# 基于误用与异常相结合的入侵检测系统的研究与设计

专业：　计算机软件与理论
硕士生：　李 伟
指导教师：　梁华金副教授

## 摘 要

随着计算机网络和通信技术的迅猛发展，网络将会无所不在地影响社会的政治、经济、军事和社会生活等各方面，网络安全已经成为世界各国共同关注的焦点。

入侵检测是一个全新的、迅速发展的领域，已成为网络安全中极为重要的一个课题。该项课题是继防火墙、数据加密等传统安全保护措施后新一代的安全保障技术。它对计算机和网络资源的恶意使用行为进行识别和响应，不仅检测来自外部的入侵行为，而且也检查内部用户的未授权活动。

本课题是广东省科技厅下达的智能交通系统的子项目——广东省高速公路联网组合式收费及管理辅助决策系统科研开发工作的子课题。我们的目标是设计一个入侵检测系统，它具有可扩展性、灵活性、安全性，使之能对现有的收费网络进行防护。在本文主要描述了一个误用与异常相结合的入侵检测系统，通过对开源软件 snort 的改进，使之具有发现新的攻击类型的能力。

本论文成果拟应用于高速公路联网收费系统等相关信息系统，联网收费系统是高速公路管理信息化最重要的组成部分，联网在带来服务快捷、方便管理和决策的同时，高速公路管理者和决策者对数据的依赖性越来越强，由于这些数据流伴随着巨大的资金流，成为决策者决策的重要参考依据，数据的丢失或者数据的失真不仅会中断正常的业务运行，而且很有可能造成巨大的经济损失，如今，随着联网规模日益庞大，由网络安全问题造成的潜在威胁也在与日俱增。

本文在分析轻量级入侵检测系统(Intrusion Detection System，以下简称 IDS)snort 的基础上，创建了异常的检测引擎，采用误用的检测手段对各种入侵特征建立规则文件，通过对规则文件的解析并进行模式匹配而建立起整个架构，作者设计了一个统计模型，对网络流量进行统计分析，这个基于异常的检测引擎与误用检测引擎并发执行。这种误用与异常相结合的方法有利于减少入侵检测的漏报率。本文原型在 Linux 操作系统下用 C 语言实现，并成功的检测了 SYN flood 攻击。

该系统采用插件机制，易于扩展，可以针对不用的应用需要增加特定的插件，由于规则语法简单，用户无需修改源代码就可以针对具体的攻击增加自己的规则。

**关键字**：误用，异常，插件，入侵检测系统。

Study and Design of a Misuse and Anomaly Combined Intrusion Detection System

System Major: Computer Science
Name: Wei Lee
Supervisor: Vice-Professor Huajin Liang

# ABSTRACT

With the rapid development of computer network and communication technology, network will affect our society anywhere and anytime in economy、politics、military and all kinds of other affairs. Network security has become the focus of the whole world.

The subject is planed by the science and technology office of GuangDong province and is the sub-project of Intelligent Transportation System. We aim at designing an Intrusion Detection System, which is extendable, flexible, safe enough to protect existing network. The author describes a misuse and anomaly combined Intrusion Detection System, which can detect new type of attacks.

The production of the paper will be applied to highway network toll system and other related information system. Network toll system is an important part of highway management information system, which brings rapid service, convenient management and decision. At the same time, the governors of highways depend more and more on data, which goes with large amount of capital, so data has become a very important factor for governors to make their decisions. Disappearance and distort of data will not only interrupt the normal operation, but also will cause huge lose in economy. Nowadays, with the expansion of network, the latent threaten caused by security problems grow day by day.

The paper is based on the analysis of lightweight intrusion detection system—snort. The author creates an anomaly detection engine. With a misuse detection engine, the system comprises a lot of rule files aiming at all kinds of attacks. By parsing the rule files and pattern matching, the whole architecture will be built. The author also designs a statistical model, which analyses network traffic and runs concurrently with the misused detection engine. The prototype is realized by using C language in Linux OS, and implements the detection of SYN flood attack.

The system is based on plugin mechanism, and it is extendable. The user can add different plugins depend on different requirements. The user can also add different rules depend on different attacks without modifying source code.

Keywords: misuse, anomaly, plugin, Intrusion Detection System.

# 引 言

入侵检测系统(Intrusion Detection System, IDS)作为一种重要的安全部件，是网络与信息安全防护体系的重要组成部分，是传统计算机安全机制的重要补充。自1980年被提出以来，IDS在20多年间得到了较快的发展。特别是近几年，由于非法入侵不断增多，网络与信息安全问题变得越来越突出。IDS作为一种主动防御技术，越来越受到人们的关注。

网络入侵检测系统（NDIS）在网络上检测原始的网络传输数据，分析捕获的数据包，主要工作是匹配入侵行为的特征或者从网络活动的角度检测异常行为，进而完成入侵的预警或记录。从检测模式言，误用检测 (misuse detetion)对已知攻击的特征模式进行匹配,包括利用工作在网卡混杂模式下，被动地进行协议分析，以及对一系列数据包解释分析特征。本系统采用基于规则的入侵检测方式,即针对每一种入侵行为，都提炼出它的特征值 ，并按照规范写成检测规则，从而形成一个规则库。然后将捕获的数据包按照规则逐一匹配，若匹配成功，则认为该入侵行为成立。目前的大多数网络入侵检测系统都采用这种基于模式匹配的误用检测模式，另外一种是基于异常的检测模式，本文采用统计分析的方法设计了一个异常检测模型。并将误用与异常两种方法结合起来，并发执行。

论文成果应用环境：

2000 年 10 月，交通部颁布了《高速公路联网收费暂行技术要求》[1]，其中明确规定了高速公路相关信息系统要建立网络安全防护措施，这也为我们将入侵检测系统应用到高速公路相关信息系统中提供了政策依据。

广东新粤机电工程有限公司致力于高速公路相关信息系统的开发，并已经有了一套功能完备的应用系统，比如高速公路收费系统、道路监控系统等，为了这套系统在安全的环境下运行，本文提出了将网络入侵检测系统应用到高速公路相关信息系统的观点。

论文的贡献：

本文在研究开源软件 snort 的基础上，创建了误用与异常相结合的入侵检测系统（MACIDS），采用模式匹配和统计分析相结合的检测方式，扩展了 snort 原来的功能和结构。

MACIDS 在结构上采用插件机制，简化了编码工作，系统本身可以很容易地增加插件，具有很大的灵活性，主要包括预处理插件，处理插件和输出插件。 MACIDS

整体上可以分为五大引擎，包括数据包获取和解码引擎、预处理引擎、基于误用的检测引擎、基于异常的检测引擎、记录日志引擎。

MACIDS 借鉴了 TCP 协议中流量控制的滑动窗口机制，采用滑动窗口的方法来实时获取数据包，动态更新，并定义了置信区间，来判断是否有异常情况发生，克服了 snort 原有系统先遭受攻击，再通过增加规则来增强检测能力的缺点，使系统的功能得到扩展。

论文的组织：

本文的结构安排：第一章介绍网络安全与入侵检测系统的综述，对 IDS 的定义、发展历史以及分类做详细论述。第二章介绍 MACIDS 的框架结构，主要对入侵检测系统的总体结构和总体流程做介绍。第三章介绍通过对 snort 的改进实现误用与异常相结合的入侵检测系统。第四章介绍详细说明误用与异常相结合的检测引擎的实现，详细说明了模式匹配的检测机制和基于平均值和方差的异常检测引擎。第五章是对该系统的测试部分，检验了异常检测引擎的效果。

# 第一章 网络安全与入侵检测系统综述

网络安全基本上是一个实践性的技术领域，在这里大多数的理论基本上是经验的汇集，因此要给一个形式化的定义是非常困难的，甚至很难给出一个安全与否的标准。

在网络安全的范畴内，网络并不是物理的网络，它包含以下三个基本要素[2]：

(1) 数据：包括在网络上传输的数据以及端系统中的数据，从本质上说这些电子意义上的数据都是 01 比特的组合，但是经过特定的程序产生和处理之后，它们就具有了多种多样的语义学上的意义。

(2) 关系：网络作为交流的重要手段，涉及到通信各方信赖关系的建立与维护，这也是攻击者比较感兴趣的一个方面，因为信赖关系的窃取就意味着能力和数据访问权力的获取，进而可以转化为物理意义上的财富。

(3) 能力：包括网络系统的传输能力以及端系统的处理能力，前者意味着网络连接能力的充分运用，而后者则意味着数据处理能力和服务提供能力等。

网络安全的意义，就在于为以上三个要素提供保护，保证这三者能够为所应为，为合适的人服务，而且只为合适的人服务。相应地，网络安全也就包含以下三个方面：

(1) 数据保护：包括数据的机密性保护和完整性保护，主要针对数据窃取、数据篡改等攻击，其基本的手段包括加密和访问控制。这方面的理论比较完备（主要是加密体制的建立和加密算法的运用），实现手段也比较完善。

(2) （信赖）关系保护：包括身份鉴别与安全地建立、维护信赖关系，主要针对网络身份冒充、连接截取等攻击，基本的手段包括加密与协议的安全设计。相对来说这方面理论也比较完备，但在实现手段上会有些漏洞。

(3) 能力保护：包括对网络系统的传输功能以及端系统的处理能力的保护，主要针对拒绝服务、远程权力获取等攻击。这方面的理论基本上是实践经验的总结，运用的手段也基本上是试验性的。能力保护相关的工作也是入侵检测系统发挥作用之处。

## 1.1 网络安全模型

入侵检测系统（Intrusion Detection System，以下简称 IDS）是近年来发展很快的一种新型的安全技术。相对于传统的纯粹防御为主的技术，例如防火墙，IDS 技

术可以更有效地发现网络或系统中出现的各种入侵行为。

其实，从网络安全技术的发展过程来看，各种技术的出现和应用都有着较为明显的特征：第一代是以保护数据完整性、有效性、机密性为目标的各种加密技术；第二代是以包过虑和代理机制来进行访问控制的防火墙技术；而第三代，则以入侵行为检测及事件响应为特征的 IDS 技术。其实这并不是严格按照时间阶段来划分的，只是从技术的应用范围来看，有了 IDS，整个网络安全技术才有了较为完整的框架。再加上主动检测技术（扫描），一个动态可适应的安全模型就诞生了——P2DR（Policy，Protection，Detection，Response）[3,4,5]。

传统的网络安全建设过程，往往是事后的、被动的、单一的，无非是针对出现的问题单纯运用一些防护措施，并以某个问题的暂时解决为过程结束的标志，这样的网络安全建设模式，往往带有很大的盲目性，远不能适应网络安全保护的发展要求。

目前，国际上被广泛采用的是 P2DR 安全模型，它是一种动态可适应性的网络安全概念模型，是由 PDR（Protection、Detection、Response）模型引申而来的，其中增加了 Policy 功能，并着重强调管理策略在信息安全工程中的主导地位。安全技术措施（安全产品）无须盲目引进，而是围绕安全策略的具体需求有序地组织在一起，架构一个动态的安全防护体系。P2DR 模型指出，在整体的安全策略控制和指导下，在综合运用防护工具（如防火墙、操作系统身份认证、加密等手段）的同时，利用检测工具（如漏洞评估、入侵检测等系统）了解和评估系统的安全状态，将系统调整到“最安全”和“风险最低”的状态。

P2DR 模型包含四个主要部分：Policy（安全策略）、Protection（防护）、Detection（检测）和 Response（响应）。防护、检测和响应组成了一个完整的、动态的循环过程，并在安全策略的指导下保证信息系统的安全。

根据 P2DR 模型的理论，安全策略是整个网络安全的依据。不同的网络需要不同的策略，在制定策略以前，需要全面考虑网络各个方面的安全性要求，并确定响应的防护手段和实施办法。策略一旦制定，应当作为整个企业安全行为的准则。

在 P2DR 模型中，动态的防护和响应检测机制是重要的过程环节。在传统的网络安全解决方法中，采用的都是静态安全防护措施，它只能对网络系统中的某几个环节起到保护作用，但面对网络中的整体安全和变化性要求，并不能起到有效的保护作用。P2DR 则要求，主动扫描审计、检测响应机制，应该同防护环节相结合，在协作过程中，建立起相互促进相互依据的关系，在安全策略的总体指导下，共同维护网络安全的动态发展。

下图为 P2DR 模型的示意图：


图 1－1 P2DR 模型示意图

## 1.2 网络安全威胁模型

和规划其他任何安全系统一样，定义网络安全威胁模型[6]是必须面临的首要问题，所谓威胁模型，主要描述攻击者渴望拥有的资源以及渴望采取的攻击，从而限定我们在特定系统中重点考虑的不安全因素，排除不需要考虑的各类因素。

