第五章 反入侵技术(II): 基于网络的机制

这一章阐述基于网络的反入侵技术，包括防火墙和运行在边界网关上的 IDS(即 NIDS)。这些机制的共同特点在于它们用以实施入侵检测的依据都是网络上的数据流，即通过观察流经它们的 IP 分组来推断和判定这些分组是否蕴涵潜在的攻击。因为只能间接地推断、而非如 HIDS 那样能直接观测到 IP 分组在某些目标机器上引发的后果，因此 NIDS 的设计与实现相对要更为复杂，对性能的要求也更高，另一方面。NIDS 所保护的对象是网络上的一群机器而非仅限于其运行的那一台主机，这是 NIDS 较之 HIDS 的优越性之一。

5.1 较简单的机制

这一节首先概要讨论几类简单的网络反入侵机制，目的是借此阐明一些有益的设计思想和概念。这些机制本身能力都有限，但在某些情况下也确实简单实用。

设计和实现 NIDS 要解决的首要问题，无疑就是如何可靠地识别出 IP 分组是否蕴涵入侵行为。不难想象，特定的入侵行为或过程在 TCP/IP 的层次上必有其特殊的特征，例如表 5-1 是对几类典型网络病毒在这方面的经验总结，从这里可以归纳出很多明显的特征。

例 5-1 一些网络病毒入侵时必须要创建或利用的端口

病 毒	被攻击或被利用的 TCP 或 UDP 端口
W32/CodRed	TCP 80
W32/Blaster	TCP 135, 4444, UDP 69
W32/Slammer	UDP 1434
W32/Sasser	TCP 445, 5554, 9996
W32/Dabber	TCP 5554,8967, 9898-9999
W32/Kogro	TCP 445, 113, 3067-3076, 6667
W32/Welchia	TCP 135, 80
W32/Welchia.D	TCP 80, 135, 445, 3127
Linux/Slapper	TCP 80, 443, 2002
W32/Witty	UDP 40000

除了以上方面，携带病毒代码的 IP 分组本身也具有明显的数据特征，例如 Linux/Slapper 病毒利用 Open SSL 的漏洞进行溢出攻击，在 HTTPS(TCP 443 端口)建立起加密信道之前就入侵了目标系统，并且以明文（而非密文）传输其数据，在其消息中含有以下典型的特征信

息：

rm –vf /tmp/.bugtraq.c; cat > /tmp.uub ugtraq<__eof__; .begin 655 .bugtraq.c

熟悉 Unix/Linux 的读者立刻能看出来以上序列在前面包含有两条带参数的 shell 命令。这一序列中独一无二(该病毒特有)的字节序列用 16 进制表达出来就是

36 35 35 20 2E 62 75 67 74 72 61 2E 63

再以病毒 W32/Slammer 为例，它在 UDP 1434 端口上的消息总包含以下特征字节序列，用 16 进制表示就是

68 2E 64 6C 68 65 6C 33 32 68 6B 65 72 6E

从以上这些事实读者不难归纳出检测网络病毒的一种方法：如果 IP 分组的数据部分被观测到携带以上字节序列，则可以判定该 IP 分组源自对应的病毒。读者可能会担心，毕竟普通用途的数据也有可能出现以上形态，那样一来不是会出现误警吗？但进一步思考就不难看出出现误警的概率很小。例如对病毒 W32/Slammer 的特征字节序列，一共有 14 个字节，从而一个数据序列包含与以上特征序列完全相同的子序列的概率只有 2^{-112}，这在许多情形下都是可以接受的概率水平。显然，病毒的特征字节序列越长，判断越可靠。

然而一个严重问题是：病毒代码总具有固定的特征字节序列吗？看下面这一小段 intel x86 汇编程序，你不妨猜猜它的功能：

```
        LEA SI, Start
        MOV SP, 0682 /*加密的病毒代码段的长度：682 字节*/
Decrypt:
        XOR [SI], SI
        XOR [SI], SP
        INC SI
        DEC SP
        JNZ Decrypt
Start: /*加密的病毒代码的起始地址，即 A0*/
        …….
```

实际上，以上代码对从地址 A_0 开始、长度为 682 字节的病毒程序进行解密，对应的加密算法是 $y[i]=x[i] \oplus A_i \oplus (N-i)$, N=682, i=0,1,2,…,N-1, x[i]是第 i 个明文字节、A_i 是 x[i]的地址、y[i] 是第 i 个密文字节，即从 Start 处开始的字节流。

实际上，当代的许多病毒都有利用加密技术进行自我隐形的能力，结果使得同源病毒也可以有很不相同的表观特征。如此一来，上面那种直接的检测方法就不能总是有效识别出这类病毒了。但另一方面，细致的分析表明大量病毒仍然具有特定的数据或指令特征，即独一无二的特征字节序列，例如病毒的加密和解密指令序列本身就能表现出某些特征，这种特征

对有些病毒能构成实施入侵检测的依据。

5-2 信息流控制：防火墙

防火墙这种网络设备的主要功能是对 IP 分组进行过滤和实施访问代理。从本质上讲，防火墙基于信息流(*information flow*)强制实施网络边界安全策略。图 5-1 是一个典型的在网络边界配置有防火墙保护的网络的例子，防火墙设备位于边界路由器外面(有时两个设备可以合二为一，具体取决于网络设备)，按照网络管理者的策略，源自外部的任何 IP 分组不允许访问 SQL Server 数据库服务器，而源自内部的 IP 分组可以向外访问 TCP 80 号端口。

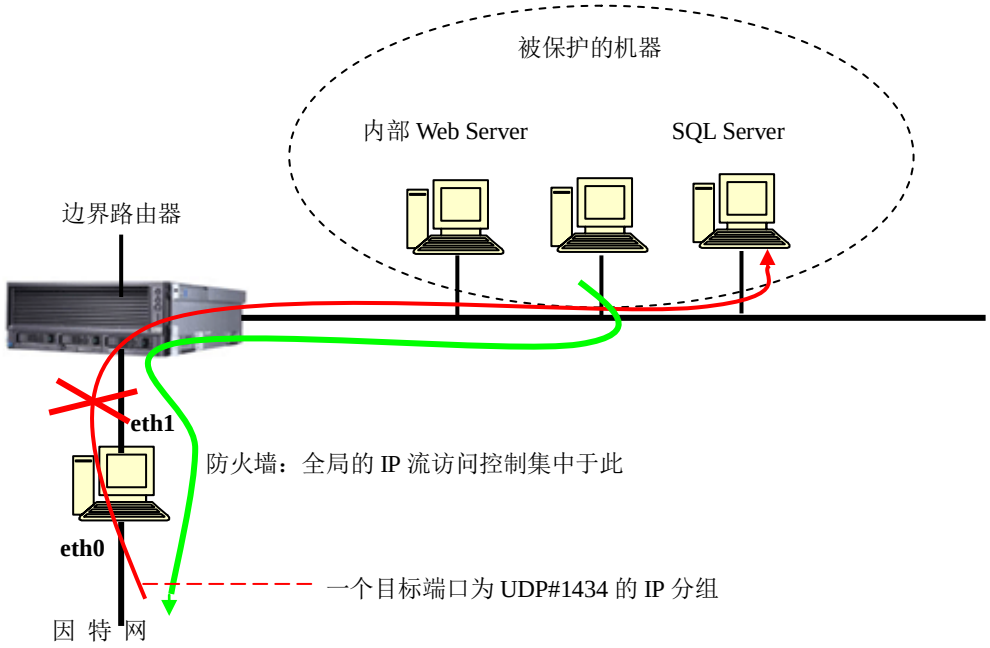


图 5-1 一个企业网及其边界防火墙

防火墙实施 IP 分组过滤的方式是依照所谓访问控制列表(Access Control List，简称 ACL)。防火墙的每个接口都可以配置至少两个 ACL(有些防火墙如 Linux iptables 可以配置三类，见下面的例子)，分别作用于进入和离开这一接口的 IP 分组。ACL 是一组表项的有序集合，每个表项表达一个规则，称为 IP 分组过滤策略，内容包含被监测的 IP 分组的特征、对该类型的 IP 分组所实施的动作(放行、废弃等)。每当一个 IP 分组经过防火墙的接口，防火墙都要顺序检索对应的 ACL，找到特征匹配的表项，然后根据该表项指示的动作处理该 IP 分组。在形式上，不同的防火墙设备对 ACL 有其特定的表达语法。我们不深入解释任何特定的 ACL 语法，仅在此给出一些例子(同时对 ACL 的表达形式做简要说明)，以使读者具

