Solaris 下基于 VFS 加密组件模型的研究

牟永敏,李 贝
(北京信息工程学院计算机开放系统实验室,北京 100101)

摘 要: 主要分析了如何利用系统调用重定向技术,在基于 Solaris 的 VFS(Virtual File System)构架下开发文件加密组件,并实现与操作系统的无缝工作. 在简单文件加密模型的基础上作了进一步改进,提出了改进后的文件加密组件模型. 结果表明,改进后的模型具有更高的效率,并且有效地降低了该模块对操作系统性能的冲击.

关键词: 系统调用重定向; 文件加密; VFS 文件系统
中图分类号: TP393.08 文献标识码: A 文章编号: 0372-2112 (2006) 12A-2547-04

The Model and Implement of File Enc/Decryption Based on Solaris VFS

MU Yong-min, LI Bei
(*Open Computer System Laboratory, Beijing Information Technology Institute, Beijing* 100101, *China*)

Abstract: On the base of analyzing how to develop file enc/decryption system based on VFS of Solaris with the technology of redirection of system call and how to make it work with OS seamlessly, the author make progress on the simple model of file enc/decryption system and gives the improved model of file enc/decryption system. The result shows that the improved model has better efficiency and reduces the impact to the OS on performance effectively.

Key words: redirection of system call; file enc/decryption; virtual file system(VFS)

1 引言

在许多企业中都存在着多个用户共用一台计算机的情况,未经授权的用户可能希望读取存储在计算机中的数据. 有些用户外出时使用便携式计算机,并在没有企业物理保护的情况下接入因特网. 所有这些情况下,计算机中的重要数据都将会面临被盗取的危险.

Solaris 是 SUN Microsystems 公司研发的计算机操作系统,它是 UNIX 操作系统的衍生版本之一,以高稳定性著称. 不少企业使用它作为内部的文件服务器供多用户访问. Solaris 的开放性使得用户可以基于 Solaris 的 VFS 构架编写自己的文件加密模块,并加入内核,实现定制文件加密的功能. 因此,研究 Solaris 文件加密系统模型及实现原理,具有很大的现实意义.

2 简单的文件加密模型

对于用户来说,访问文件的唯一途径是通过系统调用. 只有通过系统调用,才可以对文件进行打开、关闭、读写等操作. 要想实现对用户透明的文件加密/解密功能,则必须在文件读写的时候自动启动加密/解密功能: 将数据写到磁盘之前,先将数据加密,这样写到磁盘文件中的数据都是秘文; 从磁盘文件读取的数据返回给用户之前,如果用户身份合法,则先将数据解密,这样读出的才是明文.

一个最简单和直接的办法就是截取到用户读写文件的系统调用,在系统调用执行更深层次调用的前后,分别注入加密/解密的代码,这种方法称为系统调用的重定向. 即,你可以将普通的系统调用重定向到你自己写的伪调用函数. Solaris 下的系统调用保存在一个 sysent[] 数组里面. 这个数组中的每一项都指向一个结构体,该结构体包含有关某个系统调用的信息. 每个系统调用在该数组中的偏移量(即,下标值)都可以从 /usr/include/sys/syscall.h 中得到. 下面的部分代码片断显示了如何用新的 newread 函数替换掉现有的 read 系统调用.

```
/* 声明一个函数指针,用来保存原有的 read 系统调用 */
int(* oldread) (int fildes, void * buf, size_t nbyte);
/* 定义我们自己的 read 系统调用 */
int (* newread) (int fildes, void * buf, size_t nbyte);
/* 保存原来的系统调用的地址 */
oldread= (void * ) sysent[SYS_read].sy_callc;
/* 将新的系统调用的地址填到 sysent 数组的相应位置 */
sysent[SYS_read].sy_callc= (void * ) newread;
```

这样,当用户调用 read 函数读取文件时,系统将通过数组 sysent[] 找到 newread,并执行之.

