软件哨兵安全动态检测模型的研究与实现

武少杰[1]，鹤荣育[1]，薛长松[2]，谢江涛[1]
（1. 信息工程大学 电子技术学院，郑州 450004；2. 成都 95879 部队，成都 610081）

摘　要：软件哨兵是保护软件代码不被非法修改的重要方法之一，但它本身的安全面临着重大挑战。为了能动态检测软件哨兵是否遭到篡改，基于 TPM 技术提出了一个检测软件哨兵完整性的模型。该模型利用哨兵进程在内存页面中的变化情况，判断哨兵是否可信，提高了哨兵实时性和动态性。实验证明，该模型能够准确地判断出哨兵进程是否遭到篡改并满足计算平台对安全性和执行效率的要求，具有较好的可行性。

关键词：完整性度量；动态检测；软件哨兵；标准值
中图分类号：TP309　　**文献标志码：**A　　**文章编号：**1001-3695(2012)08-3008-04
doi:10.3969/j.issn.1001-3695.2012.08.053

Study and implementation of software guards' security dynamic testing model

WU Shao-jie[1], HE Rong-yu[1], XUE Chang-song[2], XIE Jiang-tao[1]
(1. *Institute of Electronic Technology*, *Information Engineering University*, *Zhengzhou* 450004, *China*; 2. *Chengdu* 95879 *Army*, *Chengdu* 610081, *China*)

Abstract: Software guards is an important mean to protect software code from illegal tampering, but its security becomes a critical issue. To be able to detect whether the software guards had been tampered, this paper presented a dynamic measuring model based on TPM, which used the changes that guards in the memory page to judge its credibility and improved the nature of dynamical and real-time. The experiments show that the model can detect whether the software guards have been tampered and satisfied the security and efficiency to the computing platforms.

Key words: integrity measurement; dynamic measuring; software guards; reference value

0　引言

当前，计算机软件是计算机中最重要的计算平台和信息资源之一，但由于运行环境的不可信性，使软件非常容易受到非法侵害[1]，如被非法拷贝、传播，以及被病毒或攻击者恶意篡改和访问。特别是随着网络计算的发展，更多软件运行在不确定的环境下，主机可以任意对软件进行分析和跟踪，而且随着各种逆向工程计算的发展，使得对软件的攻击变得更加容易。随着 SOA 应用的普及，越来越多的软件以平台无关的中间代码来发布，用这种方式发布的软件代码与源码类似，比起传统的二进制可执行代码更易遭到静态分析、逆向工程破解和篡改等恶意攻击。如何制定有效的软件保护技术，成为安全领域中的研究热点。

Chang 等人[2]提出了软件哨兵技术，利用嵌入到软件中的哨兵保护软件的安全。许多学者对软件哨兵如何保护程序作了很多的研究，但哨兵本身缺乏安全性保护，如果哨兵遭到破坏，也就失去了保护功能。本文对此提出了一个检测软件哨兵完整性的模型，能动态检测出哨兵代码是否遭到篡改。

1　相关工作

哨兵的概念首先由 Chang 等人提出，用于软件防篡改机制。哨兵是应用程序的一部分代码，不易影响应用程序的代码。一个哨兵至少包括一条指令，而一条程序指令也至少包含一个字节序列。各类哨兵的安装与具体工作详见文献[2～7]。然而软件哨兵方法没有考虑哨兵本身的安全性，如果哨兵被篡改，将失去保护功能。因此，哨兵的安全性需要引起重视，文献[3]对此作了论证，提出保护哨兵安全的重要性，但作者并没有提出具体的实施措施。

