Snort数据包捕获性能的分析与改进

李 伟[1] 鲁士文[2]
[1] (中国科学院研究生院 北京 100036)
[2] (中国科学院计算技术研究所 北京 100080)

摘 要 基于 Snort的入侵检测系统运行在 Linux操作系统平台,捕获数据包的工作是借助 Libpcap由 Linux操作系统内核完成的。要提高入侵检测系统的效率,首先要保证捕获数据包的效率。本文对 Linux的数据包捕获机制进行分析,然后利用 NAPI技术和内存映射技术对 Snort进行改进。试验结果表明,使用 NAPI和内存映射技术后, Snort系统的性能得到明显的改善。

关键词 入侵监测系统 Linux Snort 数据报捕获

ANALYSIS AND IMPROVEMENT OF PACKET CAPTURE IN SNORT

LiWei[1] Lu Shiwen[2]
[1] (Graduate School, Chinese Academy of Sciences, Beijing 100036, China)
[2] (Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China)

Abstract Snort based IDS runs on Linux, and packet capture is realized through Libpcap and Linux kernel. High efficiency of packet capture is important for the performance of the whole IDS. This paper analyzes the mechanism of packet capture firstly, then improves it through NAPI and MMAP. Experimentation results indicate that the performance of Snort is improved significantly in this way.

Keywords IDS Linux Snort Packet capture

0 引 言

随着计算机网络的飞速发展,社会信息化程度不断提高,网络在带来巨大的经济效益和社会效益的同时,也面临日益严重的安全问题。单从防御的角度构造安全系统是远远不够的。入侵检测技术是继“防火墙”、“数据加密”等传统安全保护措施之后的新一代的安全保障技术。它与防火墙、病毒防护、漏洞扫描和数据加密等技术一起,构筑了计算机网络安全的多层防护体系。

近些年,入侵检测技术不断完善,但随着高速网络的发展,总有新的问题出现,其中非常重要的一个问题是传统的入侵检测系统已经无法及时、有效地处理高速的网络流量。对于 Snort系统而言,需要监听网络中的所有数据包。如果系统不能及时得到所有的数据包, Snort的作用就会大大降低。在 Linux环境下,当数据包流量较大时,大量的数据包就会被丢弃,从而导致 Snort系统的效率大大降低。为此,本文对基于 Snort的网络入侵检测系统的数据包捕获性能瓶颈进行了分析,并做了相应改进工作。

1 Linux捕获数据包的流程分析

在 Snort进行初始化时,调用 open_pcap()函数对网卡进行初始化,使其工作在混杂模式,对网络上的所有数据包进行监听。当有数据包到达网卡时,会产生一个硬件中断,然后调用网卡驱动程序中的函数来处理。这个中断处理程序首先要做的就是进行一些 I/O操作将数据读入;当数据帧成功接收后,收到的数据包会被封装成 sk_buff结构,并脱离驱动程序,转到通用的处理函数 netif_rx中。netif_rx的一个重要工作就是将传入的 sk_buff放到等候队列中,并置软中断标志位。其目的是快速从中断中返回,等待下一个数据包的到来,从而提高对网络数据包的处理速度。图 1是 Linux网络设备工作原理[1]示意图。 netif_rx将数据包传入等待队列之后,为了提高 CPU的处理效率,上层的处理采用软中断 do_softirq实现。由于在系统初始化时,已经将 NET_RX_SOFTIRQ软中断的处理函数定为 net_rx_action(),因此 do_softirq()将调用 net_rx_action()对数据包进行处理。在函数 net_rx_action()中,根据数据包类型的不同,调用 ip_recv()或者 packet_recv()函数,对 netif_rx()传入的数据包进行处理。处理的最终结果是将数据包放到 Socket等待队列中,并通知上层有数据包到达。

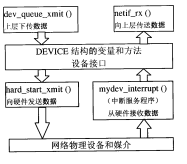

图 1 Linux网络设备工作原理图

Snort是通过调用 Libpcap的 pcap_loop()进而调用 pcap_read()从内核中获取数据包的。Libpcap位于用户空间,数据包位于内核空间, pcap_read()需要通过 recv_from()系统调用每次从内核中取一个数据包。如果内核中没有数据包, recv_from()系统调用就会将 pcap_read进程阻塞,直到 data_ready通知进程有数据包到达后继续处理。

