基于Linux的网络信息内容分析与监管技术*)

李方敏[1,2] 周祖德[1] 刘 泉[1]

(武汉理工大学信息工程学院 武汉430070)[1] (湘潭工学院计算机系 湘潭411201)[2]

Network Information Content Analysis and Monitor Technology Based on Linux

LI Fang-Min[1,2] ZHOU Zhu-De[1] LIU Quan[1]

(College of Information Engineering, Wuhan University of Technology, Wuhan 430070)[1]

(Department of Computer, Xiangtan Polytechnic University, Xiangtan 411201)[2]

Abstract With the development of computer network, network security has become a very important problem that must be solved by us. A distributed and layered skeleton has been presented, which can be deployed over all the country easily. The monitor and management system can recognize the illegal information and automatically take action according the analysis result. As a sample, an illegal VPN(Virtual Private Network) monitor and management system have been implemented based on Linux, which can dynamically find illegal IPSec tunnels and then filter them.

Keywords Linux, Network information, Content analyse, Monitor technology, VPN

1. 基于层次的监管体系结构

如何在全国范围建立一个对网络信息进行监控的管理体系，能实时地识别各种应用中的非法信息，并能自动地采取相应的行动，如记录、过滤、报警等。该监控系统应尽量减少对正常应用的影响。

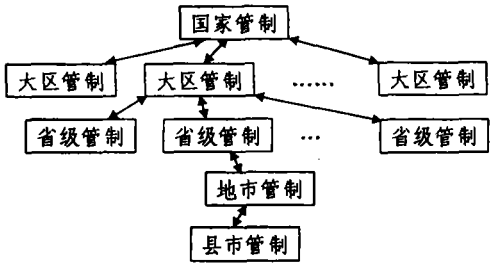

图1 基于层次的监管体系结构

在实现全国范围的监控管制时，采用分布式方式，在每个地市有一个三级管制中心，每个省有一个二级管制，国家为一级管制，根据需要可以设置县市管制和大区(位于省和国家之间)，其组织如图1所示。在层次管制节点之间的安全连接采用通过国密办认证的VPN系统，如湖北信安通公司。

在每个监管点配置的监控系统，包括两个方面：对网络信息内容的识别和拦截。

1.1 识别技术

识别技术的研究包括两个方面，首先对满足特定规则的文本、语音、图像的识别方法的理论研究，其次，采用可重构的FPGA、DSP研制硬件配合软件识别非法信息。

·*基于文本的分类方法研究* 当前，用于基于文本内容过滤的文本分类方法主要有基于规则和基于概率两种，后者已成为一种主要研究趋势。我们拟研究一种新的分类方法，在实时性、准确性、效率等方面有较高的性能，能有效应用于基于文本的网络应用，如BBS、电子邮件、OCIQ等。

·*语音信息识别分析技术* 研究语音技术与网络技术的有效融合。重点解决网络环境下语音识别系统自适应性、实时响应和实用性，以便对基于语音的网络应用进行监控，能识别特定的关键词，如"爆炸"等，并对可疑的语音信息进行记录。

·*图像识别分析技术* 主要对色情图片能有效识别，根据识别结果进行拦截。

1.2 拦截

采用专用的软、硬件设计，识别机能实时地对200Mbps的数据流进行分析，然后再根据分析的结果，与拦截机器通信，实现拦截功能。拦截机器的高速过滤实现主要包括两个层面：定制可重构网卡的FPGA和操作系统的内核。

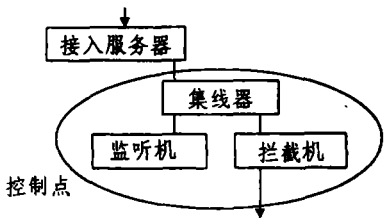

图2 监控点的配置示意图

作为实例，目前，课题组已实现非法VPN的发现和拦截系统，下面我们分析该系统实现的主要技术。

2. 非法VPN的发现和拦截

2.1 发现的前提

所谓非法的VPN[1]连接指的是建立连接的端点IP未在指定的机构登记，我们假定登记中心采用层次的、分级的树形结构，并且假设每个管制中心，即非法VPN识别和拦截点能获得相应的合法VPN登记数据。因此，我们在识别的时候可采用隧道的端点IP与合法VPN登记数据库中IP对比较，如