Chang 等人[2]提出构造保护系统的框架，利用哨兵保护程序代码抵御篡改的安全框架。为更好抵御篡改，哨兵在内部形成网状结构进行相互保护。网状哨兵很难被击败，因为哨兵之间共享安全性，它们之间相互进行保护，这使攻击者很难预知内部结构。而且，更多的哨兵还可以增加到程序中满足安全级的需要，进一步加强程序代码的安全性。网状哨兵虽然能保护哨兵，但具有一定的局限性。因为受损代码执行之前需要支配目标代码，这会限制哨兵位置的选择。文献[4]对此作了改进，折中静态和动态签名，散列法应用于静态检测能抵御内存地址攻击模型。但哨兵框架在实施过程中仍然面临许多问题。

文献[5]提出在 VM(虚拟机)中运用校验和哨兵来保护程序。哨兵嵌入到程序和虚拟机中，使用轻量级的哈希函数负责检查内存地址，保护应用程序和 VM。整个程序被加密后再计算其校验和值，这样攻击者必须首先解密程序，然后修正校验值才能篡改程序，一定程度上保护了程序的安全。作者考虑到

收稿日期：2012-01-13；**修回日期：**2012-02-20

作者简介：武少杰(1986-)，男，郑州人，硕士研究生，主要研究方向为信息安全(wushaojie539@163.com)；鹤荣育(1966-)，男，郑州人，教授，硕导，主要研究方向为信息安全；薛长松(1982-)，男，安阳人，工程师，硕士，主要研究方向为计算机安全；谢江涛(1988-)，男，郑州人，助理工程师，主要研究方向为网络安全.

应用哨兵的安全性,采用备份哨兵代码的方法,即使哨兵代码被篡改,还可以用备份的代码确保程序的安全。如果备份的哨兵代码也被篡改,程序就不能继续执行,所以哨兵的安全性保护程度不高。

从本质上来说,上述技术都属于静态度量的范畴,通过匹配散列值的方法来判断软件是否可信,但只能确保软件加载前的可信,软件运行过程中缺乏可信性的检测,在动态性和实时性方面都有很大缺陷。

针对上述技术的不足,本文基于 TPM 提出了一个动态检测软件哨兵完整性的模型,能实时检测软件哨兵是否遭到破坏,从而确保哨兵的安全。

2　软件哨兵的动态安全检测模型

软件哨兵的动态检测模型主要分为三层,从下到上依次是硬件层、操作系统(QS)内核层和应用软件层(图 1)。

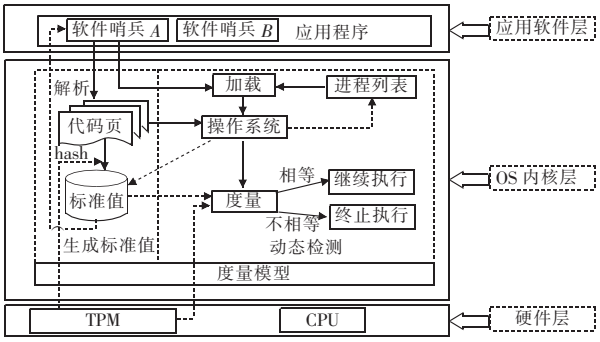


图 1　软件哨兵的安全检测模型

2.1　应用软件层

应用软件层包括将要执行的应用程序,软件哨兵嵌入其中,在应用程序运行时,负责保护它的安全。软件哨兵在应用程序中,按照程序的指令顺序执行。执行哨兵代码前,先对其进行完整性校验,判断哨兵代码是否遭到篡改。如果完整性被破坏,则不执行哨兵代码,终止程序的执行;否则,继续执行,不做任何处理。

软件哨兵在初始阶段生成的 RV(reference value)通过嵌入技术保存在哨兵代码中,待哨兵执行由内核把它分离出来并加载进内存。

2.2　OS 内核层

QS 内核层为动态检测提供运行环境和支持应用程序与 CPU 的通信。

在初始阶段负责生成软件哨兵的 RV,并把它嵌入在软件哨兵中,使得应用程序具有可迁移性;哨兵运行时负责加载代码文件和 RV 进内存,执行程序。

2.3　硬件层

计算机系统中涉及到的所有硬件,主要包括 CPU、各级存储系统、TPM 等。其中,TPM 为逻辑可信计算平台,对哨兵代码进行完整性度量,判断软件哨兵是否遭到篡改。

2.4　动态检测过程

软件哨兵安全动态检测模型分为两个阶段,即 RV 的生成和动态检测。

a)RV 的生成

该阶段完成对软件中哨兵 RV 的提取,具体过程为(图 2):

