出篡改报告,并终止该软件的执行(响应模块的动作可以按需求定制)。

3　性能分析

在建立原型系统的基础上,从有效性、系统开销和兼容性方面对本文提出的基于函数级控制流监控的软件防篡改方法进行了测试分析。

3.1　有效性测试

使用图 1 中的 example 程序为例,首先获取控制流图(如图 1 所示),然后对程序进行哨兵植入。该程序函数 foo2 中含有缓冲区溢出漏洞,通过利用缓冲区溢出漏洞,修改 foo2 返回地址,这也是缓冲区漏洞攻击的一种普遍方式[14]。通过修改返回地址,使程序跳过 foo2 之后的函数 foo3 继续执行,从而篡改原程序 example 的控制流。最后运行被篡改的程序验证系统能否检测到控制流篡改。

下面展示了实验中哨兵在 example 正常运行时和控制流被篡改后发送的不同报告序列,由于正常执行时该程序有两条执行路径(如图 1 所示),所以对应哨兵报告序列也有两种。其中正常报告序列一为:

(__libc_csu_init, start)
(_init, __libc_csu_init)
(frame_dummy, _init)
(__do_global_ctors_aux, frame_dummy)
(main, __do_global_ctors_aux)
(foo5, main)
(_fini, foo5)
(__do_global_dtors_aux, _fini)

正常报告序列二为:

(__libc_csu_init, start)
(_init, __libc_csu_init)
(frame_dummy, _init)
(__do_global_ctors_aux, frame_dummy)
(main, __do_global_ctors_aux)
(foo1, main)
(foo2, foo1)
(foo4, foo2)
(foo3, foo4)
(_fini, foo3)
(__do_global_dtors_aux, _fini)

被篡改后的报告序列为:

(__libc_csu_init, start)
(_init, __libc_csu_init)
(frame_dummy, _init)
(__do_global_ctors_aux, frame_dummy)
(main, __do_global_ctors_aux)
(foo1, main)
(foo2, foo1)
(foo4, foo2)
(_fini, foo4)
(__do_global_dtors_aux, _fini)

其中状态 start 是自己定义的初始状态,而程序在进入 main 函数之前执行函数序列为 start→_libc_csu_init →_init→frame_dummy→_do_global_ctors_aux→main,程序执行完 main 函数之后执行函数序列为_fini→_do_global_dtors_aux,这都是 Linux 系统中运行程序时的固有流程。

监控程序在 example 控制流被篡改时,即在(_fini, foo4)处成功检测到报告异常(由图 1 可知,紧跟 foo4 运行的只有 foo3,而非_fini),且抛出警告。

根据整个防篡改机制的设计原理和实现方式,并结合该实验结果,可以表明基于函数级控制流监控的软件防篡改技术对于函数级篡改是有效的,能够成功检测出控制流篡改并发出警告。

基于函数级控制流监控的软件防篡改方法采用函数级控制流图作为判断标准,能有效检测出函数级控制流篡改,但对于少数函数单元内部内容的篡改会存在漏报,这类篡改可通过在监控中加入函数执行参数等数据流信息来防范,这也是未来的研究内容。

3.2　系统开销

该系统的时间开销可分为两部分:预处理阶段和软件实际运行阶段。预处理阶段需要对可执行文件分析获取控制流图,并向可执行文件中植入哨兵,这部分时间开销比较大。但预处理阶段开销与软件处理完成后的实际执行开销没有必然联系,能对软件运行开销产生影响的是哨兵发送状态报告所需时间。为评估程序在哨兵植入前后时间开销变化情况,使用 SPEC CPU2006 Benchmark 中 CFP2006 基准测试集进行测试,表 1 列出了各测试程序被植入哨兵数目,图 4 为哨兵植入后运行时间变化情况,图中数值为程序在哨兵植入后运行时间增加部分与原运行时间的比值。

