DOI:10.15966/j.cnki.dnydx.2008.07.024

基于 RelayFS的内核态内存泄露的检测和跟踪

张爱国[1] 刘晓鹏[2]

(1. 中山火炬职业技术学院，广东 中山 528436；

2. 电子科技大学计算机科学与工程学院，四川 成都 610054)

[摘 要] 本文介绍了利用 RelayFS 对 kmalloc 和 kfree 函数进行改造的方法，在这两个函数中建立 Relay 信道并对内核态内存进行跟踪，有效防止内存泄露。

[关键词] RelayFS; kmalloc; kfree; 内存泄露

1. 引言

内核态内存泄露是指驱动程序或模块程序在申请获得内存和使用完毕后，没有释放内存就将保存内存地址的变量用于其它用途，使得这些内存不可能再被程序使用，也无法被操作系统回收，最终导致程序因耗尽系统所有的内存，无法再分配到内存而崩溃。驱动开发人员在设备驱动程序需要申请/释放内存时，不能使用用户级的 malloc/free 函数，而需由内核级的函数 kmalloc/kfree () 来实现，在调用函数 kmalloc 分配内存时，如果某一函数覆盖了 kmalloc()之前的指针，由于不能释放内存就会导致内存泄露，或者编写驱动生成了设备文件，并且多次运行 cat 打开设备文件也会产生内存泄漏，但是因为在内核态，很难用调试工具进行跟踪和检测内存的分配情况，给开发者造成很多麻烦。

目前在内存检测和跟踪方面常用的工具是动态监控工具，像 state-of-the-art Purify，但是这些工具以牺牲运行速度来实现目的，有的甚至慢了将近 20 倍，因此很不适合监控内核内存的分配和释放[1]，现在国内外在避免内存泄露方面的研究主要集中在 Garbage collection[2]。但是这种机制只能为 java 等高级语言使用，不能够在 C 语言等低级语言使用。

本文介绍了利用 RelayFS 对 kmalloc 和 kfree 函数进行改造的方法，在这两个函数中建立 Relay 信道，对内核态内存进行跟踪，能够将内存的使用分配情况显示给开发人员，定位进程所使用的内存地址，有效地防止内存泄露，保护内存使用的安全性。

2. 原理分析

2.1 RelayFS 介绍

RelayFS是一个快速的转发 (Relay) 数据的文件系统，它为那些需要从内核空间转发大量数据到用户空间的工具和应用提供了快速有效的转发机制。2003 年 3 月，RelayFS的第一个版本的代码被开发出来，经过不断的改进和更新，从 linux 内核 2.6.17 开始，RelayFS不再作为单独的文件系统存在，而是成为内核的一部分。

信道(channel)是RelayFS文件系统定义的一个主要概念，每一个channel由一组内核缓存组成，每一个CPU有一个对应于该channel的内核缓存，每一个内核缓存在RelayFS文件系统中用一个文件表示，内核使用RelayFS提供的写函数把需要转发给用户空间的数据快速地写入当前CPU上的channel内核缓存。

用户空间应用通过标准的文件I/O函数在对应的channel文件中可以快速地取得这些被转发出的数据mmap。写入到channel中的数据的格式完全取决于内核中创建channel的模块或子系统。Relay并不关心数据的格式和内容，这些完全依赖于使用Relay的用户程序。Relay的目的是提供一个足够简单的接口，从而使得基本操作尽可能的高效。