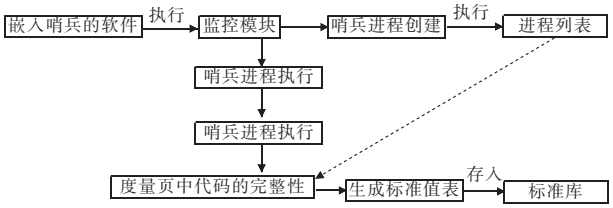


图 2　RV 的提取

(a)嵌入哨兵的应用程序执行时,监控模块监控进程的创建、调度和执行。

(b)哨兵进程创建时,登记哨兵进程占用的内存页面,创建哨兵进程列表。表中记录进程名称、进程在内存中缓存的页面数目、页的起始地址和结束地址、占用大小,便于度量时确认哨兵进程在内存中页面的位置。

(c)监控模块发现哨兵进程执行时,触发度量机制,TPM 依靠进程列表的信息度量内存页中代码的完整性。

(d)根据生成的完整性度量值创建 RV 表。表中记录标志哨兵进程的 IFID、代码页的数量 hCode_count 和 hash 值。

(e)生成的 RV 表存入标准库,最后将标准库嵌入哨兵代码,从而完成了哨兵完整性 RV 的提取。

(f)利用 TPM 对标准库进行签名,签名保存在 TPM 的 PCR 中,便于以后加载时检验其是否可信。

b)动态检测

当监控模块监测进程形式运行时,触发动态度量机制,度量内存中页的完整性,并与保存的 RV 相比较,判断哨兵的完整性是否遭到篡改,从而采取相应的保护措施。具体过程如图 3 所示。

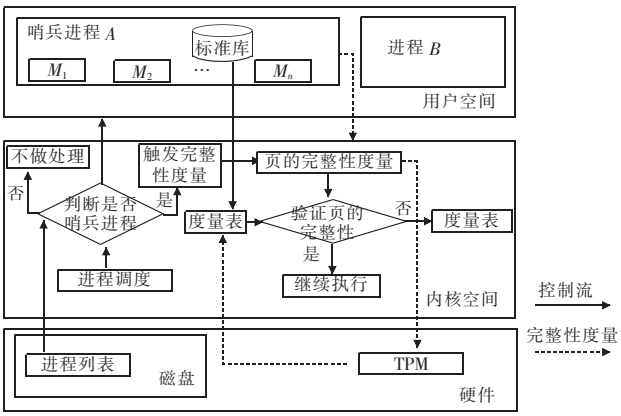


图 3　哨兵进程的动态检测过程

软件哨兵动态检测步骤如下:

(a)监控模块发现进程调度时,通过将新创建的进程标志 PFID 与进程列表中的相比较,判断调度的进程是否是哨兵进程。如果是,则触发动态度量;否则,不做任何处理。

(b)度量之前,需要检测标准库是否可信。内核把标准库从哨兵代码分量处理,调入内存之前,由 TPM 对标准库重新计算签名,并与保存在 PCR 中的签名进行比较。如果相等,说明标准库可信,可以作为 RV 使用,继续下面的操作;否则,终止程序的运行。