一般而言，系统可能面临的安全威胁主要有两种来源：人为因素和自然因素，如下图所示。

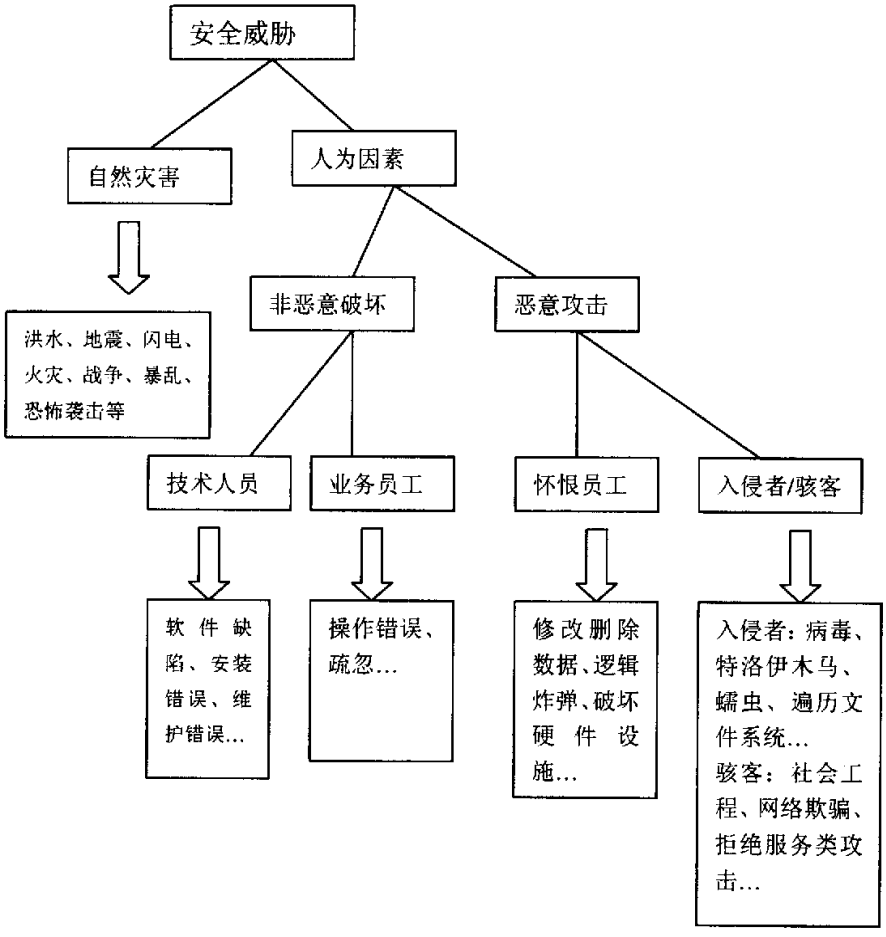



图 1—2 网络安全威胁的来源

其中，自然因素既包括地震、台风、洪水、闪电火灾等会导致系统损毁的不可抗拒的自然灾害，也包括系统、设备、网络及其他相关器件等物理设施功能失效。此外，对于战争、暴乱、恐怖袭击等，尽管也是人类活动，但是因为本身难以通过信息安全策略和控制手段加以制止，习惯上也归到这个类别里面，视同为自然灾害。

自然因素构成的安全威胁里，应对自然灾害方面通常可以考虑设置一些诸如防雷、接地、消防或者异地容灾等措施给予一定的控制，但是一般比较难以从根本上加以防范，因此在讨论联网收费系统数据安全问题的过程中重点要考虑的还是功能失效造成的安全威胁，采用冗余、备份、故障检测等技术手段，削弱可能带来的安全威胁。

人为因素造成的威胁可以分为恶意和非恶意两种情况。一般来说，非恶意的破坏主要来自缺乏专门训练或者对系统安全了解不足的用户或者员工；而恶意攻击主要来自那些为了达到某种目的而实施蓄意的破坏性操作的入侵者、“骇客”及怀有不满情绪的员工等。

“入侵者”通常是一些怀有目的却又不能通过合法手段访问系统的人，其实施攻击的动机不确定，但可以会危害到计算机上的所有组件。入侵者经常使用的手段有病毒、特洛伊木马、蠕虫或者遍历文件系统等。病毒和蠕虫往往以增加系统的处理或存储负荷来降低系统的可用性，严重的甚至可以导致系统瘫痪；特洛伊木马程序可能威胁到系统信息的一致性和机密性；通过遍历文件系统，可以窃取机密信息。

“骇客[7]”是那些擅自闯进为授权的系统或者非法越过系统边界的人。骇客们常常使用的手段有：密码破解、利用操作系统、数据库、应用程序中已知的安全漏洞、邮件攻击、拒绝服务（DoS）攻击、网络欺骗、社会工程等。

## 1.3　入侵检测定义

本文中的“入侵”（Intrusion）是个广义的概念，不仅包括被发起攻击的人取得超出合法范围的系统控制权，也包括收集漏洞信息、造成拒绝访问（Denial of Service）等对计算机系统造成危害的行为。

入侵检测（Intrusion Detection，简称 ID），顾名思义，便是对入侵行为的发觉。它通过对计算机网络或计算机系统中的若干关键点收集信息并对其进行分析，从中发现网络或系统中是否有违反安全策略的行为和被攻击的迹象。

入侵检测的一个定义是“识别不经过授权而进入计算机系统和那些有合法的访问权限，但是滥用了他们的权限的行为” [8]。E.H. Spafford 往这个定义里增加“识别不经过授权而访问计算机系统或者滥用已有权限的尝试”[9]。本文的定义和 Heady et al.[10]里面所讲的是一致的，在那里，一个入侵被定义为“任何尝试危害资源的完整性、机密性和有效性的行为集合”，而不管这些行为是成功的还是失败的。

具体来说，入侵检测系统的主要功能有[11]：

（1）监测并分析用户和系统的活动；

（2）核查系统配置和漏洞；

（3）评估系统关键资源和数据文件的完整性；

（4）识别已知的攻击行为；

（5）统计分析异常行为；

（6）操作系统日志管理，并识别违反安全策略的用户活动。

由于入侵检测系统的研究在近几年中飞速发展，许多公司投入到这一领域上来。除了国外的 ISS、axent、NFR、CISCO 等公司外，国内也有数家公司（如中联绿盟，中科网威等）推出了自己相应的产品。但就目前而言，入侵检测系统还缺乏相应的

标准。目前，试图对 IDS 进行标准化的工作有两个组织：IETF 的 Intrusion Detection Working Group （IDWG）[12]和 Common Intrusion Detection Framework （CIDF）[13]，但进展非常缓慢，尚没有被广泛接收的标准出台。

从实验室原型研究到推出商业化产品、走向市场并获得广泛认同，入侵检测系统（IDS）经过了二十多年的历史。

## 1.4　入侵检测发展历程

1980 年 4 月，James P. Anderson 为美国空军做了一份题为《Computer Security Threat Monitoring and Surveillance》[14]（计算机安全威胁监控与监视）的技术报告，第一次详细阐述了入侵检测的概念。她提出了一种对计算机系统风险和威胁的分类方法，并将威胁分为外部渗透、内部渗透和不法行为三种，还提出了利用审计跟踪数据监视入侵活动的思想。这份报告被公认为是入侵检测的开山之作。

从 1984 年到 1986 年，乔治敦大学的 Dorothy Denning 和 SRI/CSL（SRI 公司计算机科学实验室）的 Peter Neumann 研究出了一个实时入侵检测系统模型，取名为 IDES（入侵检测专家系统）[15,16]。该模型由六个部分组成：主体、对象、审计记录、轮廓特征、异常记录、活动规则。它独立于特定的系统平台、应用环境、系统弱点以及入侵类型，为构建入侵检测系统提供了一个通用的框架。

1988 年，SRI/CSL 的 Teresa Lunt 等人改进了 Denning 的入侵检测模型，并开发出了一个 IDES。该系统包括一个异常检测器和一个专家系统，分别用于统计异常模型的建立和基于规则的特征分析检测（如图 1-3 所示）。

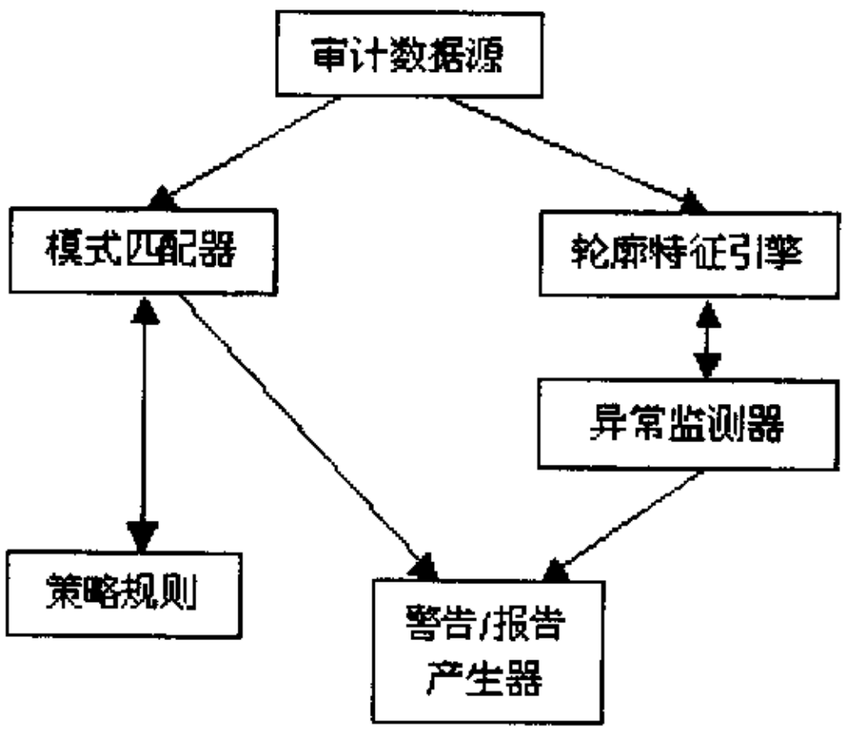



图 1-3　IDES 结构框架图

1990 年是入侵检测系统发展史上的一个分水岭。这一年，加州大学戴维斯分校

的 L. T. Heberlein 等人开发出了 NSM（Network Security Monitor）。该系统第一次直接将网络流作为审计数据来源，因而可以在不将审计数据转换成统一格式的情况下监控异种主机。从此之后，入侵检测系统发展史翻开了新的一页，两大阵营正式形成：基于网络的 IDS 和基于主机的 IDS。

1988 年的莫里斯蠕虫事件发生之后，网络安全才真正引起了军方、学术界和企业的高度重视。美国空军、国家安全局和能源部共同资助空军密码支持中心、劳伦斯利弗摩尔国家实验室、加州大学戴维斯分校、Haystack 实验室，开展对分布式入侵检测系统（DIDS）[17]的研究，将基于主机和基于网络的检测方法集成到一起，其总体结构如图 1-4 所示。

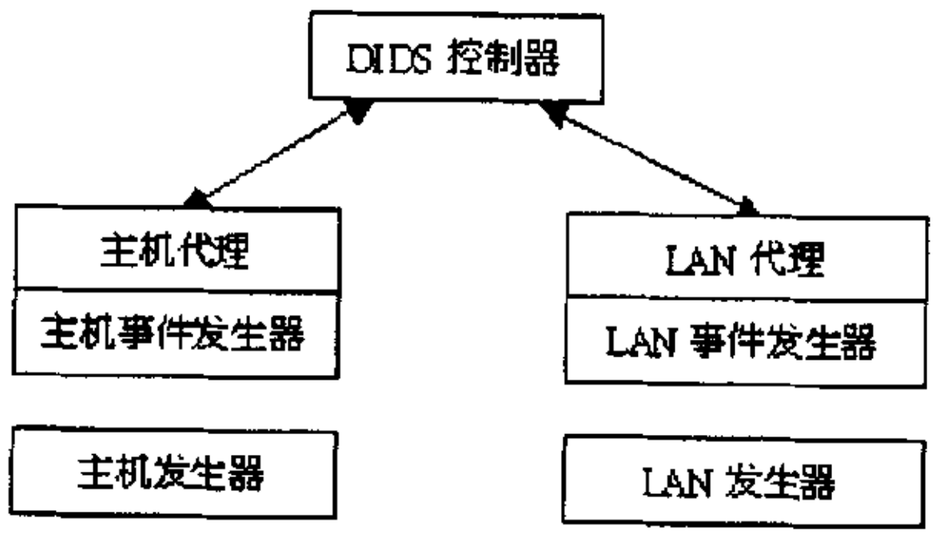



图 1-4　DIDS 结构框图

DIDS 是分布式入侵检测系统历史上的一个里程碑式的产品，它的检测模型采用了分层结构，包括数据、事件、主体、上下文、威胁、安全状态等 6 层。

从二十世纪九十年代到现在，入侵检测系统的研发呈现出百家争鸣的繁荣局面，并在智能化和分布式两个方向取得了长足的进展。目前，SRI/CSL、普渡大学、加州大学戴维斯分校、洛斯阿拉莫斯国家实验室、哥伦比亚大学、新墨西哥大学等机构在这些方面的研究代表了当前的最高水平。

## 1.5　入侵检测分类

研究一种技术或者系统的基本方法之一是分类，分而制之也是人类学习的一种习惯。对于入侵检测技术而言，也是如此。

根据着眼点的不同，对入侵检测技术的分类方法很多，下面逐一进行介绍。

### 1.5.1　按数据来源分类

按照数据来源的不同，可以将入侵检测系统分为三类：

(1) 基于主机：系统获取数据的依据是系统运行所在的主机，保护的目标也是

系统运行所在的主机。

(2) 基于网络：系统获取的数据来源是网络传输的数据包，保护的目标是网络的运行。

(3) 混合型：勿庸置疑，混合型就是既基于主机又基于网络，因此混合型一般也是分布式的。

### 1.5.2 按时效性分类

按照数据分析发生的时间不同，可以分为：

(1) 脱机分析：就是在行为发生后，对产生的数据进行分析，而不是在行为发生的同时进行分析。如对日志的审核、对系统文件的完整性检查等都属于这种。一般而言，脱机分析也不会间隔很长时间，所谓的脱机只是与联机相对而言的。