图 1 给出了Relay的基本结构和典型操作：

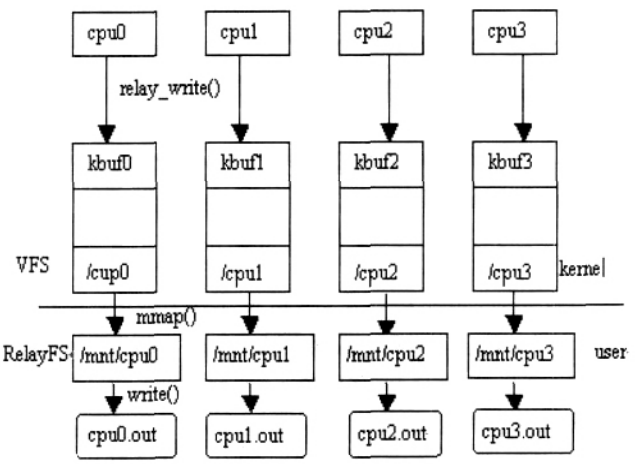

图 1 RelayFs 结构

由图 1 可知 Relay 通道由四个 Relay 缓冲区 (kbuf0 到 kbuf3)组成，它们分别对应于内核系统中的 cpu0 到 cpu1。每个 CPU 上的代码调用 relay_write()的时候将数据写入自己对应的 Relay 缓冲区内。每个 Relay 缓冲区称一个 Relay 文件，即 /cpu0 到 /cpu3。当 RelayFS 被 mount 到 /mnt/ 以后，这个 Relay 文件就被映射成映射到用户空间的地址空间。一旦数据可用，用户程序就可以把它的数据读出来写入到硬盘上的文件中，即 cpu0.out 到 cpu3.out。

2.2 Relayfs 的 API 函数

作者简介：张爱国，女，湖南永州人，副教授，研究方向：优化组合与统计。

基金项目：十一五国家支撑计划资助，项目编号：2006BAHO2A00。

2.2.1 面向用户空间的 API 函数

RelayFS提供用户空间的 API 接口，这些接口用于访问通道缓冲区中的数据，接口包括：

open () 允许在用户空间打开一个已经存在的通道缓冲区；

mmap() 将通道缓冲区映射到位于用户空间的调用者的地址空间。但是映射的是整个缓冲区文件，其大小是 CPU的个数和单个 CPU 缓冲区大小的乘积；

read() 读取通道缓冲区的内容。但这些缓冲区中的数据只能被程序读取一次。一旦数据被读取，则再也不能执行读操作；

sendfile () 将数据从通道缓冲区传输到一个输出文件描述符，其中可能的填充字符会被自动去掉，不会被用户看到；

poll() 用于通知用户空间应用转发数据跨越了子缓存的边界，支持的轮询标志有 POLLIN、POLLRDNORM 和 POLLERR；

close() 将通道缓冲区的引用数减1。当引用数减为0时，表明没有进程或者内核用户需要打开它，从而这个通道缓冲区被释放。

用户态在使用上述API时必须保证已经挂载了RelayFS文件系统，但内核在创建和使用 channel 时不需要RelayFS已经挂载，挂载的命令是 mount -t relayfs relayfs /mnt/relay。

2.2.2 面向内核空间的 API 函数

RelayFS提供内核空间的API接口，这些接口作用是向内核空间的用户提供创建、关闭relay通道、数据写入等功能，接口包括：

relay_open() 创建一个relay通道，并分配每个CPU对应的relay缓冲区，用户空间应用通过在 RelayFS 文件系统中对应的文件可以访问 channel 缓存的内容；

relay_close() 关闭一个relay通道，释放所有的relay缓冲区，在此之前会调用relay_switch()来处理这些relay缓冲区以保证已读取但是未满的数据不会丢失 ；

relay_write() 将数据写入到当前 CPU 对应的 relay 缓冲区内。由于它使用了local_irqsave()保护，所以也可以在中断上下文中使用；

relay_reserve() 在 relay 通道中保留一块连续的区域来留给未来的写入操作。这通常用于那些希望直接写入到 relay 缓冲区的用户；

relay_flush() 强制在所有的 channel 缓存上做一个子缓存切换，在 channel 被关闭之前使用来终止和处理最后的子缓存；

relay_reset() 用于将一个channel恢复到初始状态，因而不必释放现存的内存映射并重新分配新的channel缓存就可以使用channel，但是该调用只有在该channel没有任何用户在写的情况下才可以安全使用。