(c)把标准库调入内存,映射成内存度量表。如果是这样,新生成的哨兵进程的度量值就可以与 RV 在内存中进行比

较,判断哨兵的完整性(图 4)。

(d)根据进程列表中哨兵进程的信息,运用 SHA-1 算法度量内存页中代码的 hash 值,与内存中度量表中的 RV 做比较,完成哨兵的动态完整性检测。

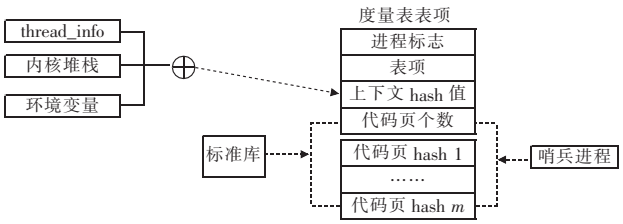


图 4　内存度量表格式

3　模型的实现

3.1　实现环境

为了验证模型的正确性和测试模型的效率,本文在 Ubuntu 上对模型进行了实现。

硬件环境:实验需要安装有 TPM 模拟器的计算机,计算机硬件配置为:CPU 为 Intel® Celeron® CPU530 1.60 GHz,内存容量 1 GB,硬盘空间 500 GB,操作系统为 Ubuntu Linux 10.04。

软件环境:TPM 软件模拟器 TPM Emulator 0.7;支持 Java 平台的 TCG 软件栈 jTSS 0.3.6;提供密码学算法的 Gmplib 库、模拟器需要的 Cmake 工具;开发工具 C 语言。

3.2　RV 的生成

不少学者很早就开始了对文件系统完整性保护技术的研究:基于信息流的完整性度量、预定内存区域的完整性度量、Linux 内核中关键数据结构的完整性度量、CPU 中 NX 机制的完整性度量。

本文出于实现上简单的考虑,采用 hash 值作为完整性度量值。初始阶段生成的摘要值称为 RV。哨兵的完整性检测,即对哨兵在内存页中的变化情况进行检查,每次检查时,重新计算内存页面的摘要值并与保存的 RV 相比较。软件哨兵在内存页面的摘要值通过 TPM 的 hash 函数计算得来,任何页面的 hash 函数计算结果都是一个固定长度的数字,采用安全性高的 SHA-1 算法,两个不同的页面几乎不可能得到相同的 hash 结果。从而,软件哨兵一旦被窜改,则其在内存中页面的内容会发生变化,摘要值也就随之改变,就会被检测到。

为了实现哨兵的动态检测,把软件哨兵的运行形式——进程作为度量对象,称为哨兵进程。哨兵进程在内存中的代码页表示如下:

$$\mathrm{Guard}_M = \langle M_1, M_2, \cdots, M_s \rangle$$

$$M_i \cap M_j = \varnothing (i \neq j, 1 \leqslant i, j \leqslant s)$$

其中:Guard_M 代表哨兵进程,M_i 代表内存中单个的代码页。单个实体的度量过程如下所示,表示度量内存中单个的页面,获得度量值 V_{M_i}

$$V_{M_i} = \mathrm{SHA1}(M_i)(1 \leqslant i \leqslant s)$$

对于每个哨兵,在标准库中为其创建一张表(即对应一个文件),表中内容包括用来唯一标志可执行哨兵的 IFID、代码页数量 hCode_count 和 hash 值。这些内容都以表的形式存放在标准库,嵌入到软件哨兵。哨兵开始运行时,标准库中相应表的内容被装入位于内存的度量表中(图 5)。

哨兵进程的摘要值都以表的形式保存在 RV 表中,表项对应着页面的摘要值。所有的表保存在标准库中,形成软件哨兵的 RV 集合,为了确保它的安全,把标准库嵌入到软件哨兵代码中。哨兵运行时,标准库中相应的 RV 表的内容被装入位于内存的度量表中,与新生成的摘要值作比较,动态检测哨兵的完整性(图 6)。