怎么将我们的代码加入到系统内核中呢? 通过编写一个内核模块[1,3]并将它加载可以做到这一点. 内核模块是可以让操作系统内核在需要时载入和执行的代码,这同样意味着

收稿日期: 2006-10-09; 修回日期: 2006-12-20

它可以在不需要时由操作系统卸载.它扩展了操作系统内核的功能却不需要重新启动和编译系统,这正符合我们的要求.可以在该内核模块加载的时候重定向 read 和 write 系统调用,新的系统调用策略如图 1 所示:

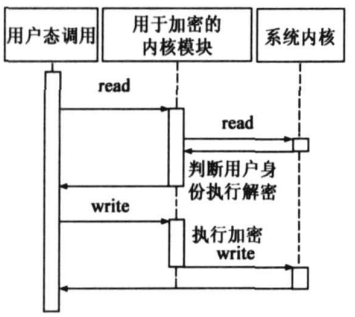

图 1 新的系统调用策略

这样一个最简单的文件加密/解密系统模型就形成了.只要用户的身份合法,该模块将在系统内核中自动完成加密解密的工作,对于用户来说,并不能感觉到该模块的存在,用户对文件的读写没有加载该模块时没什么区别.但是对于非法用户,由于该模块并不做加密解密的操作,因此读到的数据都是秘文,这种数据对于用户来说是毫无用处的.

但是这个模型存在一个缺陷:UNIX 下文件访问可以通过两种方式,一种是 read/write 系统调用直接读写;另外一种是通过 mmap 系统调用[5,6],该调用在指定的文件与一段连续内存之间建立映射,用户对该内存的访问最终被映射为对指定文件的读写.上述的简单模型可以应对第一种情况,但是对于第二种情况却是不能正常工作的.因为 mmap 本身并没有文件 I/O 的操作[4],它只是在用户内存空间与文件之间建立映射,当用户第一次访问这段内存时,系统产生一个缺页中断[2],从而载入该段内存对应的文件页.按照上面介绍的步骤那样重定向 mmap 系统调用是无法使该模型正确完成加密解密的工作的,因为我们无法截取对内存的读写操作.因此简单模型只能完成一半的工作,对于另一半的需求却无能为力.但是它所引出通过系统调用重定向技术注入"加密/解密代码"的思路为进一步的研究提供了基础.下面的改进模型沿用了该重定向技术,并通过它进一步对文件系统的函数进行替换.

3 改进的文件加密模型

首先,我们再来进一步分析文件 I/O 的两种方式.实际上这两种方式底层的实现是一致的[2]:都是先将一个文件映射到内存空间,然后再以页为单位对这段内存进行访问,如果要读的文件页尚未载入内存,则产生缺页中断,载入所需的页;当需要将内存中的数据写回文件时,如果数据被改变过,才真正执行回写.这两种方式的不同点仅在于"文件被映射到哪里"和"由谁来执行映射":当 mmap() 被调用时,进程将文件映射到进程的地址空间,从而可以进行内存映射 I/O;当 read()/write() 被调用时,由内核先将文件映射到内核的地址空间,然后再读写.真正的文件 I/O 的代码并没有写在系统调用的函数里,而是在文件对应的 V 节点的操作函数集里,当通过不同的方式访问文件时,系统将调用 V 节点的操作函数,并传入不同的参数,从而将文件映射到不同的内存空间.下面对 V 节点作简单的介绍.

V 节点(vnode)是一种数据结构,Solaris 内核中采用 vnode 来表示一个文件.这种数据结构是面向对象的,因为它不仅包含了文件的状态信息,还包含对文件数据操作的函数指针.需要注意的一点是:V 节点所代表的文件不一定是我们通常所说的普通文件,它可能有多种类型,比如:普通文件、目录文件甚至还能代表块设备或是字符设备.需要加密什么类型的文件要视需求而定,一般情况下都是普通文件或者目录文件,可以通过引入监视路径来设置,关于监视路径在后面有详细的讲解.