*)本文受湖南省自然科学基金(编号:01JJY2064)和"国家信息关防与网络安全保障持续发展计划"项目(编号2001-研2-B-003)的资助。李方敏 博士后，教授，研究方向为计算机网络及安全。周祖德 教授，博导。刘 泉 教授，博导。

果有，则为合法 VPN，否则为非法 VPN。

2.2 IP 头结构分析及非法 VPN 识别

不管 VPN 采用传送模式还是通道模式，采用 ESP 或 AH 协议，我们在识别的时候，只需处理 IPSec[1]头之前的 IP 头，其结构如图3所示。

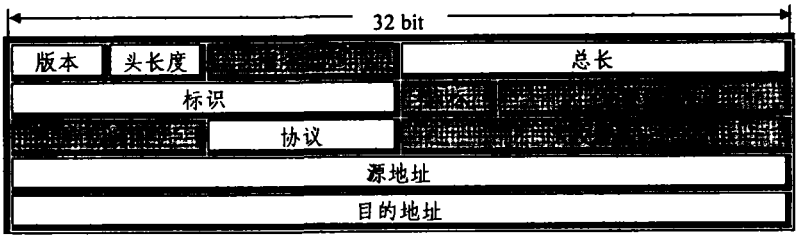

图3 IP 头结构

我们需用到协议、源地址、目的地址三个字段。如果，协议字段的值为50则为 ESP 包，如为51则是 AH 包，其它则不是 IPSec 数据，无需作进一步的处理。对 IPSec 数据过滤目前可以考虑两种方法：(1)如果路由器支持过滤，且不影响现有业务，可以采用类似的方法，将所有的 VPN 数据旁路到某个指定端口，则在该端口接收的所有数据都是 VPN 数据；(2)在识别机器上采用 BPF 在内核进行，然后将其 IP 头位于用户空间的识别进程进行处理。

在两种情况下，识别进程采用高效的算法基于2.1节识别非法 VPN 连接。

2.3 监听与拦截的交互

现在我们看一下整个设计的程序关系交互图(图2)：

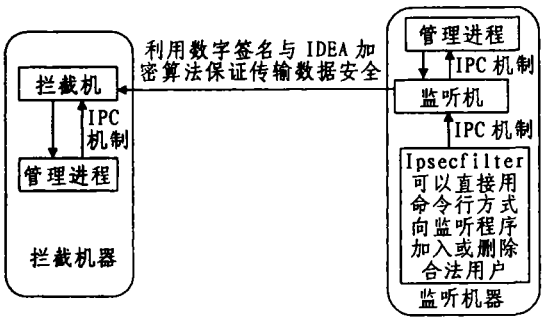

图4 程序交互关系图

拦截机的主要程序是 dae_route.c，proc_net.c，add_netflt.c，其中 dae_route 作为一个守护进程运行，负责从网络接受从监听机传送过来的非法 VPN 的两端 IP 地址，然后写入由 proc_net 创建的/proc 文件 netfltdata，它在内核里维护了一个存放非法连接的数据链表，而 add_netflt 就利用 Linux 2.4内核的 netfitler 技术，通过钩子挂上了自己的处理函数来过滤非法的数据。为了保证 dae_route 的运行，应该设计一个用来专门管理 dae_route 程序的进程，它可以监视 dae_route 的运行状态，一旦 dae_route 由于某种原因而死去，这个管理进程可以在一段时间内自动检测到，然后重新启动 dae_route 进程。具体实现细节如下：dae_route 有个定时器，它每隔一段时间向管理进程发一个信号，而在管理进程中也有一个定时器，它的触发时间是 dae_route 的一倍以上，当收到 dae_route 发出的信号时，就把定时器重置为原来的设置时间。这样在一段时间内如果没有收到 dae_route 发出的信号，就会导致定时器到时，这就认为 dae_route 进程已死去，重新启动一个。