表 1　各基准测试程序被植入的哨兵数目

程序	语言	插桩数目
410. bwaves	Fortran	16
416. gamess	Fortran	2 879
433. milc	C	244
434. zeusmp	Fortran	86
435. gromacs	C/Fortran	3 879
436. cactusADM	C/Fortran	1 402
437. leslie3d	Fortran	30
444. namd	C ++	161
450. soplex	C ++	1 600
454. calculix	C/Fortran	1 393
459. GemsFDTD	Fortran	108
465. tonto	Fortran	4 092
470. lbm	C	29
482. sphinx3	C	380

根据实验结果,约 80% 测试程序在哨兵植入后运行时间增加 50% 左右,所有测试程序时间开销平均增加 63%。从时间开销上来说,这与控制流完整性防篡改(开销增加低于 45%[3])相比略大;而与采用数据流分析的数据流完整性防篡改方法(平均时间开销可高达 103%[4],未考虑数据流分析所需的大量时间)和函数调用建模防篡改方法(仅用于截获系统调用的开销就可高达 250%[7])相比则要小很多。从空间开销上来说,控制流完整性防篡改需在每个函数的入口和出口处均植入代码(ID 和 ID_check),而基于函数级控制流监控的软件防篡改方法只需在调用函数入口处植入哨兵,故后者哨兵植入数目仅是前者的一半;由于数据流完整性方法不仅需要维护每个变量的到达定义集合,还需在每个变量使用处植入判断代码,使其空间开销可高达 50%[4],而基于函数级控制流监控的软件防篡改方法由于将哨兵功能代码通过动

态链接库实现,使所有哨兵共享此部分代码,在被保护软件中仅植入跳转指令,这些跳转指令相对于整个软件来说所产生的空间开销是微不足道的,远低于前者的 50%。

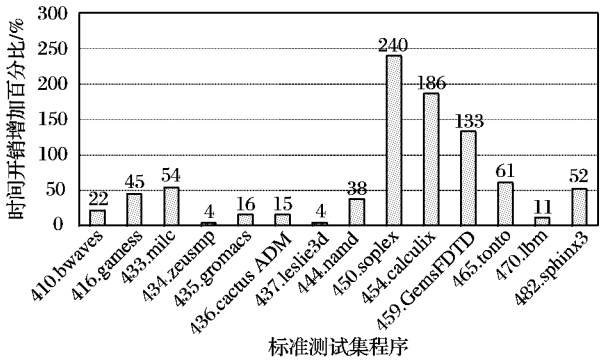

图 4 各基准测试程序植入哨兵后的运行时间

3.3 系统兼容性

该系统在进行哨兵植入时借助 Dyninst[15] 插桩框架和在其基础上实现的 Cobi[16] 可配置二进制插桩器实现。由于 Cobi 目前只能实现对 ELF 文件的分析操作,本系统目前只支持 Linux 操作系统。由于系统实现不需改变底层硬件和操作系统,该系统可在使用 Linux 操作系统的主机上快速移植、配置和应用。

4 结语

软件防篡改作为软件保护的一种重要手段,在软件保护领域具有重要作用和地位。为了防止控制流被篡改,本文提出一种基于函数级控制流监控的软件防篡改方法。方法为解决软件标准行为信息的存储问题,提出了信息树的标准行为描述方式,并借助 TPM 可信机制保证该信息的安全性。另外,方法通过采用植入哨兵并由哨兵发送软件运行状态报告的方式实现了软件行为信息的动态提取。最终在 Linux 系统上实现了一个原型系统,并验证了方法的有效性。下一步工作的重点是在现有方法的基础上,结合数据流分析和监控技术以提高对软件篡改的检测精度。

参考文献:

[1] ONE A. Smashing the stack for fun and profit[J]. Phrack, 1996, 7(49): 14 - 15.

[2] 王朝坤, 付军宁, 王建民, 等. 软件防篡改技术综述[J]. 计算机研究与发展, 2011, 48(6): 923 - 933.

[3] ABADI M, BUDIU M, ERLINGSSON U. Control - flow integrity principles, implementations, and applications[J]. ACM Transactions on Information and System Security, 2009, 13(1): 1 - 40.

[4] CASTRO M, COSTA M, HARRIS T. Securing software by enforcing data-flow integrity[C]// Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Washington, DC: USENIX Association, 2006: 147 - 160.