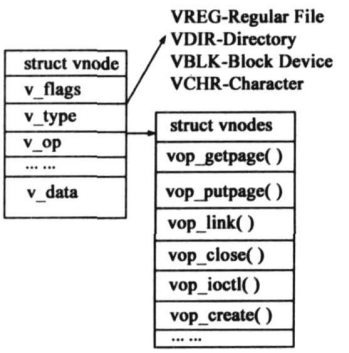

图 2 V 节点的结构

如图 2 所示,Solaris 通过 vnode 隐藏了不同的文件系统的实现细节,vnode 只向客户端暴露与文件系统无关的数据(如引用计数、vnode 类型)和一些操作接口,客户端必须通过这些操作接口才能访问与文件系统相关的私有数据.

v_flags 和 v_type 是与文件系统无关的数据,可以认为是公有数据,可以直接访问.v_data 是与文件系统相关的数据,v_op 指向 struct vnodeops,这个结构体是一个函数指针的集合,通过相应的函数可以对该 vnode 文件系统相关数据进行操作.而文件读写正是属于该类操作!因此如果能在此处重定向负责文件读写的 vop 操作,就可以满足各种方式的文件读写的加密/解密需求.

应该重定向哪些 vop 操作呢?从表面上来看似乎是 vop_read() 和 vop_write(),经过仔细研究这部分代码就会发现这些 vop 操作并不完全是在同一个层次上,它们之间仍有调用关系,例如:vop_read() 只提供了策略性的操作,将文件被请求的部分映射到内核空间,然后再调用 vop_getpage(),后者才提供了具体的实现,完成 I/O 的实际工作.vop_write() 和 vop_putage() 的关系也类似.因此我们需要对真正提供读写机制的 vop_getpage() 和 vop_putpage() 作进一步的研究,从而完成这两个操作的重定向.

与上层的读文件接口不同,vop_getpage() 接收一个 vnode 对象、偏移量 off 和长度 len,这个 vnode 对象是将被读取的文件在内核中的数据表示,off 和 len 表示读取的起始位置和要读取的长度.该函数返回的不是读入的内存首址,而是内存页(struct page).在这里理解内存页的概念是完成重定向的关键,因为我们必须在该内存页返回给调用者之前对它解密.在 Solaris 内存管理子系统中,页是物理内存的基本单位,它是一种全局性的资源,每一个页代表一块物理内存.但要访问一个页,则必须先将它映射到内核或进程的地址空间,得到一个虚拟内存地址(也就是我们通常所说的内存地址),再对这个虚拟内存地址进行操作.那么为什么 vop_getpage() 不直接返回一个内存地址呢?因为内存地址是依赖于进程的概念,每一个进程都有自己从 0 开始的虚拟内存空间,同样的内存地址在不同的进程中对应的物理内存是不同的,因此只返回一个内存地址而不指定进程是没有意义的.作为内核底层通用的

I/O 操作, 很显然 vop_ getpage( ) 不应当依赖于特定的进程, 因此它只返回内存页, 至于这块内存应当映射到用户进程地址空间还是内核地址空间则取决于调用者. vop_ putpage 是 vop_ getpage 的逆过程, 这里就不再赘述. 经过上面的分析, 重定向以后的 vop 操作策略可归纳如图 3 所示:

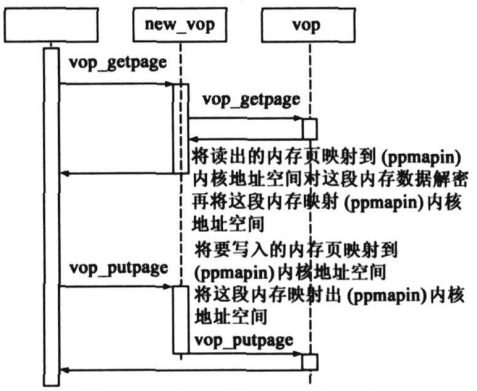

图 3　重定向后的 vop 操作策略

将内存页映射到内核地址空间的唯一目的就是取得其虚拟内存地址, 从而可以对它加密/ 解密, 当完成了加密/ 解密操作以后, 应当使用 ppmapout 取消该映射.