在监听机端，主要是 sniff_ipsec.c，它利用 BPF 机制对协议头进行分析，获取所有 IPSec 数据，因为所有合法 VPN 的用户数据把它的隧道两端的 IP 地址保存在一个平衡二叉树(legal)中，当 sniff_ipsec 监听到一个 IPSec 数据分组，它马上查找这个平衡二叉树，如是合法的不理它，否则先查找一个专门存放已经发送给拦截机的非法 VPN 数据的平衡二叉树(illegal)，如果它已经被拦截了，不理它，否则，用电子签名＋IDEA 加密算法的方法处理数据，发给拦截机进行拦截。当然，illegal 在一定的时间后会自动刷新，在程序中我们设置为24个小时，同时重新交换一个密钥。ipsecfilter 是一个单独的程序，它利用了 FIFO(有名管道)与 sniff_ipsec 进程通信，直接在 legal 中加入或删除数据。

3. 平衡二叉树

在整个设计中，我们需要一个数据结构来存放合法 IPSec 隧道的两端 IP 地址，考虑到这个数据不是需要经常更新的，而且数据变化不是非常大，例如：增加或删除一个合法的用户。而我们要求它的查找速度要足够地快，因为对每一个 VPN 分组我们都要查找这个数据结构看它是否合法，如果这个数据结构的查找速度不够快，就会发生丢包现象，这是应该尽量避免的。所以经过分析，这个数据结构的特点有：1. 实现较容易。2. 查找速度快。3. 修改数据的速度可以相对较慢。4. 除了必要的数据以外，其他数据(比如维护整个数据结构的内部数据)应该尽量少。

平衡二叉树(Balanced Binary Tree 或 Height-Balanced Tree)又称为 AVL 树。它或者是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。若将二叉树上结点的平衡因子 BF(Balance Factor)定义为该结点的左子树的深度减去它的右子树的深度，则平衡二叉树上所有结点的平衡因子只可能是－1、0、1。只要二叉树上有一个结点的平衡因子的绝对值大于1，则该二叉树就是不平衡的。

由于平衡二叉树的整个树型结构是平衡的，而它又有二叉排序树的特点：(1)若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；(2)它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；(3)它的左、右子树也分别为二叉排序树。所以它的查找速度相当于排序数组中的二分查找法，它查找的时间复杂度为○(log n)。

一个合法用户需要记录两个 IP 地址，共8个字节，要维护一个平衡二叉树数据结构需要两个指针(一个指向左子树，一个指向右子树，各4个字节)，还有一个 char 型的平衡因子(一个字节)如下：

```
struct avlnode{
char data[8];
char bf;
```

```
struct avlnode * lchild, * rchild;
}
```

那么1M=(1024*1024)/(8+8+1)≈61680,即1M的内存空间能放6万来个用户数据,这已经是足够的了。对于有大量数据的修改,我们先把它保存在一个文件里,然后设置一个时间间隔来定时刷新,几天甚至几个月。

4. Netfilter 框架分析

为了在 Linux 内核方便地添加功能,使用了 netfilter[2,3] 框架,它是 Linux 2.4内核采用的防火墙技术,以更好的结构重新构造,并实现了许多新功能,如完整的动态 NAT(2.2内核实际是多对一的"地址伪装"),基于 MAC 及用户的过滤,真正的基于状态的过滤(不再是简单地查看 tcp 的标志位等),包速率限制等。

在原有的网络部分的 LKM(Loadable Kernel Module)中,如果对网络部分进行处理,一般是先生成 struct packet_type 结构,在用 dev_add_pack 将其插入网络层(注意此时的 packet_type 实际相当于一个三层的协议,如 ip_packet_type,ipx_8023_packet_type 等)。而 netfilter 本身在 IP 层内提供了另外的5个插入点(其文档中称为 HOOK):NF_IP_PRE_ROUTING,NF_IP_LOCAL_IN,NF_IP_FORWARD,NF_IP_LOCAL_OUT,NF_IP_POST_ROUTING。它们分别对应 IP 层的五个不同位置,这样理论上在写 LKM 时便可以选择更适合的切入点,再辅以 netfilter 内置的新功能(如 connect tracking),应该会帮助写出功能更强的 LKM。