体理解防火墙设备的功能与能力。

首先参照 Linux iptables 防火墙命令语法解释如何用防火墙的 ACL 表达网络安全策略。Linux iptables 是一种运行于 Linux 上的软件防火墙系统。对图 7-1 的网络，要实施的策略之一是“阻塞从接口 eth0 到达的、拟去到 UDP 1434 端口的任何消息”，ACL 的相应表项的形式是：

iptables –A INPUT –i eth0 –p udp –m multiport −−destination-port 1434 –j DROP

性 质 IP 分组的匹配特征 处理动作

以上表达式中 iptabes, NPUT, DROP 等是关键字，-A, -i, -p, -m, -destination, -j 是表示参数类型的选项符号，其他都是参数值。-A 的作用是将该表项加入到 ACL 的末尾，这些 ACL 分成三类：针对到达本接口的 IP 分组的 ACL(名字约定为 INPUT)、针对离开本接口的 IP 分组的 ACL(名字为 OUTPUT)、针对通过该接口后转发到其它接口的 IP 分组的 ACL(名字为 FORWARD)；−i 和−o 后面紧跟的参数(在语法上隔一个空格，下同)分别指定本防火墙上的物理接口的名字；−p 后面紧跟的参数指定 IP 分组中的协议标识号；−j 后面紧跟的参数指定对模式匹配的 IP 分组的处理动作，包括 ACCEPT(放行)、DROP (丢弃且不向发送方报告)、REJECT (丢弃且向向发送方报告)。 据此,读者就不难理解上面这条 ACL 表项的含义了。

再举几个 Linux iptables 防火墙的 ACL 的例子。策略“禁止向目标端口 TCP#2049、TCP#1080 和 TCP#3128 建立连接的消息从接口 eth1 离出”用以下 ACL 表项表达：

iptables –A OUTPUT –o eth1 –p tcp –m multiport −−destination-port 2049,1080,3128 −−syn –j REJECT

其中-p tcp 指定该表项所针对的是携带 TCP 协议的 IP 分组，—syn 进一步指名 TCP 段首部的 SYN 位为 1，这显然是指用以请求建立 TCP 连接的消息。

策略“允许源自 TCP 端口 80 和 443 的、请求建立 TCP 连接的消息通过” 用以下 ACL 表项表达：

iptables –A INPUT –i eth0 –p tcp –m multiport −−source-port 80,443 −−syn –j ACCEPT

虽然以上是以 iptabes 防火墙为例，但防火墙的配置有普遍的原则：

a) 如果防火墙的默认行为是阻塞一切 IP 分组，则仅当显式指定过滤规则时，防火墙按照指定的过滤规则对满足特征匹配的 IP 分组实施规定的处理动作；所有不匹配的 IP 分组一概默认为实施阻塞；

类似地，如果防火墙的默认行为是允许一切 IP 分组，则仅当显式指定过滤规则时，防火墙按照指定的过滤规则对满足特征匹配的 IP 分组实施规定的处理动作；所有不匹配的 IP 分组一概默认为允许通过；

b) 防火墙对每个接口、每个方向(到达和离出)单独建立一组过滤规则，单独实施匹配和处理，因此每个接口一般具有分别针对两个传输方向的两组过滤规则；

c) 每组过滤规则是有特定顺序的规则表，对 IP 分组，防火墙永远(按照排列顺序)实施匹配的第一个规则；最后一条规则是本表的默认规则。

需要指出的是，除了作为专用设备的防火墙系统，当地许多中高档路由器也都具有较强的分组过滤功能，在实现分组转发的同时进行过滤。事实上一般复杂程度的安全策略都可以通过直接配置路由器的 ACL 来实现而不必借助于专用防火墙。以下是 Cisco 路由器上的过滤规则的例子，配置原则与上例相同但语法形式不同，这里语法结构为

access-list <规则号> <动作> <协议> <源 IP 地址> <[源端口号]> <目标 IP 地址> <[目标端口号]>

其中 access-list 是关键字，动作取值 permit 或 deny，源端口号和目标端口号是含等于、大于或小于关系的算术表达式。

以下表项允许来自任何源 IP 地址和任何源端口的、目标 IP 地址为 160.10.1.1 且目标端口号为 80 的 TCP 消息通过：

access-list 101 permit tcp any 160.10.1.1 eq 80

以下规则允许来自任何源 IP 地址、源端口号为 20、目标网络地址为 160.11.0.0/16 且目标端口号在 1023 以上的 TCP 消息通过：

access-list 101 permit tcp any eq 20 160.11.0.0 0.0.255.255 gt 1023

(这里 0.0.255.255 表示 16 位子网掩码，与通常的表示方式(255.255.0.0)刚好相反，是 Cisco 设备的约定)。

最后举一个更复杂的配置实例：

```
interface serial 0 /*指定接口 serial 0*/
    ip access-group 101 in /*所有标识号为 101 的规则项均用以过滤到达接口 serial 0 的 IP 分组*/
access-list 101 permit tcp any 160.10.1.188 eq 80
access-list 101 permit tcp any 160.10.2.118 eq 21
access-list 101 permit tcp any 160.10.2.118 eq 20
access-list 101 permit tcp any eq 20 160.30.0.0 0.0.255.255 gt 1023
```

防火墙在网络边界位置通过对 IP 分组的过滤实施对 IP 分组的访问限定。对更复杂的企业网络，即使企业网内部的不同网段也可能具有不同的安全访问策略，例如财务部门可能就需要特别限定 IP 分组的源和目标，因此这类网络具有更复杂的安全策略，这时需要实施多防火墙配置，图 5-2 是一个例子。表 5-2 概要总结当代各种防火墙系统的类型。

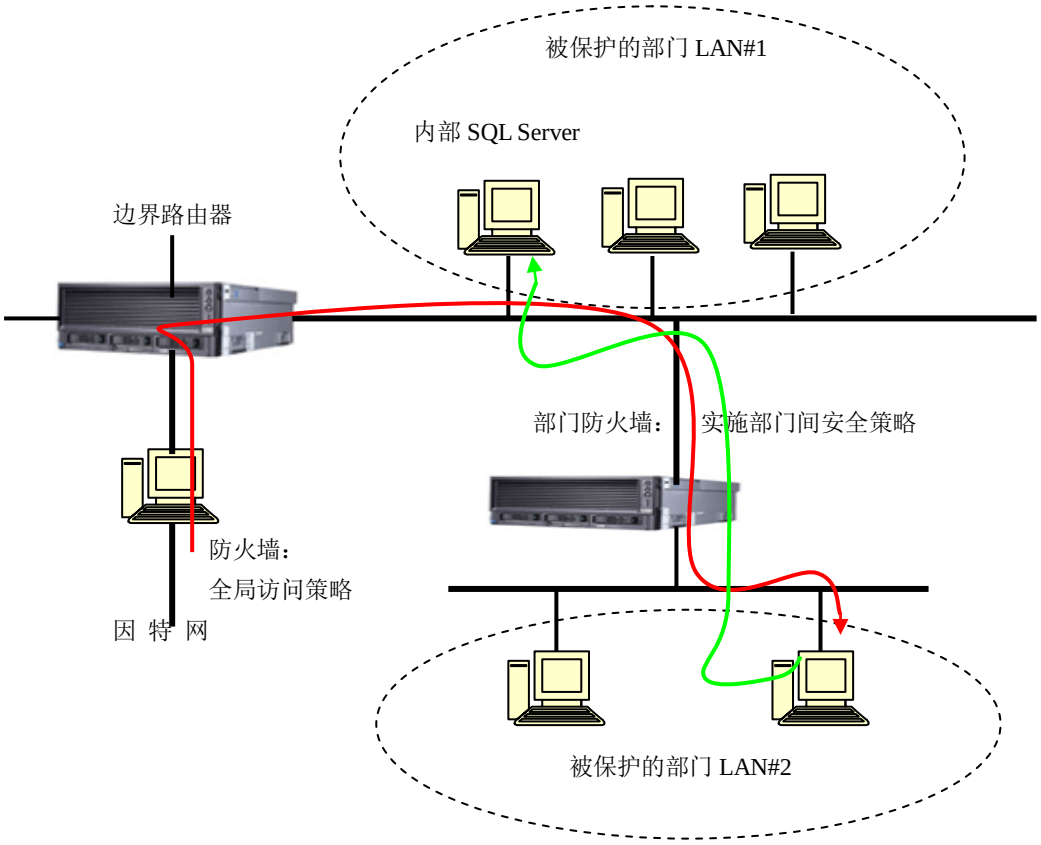


图 5-2 多防火墙配置

表 5-2 防火墙的类型

类 型	特 点
硬件防火墙	基于专用硬件平台和专用操作系统实现； 能较有效地抵抗一般性的攻击手段； 典型产品：Cisco PIX
带防火墙功能的安全路由器/交换机	集成分组转发与过滤功能；
纯软件防火墙	多为专用硬件平台（例如基于网络处理器）、专用或通用操作系统（前者如 Cisco IOS, pSOS+）、后者如嵌入式 Linux 、FreeBSD。 运行于通用操作系统平台，典型如 Linux iptables, CheckPoint 等。
个人防火墙	仅针对本机进行消息流的访问控制，例如 Windows 2000/XP 的内置产品。