2.3 kmalloc 与 kfree 函数分析

在内核空间中 kmalloc 函数用来分配驱动程序或内核模块所需的内存页，根据所需分配的对象空间的大小查找 cachs_size 数组，找到合适大小，然后根据 flags 中的 GFP_DMA 是否被设置来决定用两条链表中的哪一条去寻找空闲内存块，再遍历缓存区的所有的 slab 块，查看是否有可用的对象空间，如果所有的 slab 块都没有，则在缓存区分配一个新的 slab 块。kmalloc 函数调用 _kmalloc，在 kmalloc 中分配内存的大小。

kmalloc函数的调用层次如图 2 所示：

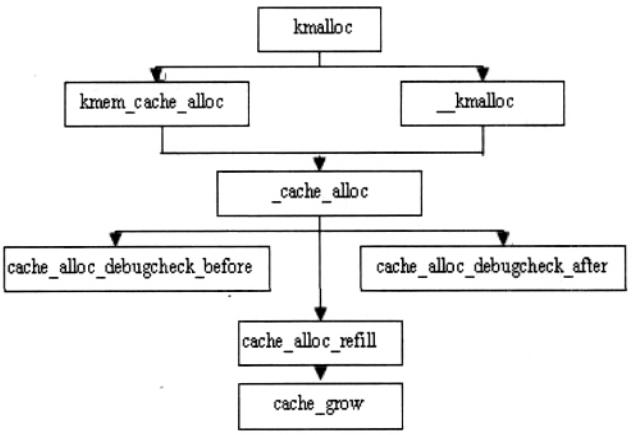

图 2 kamlloc 函数调用图

kfree函数是用来释放分配的内存空间，内存使用结束，不释放就会造成资源浪费，很容易造成内存泄露。这些释放的对象空间并没有立即返回系统，只是简单地让缓冲区中一个指向空闲对象的指针释放这个对象空间，并对由此返回的slab链表的结构变化进行相应的调整。真正释放对象的空间的工作是由内核守护进程kswapd发现内存紧张的时候，将选中的缓存区释放的。

2.4 RelayFS 对 kmalloc 和 kfree 函数进行改造

由图 2 可知在kmalloc中，调用函数 _kmalloc，所以将对_kmalloc进行改造。

```
#include<linux/relayfs_fs.  h>
extern struct rchan *kleak_chan ;// 定义一个信道结构
#define SUBBUF_SIZE 262144  // 定义通道缓冲区的大小
#define N_SUBBUFS 4  // 定义通道缓冲区的个数
viod *__kamlloc(size_t size,int flags)
{
  kmem_cache_t *cachep;
    void *a;
  cachep = __find_general_cachep(size,flags);
    if (unlikely(cachep == NULL))
          return NULL;
    a = __cache_alloc(cachep,flags);
  struct
    {
    unsigned char event_id;
    void *a;
    void *c;
    size_t size;
    int objsize;
```

```
} data = { 0x1,a,__builtin_return_address (0), size, obj_size (cachep) } ;
  kleak_chan = relay_open("kleak", NULL, SUBBUF_SIZE,
      N_SUBBUFS,1,NULL) ;// 创建一个通道，缓存对应的文件名“ bleak ”,缓存的大小为 262144,建立 4 个 cpu 缓存信道。
      if (kleak_chan) {
          char buf[64] ;
          int size = sprintf(buf,"kmalloc :addr %p caller %p, size %d\n",
                  a,__builtin_return_address(0),size) ;
          relay_write(kleak_chan,buf,size) ;
      }
      return a ;
}
```