[5] FORREST S. A sense of self for UNIX processes[C]// Proceedings of the 1996 IEEE Symposium Security and Privacy. Piscataway, NJ: IEEE Press, 1996: 120 - 128.

[6] HOFMEYR S A, FORREST S, SOMAYAJI A. Intrusion detection using sequences of system calls[J]. Journal of Computer Security, 1998, 6(3): 151 - 180.

[7] SEKAR R, BENDRE M, BOLLINENI P, *et al*. A fast automaton-based method for detecting anomalous program behaviors[C]// Proceedings of the 2001 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE Press, 2001: 144 - 155.

[8] 李闻, 戴英侠, 连一峰, 等. 基于混杂模型的上下文相关主机入侵检测系统[J]. 软件学报, 2009, 20(1): 138 - 151.

[9] MAGGI F, MATTEUCCI M, ZANERO S. Detecting intrusions through system call sequence and argument analysis[J]. IEEE Transactions on Dependable and Secure Computing, 2010, 7(4): 381 - 395.

[10] 陶芬, 尹芷仪, 傅建明. 基于系统调用的软件行为模型[J]. 计算机科学, 2010, 37(4): 151 - 157.

[11] BLIETZ B, TYAGI A. Software tamper resistance through dynamic program monitoring[C]// SAFAVI-NAINI R, YUNG M. Digital Rights Management: Technologies, Issues, Challenges and Systems. Berlin: Springer, 2006: 146 - 163.

[12] KINDER J, VEITH H, ZULEGER F. An abstract interpretation-based framework for control flow reconstruction from binaries[C]// Proceedings of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation. Berlin: Springer, 2009: 214 - 228.

[13] XU L, SUN F Q, SU Z D. Constructing precise control flow graphs from binaries[R]. Davis: University of Computer Science, 2009.

[14] PADMANABHUNI B, TAN H. Techniques for defending from buffer overflow vulnerability security exploits[J]. IEEE Internet Computing, 2011, 44(11): 53 - 60.

[15] BERNAT A R, MILLER B P. Anywhere, any-time binary instrumentation[C]// Proceedings of the 10th ACM SIGPLAN- SIGSOFT Workshop on Program Analysis for Software Tools. New York: ACM Press, 2011: 9 - 16.

[16] MUSSLER J, LORENZ D, WOLF F. Reducing the overhead of direct application instrumentation using prior static analysis[C]// Proceedings of the 17th International Conference on Parallel Processing. Berlin: Springer, 2011: 65 - 76.

(上接第 2519 页)

[4] KAMAT P, BALIGA A, TRAPPE W. An Identity-based security framework for VANETs[C]// Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks. New York, Press, 2006: 94 - 95.

[5] CALANDRIELLO G, PAPADIMITRATOS P, HUBAUX J P, *et al*. Efficient and robust pseudonymous authentication in VANET[C]// Proceedings of the 4th International Workshop on Vehicular Ad Hoc Networks. New York: ACM Press, 2007: 19 - 28.

[6] KIDO H, YANAGISAWA Y, SATOH T. A anonymous communication technique using dummies for location-based services[C]// ICPS 2005: Proceedings of International Conference on Pervasive Services 2005. Piscataway, NJ: IEEE Press, 2005: 88 - 97.

[7] GHINTA G, KALNIS P, KHOSHGOZARAN A, *et al*. Private queries in location based services: anonymizers are not necessary[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2008: 121 - 132.

[8] 徐建, 黄孝喜, 郭鸣. 动态 P2P 网络中基于匿名链的位置隐私保护[J]. 浙江大学学报: 工学版, 2012, 46(4): 712 - 718.

[9] 王岳. 稀疏车载网络路由算法的研究[D]. 桂林: 广西师范大学, 2012.

[10] 陈万顺. 基于 Chord 分布式哈希表的网络过载均衡方法[J]. 常州工学院学报, 2012, 25(6): 9 - 12.

[11] 路红. 物联网空间 LBS 隐私安全模型研究[D]. 武汉: 华中师范大学, 2012.

[12] 郭艳华. 位置服务中轨迹隐私保护方法的研究[D]. 武汉: 华中师范大学, 2011.