采用该模型进行文件加密/ 解密需要付出性能上的代价, 因为它不仅截获用户指定文件的 I/O 请求, 也截获系统文件包括虚拟内存换进换出的 I/O 请求, 并执行加密/ 解密. 因此若对所有请求不加过滤, 则会对系统的整体性能带来很大的冲击. 下面, 再引入一个请求过滤机制. 让我们再考虑一下现实的需求: 即使对于用户文件, 真正需要加密的通常只是一小部分. 因此可以考虑维护一个监视路径的列表, 只有访问该列表中某个路径下的文件时才执行加密/ 解密. 可以在该模块的. conf 文件[1] 中记录下所有监视路径, 这样当该模块被加载的时候初始化监视路径列表, 在文件加密/ 解密之前遍历该列表判断该文件是否属于在某个监视路径下, 最简单的办法是在 new_ vop_ getpage( ) 和 new_ vop_ putpage( ) 中加入判断. 但是, 这么做虽然不再对所有文件都执行加密解密, 却要在所有文件 I/O 的时候多一次判断, 这似乎仍是缺憾, 最理想的方案应该是尽可能的不对我们不感兴趣的文件产生影响. 迄今为止该模型只在系统的两个地方有过改动: (1) 重定向系统调用, (2) 重定向 vnode 的 vop 操作. 既然在后者加入过滤机制还不能取得令人满意的效果, 那么让我们看一下如果放在前面呢? 考虑如果重定向系统调用 open, 则新的 open 操作策略如图 4 所示:

把过滤机制加在替换 vop 操作之前是个很不错的选择: 当打开的文件在某个监视路径下时才替换 vnode 的 vop 操作. 这么做的优点很明显: (1) 尽量减少了加密解密代码对系统的影响, 因为对

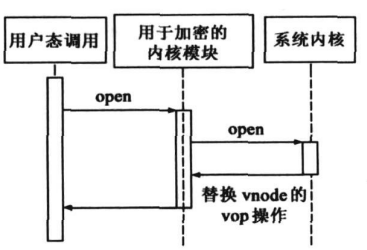

图 4　新的 open 操作策略

于非监视路径下的文件操作并不错任何替换, 减小了运行时的风险和调试的难度. (2) 这样做对于文件系统性能的影响也降至最低——非监视路径下的文件仅在打开的时候多一次判断, 读写的时候没有任何影响; 只对监视路径下的文件需要作必要的判断和额外的操作. 如果用户给出的监视路径越精确, 范围越小, 那么该加密/ 解密模块对系统造成的影响也就越小. 按照这种方案, 新的 open 系统调用策略如下:

下面让我们重新整理一下思路, 列出改进后的文件加密模型的要点:

(1) 从. conf 文件读入并初始化监视路径.

(2) 替换 open( ) 、open64( ) 、creat( ) 和 creat64( ) 等与文件打开有关的系统调用. 定义一个新的 struct vnodeops new_ v_ op 操作函数集, 按照图 5 所示的策略, 对于属于监视路径下的文件, 将其 vnode 的 v_ op 指针指向 new_ v_ op; 对于非监视路径下的文件, 则不做额外操作.