5.3* 防火墙的实现

这一节概要介绍防火墙系统的实现，其中某些方面与 NIDS 有共同之处。完整的防火

墙系统由软件+系统软件(操作系统)+硬件平台构成，而且软件在防火墙系统中占很大成分，至少包括分组的高速过滤引擎和策略管理系统两个重要部分。过滤引擎进行 IP 分组的匹配、过滤处理，要求高效、可靠，为追求高性能可能直接实现为防火墙操作系统的内核组件；策略管理系统对过滤规则进行配置管理，一般实现为用户模态的应用程序，为使用便捷多数还配有直观的图形用户界面。图 5-3 是完整防火墙系统的功能架构。

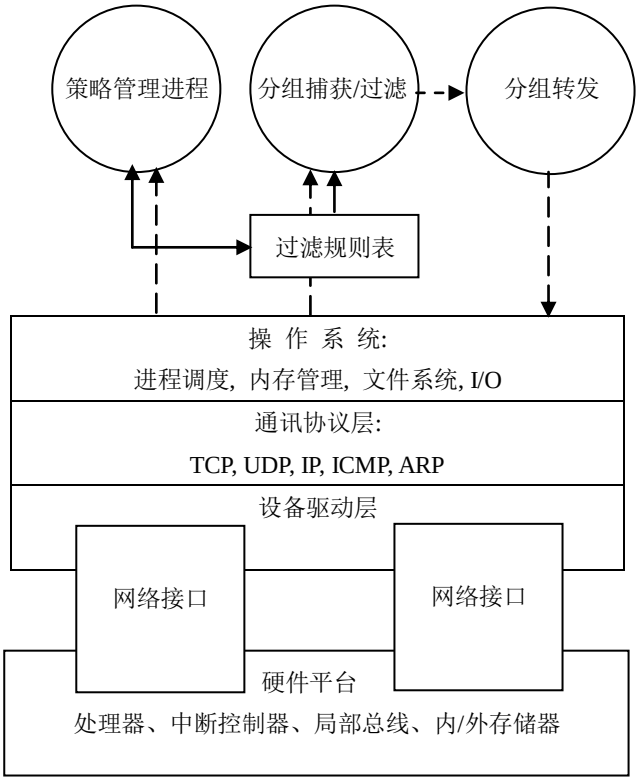


图 5-3 防火墙设备的逻辑结构

IP 分组的过滤和分类是这类系统最重要、最基本的功能，上一节已经看到一个(带分组过滤功能的)路由器或防火墙常需要限定到达分组的源 IP 地址、目标 IP 地址或目标 TCP 端口号的范围，不在限定范围内的 IP 分组将被废弃；一个入侵检测系统需要监测所有 IP 分组的各种特征并分析其上下文，以识别这些 IP 分组所构成的消息的涵义，等等。

所谓分类，是指按照某种规则将 IP 分组归结为逻辑范畴—流(flow)，有共同特征的分组归结为同一个流，然后进一步做基于流的处理。分类是许多应用的基本功能，除了基于流的安全访问控制之外，重要的应用还包括基于流的负载均衡等(现代防火墙产品经常也需要实现这类功能)。图 5-4 是一个例子，其中网关 G 是具有负载均衡能力的防火墙系统，作为服务器集群的网络前端。所有服务器对外共用一个 IP 地址(例如高吞吐量的 Web 集群)。假如

服务器事务是基于 TCP 的(例如 Web)，网关 G 可以将每个 TCP 连接对应一个流，针对每个流实施安全策略同时在所有服务器上均匀分配当前存在的流。

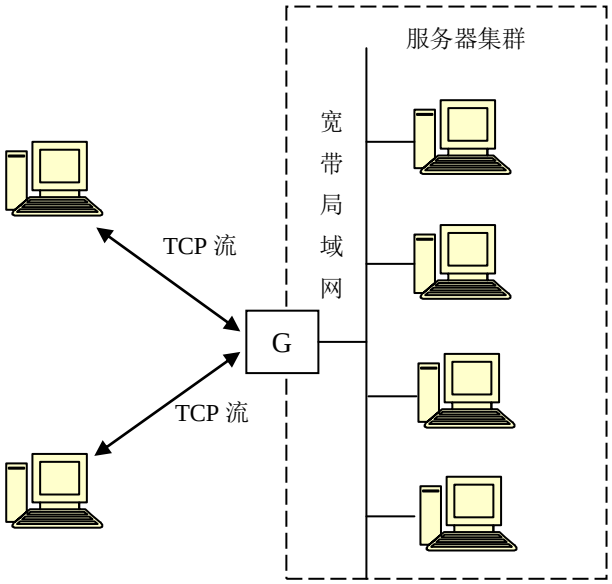


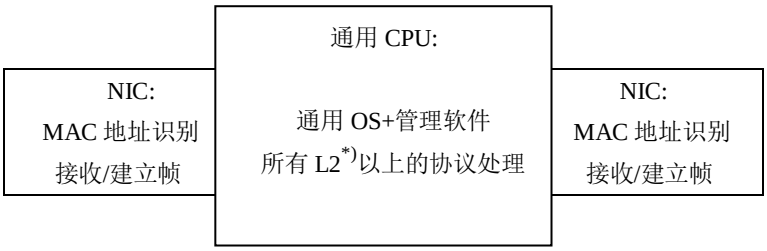
图 5-4 保护服务器集群的防火墙系统

分类是一个计算量较大的操作，特别是软件实现，虽然非常灵活，但因为这种分类计算需要大量的条件判定-程序分支，使软件分类的计算速度不高，成为实现高性能网络系统的瓶颈之一。如何综合软/硬件功能，优化分类处理，是实现高性能网络系统的关键技术之一。基于目前通用计算平台和通用操作系统上的防火墙方案本质上只适合于中低性能的网络系统，例如最高分组转发率在 50kpps 左右的路由器。对于工作站上的个人防火墙系统，这类方案都可以达到性能要求，因为主机的主要任务是计算而非通讯处理，即使是服务器也只有为数不多的网络接口。然而，对于专用的高性能网络系统，情况完全不同。以高性能路由器为例，如果要求该路由器有能力联结 10 个 1Gbps-以太网链路，假设平均每个分组含 100 字节(800 位)，则路由器所要处理的总吞吐量大约为 10Gbps/800b=12.5M 分组/秒。目前较好的 TCP/IP 的软件实现在处理一个分组时大约需要 5000 条左右的指令，这样就要求该路由器的处理器能力要达到约 12.5M 分组/秒×5000 指令/分组=62.5GIPS[1]！这一速度远远超越了当代任何通用处理器的现实能力。

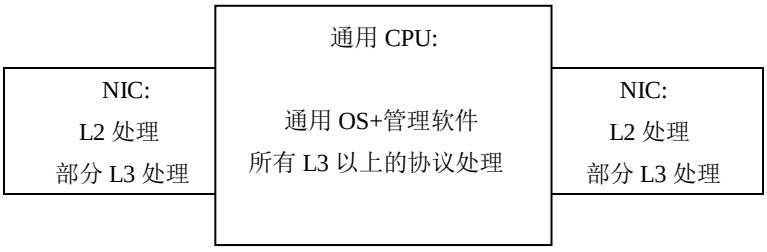
这个粗略的数量级估计表明，不能仅仅依靠提高处理器的速度来达到防火墙系统的高

[1] 读者在计算机组成原理课程中学习过处理器的性能度量之一是所谓每秒指令数，单位一般为 MIPS，即 Mega Instructions Per-Second。这里 GIPS 即是 Gega Instructions Per-Second 的缩写，即每秒能执行多少千兆(10^9!)条指令。