通俗地说,netfilter 的架构就是在整个网络流程的若干位置放置了一些检测点(HOOK),而在每个检测点上登记了一些处理函数进行处理(如包过滤,NAT 等,甚至可以是用户自定义的功能)。

IP 层的五个 HOOK 点的位置如图5所示(copy from 〈packet filter howto〉):

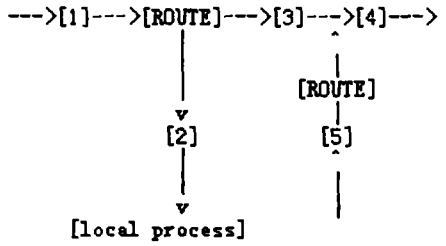

图5 netfilter 结构图

[1]:NF_IP_PRE_ROUTING:刚刚进入网络层的数据包通过此点(刚刚进行完版本号、校验和等检测),源地址转换在此点进行;

[2]:NF_IP_LOCAL_IN:经路由查找后,送往本机的通过此检查点,INPUT 包过滤在此点进行;

[3]:NF_IP_FORWARD:要转发的包通过此检测点,FORWORD 包过滤在此点进行;

[4]:NF_IP_LOCAL_OUT:本机进程发出的包通过此检测点,OUTPUT 包过滤在此点进行;

[5]:NF_IP_POST_ROUTING:所有马上便要通过网络设备出去的包通过此检测点,内置的目的地址转换功能(包括地址伪装)在此点进行。

在 IP 层代码中,有一些带有 NF_HOOK 宏的语句,如 IP 的转发函数中有:

```
〈-ipforward.c ip_forward()-〉
NF_HOOK(PF_INET, NF_IP_FORWARD, skb, skb->
                dev, dev2,ip_forward_finish);
```

其中 NF_HOOK 宏的定义提炼如下:

```
〈-/include/linux/netfilter.h-〉
#ifdef CONFIG_NETFILTER
#define NF_HOOK(pf, hook, skb, indev, outdev, okfn) \
(list_empty(&nf_hooks[(pf)][(hook)]) \
? (okfn)(skb) \
:nf_hook_slow((pf), (hook), (skb), (indev), (outdev),
  (okfn)))
#else /* !CONFIG_NETFILTER */
# define NF_HOOK(pf, hook, skb, indev, outdev, okfn) (okfn)
  (skb)
#endif /* CONFIG_NETFILTER */
```

如果在编译内核时没有配置 netfilter 时,就相当于调用最后一个参数,此例中即执行 ip_forward_finish 函数;否则进入 HOOK 点,执行通过 nf_register_hook()登记的功能(这句话表达得可能比较含糊,实际是进入 nf_hook_slow()函数,再由它执行登记的函数)。

NF_HOOK 宏的参数分别为:1.pf:协议族名,netfilter 架构同样可以用于 IP 层之外,因此这个变量还可以有诸如 PF_INET6,PF_DECnet 等名字;2.hook:HOOK 点的名字,对于 IP 层,就是取上面的五个值;3.skb:Linux 下用它来指向每一个数据分组;4.indev:进来的设备,以 struct net_device 结构表示;5.outdev:出去的设备,以 struct net_device 结构表示;(后面可以看到,以上五个参数将传到用 nf_register_hook 登记的处理函数中。)6.okfn:是个函数指针,当所有的该 HOOK 点的所有登记函数调用完后,转而走此流程。

这些点是已经在内核中定义好的,除非你是这部分内核代码的维护者,否则无权增加或修改,而在此检测点进行的处理,则可由用户指定。像 packet filter,AT,connectiontrack 这些功能,也是以这种方式提供的。正如 netfilter 当初的设计目标——提供一个完善灵活的框架,为扩展功能提供方便。为了加入自己的处理功能,采用 nf_register_hook 函数,其函数原型为:

```
int nf_register_hook(struct nf_hook_ops * reg)
```