对 kfree 函数的改造如下：

```
void kfree(const void *objp)
{
      kmem_cache_t *c ;
      unsigned long flags ;
      if (unlikely(!objp))
          return ;
      if (kleak_chan) {
          char buf[64] ;
          int size = sprintf (buf,"kfree :addr %p caller %p, size %d\n",
              objp, __builtin_return_address(0), ksize(objp)) ;
          relay_write (kleak_chan, buf, size) ;// 将要释放的数据写入到当前 CPU 对应的 relay 缓冲区内,方便用户在用户态获得。
      }
      local_irq_save(flags) ;
      kfree_debugcheck(objp) ;
      c = GET_PAGE_CACHE(virt_to_page(objp)) ;
      __cache_free(c, (void*)objp) ;
      local_irq_restore(flags) ;
}
```

通过对 kfree 函数和 kmalloc 函数的改造，即原有的函数基础上添加 struct data 数据结构和建立 kleak_chan 通道。可以利用这两个函数在内核态分配内核释放内存时,用来获取函数中内存的初始地址,分配内存大小,通过RelayFS文件系统的relay_write()函数将其写入已经建立好的文件kleak,用户空间使用标准I/O函数就可以读取文件中进程分配的起始地址大小信息。通过使用 RelayFS 建立的信道,就可以很容易跟踪内核空间中的kmalloc和kfree分配情况。RelayFS作为内核空间和用户空间写入和读取的桥梁,也就是将内核用户写入的数据缓存并转发到用户空间 。

3．测试数据

在Red Hat Linux 9.0上经过测试显示，本方法可以检测到所有的内存泄漏,并且可以提供用户修改内存错误的足够信息。 编写一个简单的驱动程序,使用改造的分配内存函数kmalloc和kfree,将这个程序insmod到内核中去,在用户空间编写一个测试程序,读取存放在文件“ kleak ”中的信息内容。由下面的实验数据看出,do_fork在kmalloc中分配了40k 大小的内存,由于没有释放,所以在kfree中被记录下来。

(1) kmallocs的实验数据：

total kmallocs by caller	size(k)
stat_open+0x0	2
svs_poll+0x54	2
d_alloc+0xf8	6
alloc_skb+0x28	327
load_elf_binary+0x47	2
do_fork+0x100	40
alloc_fdset+0xb7	2
select_bits_alloc+0x22	240

(2) kfree 的实验数据：

unfreed kmallocs	size(k)
svs_poll+0x54	2
alloc_skb+0x28	1
d_alloc+0xf8	6
select_bits_alloc+0x22	1
do_fork+0x100	40
_jbd_kmalloc+0x192	1

4．结论

针对 Linux 操作系统下驱动程序和内核模块中异常内存使用问题,本文实现了利用 RelayFS 在 kmalloc 和 kfree 函数中添加映射信道,跟踪内存的分配和释放的情况,弥补了国外类似工具在易用性、附加开销和性能等方面的不足。下一步将结合 Debugfs 一起对 Linux 内核空间内存使用异常问题的检测技术进行研究。

参考文献:

[1] Feng Qin, Shan Lu, Yuanyuan Zhou SafeMem: Exploiting ECC-Memory for Detecting Memory Leaks and Memory Corruption During Production Runs [J].Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture, HPCA-11 2005

[2] Jungeun Kim. Memory Access Optimization Through Combined Code Scheduling, Memory Allocation, and Array Binding in Embedded System Design2005[J]. ACM1-59593-058-210510006

[3] P.Zhou, F. Qin, W. Liu, Y. Zhou, J. Torrellas. iWatcher: Efficient architecture support for software debugging[J]. InProceedings of the 31st International Symposium on ComputerArchitecture. Jun 2004, pages 224-237. ISCA

[4]W. Richard Stevens. Unix 环境高级编程[M]. 尤晋元等译. 北京:机械工业出版社, 2002.

[5]倪继利. linux 内核分析及编程[M]. 北京:电子工业出版社, 2006.

[6]何杭军,朱利. 基于 Linux 的动态内存检测工具的设计与实现[J]. 计算机工程, 2006, 1.