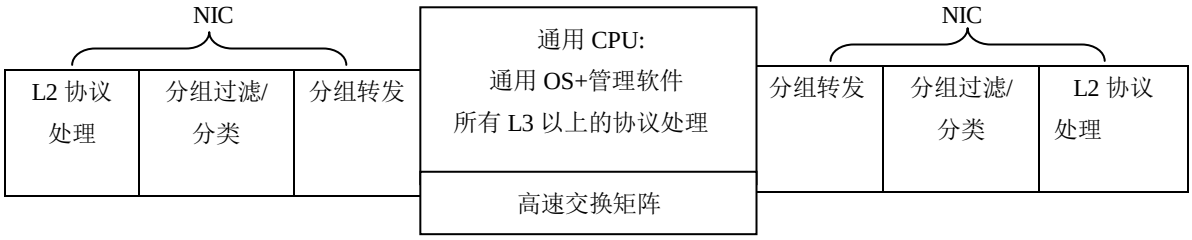
性能。和高性能计算领域的思路类似，巧妙的体系结构设计在这里成为解决问题的关键性因素，在硬件方面，这包括多处理器系统、专用处理器、专用协处理器系统、带板载智能处理功能的 NIC、信元交换、数据流水线结构等等措施。同时，高性能网络系统的硬件体系结构与软件体系结构是相互影响的，两方协同才有可能构造出真正满足高性能、同时成本合理的系统。图 5-5 是高性能防火墙系统典型的逻辑体系结构。



*) L_N 意指 OSI 参考模型的第 N 层，下同. a)



b)



c)

图 5-5 防火墙系统体系结构的几种方案

图 5-5(a)是低性能系统的构成方案，它以通用 CPU 及通用操作系统完成一切协议处理功能，因此性能不高，特别是系统的聚合速率不可能达到很高水平，但它突出的优点是开发简单、成本低廉。图 5-5(b)是低性能系统的构成方案的演化，将数据链路层的处理完全由通用 CPU 转移到网络接口上，并且尽可能地将一部分网络层功能，例如校验计算，也转移到输入/输出接口上，这样不仅使通用 CPU 避免了一部分它所不擅长的计算，而且提高了网络协议处理的并行程度，直接效果是加快了处理速度。图 5-5(c)是沿着方案(b)的继续改进，将越

来越多的、通用 CPU 不适合的针对网络协议处理功能转移到输入/输出接口上。由于接口要完成的功能变得更为复杂，因此在方案(c)中，每个接口实际上已演化为带嵌入式处理器的可编程子系统，嵌入式处理器具有优化的 RISC 指令集，接口程序也专门针对其要完成的第二层到第四层的功能进行优化。实际构造接口子系统时可以采用通用嵌入式处理器，因此相对其性能而言，系统的总体成本并不十分高昂。

在防火墙软件方面，除了实施基本的上下文无关检测功能(例如上一节的 ACL 所表达的那类功能)，高级的软件也能实现上下文有关的检测。这时防火强不是仅针对每个孤立的 IP 分组单独识别、过滤和处理，而是将这些 IP 分组关联起来，放在特定的协议上下文中识别和处理，以发现更复杂的行为。这也是网络入侵检测系统具有的重要功能之一。例如，对 TCP 连接的状态跟踪算法就是这类技术，这一算法为识别出的每个 TCP 连接保持其状态以支持复杂的协议分析，梗概如下：

```
TCP_connection_tracking()
{
    /*C 是当前活跃连接的表, 表项为<源 IP 地址,目标 IP 地址, 源 TCP 端口号,
      目标 TCP 端口号, 状态, …>*/
    C = 空;
    while(1){
        P = 下一个承载 TCP 段的 IP 分组;
        以<P 的源 IP 地址, P 的目标 IP 地址, P 的源 TCP 端口号, P 的目标 TCP 端口号>
          为关键字检索 C 中的表项, 若不存在则创建一个新表项;
        if(TCP 首部的 RST 位为 1) { /*发生 TCP 异常*/
            删除该表项;
        } else {
            if(TCP 首部的 FIN 位为 1) {
                标记对应方向上的 TCP 连接状态为“关闭”;
                if(表项中另一方向上的 TCP 连接状态为“关闭”) {
                    从 C 中删除该表项;
                } else {
                    if(TCP 首部的 SYN 位为 1) {
                        标记对应方向上的 TCP 连接状态为“建立”;
                        if(表项中另一方向上的 TCP 连接状态为“建立”) {
                            将该表项的状态置为“连接完成”;
                        }
                    }
                }
            }
        }
    }
```

}

5-4 网络入侵检测系统(NIDS)

这一节阐述 NIDS 的工作机理并以两个典型的 NIDS 作为实例具体剖析。虽然 NIDS 在概念上适用于任何网络和网络协议，但以下阐述均依照因特网及 TCP/IP 协议族进行。

NIDS 的目的是从网络上的 TCP/IP 消息流中识别出潜在的攻击行为。在识别入侵的具体方法上，基于误用类型的(*misuse-based*) NIDS 依照已知的各种网络入侵在协议消息方面所表现出来的、可靠的消息特征来判定是否存在入侵，而基于反常类型的(*anomaly-based*)NIDS 依据所观测到的消息流是否偏离网络协议本身的正常行为规范来判定是否存在入侵。

5.4.1 NIDS 实例: Snort

Snort 是一个所谓轻量级 NIDS(“轻量级”意味着使用方便、配置简捷、主要针对中小规模网络)，实时记录网络消息流并进行协议分析，对内容进行搜索/匹配，能够检测各种不同的攻击方式，对攻击进行实时报警。此外，Snort 具有很好的扩展性和可移植性。事实上，目前许多 NIDS 商品软件都是在 Snort 检测引擎的基础上进一步开发而来，此外多数协议分析工具也具有与 Snort 的接口以提取 Snort 的记录进行分析。本节依照较新的 Snort 2.0 版本进行阐述。

Snort 由四部分组成：分组捕获器、分组的分类/预处理器、检测引擎和日志/报警输出模块，系统体系结构如图 5-6 所示。

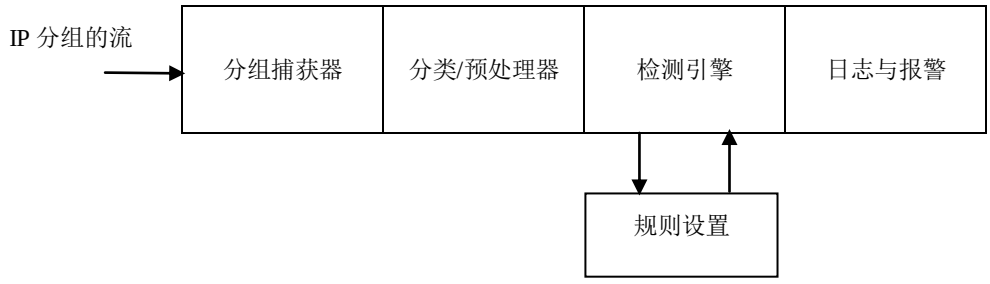


图 5-6 Snort 的系统结构

预处理器、检测引擎和日志/报警模块都是插件结构，插件程序按照 Snort 提供的插件接口实现，使用时动态加载，这样一来不必修改 Snort 的核心代码就可以扩展功能，例如增加针对新型网络协议的入侵检测机制，既保证了插件程序和 Snort 核心代码一致，又保证核

心代码的稳定可靠。

Snort 具有实时的日志记录和协议消息流分析能力，能够快速到检测网络攻击并报警。Snort 的报警机制很丰富，例如直接通过 syslog[2]、通过向用户指定的文件写入、通过 UNIX 套接字发送报警消息，甚至使用 SAMBA 协议[3]向 Windows 客户程序发出 WinPopup 消息。利用 XML 插件，Snort 还可以使用简单网络标记语言把日志存储到一个 SNML 文件。

Snort 能够进行在线协议分析，包括协议消息内容的搜索与匹配。有能力分析的协议包括 TCP、UDP、ICMP，还将包括 ARP、ICRP、OSPF、RIP 等协议。Snort 能够检测多种方式的攻击和入侵探测，例如缓冲区溢出攻击、秘密端口扫描、CGI 攻击、SMB 探测、操作系统指纹特征探测等。

Snort 的日志格式丰富，除了包括最经典的 tcpdump 格式和自定义 ASCII 字符形式，若使用数据库输出插件，Snort 可以把日志记入特定类型的数据库，例如:Postgresql、MySQL、支持 ODBC 的关系数据库甚至 Oracle 这类大型企业数据库。