(2) 联机分析：就是在数据产生或者发生改变的同时对其进行检查，以发现攻击行为。这种方式一般用对网络数据的实时分析，对系统资源要求比较高。

### 1.5.3 按分布性分类

按照系统各个模块运行的分布式方式不同，可以分为：

(1) 集中式：系统的各个模块包括数据的收集与分析以及响应模块都集中在一台主机上运行，这种方式适用于网络环境比较简单的情况。

(2) 分布式：系统的各个模块分布在网络中的不同的计算机、设备上，一般来说分布性主要体现在数据收集模块上，例如有些系统引入传感器（Sensor），如果网络环境比较复杂、数据量比较大，那么数据分析模块也会采用分布式结构。

### 1.5.4 按分析模型分类

根据数据分析方法（也就是检测方法）的不同，可以将入侵检测系统分为 3 类[18]：

(1) 异常检测模型（Anomaly Detection Model）：这种模型的特点是首先总结正常操作应该具有的特征，例如特定用户的操作习惯与某些操作的频率等；在得出正常的操作模型之后，对后续的操作进行监视，一旦发现偏离正常的统计学意义上的操作模式，即进行报警。可以看出，按照这种模型建立的系统需要具有一定的人工智能，由于人工智能领域本身的发展缓慢，基

于异常的检测模型建立入侵检测系统的工作进展也不是很好。

(2) 误用检测模型（Misuse Detection Model）：这里翻译成误用有一些习惯性的因素，它的意思应该是不正确地使用。这种模型的特点是收集非正常的操作也就是入侵行为的特征，建立相关的特征库；在后续的检测过程中，将收集到的数据与特征库中的特征代码进行比较，得出是否入侵的结论。可以看出，这种模型与主流的病毒检测方式基本一致。当前流行的系统基本上采用了这种模型。

(3) 误用与异常相结合的检测模型（Misused and Anomaly Combined Detection Model）：这种模型将误用和异常两种检测模型结合起来，并发执行，本文就采用这种方式。

# 第二章 MACIDS 的应用环境和总体结构

## 2.1 MACIDS 的应用环境

过去的 15 年，是我国高速公路从实现零的突破到大发展的 15 年，国内高速公路建设领域屡创奇迹，至 2003 年 9 月，全国高速公路里程已达到 27000 公里，居世界第二位，预计至 2010 年，高速公路里程还要翻一番。

高速公路的建成、联网、加密使我国交通基础设施的面貌和交通状况发生了质的改变，同时，与高速公路配套各项管理措施也在与时俱进，高速公路收费系统运营模式也已经从高速公路诞生之初的分段收费发展到区域联网和省域联网收费。

高速公路实行联网收费的要求提出后，各省市陆续开始结合本省的实际情况着手组织实践工作，目前已经有广东、山西、河北、贵州等多个省份的高速公路实现了区域的或者省域的联网收费，2003 年 9 月京沈高速公路联网收费试点的成功，更是标志着高速公路收费系统正在朝着更广阔的范围和更高的水平在不断发展。

联网收费系统是高速公路管理信息化、智能化最重要的组成部分，联网在带来服务快捷、方便管理和决策的同时，高速公路管理者和决策者对数据的依赖性越来越强，由于这些数据流伴随着巨大的资金流，成为决策者决策的重要参考依据，数据的丢失或者数据的失真不仅会中断正常的业务运行，而且很有可能造成巨大的经济损失，如今，随着联网规模日益庞大，由数据安全问题造成的潜在威胁也在与日俱增。

保证联网收费系统的安全不仅是广大联网单位和联网收费管理单位关注的问题，也是关系到联网收费系统今后的发展和提升水平的关键问题，但是通过资料调研表明，目前还没有看到专门针对联网收费系统安全这一研究主题的相关报道资料。

### 2.1.1 高速公路联网组合收费管理系统的构成

联网组合收费管理系统总体框架结构一般由收费结算中心和联网收费区域内各路段的收费系统两部分组成，由于高速公路多采用分级管理模式，路段收费系统本身一般又可以进一步分为若干层次，其中，最具有代表性的就是区域收费分中心、收费站以及收费车道等三层，其总体结构设置如图所示。

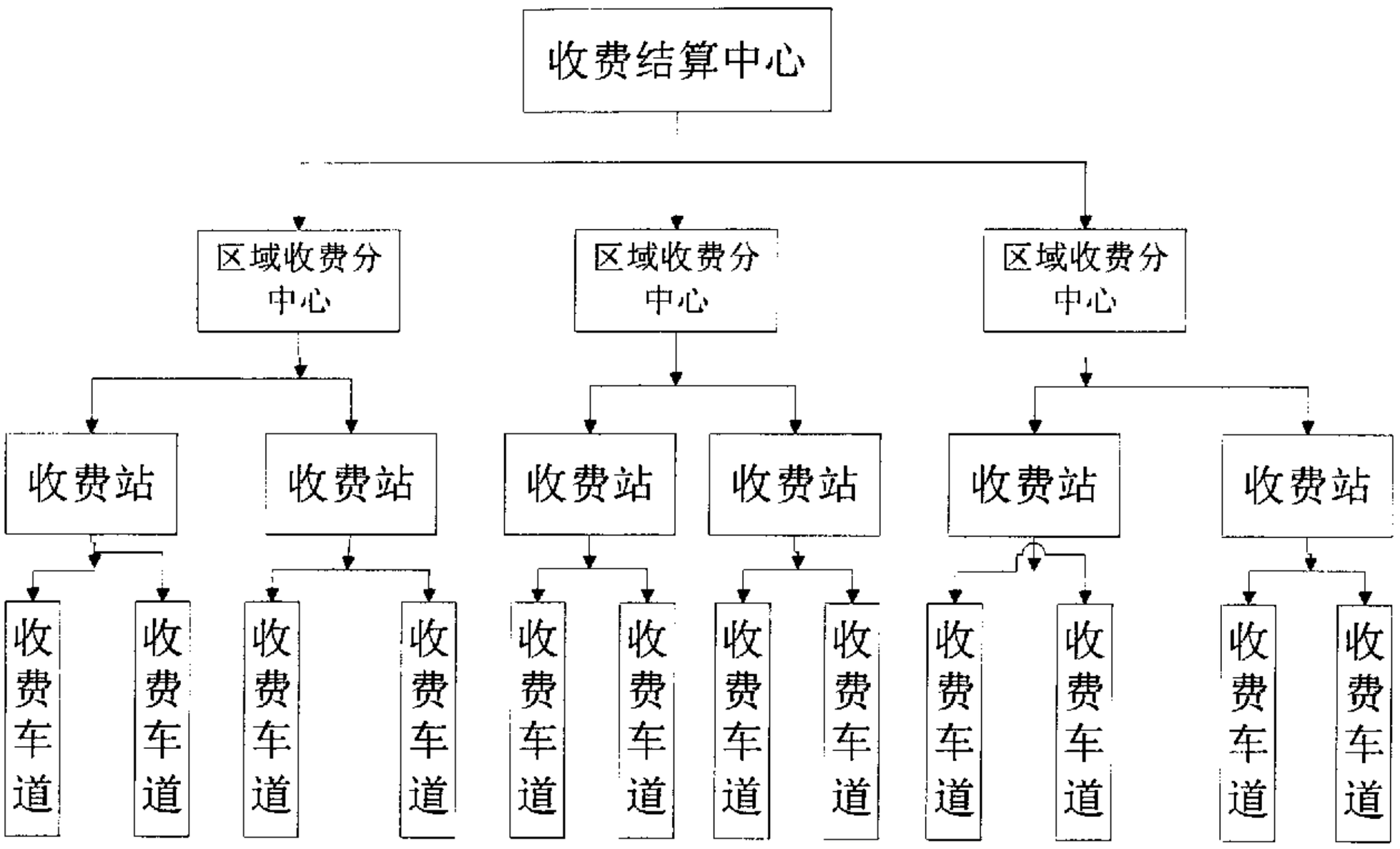



图 2—1 高速公路联网组合收费管理结构图

2.1.1　各级业务管理系统的主要功能如下：

2.1.1.1　收费结算中心的基本功能包括：

制定和下传联网收费系统运行参数（包括费率表、时间同步、系统设置参数等），帐务分割的报表数据；

接收收费站、收费中心上传的所有原始收费数据和收费统计数据；

接收银行提供的资金明细数据；对通行费进行拆分和复核，与指定银行进行帐目信息交换和通行费结算、帐务分割；并将拆分结果形成划帐指令，交给结算银行完成资金划拨；

联网收费通行券、票证的管理；数据库、系统维护、网络管理；汇总、统计、查询、打印报表；数据存储、备份和安全保护。

可扩展的主要功能有：预付卡和电子不停车收费的管理；客户服务和抓拍图像的管理等。

2.1.1.2　区域收费中心的基本功能包括：

接收和下传联网收费系统运行参数；

采集收费车道每一条原始数据；准确可靠地收集管辖区内的原始收费数据与资料；

处理收集到的数据与资料，汇总、统计、查询、打印收费、管理、交通量等报表；

本路段内的通信券、票证的管理；

联网收费系统中的操作、维修人员权限的管理；数据库、系统维护、网络管理等；

数据、资料的存储与备份和安全保护等。

2.1.1.3　收费站的基本功能包括：

轮询所有收费车道，采集收费车道每一条原始数据；

对收费车道的运行状况实施检测与监视，具有故障自动检测功能；向收费中心/收费结算中心传输收费业务数据（收入、交通、管理）；

接收收费结算中心和收费中心下传的系统运行参数并下传给收费车道；收费员录入班次的收费额；

值班员录入欠（罚）款和银行缴款数据；

通行券、票证的管理；

抓拍图像的管理等。

2.1.1.4　收费车道的基本功能是：

按车道操作流程正确工作，并将收费处理数据上传收费站、分中心和收费结算中心的计算机系统；

接收收费站下传的系统运行参数；

对车道设备的管理与控制，具有设备状态自检功能；

当通信中断时可降级使用，但不丢失数据，具有后备独立工作能力；

为车辆通行提供控制信息。

2.1.1.5　MACIDS 的应用对象

利用防火墙技术，经过仔细的配置，通常能够在内外网之间提供安全的网络保护，降低了网络安全风险。但是仅仅使用防火墙，网络安全还远远不够，主要表现在以下几个方面：入侵者可寻找防火墙背后可能敞开的后门[28]；入侵者可能就在

防火墙内；由于性能的限制防火墙通常不能提供实时的入侵检测能力；保护措施单一。

高速公路联网收费系统安全体系必须建立一个实时攻击识别系统，管理和降低安全风险，保证网络安全防护能力能够不断增强。

## 2.2 MACIDS 的体系架构

本系统在体系架构上主要有以下特点：

(1) 检测机制上看，它不仅具有基于规则的误用检测方法，而且还有基于异常的检测方法（第三方添加）。
(2) 体系结构上看，它充分考虑到扩展的功能，使用了大量的插件。
(3) 功能模块上看，各个模块功能明晰，相对独立，设计合理。
(4) 编码上看，它具有很好的编程风格和详细的注释，易于理解。