数据包从网络设备到 Linux内核,需要通过中断——每到达一个数据包,都会通过一个中断将数据包送到内核中。数据

收稿日期: 2004 - 09 - 03。李伟,编辑,主研领域:网络安全。

包从内核中的缓冲队列到用户空间,需要经过一次系统调用。同时在这个过程中,数据包经历了从网卡到内核空间、从内核空间到用户空间两次数据包拷贝。在操作系统中,中断、系统调用以及内存操作都是非常消耗系统资源的,频繁的系统调用和内存拷贝会成为系统性能的瓶颈。

2　使用 NAPI技术提高数据包捕获的效率

NAPI[2]是 Jamal Hadi Salim等提出的对当前的 Linux协议栈进行改进的一种方法。其主要思想是结合中断与轮询的优点,有效解决网络高负载情况下带来的拥塞冲突问题。本文借鉴了 NAPI技术的思想,用于解决 Snort入侵检测系统中网络高负载情况下中断次数过于频繁的问题。

2.1　中断方式与轮询方式

实际上,物理网络设备接收到数据时,系统可通过两种途径知道数据包到达,并读取数据。一种是轮询方式,系统每隔一定时间间隔就去检查一次物理设备,若设备"报告"有数据到达,就调用读取数据的程序。另一种是中断方式,利用硬件体系结构的中断机制实现设备和系统的应答对话,即:当物理设备需要 CPU处理数据时,设备就发一个中断信号给系统,系统则在收到信号后调用相应的中断服务程序响应设备的中断请求。

轮询在重负载的情况下非常有效,但是在负载较轻的情况下,可能会带来较大的处理延时,因为网络设备有数据到达时可能不能马上得到 CPU的响应,同时,不断轮询没有任何数据包的网络设备是对 CPU资源的浪费。

中断方式有效地解决了设备与 CPU的对话交流问题,并将 CPU从繁重的设备轮询中解脱出来,大大提高了 CPU的利用率。在轻负载的情况下减少了处理的时间延迟,但是当中断量超过 MLFFR (Maximum Loss Free Forward Rate)时,中断活锁(livelock)[3]问题会导致系统性能降低。

因此,轮询方式适用于重负载情况,如果在轻负载情况下,它会带来 CPU资源的浪费以及较大处理延时;中断方式适用于轻负载情况,如果在重负载情况下,它可能带来中断活锁问题,导致系统效率的严重下降。

2.2　NAPI技术

NAPI是一种能够在 Linux系统中提高网络性能的方法。它结合了中断方式与轮询方式的优点,提出了一个中间道路——在轻负载情况下,处理方式趋近于中断;在重负载的情况下,处理方式趋近于轮询。同时,NAPI还充分考虑到了多个网络设备的公平调度问题。

NAPI的基本思想是:一批数据包中的第一个数据包到达时,采用中断方式通知系统,系统将该设备注册到一个设备轮询队列中,并关闭对该设备的中断响应。同时,激活一个软中断,对轮询队列中注册的网络设备进行轮询,从中读取数据包。为了能够保证对各个网络设备的公平调度,NAPI技术引入了配额概念。所谓配额,是指网络设备每次向当前 CPU发送的数据包的最大个数,这个数值是可以配置的,记为 P。如果网络设备发送了 P个数据包,而设备的数据包接收缓冲区中仍然有数据,就将该设备重新注册到轮询队列的末尾,等待下一次轮询到该设备时继续向内核提交数据包;否则,将该设备从轮询队列中注销,同时打开对该设备的中断的响应。这样,在系统中有多个网络设备的情况下,可以保证各个设备的公平调度。使用 NAPI技术的处理流程如图 2所示。

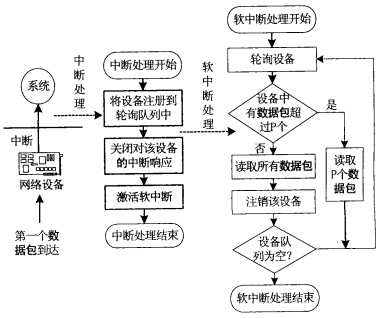

图 2　NAPI的流程图

当网络负载较低时,每次到达网卡的数据包个数比较少,极端情况下只有一个数据包到达。那么,当网卡将该数据包提交到上层之后,就从轮询队列中注销,同时开中断,等待下一个网络数据包的到达。这与 Linux 2.4采用的中断方式是一致的。