Snort 不仅对 IP 分组的数据负载部分进行病毒模式匹配，而且对 TCP 段进行重组以强制检测完整的 TCP 流(关于这方面更详细的讨论参见下一节)。Snort 能够对端口扫描事件进行有效检测。所有这些都基于软件插件实现，非常灵活。在检测到入侵事件后，Snort 能够主动断开恶意的 TCP 连接。

Snort 的核心功能之一是对 IP 分组高速过滤，过滤规则由用户配置，为此 Snort 使用一种简单易用的规则描述语言。典型的 Snort 过滤规则包含四个要素：处理动作、被过滤的协议、消息的方向、发生过滤的端口。一个典型的过滤规则如 log tcp any any->10.1.1.0/24 79。通过组合使用过滤规则可以实现复杂的检测功能。因为这种语言简单、易用，所以当新的网络攻击被发现确认后能很快根据入侵特征码写出相应的过滤/检测规则。

以上是 Snort 的主要特点。在工作程序上，Snort 在捕获 IP 分组后先提交分类/预处理插件处理，然后被捕获的分组将经过检测引擎中对应的规则链，如果检测到有匹配规则链所表达的病毒模式的情况，Snort 就根据输出设置把该信息记录到对应的文件并报警。

5.4.2 NIDS 实例：Bro

较之轻量级的 Snort，同样是开放源代码的 Bro 要复杂得多，但功能也更完善、性能更高。Bro 的主要设计目标包括以下诸方面：

[2] 这是 Unix/Linux 系统上最常用的日志功能调用。
[3] 这是 Windows 系统的网络文件系统的协议，类似于 Unix/Linux 网络文件系统协议 NFS。

大吞吐量与高速率，即着眼于针对大型网络的消息流进行高速检测。前一节提到过，当代最新的有线宽带网络的传输速率已开始超过典型 CPU 的处理速率，因此 Bro 的设计要特别避免因 NIDS 处理速度滞后于网络分组传输速率而导致的分组丢失。换句话说，即使在极其重载的情况下，Bro 的设计也要保证正确分析和检测入侵事件[4]。

实时检测与在线报警。许多 NIDS 本质上仅是一个消息日志系统，特别是在重载情况下只能在线识别和处理简单的事件，而把复杂的入侵检测推给专用的离线日志分析系统。这类方案的优点是能深入、详细地分析入侵事件，但其缺点也不言而喻。与多数 NIDS 不同，Bro 的设计目标明确追求实时分析大吞吐量的协议消息和在线检测入侵事件。

灵活的可扩展能力。这意味着无论是针对新入侵类型、还是针对新型网络协议，对 Bro 追加相应的新功能都能在不改变现有核心软件的前提下以最小的复杂性完成。

具有抵抗攻击的能力。Bro 的设计明确假定 NIDS 本身有被攻击的可能，因此其设计内容之一就是使 Bro 本身能够最大程度地抵抗各类针对 NIDS 的典型攻击。这里要指出一点：Bro 假定针对 Bro 的攻击总是仅仅来自于一侧(Bro 运行于边界网关，一侧是被保护的内部网络、另一侧是不可信任的外部网络)，这是一个合理而现实的假设。实际上，当 NIDS 两侧均不可信任且合谋攻击时，NIDS 几乎不可能进行检测和抵御，但对一个可靠的 NIDS，这是实际很少发生的情形。Bro 的系统结构如图 5-7，其中虚线表示控制流、实线表示数据流。

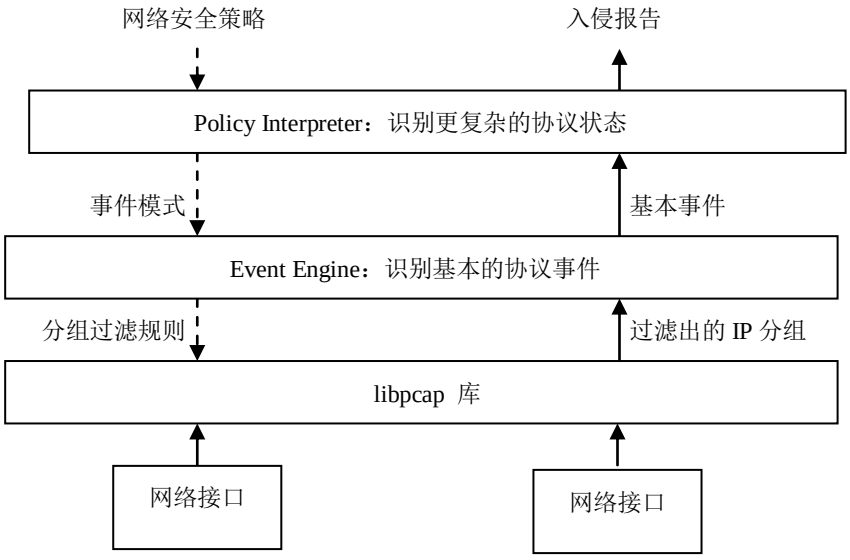


图 5-7 Bro 的系统结构

除了最基层的 IP 分组捕获模块，Bro 最重要的两个模块是事件引擎(Event Engine)和策

[4] 考虑到网络重载本身很可能源自于 DoS 攻击, Bro 的这一设计目标非常具有现实意义。

略解释器(Policy Interpreter)。事件引擎对 IP 分组捕获模块所过滤出的 IP 分组首先进行完整性检查，确认是无缺陷的 IP 分组(例如 IP 首部校验和正确)、对经过切割的 IP 分组进行重组，识别并建立其携带的上层协议的当前状态。Bro 的事件引擎以各种基本事件的形式向上层的策略解释器输出，每个事件表示基本协议的特定状态或状态的变化。由此从解释器的观点看，网络上的消息流被抽象成一个事件序列，这些事件的涵义较之原始的 IP 分组或基层协议消息单元更能表达实施安全策略所需要的高级语义，而策略解释器的功能实质上就是从所观测到的这些事件中进一步提取(可能需要复杂的上下文关联的)入侵特征。

事件引擎对每种协议所生成的事件序列可以看做对该协议状态机模型的一种匹配。例如，对携带 TCP 段的 IP 分组，事件引擎利用类似第 5.3 节的 TCP 状态跟踪算法在线拟合图 1-9 中的 TCP 状态机模型。每当捕获一个含 SYN=1 的 TCP 段的 IP 分组，事件引擎根据该分组的源/宿 IP 地址和段的源/宿端口号建立一个特定连接的状态机事件区域、启动一个定时器(默认值 5 分钟)，若定时器超时后对方仍未响应连接请求则生成事件 connection_attempt，若在定时器超时之前对方正确响应连接请求则生成事件 connection_established，若对方响应的段内 RST=1 则生成事件 connection_rejected，所有这些事件都带有表达 TCP 连接属性的参数。另一个例子是当观测到 FIN=1 的 TCP 段时生成事件 connection_finished。事件引擎对非连接式协议，例如 UDP，也建立类似的事件模型，例如每当所捕获到 IP 分组的源/宿 IP 地址及源/宿 UDP 端口号都与以前某个 IP 分组的对应属性反转，则生成事件 udp_reply；若没有对应关系则建立一个新的事件区域、记录属性参数并生成事件 udp_request。通过这中方式，事件引擎从单纯的、看似孤立的 UDP 消息流提取出了会话(session)。

在策略解释器中，每个事件对应一个特定的、事先配置好的事件处理例程(event-handler)。事件处理例程的重要功能是将多事件的上下文关联起来以推断这些事件所蕴涵的协议行为是否符合该协议的规范，或从中检测出潜在的入侵特征。每个事件处理例程的输出可能更新协议机模型的内部状态、生成实时入侵报告或引发新的事件。

Bro 的安全策略是一个类似于 C++风格的语言所表达的程序脚本，Bro 的策略解释器同时执行多个这样的脚本。一般说来网络中每个被允许的应用层协议都被指派一个安全策略，例如 FTP、portmapper、Telnet、Rlogin、HTTP 等。安全策略的程序脚本包括一组事件、参数和对应的处理例程，处理例程定义对事件的处理代码。例如下面是 rlogin 安全策略中的一行语句，定义事件 rlogin_request 的处理例程，其中参数包括 c(TCP 连接)和 user (用户名)等：

```
event rlogin_request(c: connection, user: string, pwd: string )
```

类似地，读者不难理解 rlogin 安全策略中以下这行语句的作用及其参数的涵义：

event rlogin_reply(c: connection, line: string)