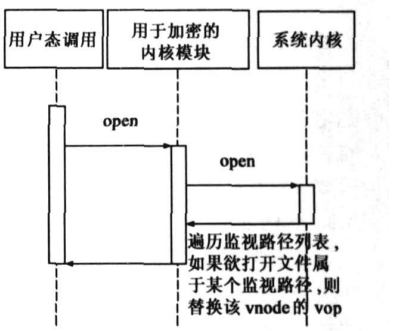

图 5　引入监视路径后的 open 操作的策略

(3) 重写 new_ v_ op 的 vop_ getpage( ) 和 vop_ putpage( ) 函数, 策略如图 5 所示, 对于不需要替换的 vop_ XXX 函数, 则只需要将其指针指向系统原有的函数地址[7].

4　实现

上面只给出了实现文件加密/ 解密的模型, 由此模型转化成具体的实现还需要 Solaris 内核模块编程、Sloaris 文件系统、内存管理和加密解密算法[8~10] 等各方面的知识, 由于篇幅有限, 不再一一讲解. 这里仅对几个关键的实现点作一下分析.

4.1　监视路径的维护

每次打开文件的时候都要遍历监视路径列表, 因此采用适当的数据结构以提高对监视路径列表的遍历操作仍然可以在一定程度上提升本系统性能.

最简单的情况是不支持监视路径的动态修改, 即: 一旦模块被加载, 监视路径从. conf 文件中读入, 就固定下来了, 不允许用户再增加、修改和删除. 很显然此时的数据结构应当选用顺序的数组, 查找算法可采用折半查找. 这样虽然效率是最高的, 却给用户带来了很大的限制和不便——当用户需要改变监视路径的时候将不得不修改. conf 文件, 并重起系统才能生效. 因此在模块的 xxx_ ioctl( ) 接口[1] 中加入监视路径的动态增、删、改、查的功能将大大提高本系统的可交互性, 我们也因此要对采用什么样的数据结构来维护监视路径在时间和空间上需要做更多的权衡: 如果监视路径不是很多, 可以使用顺序的单向链表或双向链表; 否则可以考虑采用 Hash 表. 另外, 由于 Solaris 内核是多线程的, 此处必须考虑线程安全问题——有可能多个内核线程同时调用 xxx_ ioctl( ) 接口, 对监视路径列表进行修改. 这是一个典型的读者写者问题, 应当采用信号

量机制对监视路径进行互斥访问.

另外,监视路径的精度也是应当权衡的问题:如果监视路径范围过大,将起不到引入监视路径的作用,反而影响了系统的效率;如果监视路径过于精确,则可能需要经常调整,这给使用带来了不便.最好能将监视路径限制在用户经常使用且包含了重要数据的的目录下.

4.2 用户地址空间和内核地址空间

在UNIX下,内核模块编程和应用程序编程最大的区别之一就是二者要运行在不同的地址空间:内核模块运行在内核地址空间,而应用程序运行在用户地址空间.系统软件和用户应用程序所运行的地址空间是隔离开的,因此内核模块不可以直接访问用户空间的数据,而应当使用 ddi_copyin 和 ddi_copyout[5] 将用户地址空间的数据拷出或拷入到内核地址空间.在实现模块的 xxx_ioctl() 接口的时候会遇到这种情况:当用户增加一条监视路径的时候,传入的参数是该监视路径的字串,该字串就是在用户地址空间的,而此加密/解密系统维护的监视路径列表则是在内核地址空间的,因此必须先将参数中的数据拷贝到内核地址空间再作其它操作.

4.3 性能分析

由于本系统运行在操作系统的底层,它对操作系统性能上有多大的影响并不能简单的通过统计若干次读写文件来得出结论,它还与文件读写的频繁度、文件系统读写磁盘的策略、监视路径的设置、加密算法的选择等有很大的关系,这些有的属于上层逻辑,有的则是更具体的实现细节,要视不同的应用而定.但是我们还是可以找出可能对性能产生重大影响的关键点,来分析该模型的适用性.