(下转第 24 页)

中用来构建 Web Service 的文件扩展名）把原来的 WSDL 文档变成了 AO- WSDL 文档，从而可以使得服务组件的功能性和非功能性的属性更加特征化，让组件的服务和方面的信息更加容易被理解和提取。该 AOCE 模型采用了 C# . NET web service 组件来构建扩展的注册机制 AO- UDDI。用这个机制实现的 Web 服务可以保证用户的安全和交易协商等横向服务的执行，并且用 AO- UDDI 构造的用户界面可以用手动操作的方式去发现 Web 服务，还可以让 Web 服务组件更好地被描述、发现、测试和规类集成。

我们可以看出 Santokh Singh 等人，成功地利用 AOCE 设计和实现了基于. NET Web Services 的旅行计划系统。该模型为我们提供了一个很好的方案去构建基于组件的系统工程，并且他们还认为 AOCE 的模型可以同样地应用于任何基于 Web Service 的系统。

5．总结和展望

从本文可以看出基于组件的系统工程（包括基于 Web services 的系统）趋向于低水平软件组件界面设计和实现。AOCE 是一个组件发展的新技术，它能很好地分析组件和组件之间的相互服务，它能识别、描述和分析高水平的组件的功能性和非功能性的系列服务。

AOP 作为一种新的程序设计范型体现出了强大的生命力，提高了软件的可理解性、可维护性、可复用性和可扩展性。AOP 引起了软件开发方式继 OOP 以来的又一次重要变革。AOP 在 Java 平台下的技术已经很成熟，如 Aspect，而 AOP 在. Net 平台下的应用，相对于 Java 平台而言还不够成熟，功能也相对较弱。但从本文的 AOCE 模型中我们可以大胆预测，未来 AOP 在. NET 平台的应用必然会变得越来越成熟。

参考文献：

[1] 徐宝文，周超洪等. 面向方面的程序设计：概念、实现与未来[J]. 计算机与数字工程，2006，33(8)：1- 10.

[2] 丁辉，姚庆文. 面向方面编程的研究[J]. 山东理工大学学报，2005，19(6)：54- 58.

[3] 曹东刚，梅宏. 面向 Aspect 的程序设计——一种新的编程范型[J]. 计算机科学，2003，30(9)5- 10.

[4] 宋小鹏，盛仲飚等. 面向方面编程方法的研究[J]. 微计算机信息 2006，22(4-3)1-3.

[5] 刘岩，毛迪林等.基于 AOP 的 Web Services 管理架构研究[J]. 计算机应用与软件，2006，23(11)11- 14.

[6] Santokh Singh，etc. "Developing. NET Web Service-based Applications with Aspe-ct-Oriented Component Engineering". In Proceedings of the Fifth Australasian Workshop on Software and Systems Arch-itecures，Melbourne，Australia，13-14 April 2004.

Research For Aspect-Oriented Component Engineering

Zhang Yichao[1,2] Zhang Lichen[1] Nie Huabei[2]

(1. College of Computer，Guangdong University of Technology，Guangzhou 510090，Guangdong；

2. Department of Computer，Guangzhou kangda vocational Technical college，Guangzhou 511363，Guangdong)

【Abstract】 In this paper we research for Aspect-Oriented Component Engineering and show how Aspect-Oriented Component Engineering can be implemented based on .NET platform.

【Keywords】 AOP；Web Services；NET；AOCE

（上接第 18 页）

The Kernelly Memory Leaked Test & Trace Using RelayFS

Zhang Aiguo[1] Liu Xiaopeng[2]

(1. Zhongshan Torch Polytechnic，Zhongshan 528436，Guangdong；

2. School of Computer Science and Engineering UESTC of China，Chengdu 610054，Sichuan)

【Abstract】 The paper presents the method of using RelayFS to reconstruct the kamlloc and kfree functions，establishes the channel between these functions，and traces the kernelly memory，avoiding the memory leak effectively.

【Keywords】 RelayRS；kmalloc；kfree；memory leak