Bro 的策略解释器根据安全策略判定所观测到的协议会话是否符合该协议的规范，并适时加以安全控制。以 FTP 协议为例，从事件引擎生成事件 ftp_request 开始，策略解释器记录该事件的参数，包括相关的 TCP 连接的属性，还对 FTP 数据进一步解析，提取出这一 FTP 会话当前请求的文件名、用户名等参数，然后等待同一 TCP 连接上的事件 ftp_reply，当这一事件发生时进一步解析 FTP 数据以提取出 FTP 服务器的状态码、数据传输端口[5]等参数。策略解释器有能力根据安全脚本识别出非法的 FTP 数据连接，特别是因此而能抑制 FTP 反射攻击，在这种攻击中，攻击者欺骗 FTP 服务器与第三方(而非攻击者)主机建立数据连接而使 FTP 服务器向第三方主机传输文件，以此扩散病毒或实现隐密的网络探测。Bro 通过严格检测、匹配每个 FTP 会话的命令参数并区分不同的 FTP 会话而切断了这一攻击途径。

5.4.3 对 NIDS 的典型欺骗及抵御方法

这一小节首先讨论对 NIDS 的一些欺骗技术，目的是使读者认识到 NIDS 的设计需要保证其自身抵抗各种攻击，开阔读者的思路，然后概要讨论抵御方法。

NIDS 的目的在于识别出反常的协议行为，因此从入侵者的观点看，如何使入侵过程在消息流的层次上表现得正常，或最大程度地掩盖其反常特征，将是一条可取的途径。在这里给出几个典型而有趣的例子。

第一个例子(图 5-8)是入侵者故意切割 IP 分组，使得每个 IP 分组中的消息载荷都不包含完整的病毒特征，以此欺骗那些基于特征匹配方式检测入侵的 NIDS。注意这一方法不影响病毒本身的完整性，因此最终的目标计算机总能将被切割的 IP 分组正确重组，从而病毒程序总能够完整地在目标计算机上被重构。

第二个例子(图 5-9)是入侵者发送“真”、“假” IP 分组，所有这些 IP 分组都经过 NIDS，但只有其中所谓的“真” IP 分组真正到达目标计算机，而那些“假”IP 分组并不会实际到达目标计算机。这里的问题在于，NIDS 从包含“真”“假”IP 分组的消息流中所“看到”的信息与推断出来的功能性涵义，与那些实际到达目标计算机的“真”IP 分组所具有的功能性涵义完全不同，前者表现出良性的语义，而后者的含义却是恶意的，具有入侵的目的。入侵者炮制“真”“假”IP 分组的方法可以很简单，例如利用 IP 分组首部中的 TTL 域，只要使“假”IP 分组的 TTL 域初值大到足以达到 NIDS 所在的网关、但同时又小到不足以到达目标计算机就可

[5] 一个 FTP 会话具有两个 TCP 连接，一个用于传输命令和控制信息，另一个用于传输文件本身，即数据。

以达到目的，而 “真”IP 分组的 TTL 域初值则总是使其大到足以达到目标计算机。

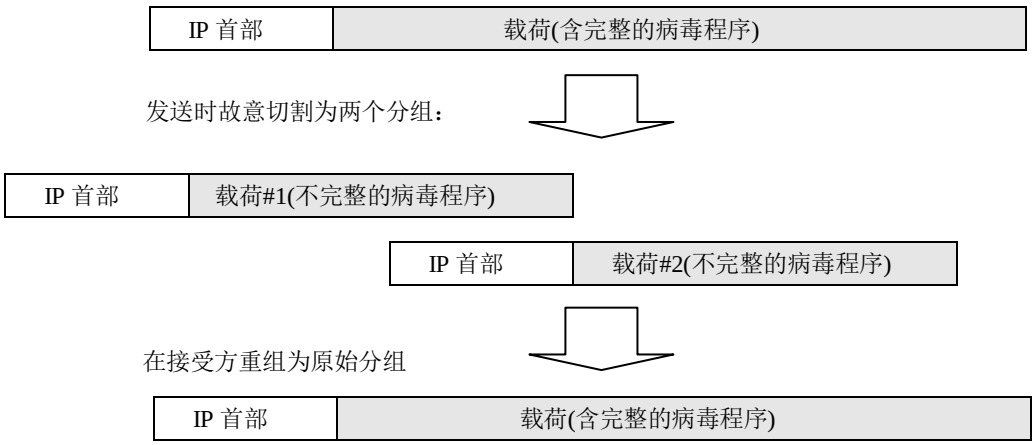


图 5-8 通过故意切割 IP 分组来隐藏载核中的病毒特征

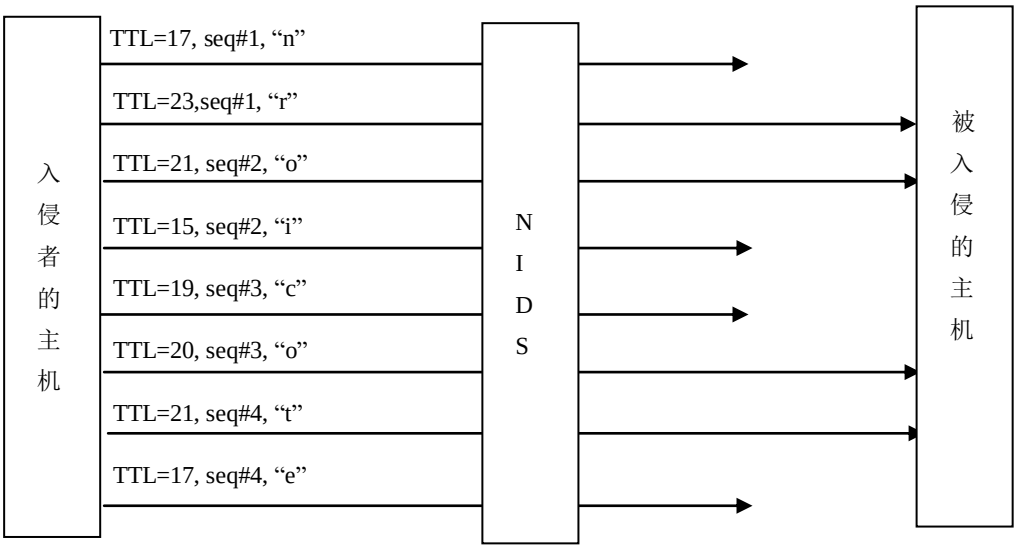


图 5-9 包含“真”“假”IP 分组的消息流

在图 5-9 的例子中，入侵者生成的 IP 分组都携带 TCP 协议载荷，这些 TCP 段属于同一个 TCP 会话。假如图中所表达的这部分载荷的含义是用户名字，seq#n 表示 TCP 段首中的序列号域值为 n，则所有“真”“假”消息载荷所表达的用户名(从 NIDS 的观点看)是“nroicote”，一个普通用户名而已；但实际上，从图中不难看出那些真正到达目标机器的 IP 分组所携带的用户名(从目标机器的观点看)却是 “root”，一个具有最高权限的特权用户！消息流中出现“root”是一个敏感的安全事件，NIDS 本来需要进行更多的处理以检测这一会话是否合法，然而“真”“假”IP 分组的故意混杂却成功地掩盖了这一事件。

第三个例子是隐形端口扫描(图 5-10)。端口扫描是入侵准备阶段最重要的步骤之一，入侵者以此判定哪些机器上的哪些 TCP 或 UDP 端口当前处于工作状态，也就是准备入侵的对象进程是否存在。直接的端口扫描很容易被检测到，因此当前入侵者已经开发各种隐形扫描技术，这些技术使扫描消息看起来几乎如同正常的协议消息，同时不暴露扫描者的真实 IP 地址。图 5-10 就是这类技术之一。