### 2.2.1 MACIDS 的总体结构

首先我们通过图示的方法来论述 MACIDS 模块的组成以及相互关系，如图 2-2 所示：

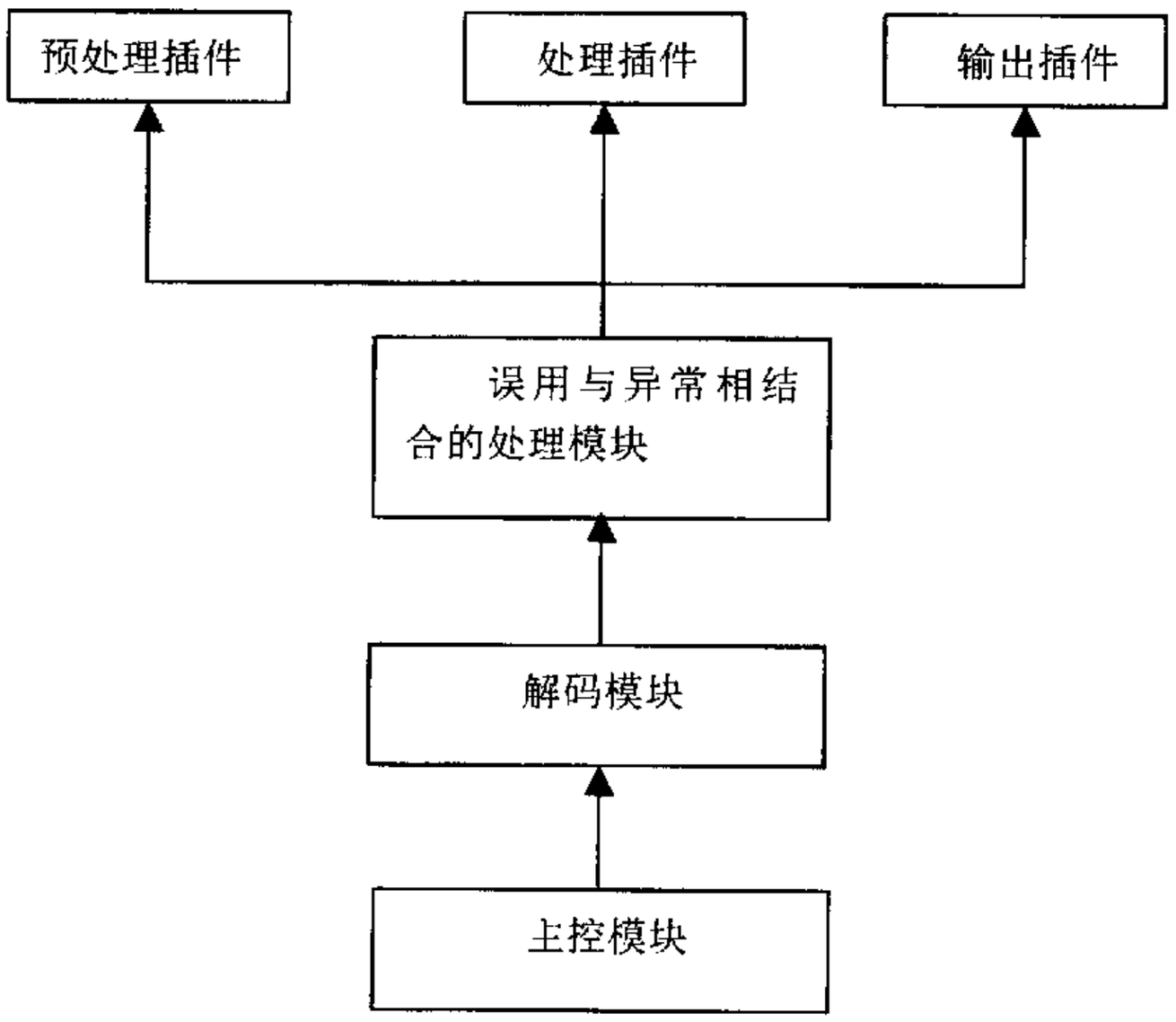



图 2-2 MACIDS 的总体结构图

下面就从功能上对图 2-2 中的各个模块进行分析说明：

(1) 主控模块实现的功能包括所有模块的初始化、命令行解释、配置文件解释、数据包捕获 libpcap[19]的初始化，然后调用 libpcap 开始捕获数据包，并进行解码检测入侵；此外，对所有插件的管理功能也属于主控模块的范围。

(2) 解码模块把从网络上抓取的原始数据包，从下到上沿各个协议栈进行解码并填充相应的数据结构，以便规则处理模块处理。

(3) 误用与异常结合得处理模块包含两部分：误用检测部分，实现对这些报文进行基于规则的模式匹配工作，检测入侵行为；异常检测部分，通过统计数据包中 TCP 标志位的情况，进行分析检测。

(4) 预处理插件在模式匹配之前进行，对报文进行分片重组、流重组等预处理操作。

(5) 处理插件主要检查数据包的各个方面，包括数据包的大小、协议类型、IP/ICMP/TCP 的选项等，辅助规则匹配完成检测功能。

(6) 输出插件实现在检测到攻击后执行各种输出和反应的功能。

### 2.2.2　MACIDS 的插件机制

在 MACIDS 中运用插件机制[20,21]，对于 MACIDS 来说，插件机制具有以下一些明显的优点：

(1) 通过增加插件，MACIDS 能够非常容易地增加功能，使程序具有很强的可扩展性。

(2) 插件机制简化了 MACIDS 的编码工作。

(3) 插件机制使代码功能内聚，模块性强，程序相对易读。

MACIDS 主要包括三种类型的插件，下面我们对这三种插件简单介绍并进行比较：

1）预处理插件

它们的源文件名都是以 spp_开头的，在规则匹配（误用检测）之前运行，完成的功能主要分为以下几类：

（1） 模拟 TCP/IP 堆栈功能的插件，如 IP 碎片重组、TCP 流重组插件。

（2） 各种解码插件：Http 解码插件、Unicode 解码插件、RPC 解码插件、Telnet 插件等。

（3） 规则匹配无法进行攻击检测时所用的检测插件：bo 检测插件、arp 欺骗检测插件等。

2）处理插件

它们的源文件名都以 sp_开头，在规则匹配阶段的 ParseRuleOption 中被调用辅助完成基于规则的匹配检测过程。每个规则处理函数通常对应规则选项中的一个关键字，实现对这个关键字的解释或者辅助解释。关于规则选项各个关键字的含义和使用方法可参见第 3 章。这些插件的主要功能是：

（1） 协议各字段的检查，如 TCPFlag、IcmpType、IcmpCode、Ttl、IpId、TcpAck、TcpSeq、Dsize、IpOption、Rpc、IcmpId、IcmpSeq、IpTos、FragBits、TcpWin、IpProto 和 IpSame 等。

（2） 一些辅助功能，如 Respond、Priority、PatternMatch、Session、React、Reference 等，这些插件分别完成响应（关闭连接）、严重级别、模式匹配（内容）、会话记录、攻击响应（高级的响应机制）、攻击参考信息等功能。

3）输出插件

它们的原文件名都以 spo_开头，这些插件分为日志和警告两种类型放入两个列表中，在规则匹配过程中和匹配之后结束之后调用，以便记录日志和警告。和其他插件相似，它们也对应一个相应的关键字，规则中响应的关键字将激活这些插件的运行。

下面以一个 Http 解码处理器插件为例来解释插件的内部结构。

(1) 每个插件都有一个安装函数，名称为 SetupXXX（），如 Spp_http_decode.c：SetupHttpDecode()。这个函数需要放在 Plugbase.c：InitPreprocessor()中，而且一定要被调用。这些安装函数会在一个列表中注册自己对应的关键字和响应的初始化函数，Http 解码预处理插件安装了两个关键字“http_decode”和“http_decode_ignore”，其相对应的初始化函数是 HttpDecodeInit()和 HttpDecodeInitIgnore()，把它们注册到函数列表 PreprocessKeywords 中。

(2) 在解释规则文件时，如果检测到相应的关键字，系统就会使用规则文件中这些关键字后的字符串作为参数来调用相应的初始化函数。初始化函数会根据这些参数初始化插件的状态，并把响应的处理函数注册到处理列表中。这里，当初始化函数 HttpDecodeInit()完成初始化后，会把处理函数 PreprocUrlDecode()注册到列表 PreprecessList()中。如果规则文件中没有相应的关键字，就不会触发相应的初始化函数，入侵检测过程中也就不会调用相应的处理函数，也就是不使用这个插件。

(3) 在检测过程中，一旦匹配成功，就会触发处理函数的执行，预处理器就会在 preprocess 预处理过程中对数据报调用 PreprocUrlDecode 进行处理。

从上面的分析可以看出，输出插件和预处理插件除了注册到不同的列表之外，其他的过程都很相似；处理插件的过程其实也是大同小异，只是在初始化过程中有

所不同而已。输出和预处理插件的初始化通常只有一次，在内存中只有一个实例，所以被注册到一个列表中；处理插件则完成每个匹配规则的一部分功能，所以处理插件为每个匹配规则初始化一次，然后插入到规则链表中这个规则的列表中。

综上所述，一个插件通常由三个框架函数组成：插件安装函数、插件初始化函数、插件的处理函数。如下表所示：

表 2—1 插件函数表

| 名　称 | 函 数 名 | 何 时 调 用 | 功　能 |
|---|---|---|---|
| 插件安装函数 | SetupXXX() | 程序初始化时调用 | 注册插件的初始化函数 |
| 插件初始化函数 | XXXInit() | 解释规则文件时调用 | 完成该插件的初始化，并注册处理函数 |
| 插件的处理函数 | XXXXX() | 检测流程调用 | 在检测过程中完成插件的功能 |

## 2.3 MACIDS 的系统流程设计

### 2.3.1 libpcap 的流程

由于本系统定位在轻量级入侵检测工具上，因此它的体系结构相对简单，基本上属于事件驱动类型；同时，它又是一个标准的基于 libpcap 库的应用，使用 libpcap 库来捕获网络上的报文，并触发 IDS 的检测过程进行检测。图 2—3 就是一个标准的基于 libpcap 库的应用程序流程。