当网络负载较高时,每次会有一批数据到达网卡,在极端情况下,数据包源源不断的到达。由于接收缓冲区是一个环形的缓冲区,网卡每向内核提交一个数据包,立刻就会有一个数据包填充进来,接收缓冲区一直处于满的状态。在这种情况下,网卡就一直处于在轮询队列中,并行地进行读取网络上数据包和向上提交数据包的工作。这与轮询方式是一致的。也就是说,采用 NAPI技术,使系统在轻负载的情况下趋近于中断方式,响应速度很快;而在重负载情况下,趋近于轮询方式,高效地处理数据包并且避免重负载情况下的中断活锁问题。

3　使用内存映射技术提高数据包捕获的效率

内存映射技术的主要思想是通过使内核空间和用户空间共享一段内存区域,避免系统调用,从而提高捕获数据包的效率。

3.1　Linux的内存管理模式

Linux是一个具有保护模式的操作系统[4]。内存被分为两个部分:内核空间和用户空间。内核空间存放并运行着核心代码,用户空间存放并运行用户程序。用户进程不能访问内核区域的地址空间也不能访问其他用户进程的地址空间,同样,内核进程也不能访问用户进程的地址空间。用户进程如果需要访问内核空间中的数据,需要通过系统调用。

因此,Snort从内核空间读取数据包必须通过 recv_from系统调用完成。我们知道,系统调用是非常消耗系统资源的,而且,每次系统调用都会带来一次上下文切换,对 Cache的命中率有很大的影响。随着网络流量的增大,网络中数据包的增加,系统调用就会非常频繁,严重影响 Snort的性能。因此,系统调用已经成为 Snort的性能瓶颈了。

为此,我们借鉴了内存管理中地址映射技术 (MMAP)的思想,通过使内核与应用程序共享一段内存空间的方法,避免系统调用,从而降低系统调用的开销,提高 Cache命中率,并提高系统性能。

3.2　内存映射技术

首先,我们构造了一个与内核空间共享的环形缓冲区。该缓冲区中存放的是 Linux内核捕获的数据包在内存中的地址。内核每捕获一个数据包,就将该数据包存放的地址放在环形共享缓冲区中。在 pcap_read()函数中,调用了一个叫做 pcap_ring_recv()的函数。该函数不断地检查共享环形缓冲区中是否有数据,如果有数据,就根据这个地址取得内存中的数据包;如果没有数据,就会将自己阻塞,直到有数据包到达时才被唤醒,开始读取数据。

(下转第 110页)

此,BBS生成器的安全性建立在与 RSA同样的基础之上。

4.2 素性检测

素性检测就是判断一个整数是否为素数的准则。令 n是一个大的奇整数,假设相确定 n是否为素数,最简单的素性检测就是"试除法",即取奇整数 m,看 m是否整除 n,当 m遍历了从 3到$\sqrt{n}$之间的所有奇数,而 n通过了所有的这些试除检测,则 n是素数。显然,这种素性检测方法是十分耗时的,在实际中我们需要更有效的素性检测方法。

Miller-Rabin检测使实际中应用最多的是素性检测,它在计算上较省时、容易实现且错误概率较低。下面是 Miller-Rabin素性检测过程 miller_rabin(a, n)的算法描述:

①设 $b_k b_{k-1}\cdots b_0$是(n-1)的二进制表示
② d←1
③ for i←k downto 0
④ do x←d
⑤ d←(d×d) mod n
⑥ if(d=1 & x≠1& x≠n-1) return TRUE
⑦ if (bi=1) then d←(d×a) mod n
⑧ if(d≠1) return TRUE
⑨ else return FALSE

其中 a为安全参数,n为待检测的整数。

5 结束语

RSA方法即可用于保密,也能用于签名和认证,目前已被广泛应用于各种安全或认证领域,如 Web服务器和浏览器信息安全、Email的安全和认证、对远程登录的安全保证和各种电子信用卡系统的核心。许多流行的操作系统上如微软、Apple、Sun和 Novell也在其产品上融入了 RSA。在硬件上,如安全电话、以太网卡和智能卡都使用了 RSA技术。而且几乎所有 Internet安全协议如 S/MIME、SSL 和 S/WAN 都引入了 RSA 加密方法。ISO9796标准把 RSA列为一种兼容的加密算法。可以预见,RSA的应用将会越来越广泛。

参 考 文 献

[1] 陈鲁生、沈世镒,现代密码学,科学出版社,2002年 7月.
[2] William Stallings[美]著,密码学与网络安全:原理与实践(第二版),电子工业出版社,2001年 4月.
[3] Douglas R. Stinson,密码学原理与实践,电子工业出版社,2003年 2月.
[4] 周玉洁、冯登国,公开密钥密码算法及其快速实现,国防工业出版社,2002年 9月.