我们考察一下 struct nf_hook_ops 结构:

```
struct nf_hook_ops
{
  struct list_head list;
  /* User fills in from here down. */
  nf_hookfn * hook;
  int pf;
  int hooknum;
  /* Hooks are ordered in ascending priority. */
  int priority;
};
```

我们的工作便是生成一个 struct nf_hook_ops 结构的实例,并用 nf_register_hook 将其 HOOK 上。其中 list 项总要初始化为{NULL,NULL};由于一般在 IP 层工作,pf 总是 PF_INET;hooknum 就是我们选择的 HOOK 点;一个 HOOK 点可能挂多个处理函数,谁先谁后,便要看优先级,即 priority 的指定了。netfilter_ipv4.h 中用一个枚举类型指定了内置的处理函数的优先级:

```
enum nf_ip_hook_priorities {
  NF_IP_PRI_FIRST = INT_MIN,
  NF_IP_PRI_CONNTRACK = - 200, NF_IP_PRI_MANGLE
  = -150,
  NF_IP_PRI_NAT_DST= -100,
  NF_IP_PRI_FILTER = 0,
  NF_IP_PRI_NAT_SRC = 100,
  NF_IP_PRI_LAST = INT_MAX,
};
```

(下转第75页)

过的部分丢失；Kk 表示总队长中预取量的阈值，当总队长达到此长度后，服务开始；Nn 表示总队长的最大值，$Nn = SubNn * M$；H_i 表示未开始服务时，总队长中有 i 个数据包的概率，$i=1,2,\cdots,Kk-1$；P_i 表示开始服务后，总队长中有 i 个数据包的概率，$i=0,1,2,\cdots,Nn$；$f_{i,j}$ 表示在状态 i 时服务 j 个数据包的概率；$g_{i,j}$ 表示在未服务的状态 i 时，由于子队长达到 $SubNn$ 启动服务后服务 j 个数据包的概率。

由于该方程涉及的未知量太多，$g_{i,j}$、$f_{i,j}$ 并没有历史数据可以给出，理论上直接推导属于 NP 难度问题。因此，下面将借助于实际系统的试验数据来获得我们需要的解。

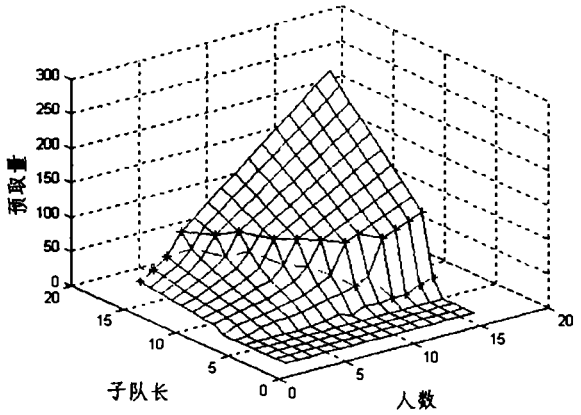

图4　人数与子队长和预取量之间的关系曲线

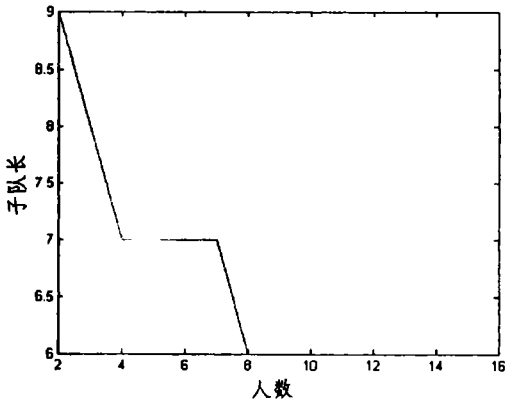

图5　人数与最优的子队长之间的关系曲线

在试验中，我们将事先按照 Poisson 分布确定的数据（代表语音包）向队列中发送，同时系统按照平均服务间隔从队列中取出数据进行服务。系统首先预取一定的数据量 Kk（可调整）；在此过程中检测系统的状态（即包含的数据包个数），若达到开始服务的条件，则开始服务。重复服务足够多次，然后观察各队列中所包含的数据量 Nn（即实际占用的队长）的概率和数据包损失的概率。改变 Kk，可以获得不同的 Nn 值。另外，调整会议规模，重复多次进行上述试验，又可以获得不同的 Kk 与 Nn 的关系。将试验数据中的多组的子队长、预取量、总队长及损失率数据，并以文件的形式记录下来。根据这些数据，我们可以得到图4和图5。