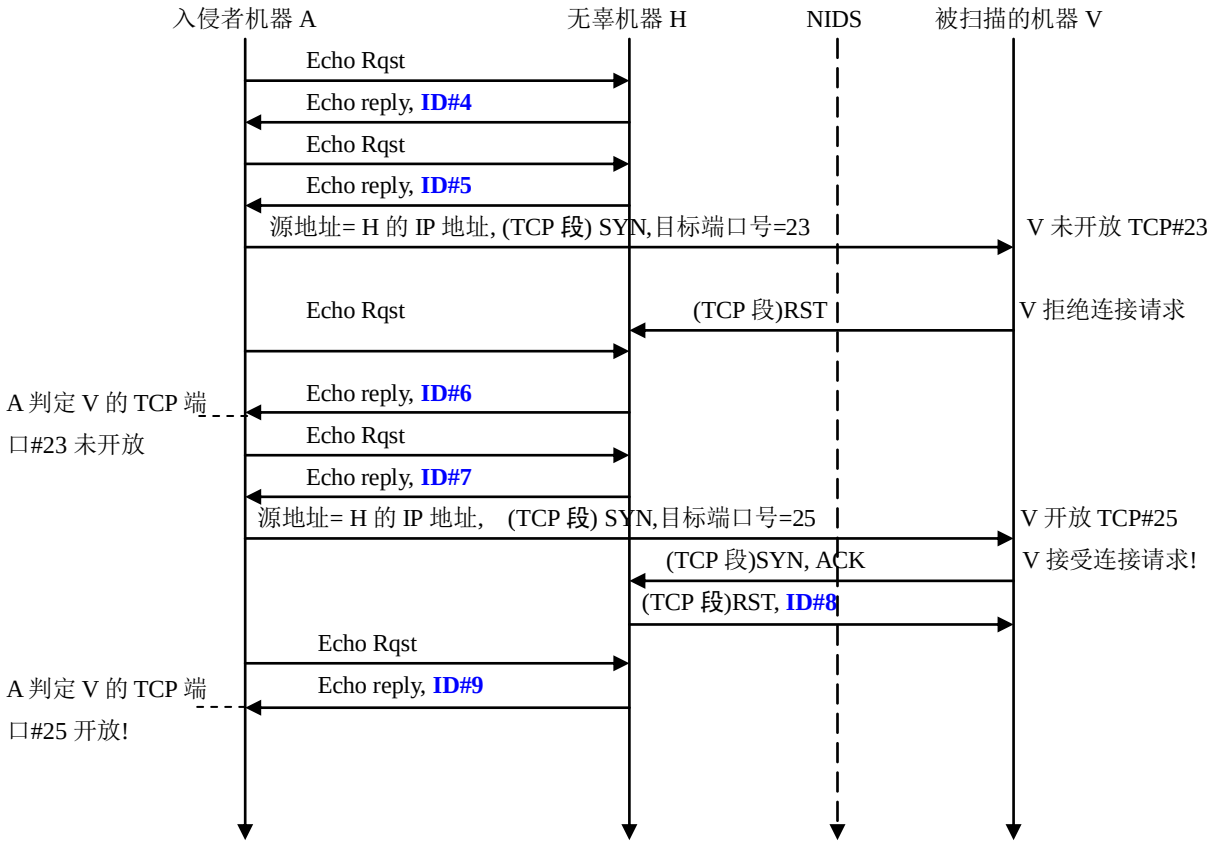


图 5-10 隐形端口扫描

图 5-10 是对 TCP 端口的隐形扫描，综合利用了 IP 协议、ICMP 协议和 TCP 协议的特点。完整的扫描过程涉及入侵者的机器 A、被扫描的机器 V 和某个无辜机器 H，这些机器上的进程用垂直的实线代表，垂直的虚线表示 NIDS。注意机器 H 位于 NIDS 所保护的网络区域之外，被扫描的机器 V 位于 NIDS 所保护的网络区域之内。Echo Rqst 和 Echo reply 分别是 ICMP 协议的两个消息，分别用以探测指定 IP 地址的目标机器和对此给出响应(常用的网络命令 ping 就是以此实现)。ID 指图中 H 生成的 IP 分组的标识号。回顾图 1-4 中的分组标识号字段，当发送方生成 IP 分组时，每个具有特定源/宿 IP 地址的 IP 分组都有一个唯一的分组标识号，这一标识号的目的是为了正确重组被切割的 IP 分组。不仅如此，很多系统

在实现 IP 协议时，(在始发方) IP 分组的标识号是被连续生成的，图 5-10 中的无辜机器 H 就是如此。正是这一点被入侵者巧妙利用以实现隐形扫描。

入侵者 A 首先连续地想 H 发送 ICMP Echo Rqst 消息，通过观测 H 所响应的 ICMP Echo reply 消息中的 IP 分组标识号了解其增长规律，然后向真正准备入侵的目标机器 V 的 TCP 23 号端口发送 TCP 连接请求，但承载该 TCP 段的 IP 分组的源地址不是入侵而是无辜机器 H 的 IP 地址。假如 V 上的 TCP 23 号端口未开放，这时 V(按照 TCP 协议规范)向 H 响应一个 RST 位为 1 的 TCP 段，用以通告这一错误，而 H 将不对此做任何进一步处理。因此，当 A 继续向 H 发送 ICMP Echo Rqst 消息时，将发现 H 所响应的 ICMP Echo reply 消息中的 IP 分组标识号继以前的标识号连续增长，由此 A 得出结论：V 上的 TCP 23 号端口并未开放。

A 继续向目标机器 V 的 TCP 25 号端口发送 TCP 连接请求，承载该 TCP 段的 IP 分组的源地址也是无辜机器 H 的 IP 地址，并且假设 V 上的 TCP 25 号端口是开放的。这时 V(按照 TCP 协议规范)将向 H 响应一个正常的连接确认及反向连接请求，也就是 TCP 3-握手过程中的第二条消息，然而 H 将对此回应一条 TCP 消息，其中 RST 位为 1 以通告这是一个错误，因为 H 实际上并未向 V 发起过任何 TCP 连接请求。注意这一 TCP 消息需要 H 生成一个 IP 分组承载，因此“耗用”了一个 IP 分组标识号，即图中的 ID#8。当 A 继续向 H 发送 ICMP Echo Rqst 消息时，将发现 H 所响应的 ICMP Echo reply 消息中的 IP 分组标识号没有继以前的标识号连续增长，由此 A 得出结论：V 上的 TCP 25 号端口当前是开放的。

以上是对这一隐形扫描过程的原理性描述，实际上入侵者要处理的情况更为复杂一些。但问题的关键在于，NIDS(垂直的虚线)所看到的消息流是什么？只不过是两次失败的 TCP 连接建立过程，NIDS 甚至始终没有看到真正的入侵者 A 的 IP 地址！当然，NIDS 能看到 H 的 IP 地址，但 H 实际上却是完全无辜的。

欺骗 NIDS(包括欺骗防火墙)的方法还有许多，这里当然不能一一列举。对第一个例子，读者不难看出如何抵御：NIDS 在进行入侵检测之前按照 IP 协议重组 IP 分组即可。对第二个例子也不难看出抵御方法，例如适当校正 IP 分组中的 TTL 域值到一个统一的、充分大的值就可以抑制这类欺骗。对第三个例子，不妨请读者自己思考如何抵御(习题 5-6)？

抑制以上攻击途径的一类普遍技术是对协议消息流的正规化处理，具体包括针对各个协议的消息中可能被用来对 NIDS 或防火墙实施欺诈的域的值进行验证或统一的校正，上面针对前两个例子的处理方法都属于对 IP 协议消息的正规化处理。回顾图 1-4 中，针对其中的 IP 首部的每个域，都存在一种相应的正规化处理，例如对版本号，若其值代表的是被保护网络所不支持的版本号(例如除 4 以外的值)，则废弃该 IP 分组；对首部长度域，若该值小

于 5(这意味着 IP 分组首部长度短于 20 字节)，则废弃该 IP 分组；对分组长度域，若其值大于实际承载该 IP 分组的链路帧的物理长度，则废弃之；若短于该链路帧的物理长度，则截断帧到恰好承载该 IP 分组的长度(这时需要重新计算帧的校验和)；对上层协议号，若其值所代表的是被保护网络所不支持的协议，则废弃该 IP 分组；对分组标识号、分段标志和分段位置域，根据其值对分割过的 IP 分组的片段先行重组然后再继续转发；对 IP 选项，一般可以去掉这一部分(这时需要正确修正 IP 首部长度域、分组长度域几 IP 校验和的值)，因为实践证明 IP 选项弊多利少，绝大多数常规应用并不使用选项功能。

对协议消息的正规化处理原则上也适用于其它协议，例如对 TCP 和 UDP，只是这类协议中的会话结构比 IP 协议更复杂，因此正规化处理所需要考虑的细节更多，例如对 TCP 协议一般不能简单去掉其选项而需要先进行识别。这里不再深入继续讨论这方面的技术，感兴趣的读者可以参考章后列举的材料。

5.5 小结与进一步学习的指南

这一章主要阐述最重要的两类网络安全系统，即防火墙和 NIDS，包括它们的工作机理和相关的技术。这两类系统在目前都被广泛应用，而且在很大程度上可以结合使用。

关于防火墙有许多优秀著作可以进一步学习，例如关于 Linux 上的 iptables 防火墙的详细使用方法与配置案例可以参考 S.Suehring 和 R.Ziegler 所著《Linux 防火墙》，何泾沙 译，机械工业出版社，2006。Cisco 教程也包含对其防火墙和安全路由器访问控制列表配置的详细解释和丰富、实用的案例。

在实用的层次上讨论 NIDS 的优秀著作是 Northcutt 主编的 *Intrusion Signatures and Analysis* 和 *Network Intrusion Detection*，两书由 New Riders Publishing 分别于 2001 2003 出版。第一部著作包含大量网络入侵的特征实例，第二部著作详细阐述典型工具及针对各类典型攻击的使用与分析技术，两部著作都包含丰富的实例。

关于 *Bro* 和 *Snort* 的以下经典论文至今价值不减，而且写作得深入浅出：

V.Paxson *Bro: a System for Detecting Network Intruders in Real-Time*, Computer Networks, 31(23):2435-2463, 1999(或见 Proc. 7th USENIX Security Symposium, 1998).