(上接第 105页)

这样,如果网络上有大量的数据包到达,内核就可以不断地向环形缓冲区中写数据,Libpcap就可以不停地从环形的缓冲区中读取数据,数据包的并行读写工作互不影响。因此,系统的处理速度加快。而且,在这个过程中,不需要通过系统调用从内核中获取数据包,从而消除系统调用的开销。因而,有效地提高了系统性能。

4 实验结果及分析

本文采用 oprofile对系统运行过程中的中断次数和系统调用次数进行了对比。改进前后中断个数的对比情况如图 3所示。图中可以看出,使用 NAPI技术后,相同流量下,系统中断的次数明显降低,尤其是在较大流量的情况下。这正是因为 NAPI技术在网络负载较小的情况下趋近于中断,而在网络负载较大的情况下趋近于轮询。

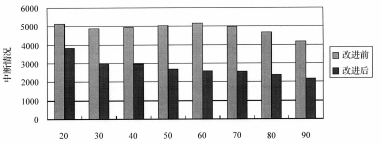

图 3 改进前后中断情况对比

改进前后系统调用的对比情况如图 4所示。从图中可以看出,通过采用地址映射技术,系统调用的次数明显减少,几乎达到可以忽略不计的程度。由此可以说明,内存映射技术可以大幅度降低系统调用次数。

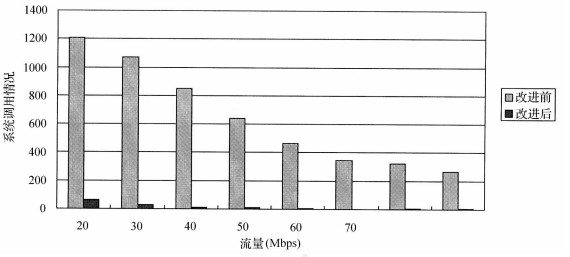

图 4 改进前后的系统调用情况对比

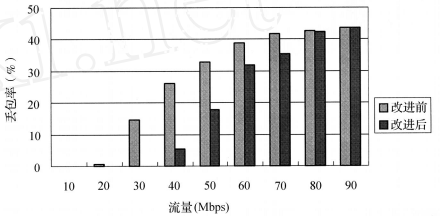

图 5 改进前后 Snort1.9丢包率对比

图 5是改进前后 Snort1.9丢包情况的对比。从图中可以看出 NPAI及内存映射技术的使用对于消除 Snort性能瓶颈起到了积极作用,系统丢包率明显降低。但是,在网络流量较高的情况下,Snort1.9仍有较高的丢包率,这有待于通过提高攻击检测的效率来解决。

5 总 结

本文首先分析了基于 Snort的网络入侵检测系统数据包捕获的瓶颈所在,然后,采用 NAPI技术,解决了高网络负载情况下,由于频繁中断导致系统性能下降的问题;采用内存映射技术,使内核空间与用户空间共享同一块内存区域,从而避免了每次读取数据包都需要经过系统调用的问题。实验结果表明,改进后的入侵检测系统数据包的捕获效率得到了明显的提高。

参 考 文 献

[1] 陈莉军,Linux操作系统内核分析,北京:人民邮电出版社,2000.
[2] Jamal Hadi Salim and Rovert Olsson, Beyond Softnet, the Proceedings of the 5th Annual Linux Showcase & Conference, 2001.
[3] J. Mogul and K K Ramakrishnan, Eliminating receive livelock in an interrupt driven kernel, Winter USENIX Conference, Jan. 1996.
[4] Alessandro Rubini & Jonathan Corbet, Linux Device Drivers, http://www.oreilly.com/catalog/linuxdrive2/chapter/bookindexpdf.html, 2001.