图4给出了不同会议规模（人数）情况下（考虑1%损失率），子队长（$SubNn$ 的可取值）、预取量 Kk 与人数之间的三维关系曲线。在图中，有两条分别连接拐点的曲线。预取量较小的曲线表明实际计算出来的预取量的可取值范围，在该曲线以下的数据均是计算得出的。预取量较大的曲线表明理论上的预取量的可取值，该曲线以上所有数据是为了绘制三维曲线而根据已有的数据估算出的，仅具有参考意义。

在实际应用中，由于考虑会议系统的实时性要求，$SubNn$ 应尽可能取小。故在取值时取可以满足要求的最小的 $SubNn$ 及此时对应的 Kk。我们绘制出最优子队长（最小的 $SubNn$）与会议规模（人数）之间的关系曲线，如图5所示。

结论　从图4可以看出，当人数（会议规模）固定时，预取量 Kk 与子队长 $SubNn$ 之间近似为线性关系，并且随着人数的增加，预取量 Kk 与子队长 $SubNn$ 的线性系数随之增加（图中，人数增加时，Kk 与 $SubNn$ 之间的关系曲线越陡）。当子队长 $SubNn$ 固定时，人数增加，所需的预取量 Kk 也增加，这同我们在实际中的一般认识是一致的，该图为我们实验的最终图形。若给出参加会议的人数，从该图中可以方便地查出所需的子队长的大小和预取值的大小。从图5中可以看出，当人数小于8时，最优子队长 $SubNn$ 随着人数的增加而减小，在此之后，会议规模再扩大（人数再增加），最优子队长 $SunNn$ 趋于一个恒定值（等于6）。

为了检验计算机模拟所得数据的可靠性，我们对数据在实际系统中进行了验证。即把该模型所得实验结果在视频会议的模拟系统上进行测试，所得的声音效果清晰可辨，完全可以达到会议的要求。

参 考 文 献

1　Yang Shutang. Multipoint Communications with Speech Mixing over IP Network, Computer Communications, (将发)

2　杨树堂，余胜生，周敬利. 基于分组网络的多点实时语音混合及调度算法. 软件学报，2001，12(9)：1413～1419

3　CCITT (ITU-T) Rec. G. 711: Pulse code modulation (PCM) of voice frequencies, Nov. 1988

4　孟玉珂. 排队论基础及应用. 同济大学出版社，1989

5　华兴(美). 排队论与随机服务系统. 上海翻译出版公司，1987

(上接第69页)

hook 是提供的处理函数，也就是我们的主要工作，其原型为：

```
unsigned int nf_hookfn(unsigned int hooknum,
                       struct sk_buff **skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff *));
```

它的五个参数将由 NFHOOK 宏传进去。

结束语　本文提出了一种基于层次的监管体系，作为实例，在 Linux 内核实现了非法 VPN 的识别和拦截功能，对实现的几个主要部分和用到的关键技术进行了详细的分析。课题组在校园网环境建立了一个仿真 Internet 的实验测试平台，在100Mbps 的满负荷运行下，非法 VPN 连接的识别和拦截率达到98%以上，达到了设计要求。

参 考 文 献

1　Doraswamy N, Harkins D 著，京京工作室译. IPSec 新一代因特网安全标准. 北京：机械工业出版社，1999

2　netfilter hacking HOWTO. http://netfilter.samba.org/unreliable-guides/netfilter-hacking-HOWTO/index.html

3　Linux 2.4 Packet Filtering HOWTO. http://netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/index.html