M.Roesch *Snort – Lightweight Intrusion Detection for Networks*, Proc. LISA'99.

Bro 和 *Snort* 自诞生至今一直在不断发展，关于这两个开放源代码 NIDS 的更多技术细节及最新进展可以阅读美国加洲大学 Berkeley 分校国立 Lawrence 实验室的网站

www.ee.lbl.gov 及网站 www.snort.org 上的丰富资料，例如下面这篇论文详细讨论了对协议消息流的正规化处理：

M.Handley, V.Paxon Network *Intrusion Detection: Evasion, Traffic Normalization and End-to-End Protocol Semantics*, In: Proc. 10th USENIX Security Symposium, 2001.

关于 libpcap 这一广泛应用的 IP 分组捕获工具的技术性论述参见其开发者的原始论文：S.McCanne, V.Jacobson *The BSD Packet Filter: A New Architecture for User-Level Packet Capture*, Proc. 1993 Winter USENIX Conference, 1993.

最后指出一个有趣的事实：虽然本章是从入侵者的角度讨论对 NIDS 和防火墙的欺骗技术(几乎所有的教科书也都如此)，但这类技术也可以被正当的软件利用来达到“无缝穿越”的目的，这样做的原因很大程度上是因为当今防火墙系统的实施在一定程度上干扰了某些正当软件的运行，例如某些基于 IP 的语音通讯系统就属于这类“无辜的受害者”。为了避免这一点，最近一些软件设计者开始考虑利用类似前述的“欺骗”技术来回避防火墙和 NIDS，一个广泛而有趣的例子就是 2003 年开发的应用于 P2P 系统上的 VoIP 协议 Skype，关于这一软件的精彩分析可以参见下面这篇深入浅出的技术报告：

S.Baset H.Schulzrine *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*, Tech. Report CUCS-39-04, Columbia University, 2004.

习 题

5-1 下面是一段病毒 W95/Mad.2736 的自解密程序的 Intel x86 汇编代码，试写出其加密算法：

```
        MOV    EDI, 00403045h
        ADD    EDI, EBP
        MOV    ECX, 0A6Bh
        MOV    AL, [key]
    Decrypt:
        XOR    [EDI], AL
        INC    EDI
        LOOP   Decrypt
        JMP    Start
  DB    key    86
  Start:  ……
```

5-2 试解释以下 iptables 的 ACL 表项的涵义，其中 1024::65535 含义是从 1024 到 65535 之间的任何数：

```
iptables –A OUTPUT –o eth1  –p tcp –m multiport --destination-port 80,443  --syn --source-port 1024::65535 \
```

-j REJECT

5-3 解释以下 Cisco 路由器上的 ACL 的含义：

```
interface serial 0 /*指定接口 serial 0*/
    ip access-group 102 out /*所有标识号为 102 的规则项均用以过滤离开接口 serial 0 的 IP 分组*/
access-list 102 permit icmp 160.11.0.0 0.0.255.255 any echo-request
access-list 102 permit icmp 160.11.0.0 0.0.255.255 any packet-too-big
access-list 102 deny icmp 160.11.0.0 0.0.255.255 any /*禁止任何其他 ICMP 消息离出*/
access-list 102 permit ip 160.11.0.0 0.0.255.255 any
```

请特别注意以上四项的顺序。如果换成其他顺序，过滤效果还一样吗？为什么？

5-4 如果不对速度做过高要求，防火墙也完全可以实现为一个用户模态的应用程序（事实上有许多防火墙产品正是这样实现的）。试设计一个防火墙软件，实现分组捕获及简单的过滤处理。尽可能详细地说明你需要的技术，并查阅相关的编程资料。关于分组捕获可以使用 libpcap。

5-5 对图 5-9，你能发现什么异常特征吗？由此可以检测出图中出现的反常 IP 分组。

提示：需要联系 IP 分组前后的其他 IP 分组进行判断。

5-6 考虑图 5-10，对 NIDS 给出一种合理可行的方法以防止这类欺骗。

提示：针对 TCP 协议进行分析。

5-7 (1)考虑以下试验：

IP#A 是一个已知的 IP 地址，从你的计算机发送一个目标地址为 IP#A 的 ICMP/echo request 分组，该 IP 分组的 TTL(寿命值)=1。

你的计算机接收到一个 ICMP 差错报告，报告该 IP 分组寿命超限，且该 ICMP 分组的源地址是 IP#1；记录下地址 IP#1。

从你的计算机发送一个目标地址为 IP#A 的 ICMP/echo request 分组，该 IP 分组的 TTL 值=2。

你的计算机仍接收到一个 ICMP 差错报告，报告该 IP 分组寿命超限，且该 ICMP 分组的源地址是 IP#2；记录下地址 IP#2。

重复以上试验，每次将 TTL 值增加 1，直到你接收到的不是报告 IP 分组寿命超限的 ICMP 消息，而是一个发自 IP#A 的 ICMP/echo reply 消息。

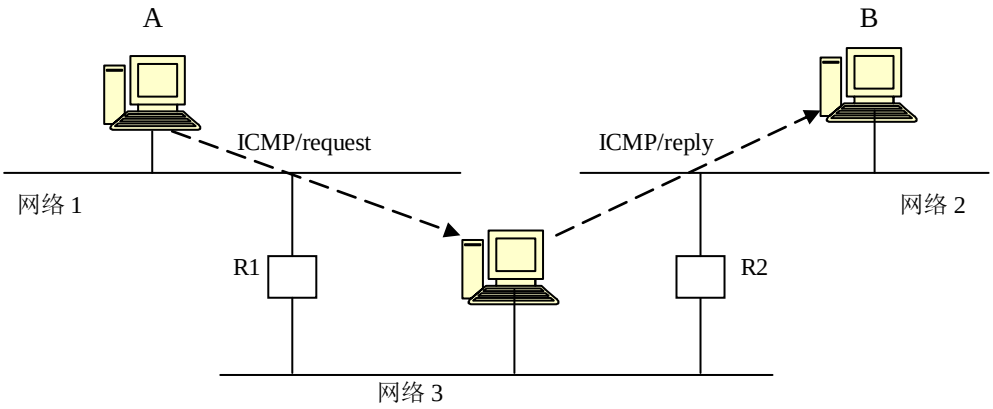
以上依次记录的地址 IP#1、IP#2、…意味着什么？你能从这里推断出些什么信息吗？

(2) traceroute 命令是一个路由跟踪命令，让你测试出从你的计算机到达一个已知 IP 地址的路由。你能根据以上试验推断出一种 traceroute 的实现方法吗？

关于 traceroute 命令的原理及源程序，感兴趣的读者可以参考 Stevens 的著作 Unix

Networking Programming, 2th ed, Vol. 1, socket-APIs and XTI, Prentice Hall, 1997。

5-8 一个利用 ICMP 实施拒绝服务攻击的例子：考虑下图，R1、R2 是路由器，假设网络 3 上有 10000 台主机，工作站 A 向网络 3 广播 ICMP/request 分组，但该分组的源地址故意不用 IP#A 而用 IP#B，这样一来会导致什么后果？如果利用路由器 R1 或 R2 上的访问控制列表来防止这种情况，过滤规则应该如何配置(不止一种方法)？



习题 5-1 的答案 密文 y[i] = x[i]⊕[key], i=0,1,…,N-1, N=0A68h.