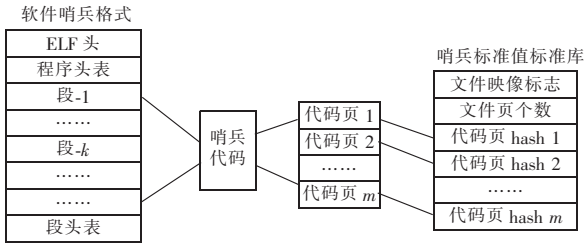


图 5　哨兵标准库中 RV 表的生成过程

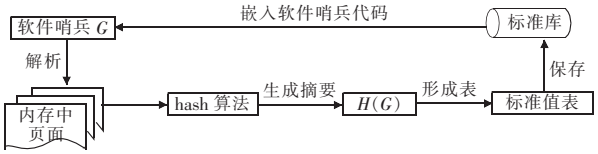


图 6　哨兵代码的完整性校验

3.3　RV 的嵌入

为了确保 RV 的安全,本文把生成的标准库嵌入到软件哨兵,待哨兵加载时内核再从哨兵代码中读出 RV,下面介绍标准库嵌入哨兵的基本原理。

ELF(executable and linkable format)即可执行连接文件格式,是 Linux 默认的目标文件格式。由 ELF 头部,包含多个单项的程序头部(program header)、多个节区(section)以及包含多个单项的节区头部(section header)所组成(图 7)。

ELF header	Program header	section	……	section	section header	section (optional)

图 7　ELF 文件结构

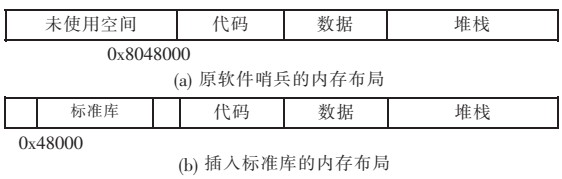
ELF 头部为固定的 52 Byte,是一个 elfhdr 结构的文件头部,结构中的 e_entry 成员保持着程序开始执行时的入口地址,e_phoff 成员保持着段头表在文件中的偏移量,e_shoff 成员保持着节头表在文件中的偏移量。每个节和段的位置、大小、属性都在节头表项或段头表项中有所体现。它标明了程序头部相对文件起始点的偏移量和程序头部中单项的个数,同时也标明了节区头部的偏移量和个数。

由于 Linux 内核是分段将 ELF 可执行文件载入内核的,因此可以考虑为嵌入的标准库增加一个可加载段,相应地也增加一个段头表项。在 Intel 平台上,进程空间大小一共 4 GB,地址范围从 0 到 0xFFFFFFFF。内核加载到 ELF 文件时,将代码作为一个可加载段从 0x8048000 开始向上读入内存。为了避免内核将新增加的段读入源哨兵程序的地址空间,新程序段只能放置在代码段装载地址 0x8048000 的前面。

为了不影响软件哨兵各段和节的偏移,需要将标准库插入到软件哨兵所有数据的后面,并且在段头表中增加一个 p_type = PT_LOAD 的表项,这样在内核执行软件哨兵时,该段就可以被加载进内存。设置新增加段的 p_vaddr 成员为 0x48000,当哨兵执行时,该段被加载到地址 0x48000。由于新增加的段被加载到地址 0x48000,因此,只需把标准库插入到哨兵末尾并且偏移量为整数页的地方,并保持代码插入的偏移

到变量 new_offset 中。

嵌入标准库前后软件哨兵的内存布局如图 8 所示。



(a) 原软件哨兵的内存布局

(b) 插入标准库的内存布局

图 8　标准库嵌入后的内存布局

当标准库嵌入完成后,需要修改软件哨兵中各头部中的信息。对于哨兵文件头部,修改程序入口地址(e_entry)为 0x48000。这样,软件哨兵在执行时,标准库首先被调入内存执行。由于增加了一个段头表项,段头表项个数的 e_phnum 成员的值修改如下:

```
elf_head. e_phnum ++ ;
elf_head. e_entry = 0x48000;
lseek( newfd,0,SEEK_SET) ;
len = write( newfd,&elf_head,elf_head. e_ehsize) ;
```