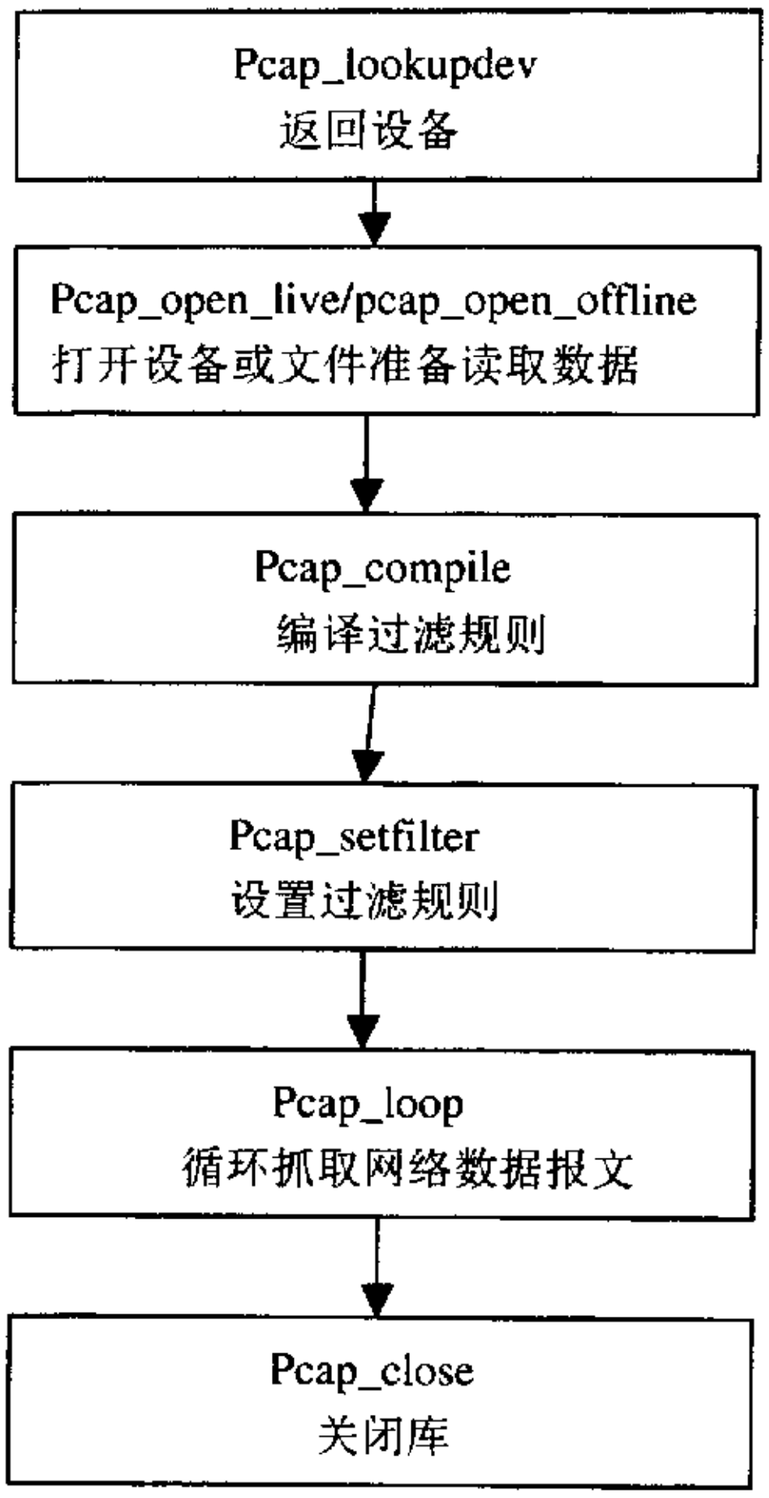



图 2－3 基于 libpcap 应用的基本流程

该图看上去似乎与 IDS 没什么关系，但其实它可是 IDS 运行的基础[22]，通过它可以增强对 IDS 底层运行情况的理解。

### 2.3.2　MACIDS 的系统流程设计

作为一个轻量级的入侵检测系统，MACIDS 的系统流程设计如图 2-4 所示。

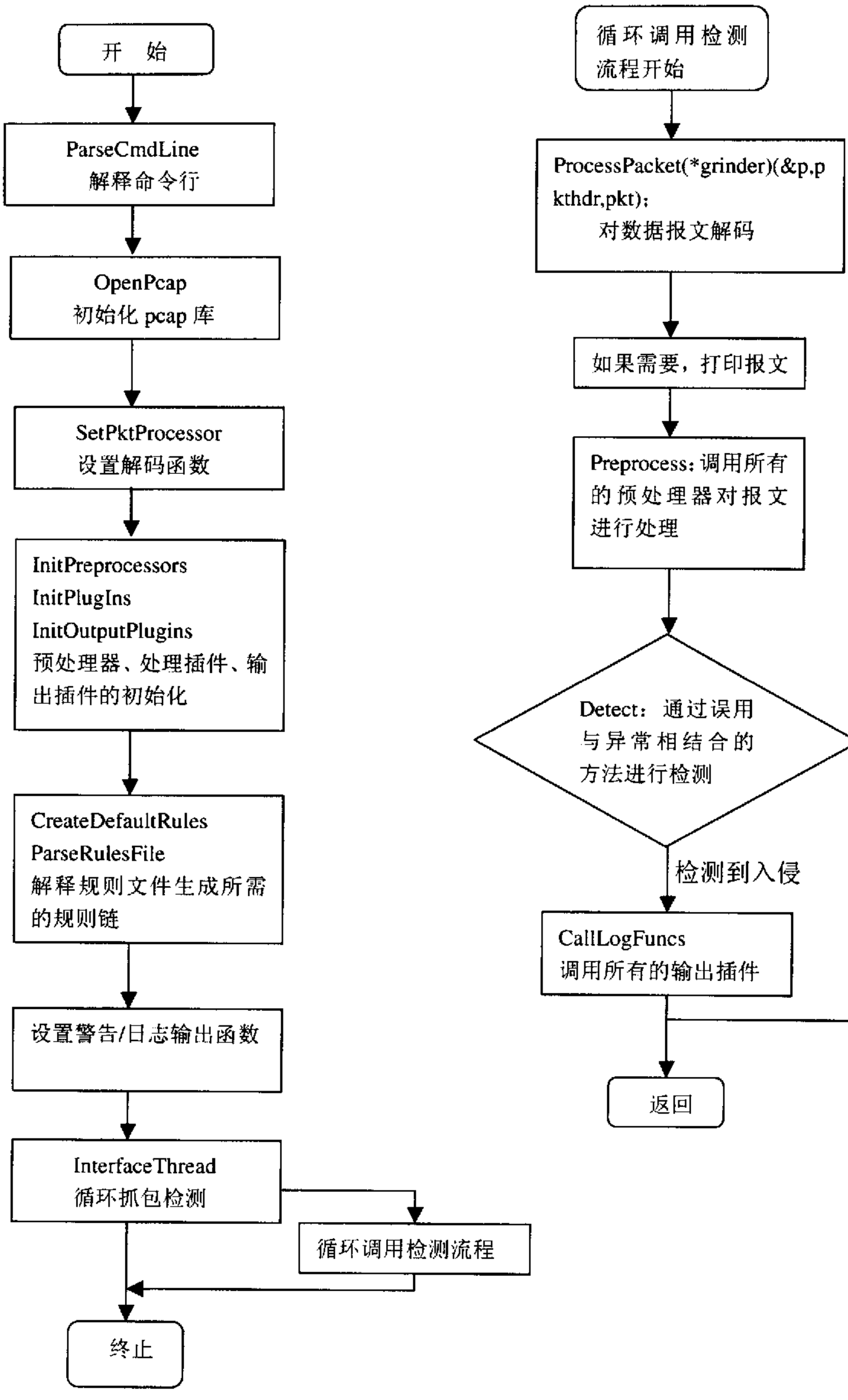



图 2－4 MACIDS 的系统流程图

# 第三章 误用与异常相结合的入侵检测系统的设计

## 3.1 对模式匹配检测架构的改进

改进前的模式匹配检测架构，如图 3-1 所示

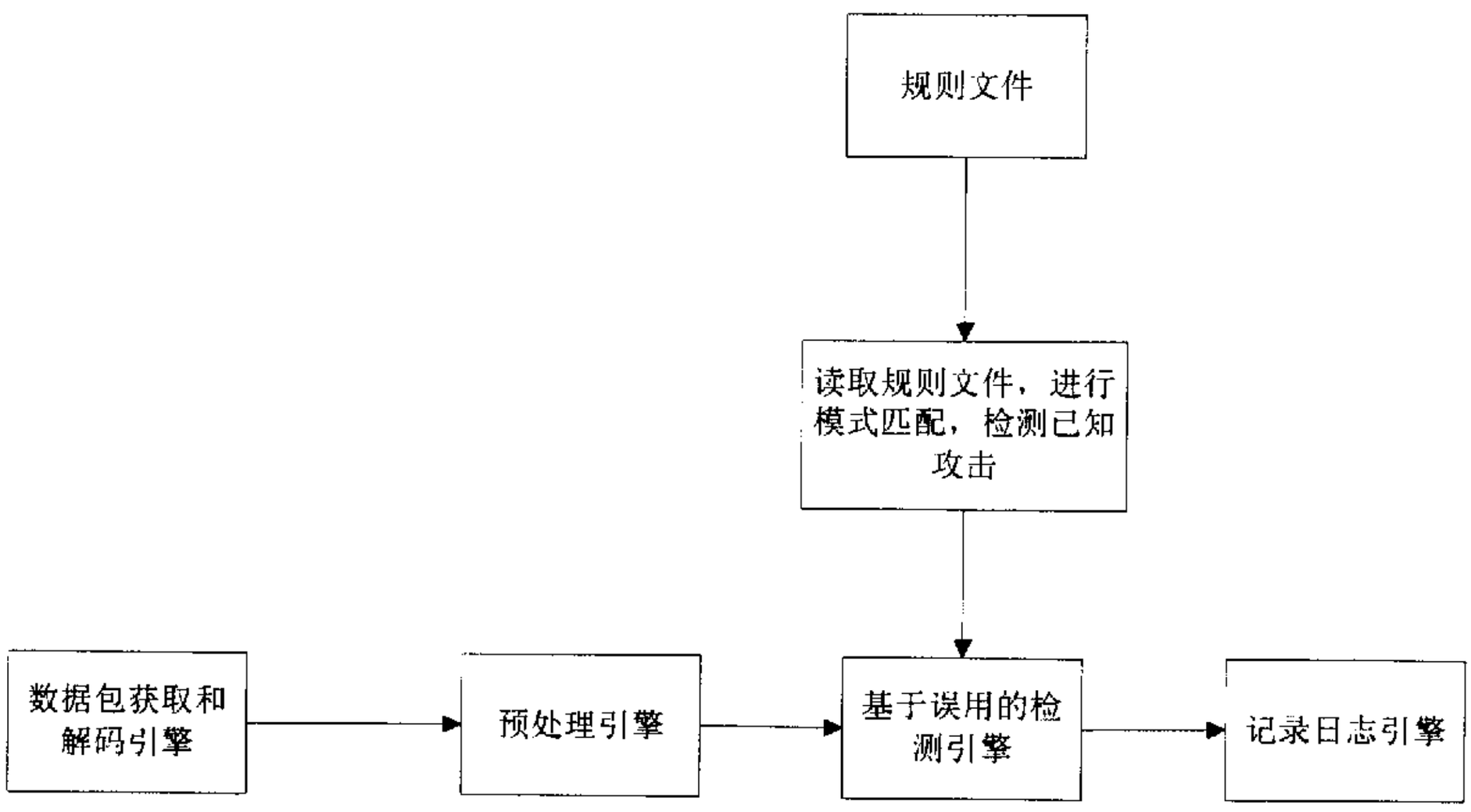



图 3－1 改进以前基于模式匹配的检测架构

本文对以上检测架构作改进，创建了基于异常的检测引擎，并将两个引擎有机结合，并发运行，如图 3-2 所示。下面将对各个模块逐一解释。

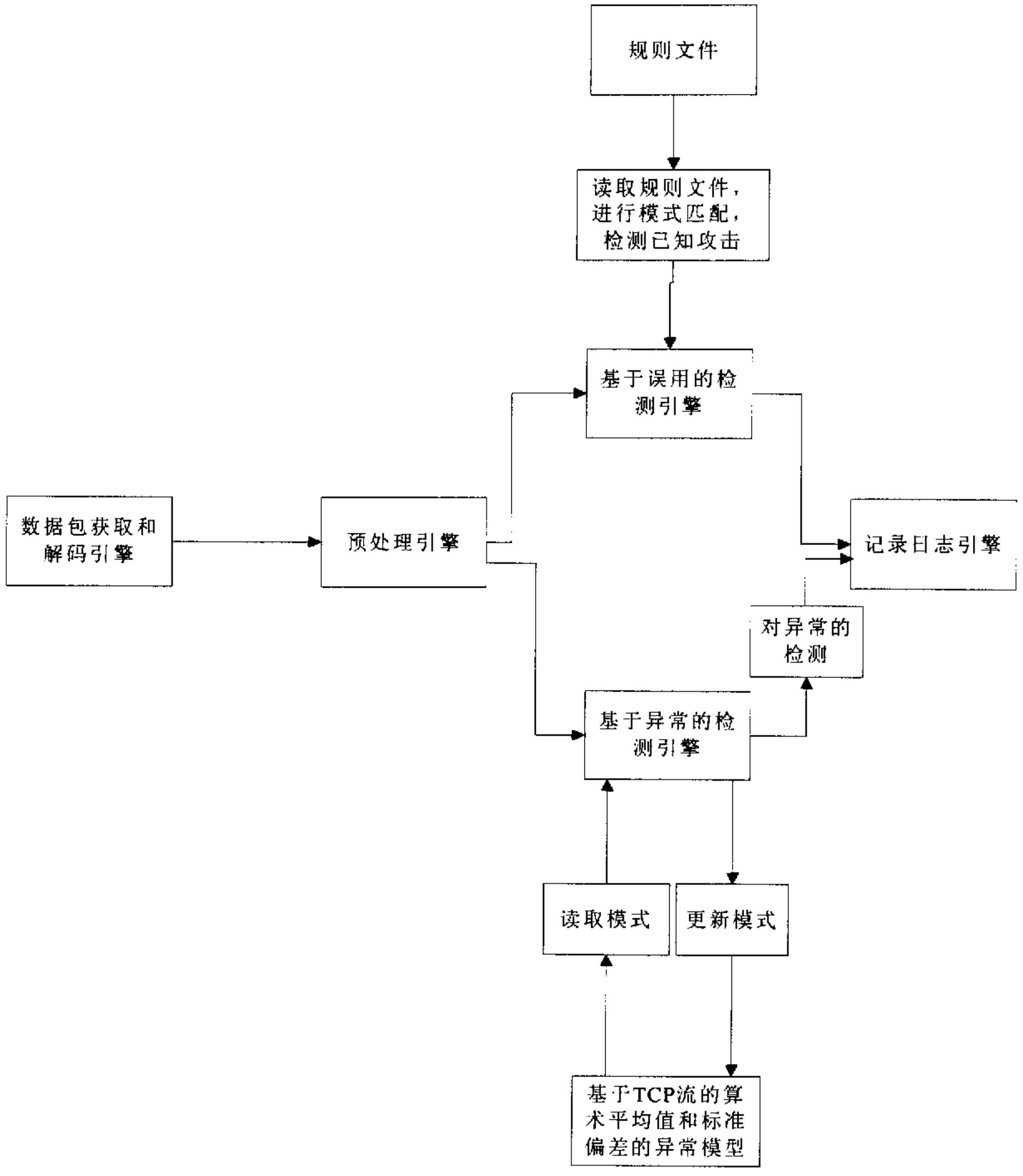



图 3－2 改进后的检测机制

## 3.2　数据包获取和解码引擎

属于这个引擎的所有函数都在文件 decode.c 中。ProcessPacket()是这个引擎的入口点。这个函数是当 IDS 抓到一个数据包时，被 pcap_loop 激活的，pcap_loop 是 libpcap 库函数。ProcessPacket()得到两个指针，一个指向 libpcap 的 pcap_pkthdr 结构，该结构包括所包括的数据包的各种信息，如数据包的总长度，数据包的时间戳

等：另一个指向 u_char 类型的数组，即抓获的数据包本身。从这里开始，我们填充 Packet 结构，该结构在 decode.h 的头文件中被定义如下：

```
typedef struct _Packet
{
    struct pcap_pkthdr *pkth;      /* BPF data */
    u_int8_t *pkt;                 /* base pointer to the raw packet data */

    Fddi_hdr *fddihdr;             /* FDDI support headers */
    Fddi_llc_saps *fddisaps;
    Fddi_llc_sna *fddisna;
    Fddi_llc_iparp *fddiiparp;
    Fddi_llc_other *fddiother;

    Trh_hdr *trh;                  /* Token Ring support headers */
    Trh_llc *trhllc;
    Trh_mr *trhmr;
    SLLHdr *sllh;                  /* Linux cooked sockets header */
    PflogHdr *pfh;                 /* OpenBSD pflog interface header */
    EtherHdr *eh;                  /* standard TCP/IP/Ethernet/ARP headers */
    VlanTagHdr *vh;
    EthLlc     *ehllc;
    EthLlcOther *ehllcother;

    WifiHdr *wifih;                /* wireless LAN header */

    EtherARP *ah;

    EtherEapol *eplh;              /* 802.1x EAPOL header */
    EAPHdr *eaph;
    u_int8_t *eaptype;
    EapolKey *eapolk;
    IPHdr *iph, *orig_iph;    /* and orig. headers for ICMP_*_UNREACH family */
    u_int32_t ip_options_len;
    u_int8_t *ip_options_data;
```

```
TCPHdr *tcph, *orig_tcph;
u_int32_t tcp_options_len;
u_int8_t *tcp_options_data;
UDPHdr *udph, *orig_udph;
ICMPHdr *icmph, *orig_icmph;
echoext *ext;           /* ICMP echo extension struct */

u_int8_t *data;         /* packet payload pointer */
u_int16_t dsize;            /* packet payload size */
u_int16_t alt_dsize; /* the dsize of a packet before munging
                                (used for log)*/

u_int8_t frag_flag;         /* flag to indicate a fragmented packet */
u_int16_t frag_offset;    /* fragment offset number */
u_int8_t mf;                 /* more fragments flag */
u_int8_t df;                /* don't fragment flag */
u_int8_t rf;                        /* IP reserved bit */

u_int16_t sp;           /* source port (TCP/UDP) */
u_int16_t dp;           /* dest port (TCP/UDP) */
u_int16_t orig_sp;          /* source port (TCP/UDP) of original datagram */
u_int16_t orig_dp;          /* dest port (TCP/UDP) of original datagram */
u_int32_t caplen;

u_int8_t uri_count;     /* number of URIs in this packet */

void *ssnptr;           /* for tcp session tracking info... */
void *state;            /* for conversation info */

Options ip_options[40]; /* ip options decode structure */
u_int32_t ip_option_count;    /* number of options in this packet */
u_char ip_lastopt_bad;    /* flag to indicate that option decoding was
                                halted due to a bad option */
Options tcp_options[TCP_OPTLENMAX];       /* tcp options decode struct */
u_int32_t tcp_option_count;
```

```
    u_char tcp_lastopt_bad;    /* flag to indicate that option decoding was
                                  halted due to a bad option */

    u_int8_t csum_flags;          /* checksum flags */
    u_int32_t packet_flags;       /* special flags for the packet */
    int preprocessors;            /* flags for preprocessors to check */
} Packet;
```