首先,对系统影响最大的就是是否采用监视路径.作者做过如下实验:采用 DES 加密算法对某个文件做读操作 10 万次,结果如图 6 所示:不引入加密的执行时间是 11.2ms,引入后的执行时间是 12.3ms——该模型使文件操作性能降低了近 10%,如果不引入监视路径,这个数字就是该模型对文件系统整体性能的影响.因为对每个文件的访问都要消耗这 10% 的时间用于加密.引入监视路径后,虽然不能减少对文件执行加密/解密的额外开销,却可以大大减少需要执行加密/解密的文件.如果能把目标文件缩小到正常读取文件数量的 1/10(通常还会远小于这个数字),那么该模型对文件系统整体性能只有 1% 的影响.这是个可以接受的范围.

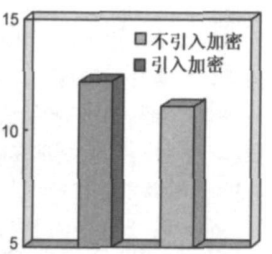

图 6 测试结果

当采用了监视路径策略后,还可以在维护监视路径的数据结构上作一些性能优化.但这种优化不能像上面那样在时间的量级上有更大的提升,因为对于一个文件只有在打开的时候才需要遍历监视路径,读写的时候则都不需要了.文件打开操作的频繁度远远的低于文件读写.因此这里更多地关注代码的灵活度和健壮性,向上层提供一个可自由调控的监视路径,比更多地考虑时间性能意义更大.

5 结束语

Solaris的内核模块机制使 Solaris 操作系统具备更强的可扩展性,使得开发人员可以根据自己系统的需要实现用户自定义的系统级软件.本文正是利用这一特点,开发出了文件加密组件模型.该模型的主要意义不仅在于它可用于文件加密,同时也为类似的文件系统操作提供了一套通用的解决方案,例如:把文件加密/解密算法代码改为数据流的监视代码,本模型就演变成了病毒实时监视系统.

参考文献:

[1] Sun Microsystems. Device Driver Tutorial[M]. Sun Microsystems, 2005.

[2] Jim Mauro, Richard McDougall. Solaris Internals[M]. Sun Microsystems Press, 2000.

[3] Sun Microsystems. Writing Device Drivers[M]. Sun Microsystems, 2005.

[4] Sun Microsystems. Solaris Modular Debugger Guild[M]. Sun Microsystems, 2005.

[5] Sun Microsystems. man pages section 9: DDI and DKI Kernel Functions[M]. Sun Microsystems, 2005.

[6] Sun Microsystems. man pages(2): System Calls[M]. Sun Microsystems, 2005.

[7] Perter Van Der LinDen. Expert C Programming[M]. PH PTR, 2002.

[8] 汪志达,叶伟.在 PC 上实现对公开密钥加密密文的解密[J].计算机应用与软件,2005,22(8):144-145.
Wang Zhi-da, et al. Cipher decryption of public key cryption on PC[J]. The Computer Application and Software, 2005, 22(8): 144-145. (in Chinese)

[9] 张串绒,尹忠海,肖国镇.不使用 Hash 和 Redundancy 函数的认证加密方案[J].电子学报,2006,34(5):874-877.
Zhang Chuan-rong, et al. Authenticated encryption schemes without using Hash and Redundancy functions[J]. Acta Electronica Sinica, 2006, 34(5): 874-877. (in Chinese)

[10] 顾纯祥,张亚娟,祝跃飞.混合可验证加密签名体制及应用[J].电子学报,2006,34(5):878-882.
Gu Chun-xiang, Zhang Ya-juan, Zhu Yue-fei. A mixed verifiably encrypted signature scheme and it's applications[J]. Acta Electronica Sinica, 2006, 34(5): 878-882. (in Chinese)

作者简介:

牟永敏 男,1961 年生于山东烟台,教授,主要研究方向为网络安全、嵌入式、软件自动生成技术.E-mail: yongminmu@biti.edu.cn

李 贝 男,1983 年生于河南郑州,硕士研究生,主要研究方向为网络安全、嵌入式.