3.4　动态完整性功能实现

内核监控模块通过替换操作系统的相关 API 调用、监控软件的进程调度,Linux 系统为每个进程维护一个映像,生成哨兵的进程列表。该映像完全反映了进程的功能和信息。对哨兵进程的动态度量,就是对度量目标——某个哨兵进程在内存中的映像进行度量。本文根据系统为进程地址空间维护的内存描述符(mm_struct),来确定哨兵映像在内存中可寻址的位置(图 9)。

图 9　哨兵进程地址空间

对于某一个哨兵进程而言,其运行过程和运行结果由进程地址空间内的数据决定。这些数据中,有几处数据最为关键。Start_code 和 end_code 标志哨兵代码所在的线性区的起始和中止地址;Start_data 和 end_data 标志哨兵数据所在的线性区地址;Arg_start 和 arg_end 标志命令行参数所在的堆栈部分地址,它们是度量的主体。

通过 Linux 内核的加载机制,加载监控模块进内核。监控模块监测到哨兵进程调度后,从进程列表中找到这个进程,获得进程的句柄;然后找到进程的各种信息所在的地址,包括进程的代码段、数据段、堆栈段等等,获得进程的线性地址;接着将它转换为物理地址并读取物理地址中的内容,映射到哨兵的进程空间中;最后将这些内容交由 TPM 芯片做散列值,并与 RV 相比较。

RV 保存在标准库中,嵌入在软件哨兵内,不能直接与 TPM 生成的散列值进行比较,所以本文在内存中设计了与它相应的度量表。内存度量表除包含标准库中的相关信息之外,还需要其他信息,例如,每个进程在内存度量表中都由一个 PFID 唯一标志,与进程运行时的上下文环境对应的 hash 值由 P_regs 表示。内存度量表的结构如下所示。

内存度量表的一个表项的数据结构设计如下:

```
struct struct_measure_table_entry
{unsigned long PFID;//唯一标志一个进程
SHA_CTX P_regs;//进程上下文的 hash 值
SHA_CTX h[MaxHash];
//进程代码页的 hash 值
}
```

3.5　实验与分析

为了计算出添加动态度量引起的系统开销,本文通过计算一些系统调用和命令在引入动态度量前后的时间来进程评估。每一项都运行 10 次,然后取平均值(表 1)。

表 1　实验结果

测试项目		安装前/s	安装后/s	开销/%
系统调用	getpid	0.142 6	0.147 6	3.51
	Open	3.833 3	3.873 4	1.01
命令	tar	0.366 8	0.369 0	0.06
	gcc	0.124 5	0.125 3	0.64

通过表 1 可知,引入动态度量后在时间性能方面上增加的开销不超过 4%,在可容忍的范围内,所以对软件的运行影响不大。

3.6　下一步的工作

本模型能够在不安全的操作系统中建立安全的应用支撑,检测到软件哨兵是否遭到病毒和木马的破坏。存在的问题是:如果某个哨兵的功能需要更新,那么哨兵的代码就要改变,进而标准值也需要更新,这样哨兵正常情况下的更新操作也会被误认为不安全操作,影响软件的正常运行。下一步将研究标准值表的更新机制,实现更加准确的检测。

4　结束语

本文基于 TPM 提出一种哨兵运行时动态度量完整性的模型。与其他度量模型相比,该模型能动态度量进程在内存中页的完整性,通过与 RV 的比较,动态检测软件哨兵是否遭到窜改。通过实验证明,动态度量的引入在增强安全性的前提下对系统性能影响不大,软件运行的时间开销不超过 4%。

参考文献:

[1]　袁苑,刘小汇. 基于 ARM 平台的嵌入式软件保护方案[J]. 电子计算应用,2009,35(9):149-152.

[2]　CHANG H,ATALLAH M J. Protecting software codes by guards[R]. 2001:49-55.

[3]　YUNG M. How to guard the guards themselves[C]//Proc of the 17th International Conference on FCT. Berlin:Springer-Verlag,2009:1-2.