(*grinder)(&p, pkthdr, pkt) 函数的声明调用了数据链路层(Data link layer, DLL)的解码函数。此时，在函数 SetPktProcessor()中初始化 grinder 指针，使其指向一个在 decode.c 中定义的数据链路层函数。

接下来，下面的数据链路层函数立即被定义：

(1) DecodeEthPkt() 定义了 10Mb/s 以太网数据包解码接口；

(2) DecodeNullPkt()定义了 loopback 数据包解码接口；

(3) DcodeTRPPkt()定义了令牌环数据包解码接口；

(4) DcodeFDDIPkt()定义了 FDDI 数据包解码接口；

(5) DecodePppPkt()定义了 ppp 数据包解码接口；

(6) DecodeSlipPkt()定义了 Slip 数据包解码接口；

(7) DecodeRawPkt()当某些平台的 ppp 接口要求返回数据连接的类型时返回值是 DLT_RAW，DecodeI4LrawPkt()，Decode*4LCiscoIPPkt()这两个函数的作用相同。

每个数据包解码器都以相似的方式工作，在 Packet 结构中填充适当的指针，当然要注意，在使用这些指针之前要初始化为 Null，如果指向数据结构的指针为非空，那么 Print2ndLayer()将打印出特定的数据头。

以下的数据结构是为数据链路层的头部定义的[23]。

(1) 令牌环头部：

```
typedef struct _Trh_hdr
{
    u_int8_t ac;              /* access control field */
    u_int8_t fc;              /* frame control field */
    u_int8_t daddr[TR_ALEN];      /* src address */
    u_int8_t saddr[TR_ALEN];      /* dst address */
}           Trh_hdr;
```

(2) 令牌环的逻辑链路控制层（LLC）头部：

```
typedef struct _Trh_llc
{
        u_int8_t dsap;
        u_int8_t ssap;
        u_int8_t protid[3];
        u_int16_t ethertype;
}               Trh_llc;
```

(3) 令牌环的 RIF 头部

令牌环的一种头部输出格式，并不是每个令牌环数据包都采用这种格式。

```
typedef struct _Trh_mr
{
        u_int16_t bcast_len_dir_lf_res; /* broadcast/res/framesize/direction */
        u_int16_t rseg[8];
}               Trh_mr;
```

(4) FDDI 头部

```
typedef struct _Fddi_hdr
{
        u_int8_t fc;                /* frame control field */
        u_int8_t daddr[FDDI_ALEN];  /* src address */
        u_int8_t saddr[FDDI_ALEN];  /* dst address */
}               Fddi_hdr;
```

(5) FDDI 的 LLC 头部

```
typedef struct _Fddi_llc_saps
{
        u_int8_t dsap;
        u_int8_t ssap;
}                       Fddi_llc_saps;
```

(6) FDDI 的 SNA 头部

```
typedef struct _Fddi_llc_sna
{
        u_int8_t ctrl_fld[2];
```

```
}                Fddi_llc_sna;
```

(7) 10m 以太网的数据包头部

```
typedef struct _EtherHdr
{
    u_int8_t ether_dst[6];
    u_int8_t ether_src[6];
    u_int16_t ether_type;

}            EtherHdr;
```

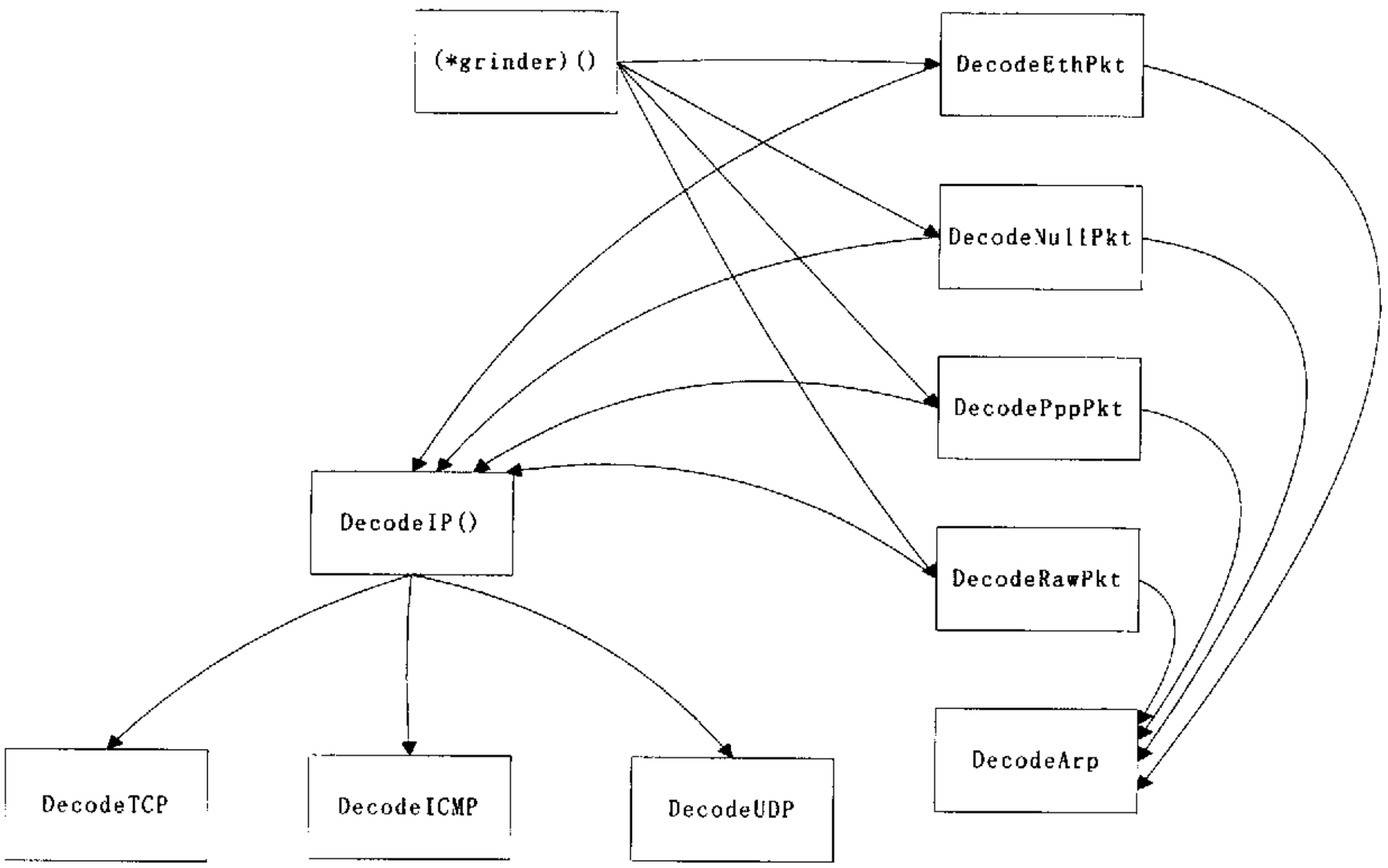



图 3-3 不同数据包解码器之间的关系

当函数 DecodeIP()被激活的时候，需要三个参数：IP 数据包长度、指向 Packet 结构的指针和指向数据包本身的指针。数据包最小值检查（IP 数据包的长度不应小于最小的 IP 头部的长度，IP 版本号要合适）在数据包被处理之前完成。如果 IP 包头的长度大于 20 个字节，说明接受到的 IP 包包含 IP 选项，那么将激活 DecodeIPOption()函数。在这之后，IP 头部的剩余部分被填充，如果碎片偏移量或

者 MF 标志位不是 0，那么 frag_flag 标志位被置 1。

IP 头结构（iph）是在头文件 decode.h 中被定义的，详细如下：

```
typedef struct _IPHdr
{
    u_int8_t ip_verhl;        /* version & header length */
    u_int8_t ip_tos;          /* type of service */
    u_int16_t ip_len;         /* datagram length */
    u_int16_t ip_id;          /* identification  */
    u_int16_t ip_off;         /* fragment offset */
    u_int8_t ip_ttl;          /* time to live field */
    u_int8_t ip_proto;        /* datagram protocol */
    u_int16_t ip_csum;        /* checksum */
    struct in_addr ip_src;   /* source IP */
    struct in_addr ip_dst;   /* dest IP */
}       IPHdr;
```

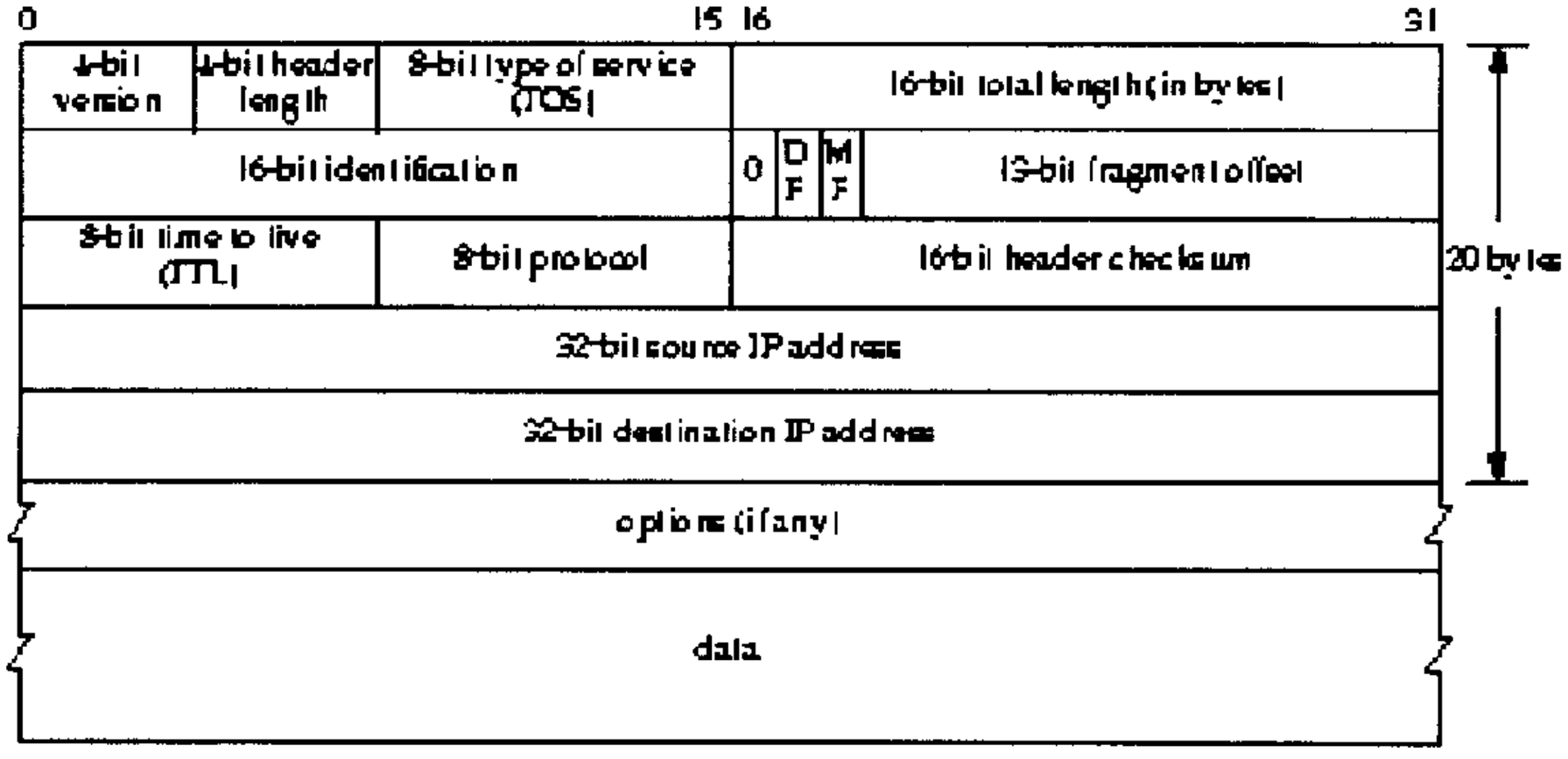



图 3－4　IP 数据包头部

当 IP 解析完成时，在 DecodeIP()函数中使用 ip_proto 变量来转向传输层的解码函数(TLD 函数)。TCP，UDP 和 ICMP 的 TLD 解码函数是在这时被定义的。如果该数据包的协议和上述三种协议都不匹配，那么数据包的数据指针指向 IP 数据包的负载部分，数据包的解码引擎结束并返回 ProcessPacket()函数。否则一个 TLD 函数搜

索整个数据包，填充相应的域到 Packet 结构并返回。

相应的数据类型在头文件 decode.h 中定义如下：

TCP 头部：

```
typedef struct _TCPHdr
{
    u_int16_t th_sport;        /* source port */
    u_int16_t th_dport;        /* destination port */
    u_int32_t th_seq;          /* sequence number */
    u_int32_t th_ack;          /* acknowledgement number */
    u_int8_t th_offx2;         /* offset and reserved */
    u_int8_t th_flags;
    u_int16_t th_win;          /* window */
    u_int16_t th_sum;          /* checksum */
    u_int16_t th_urp;          /* urgent pointer */

}       TCPHdr;
```

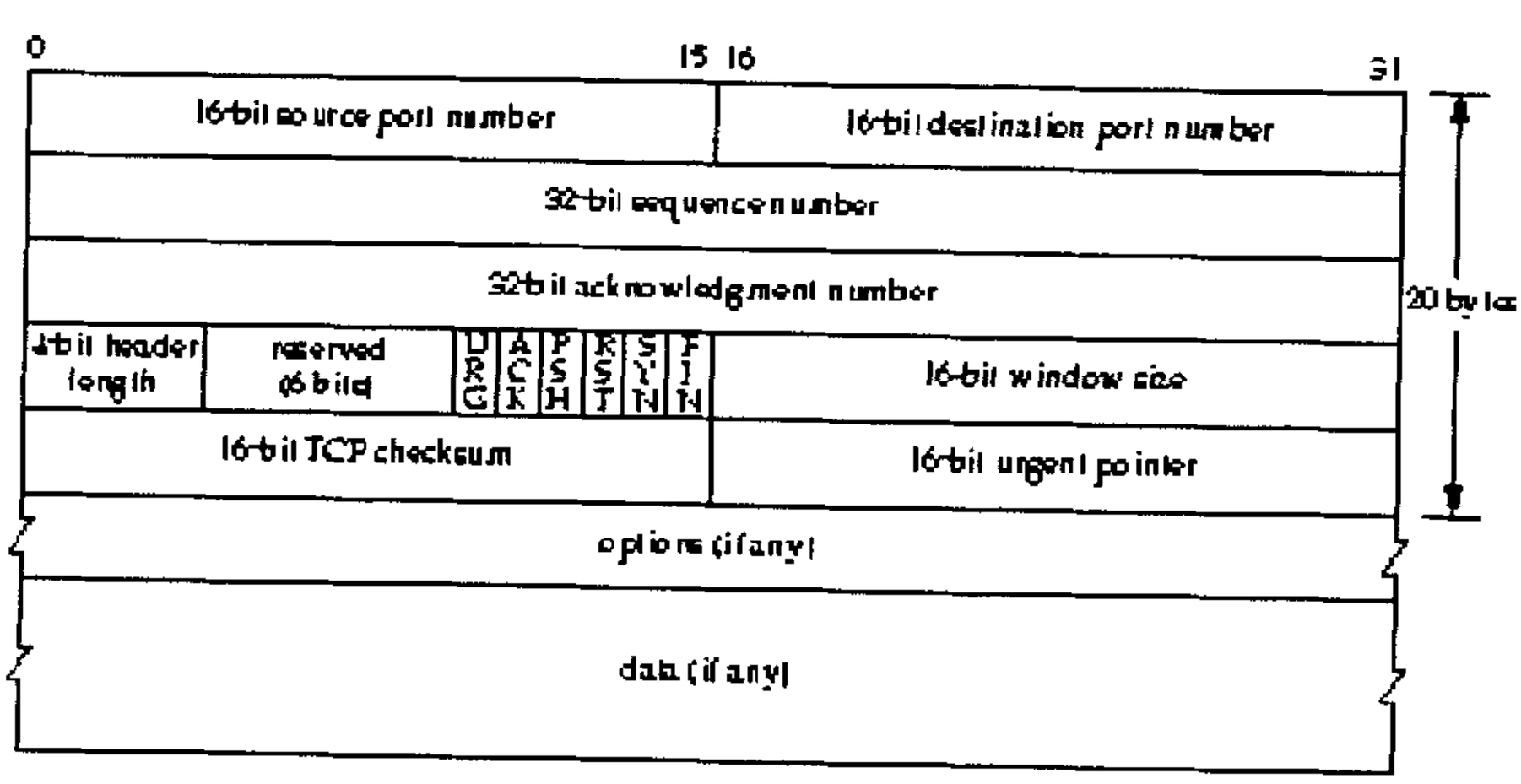



图 3－5 TCP 数据包头部

UDP 头部：

```
typedef struct _UDPHdr
{
    u_int16_t uh_sport;
    u_int16_t uh_dport;
```

```
    u_int16_t uh_len;
    u_int16_t uh_chk;

}       UDPHdr;
```

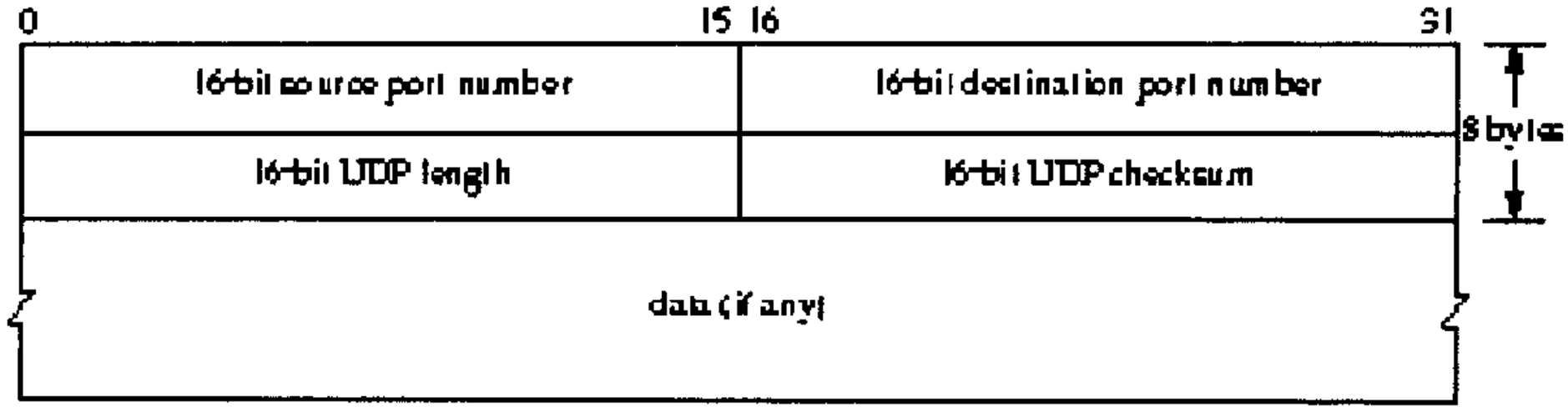



图 3－6　UDP 数据包头部

ICMP 头部：

```
typedef struct _ICMPHdr
{
    u_int8_t type;
    u_int8_t code;
    u_int16_t csum;
}
```

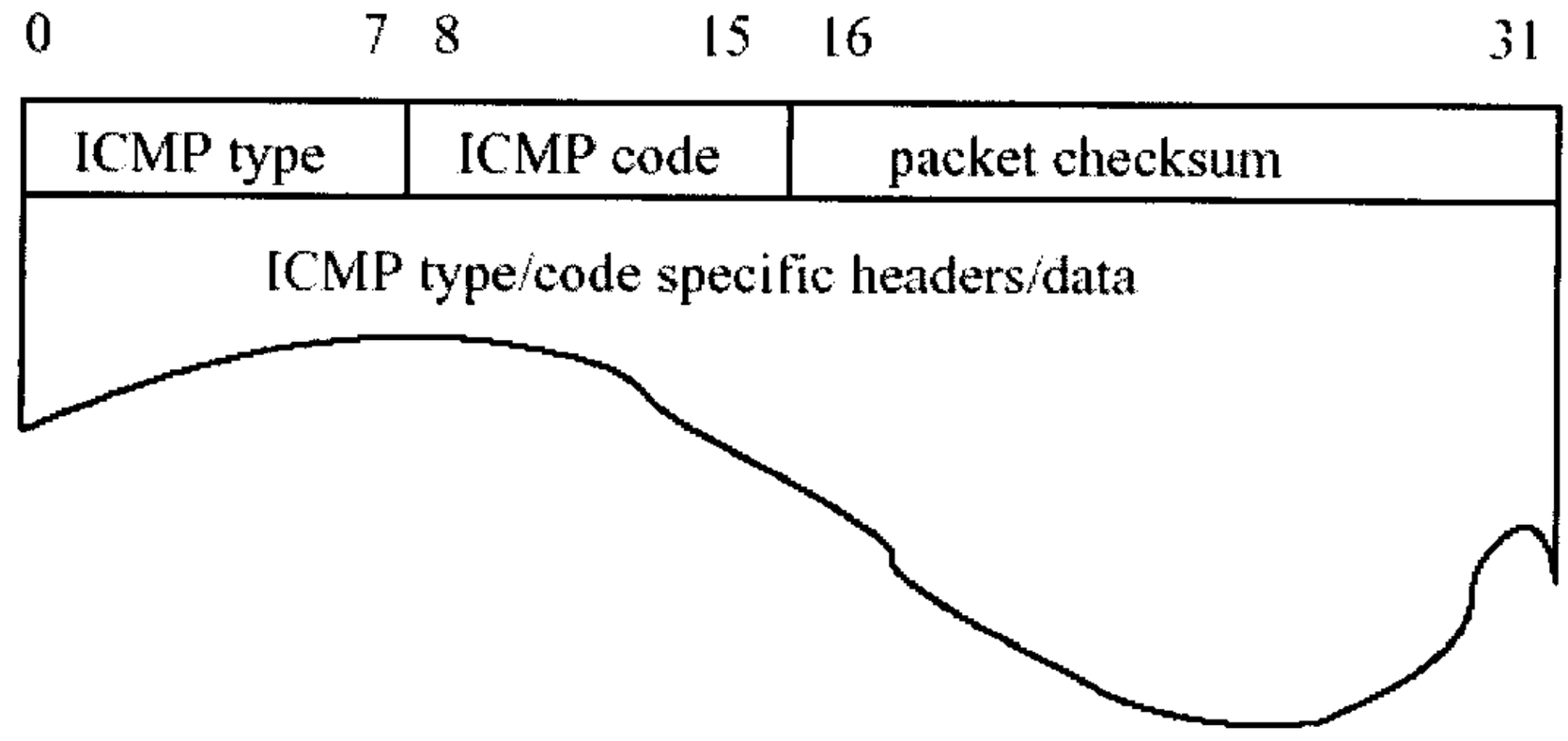



图 3－7　ICMP 数据包头部

## 3.3　预处理引擎

这个引擎模块设计于 plugbase.c 这个文件中，所有的文件名为 spp*的都是该引擎的一部分。

该引擎还设计了三个函数（InitPlugins()，InitPreprocessors()，InitOutputPlugins()），这些函数主要负责不同插件和预处理器的初始化工作，还设计了几个注册函数（RegisterPlugin()，RegisterPreprocessor()，RegisterOutputPlugin()），这些函数是在模型初始化时被调用的，此外还设计了函数（AddFucntionToRestartList()，AddFucntionToCleanExitList()，AddFucntionToSignalList()）负责帮助插件进行垃圾的回收和退出。

## 3.4　基于误用的检测引擎

### 3.4.1　IDS 的规则的概述

alert tcp any any → any any (content : "|0000 0101 EFFF|" ; msg:" Searching for Garbage! ";)

图 3—8 入侵检测的规则

IDS 的规则包括两部分：规则头和规则选项。规则头是到第一个括号之前的全部内容。规则选项是括号内的全部内容。规则头可以映射到 RTN（Rule Tree Node），规则选项可以松散地映射到 OTN(s)(Optional Tree Node)。

在本系统中有 35 个关键字可以应用到规则选项中，其中的 20 个可以用在 OTN 中。在这 20 个当中，17 个是布尔型的（其值是真或者假，例如等于或者不等于）值或者大于或者小于某个值。IDS 的引擎可以以很小的开销将这些值在连接链表中传递。多数的计算开销来自于使用以下三个关键字：内容（content），内容链表（content－list），uri 内容（uricontent）。这当中的每个关键字调用模式匹配引擎来针对具体的模式（pattern）解析数据包的数据部分。由于模式匹配存在开销，这是检测规则选项的最后一部分。在 1270 条规则当中，1086 条或者包括“内容（content）”或者包括“uri 内容（uricontent）”的关键字。