[4]　ZHANG Dong. Checking primitives with guards[D]. [S. l.]:University of Auckland,2005:138-144.

[5]　GHOSH S,HISER J D,DAVIDSON J W. A secure and robust approach to software tamper resistance[C]//Proc of the 12th International Conference on IH. 2010:33-47.

[6]　施光源,张建标,付颖芳. 基于双重完整性的可信证明模型[J]. 计算机工程,2012,38(1):28-32.

[7]　王宁,吴振强,符湘萍. 实时监控下的动态远程证明模型[J]. 计算机应用研究,2011,28(5):437-439.

[8]　曾凡平,陈明辉,尹凯涛,等. 进程完整性及其在软件漏洞检测中的应用[J]. 信息安全与通信保密,2009,6(10):91-94.

(下转第 3017 页)

对写写冲突进行了判断并中止,SSI-REP 为达到单副本可串行化同时对读写冲突进行判断。实验结果表明,由读写冲突引发的额外的中止情况占全部中止事务的很少一部分,即 SSI-REP 中增加的读写冲突的判定对事务中止的比率的影响很少。

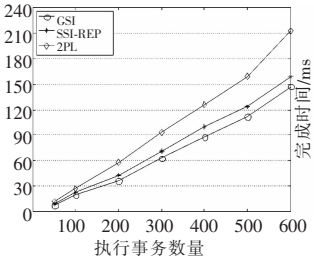


图 5　100%更新事务

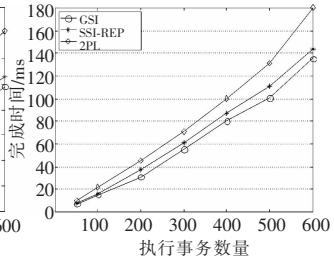


图 6　50%更新事务

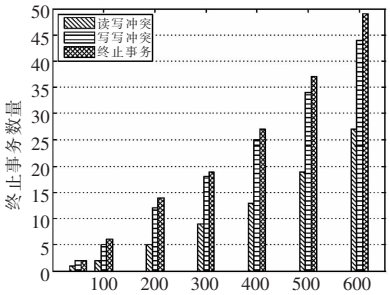


图 7　50%更新事务下的中止事务

6　结束语

维护分布式数据系统中的各节点副本一致性是数据库同步的核心问题,许多同步方案通过牺牲系统的性能来达到数据一致性。本文采用快照隔离技术实现分布式数据库的同步,并基于当前研究提出快照隔离可串行化算法 SSI,设计了分布式环境下的数据库同步协议 SSI-REP。实验结果表明,在 SSI-REP 数据库同步协议下,系统在维护节点副本一致性和保证快照隔离可串行的同时仍然有较好的性能。

参考文献:

[1] GRAY J, HELLAND P, O'NEIL P, *et al*. The dangers of replication and a solution[C]// Proc of SIGMOD. New York: ACM Press, 1996: 173-182.

[2] LIN Yin, KEMME B, PATINO-MARTINEZ M, *et al*. Middleware based data replication providing snapshot isolation[C]// Proc of SIGMOD. New York: ACM Press, 2005: 419-430.

[3] ELNIKETY S, ZWAENEPOEL W, PEDONE F. Database replication using generalized snapshot isolation [C]//Proc of the 24th IEEE SRDS. Washington DC: IEEE Computer Society, 2005: 73-84.

[4] BERENSON H, BERNSTEIN P, GRAY J, *et al*. A critique of ANSI SQL isolation levels[C]// Proc of ACM SIGMOD. New York: ACM Press, 1995: 1-10.

[5] LIN Y, KEMME B, ARMENDARIZ J E, *et al*. Snapshot isolation and integrity constraints in replicated databases [J]. ACM Trans on Database System, 2009, 34(2): 1-49.

[6] ALOMARI M, FEKETE A, ROHM U. A robust technique to ensure serializable executions with snapshot isolation DBMS[C]//Proc of IEEE ICDE. Washington DC: IEEE Computer Society, 2009: 341-352.

[7] FEKETE A, LIAROKAPIS D, O'NEIL E, *et al*. Making snapshot isolation serializable[J]. ACM Trans on Database Systems, 2005, 30(2): 492-528.

[8] ADYA A. Weak consistency: a generalized theory and optimistic implementations for distributed transactions[D]. [S. l.]: MIT Lab for Computer Science, 1999.

[9] KEMME B, ALONSO G. A new approach to developing and implementing eager database replication protocols[J]. ACM Trans on Database Systems, 2000, 25(3): 333-379.

[10] KEMME B, PEDONE F, ALONSO G, *et al*. Using optimistic atomic broadcast in transaction processing systems [J]. IEEE Trans on Knowledge Data Engineering, 2003, 15(4): 1018-1032.

[11] ATTIYA H, WELCH J. Distributed computing[M]. [S. l.]: Wiley-Interscience, 2004.

[12] KEMME B, ALONSO G. Database replication: a tale of research across communities[J]. Proc of VLDB, 2010, 3(1-2): 5-12.

[13] WU Shu-ging, KEMME B. Postgres-r(si): combining replica control with concurrency control based on snapshot isolation [C] //Proc of the 21st ICDE. Washington DC: IEEE Computer Society, 2005: 422-433.

[14] ABDELLATIF T, CECCHET E, LACHAIZE R. Evaluation of a group communication middleware for clustered J2EE application servers [C]//Proc of CoopIS/DOA/ODBASE. Berlin: Springer-Verlag, 2004: 1571-1589.

[15] KEMME B, ARMENDARIZ J E, PATINO-MARTINEZ M. Database replication[M]. [S. l.]: Morgan and Claypool, 2010.

[16] 娄燕飞,杨树强,李爱平,等. 一种基于组通信的复制数据库在线恢复策略研究[J]. 计算机研究与发展, 2007, 44(Z3): 232-237.

[17] AMIR Y, NITA-ROTARU C, STAN TON J, *et al*. Secure spread: an integrated architecture for secure group communication [J]. IEEE Trans on Dependable and Secure Computing, 2005, 2(3): 248-261.

(上接第 3011 页)

[9] 宋杨,李立新,周艳周,等. 软件防窜改技术研究[J]. 计算机安全, 2009(1): 34-37.

[10] 龙文策,唐宁九,林涛. Linux 平台下可执行文件代码嵌入技术的研究与实现[J]. 通信市场, 2009(11): 23-29.

[11] 刘昌平,范明钰,王光卫. 开放网络环境完整性按需度量模型[J]. 计算机研究与发展, 2011, 48(2): 86-90.

[12] 周振柳,栾好利,张楠,等. 基于动态安全属性保护的可信固件[J]. 计算机工程, 2011, 37(4): 56-61.

[13] WEI Cheng-long, SONG Shao-hua, HUA Wen, *et al*. Operating systems support for process dynamic integrity measurement[C]//Information, Computing and Telecommunication. 2009: 339-342.

[14] LIN Hui, LEE Gyungho. Micro-architecture support for integrity measurement on dynamic instruction trace[J]. Journal of Information Security, 2010, 1(1): 1-10.

[15] BARRON G, KIM Jung. Guards for application in software[M]. 2007: 213-219.

[16] HENNING M T. Software trusted platform module and application security wrapper: US, 7870399[P]. 2011.

[17] LIU Chang-ping, FAN Ming-yu, FENG Yong, *et al*. Dynamic integrity measurement model based on trusted computing[C]//Proc of International Conference on Computational Intelligence and Security. Washington DC: IEEE Computer Society, 2008: 281-284.

[18] ATALLAH M J, FRIKKEN K B, BLANTON M. Dynamic and efficient key management for access hierarchies[J]. ACM Trans on Information and System Security, 2009, 12(3): 13-21.