### 3.4.2　规则解析和检测引擎

当 IDS 初始化和解析规则时，它为 TCP, UDP, ICMP, IP 创建了互相分离的规则

链表。在每个规则链表当中有一个互相分离的三维连接链表，这三维指：RTNs（一维）、OTN（二维）和函数指针（三维）。RTNs 中将包括 IP 地址和端口号信息。

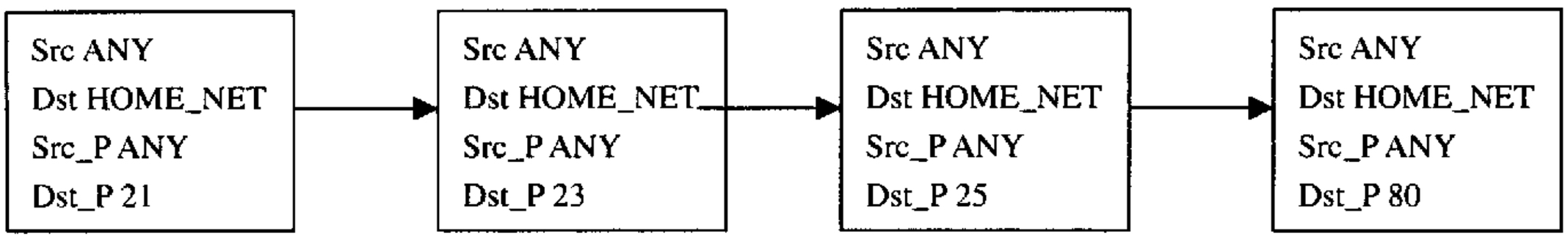



图3-9 一个连接链表头的实例（RTN）

当 IDS 通过检测引擎传送一个数据包时它首先检查当前数据包的 ip 层协议是什么以确定它应该送到哪个规则链表中。一旦数据包发送到正确的规则链表中，它将被赋值，并按照 RTN 中的选项进行对比，从左到右的顺序，直到匹配成功。当检查 RTNs 时，IDS 将首先检查 ip 地址然后是端口号，如果必要的话。如果某个 RTN 被发现与当前数据包相匹配，那么该数据包将沿着该 OTN 一个一个地查找看是否能够找到一个模式 匹配成功。每个 OTN 并不检查每个可用的选项，因为那样将浪费资源去检查根本不存在的内容，如对一个无内容（non-content）的数据包检查内容（content）。每个 OTN 有一个函数指针（第三维）的连接链表来实施需要进行的检测。

该引擎可以大体上被分为两个主要模型：规则解析器和检测引擎。规则解析器在初始化阶段被激活，该模型的入口点是 ParseRuleFile()函数，它接收函数名和深度层次计数器作为参数，没有返回值。该函数逐行读取规则文件，跳过注释（以字符‘#’和‘;’开头的行），跳过空白行，逐一删除开头的空格和需要跳过的行，将剩余的部分传送到 ParseRule()函数。如果需要，该函数将变量扩展，并通过将规则转换成内部表示形式、处理文件包含的伪指令以及变量定义等方式逐行处理指令。有五个独立的规则链：Alert，Log，Pass，Activation 和 Dynamic，每个规则链有三个互相独立的连接清单，保持了 TCP，UDP 和 ICMP 的规则集。以上结构主要是基于性能方面的考虑。规则有两种处理顺序：按照协议来处理或者按照 Pass、Alert，Log 的顺序。前一种顺序可能使性能降低但在某些情况下是必要的。每个规则内部都与 RuleTreeNode 结构关联，这个结构在 rules.h 文件中定义如下：

```
typedef struct _RuleTreeNode
{
```

```
RuleFpList *rule_func; /* match functions.. (Bidirectional etc.. ) */

int head_node_number;

int type;

IpAddrSet *sip;
IpAddrSet *dip;

int not_sp_flag;        /* not source port flag */

u_short hsp;             /* hi src port */
u_short lsp;             /* lo src port */

int not_dp_flag;        /* not dest port flag */

u_short hdp;             /* hi dest port */
u_short ldp;             /* lo dest port */

u_int32_t flags;        /* control flags */

/* stuff for dynamic rules activation/deactivation */
int active_flag;
int activation_counter;
int countdown;
ActivateList *activate_list;

struct _RuleTreeNode *right;    /* ptr to the next RTN in the list */

    OptTreeNode *down;      /* list of rule options to associate with this
                                rule node */
    struct _ListHead *listhead;

} RuleTreeNode;
```

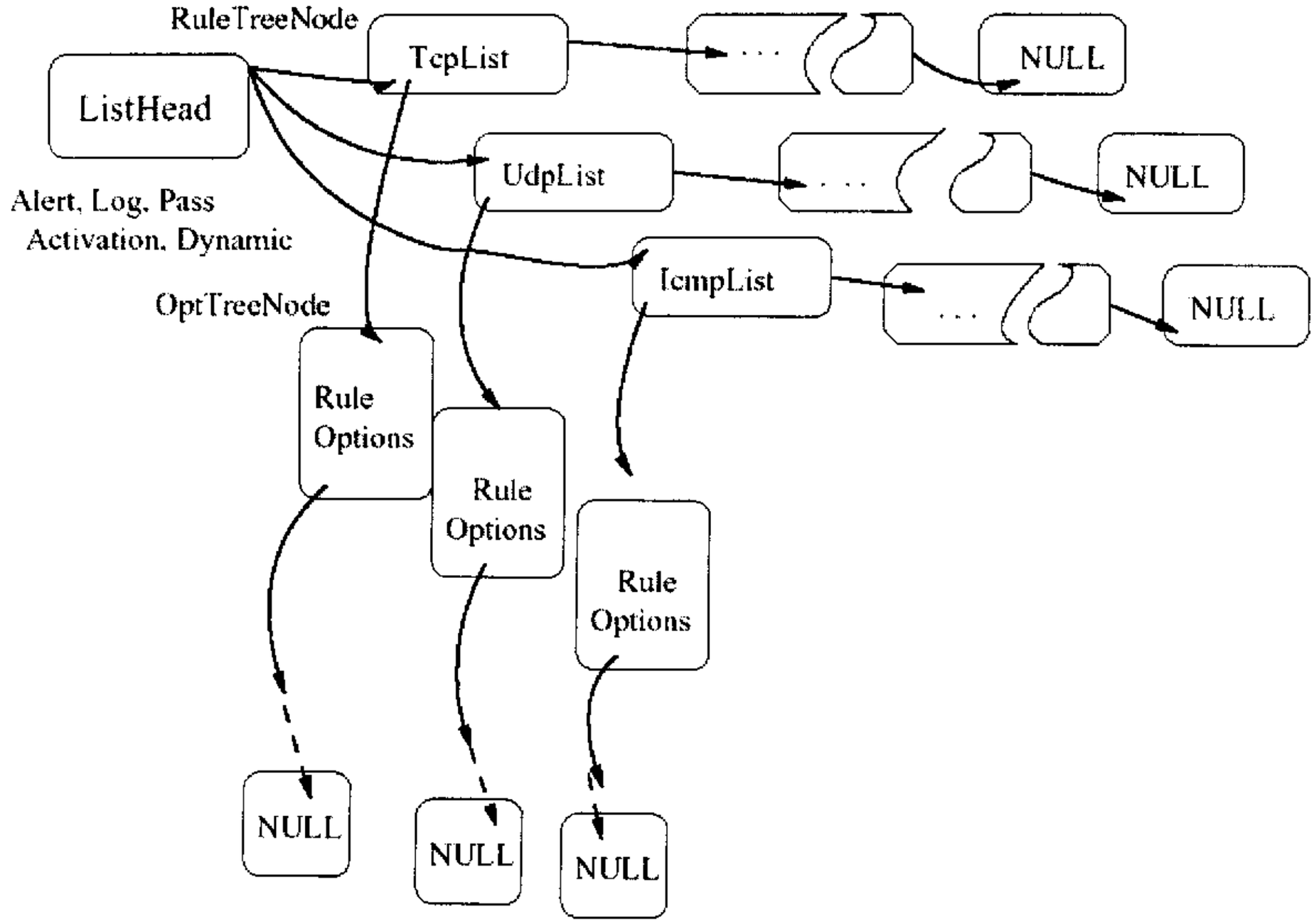



图 3-10 规则在存储器中的内部表示

当一行规则被解释成内部形式时，函数 ProcessHeadNode()被调用。这个函数遍历一个规则链表，并将规则追加到链表的末尾。RuleFpList 是规则头检测函数的链表，每个 RuleTreeNode 的变量包含一个指向 RuleFpList 链表的指针、一个指向下一个规则的指针、一个指向规则选项的指针。

还有一些头部的检测规则，其中主要是对 IP 地址的匹配（源 IP 相等和不相等，目的 IP 相等和不相等...），对端口号的匹配（源端口相等和不相等，目的端口相等和不相等...）等等。

规则选项也是以相似的机制工作的，一个单一的选项节点以 OptTreeNode 结构来描述的，在 rules.h 文件中定义如下：

```
typedef struct _OptTreeNode
{
    /* plugin/detection functions go here */
    OptFpList *opt_func;
    RspFpList *rsp_func;    /* response functions */
```

```
/* the ds_list is absolutely essential for the plugin system to work,
    it allows the plugin authors to associate "dynamic" data structures
    with the rule system, letting them link anything they can come up
    with to the rules list */
void *ds_list[64];      /* list of plugin data struct pointers */

int chain_node_number;

int type;                /* what do we do when we match this rule */
int evalIndex;           /* where this rule sits in the evaluation sets */

int proto;               /* protocol, added for integrity checks
                                    during rule parsing */
struct _RuleTreeNode *proto_node; /* ptr to head part... */
int session_flag;        /* record session data */

char *logto;             /* log file in which to write packets which
                                    match this rule*/
/* metadata about signature */
SigInfo sigInfo;

u_int8_t stateless;    /* this rule can fire regardless of session state */
u_int8_t established; /* this rule can only fire if it has been marked
                                    as established */
Event event_data;

TagData *tag;

/* stuff for dynamic rules activation/deactivation */
int active_flag;
int activation_counter;
int countdown;
int activates;
int activated_by;
```

```
u_int8_t    threshold_type; /* type of threshold we're watching */
u_int32_t threshold;       /* number of events between alerts */
u_int32_t window;          /* number of seconds before threshold times out */
struct _OptTreeNode *OTN_activation_ptr;
struct _RuleTreeNode *RTN_activation_ptr;
struct _OptTreeNode *next;
    struct _RuleTreeNode *rtn;
} OptTreeNode;
```

还有指向 opt_func 的指针，opt_func 中包含选项检测引擎和参数声明，opt_func 可以是指向插件函数的指针或者是指向内部声明的检测函数的指针。

检测模型以 Preprocess()函数开始，它接受到一个指向 Packet 结构变量的指针作为参数。它激活了 PreprocessList 链表中所有的预处理器函数，并转移到 Detect()函数。Detect()函数将规则逐一应用到当前数据包，使用 EvalPacket()函数并按需要激活输出报警或者日志函数。检测引擎从“左”到“右”遍历规则，从“上”到“下”遍历规则选项，如图 3-10 所示。

## 3.5　基于异常的检测引擎的设计

异常检测引擎的作用是建立一个统计分析模型，来描述服务器和多个客户端之间的流量。这个模型仅仅需要对每一段时间内数据包 TCP 标志位保存并进行一些计算。当检测到流量和该模型设计的流量不符合之后，将向控制台发送一条检测到异常的消息。

这个工作的一个实例是对 SYN 泛洪攻击[24]的检测。SYN 泛洪是一个古老的但曾经很流行的拒绝服务（DoS）攻击。当系统 1 向系统 2 的特定端口进行 TCP 连接的时候，首先要进行三次握手的过程。该过程主要包括系统 1 向系统 2 先发送一个 SYN（synchronize，同步）标志位置 1，ACK（acknowledge，回复）标志位置 0 的数据包；之后系统 2 向系统 1 回答一个 SYN 和 ACK 标志位都置 1 的数据包；然后，系统 1 向系统 2 发送另外一个 SYN 标志位为 0 和 ACK 标志位被置 1 的数据包，这样，该连接正式建立起来。如果由于某种原因最后一步没能够成功，系统 2 通常会等待数分钟才放弃连接。多数的操作系统只能保持很少的连接处于等待状态。正常情况下，三次握手会很快完成，等待队列同时等待的连接个数不会超过十。然而，入侵者发送了一些 SYN 标志位置 1 的数据包，但将源 IP 隐藏为一个系统 1 不可达的系统，那么系统 1 永远也不会得到三次握手的终止数据包（因为系统 1 根本不存在），系统 2 的连接队列将被塞满，使得系统 2 的端口不可达，如果这种保持下去，

那么攻击者只要每过几分钟就向系统 2 发送几个伪造的数据包，就完全有可能使用一个仅 14.4k 的 modem 就让几台服务器死机。然而，在一个 SYN 泛洪攻击下，我们分别统计数据包中 tcp 和 ack 标志位的算术和，并求其比值，这个值会落在我们建立的置信区间之外，此时，我们将产生报警。

IDS 的异常检测部分对发送大量数据包的攻击的检测是非常有效的。它还可以检测误用检测引擎检测不到的新的类型的攻击。

## 3.6　记录日志引擎

该引擎主要设计于 log.c 文件中，输出插件可以看作该引擎的一部分。

引擎的入口点有两个：CallAlertPlugins()和 CallLogPlugins()函数（见图 3－10）。指针 AlertFunc 和 LogFunc 分别指向这两个函数。CallAlertPlugins()和 CallLogPlugins()函数可以激活每个规则链的告警/日志工具。

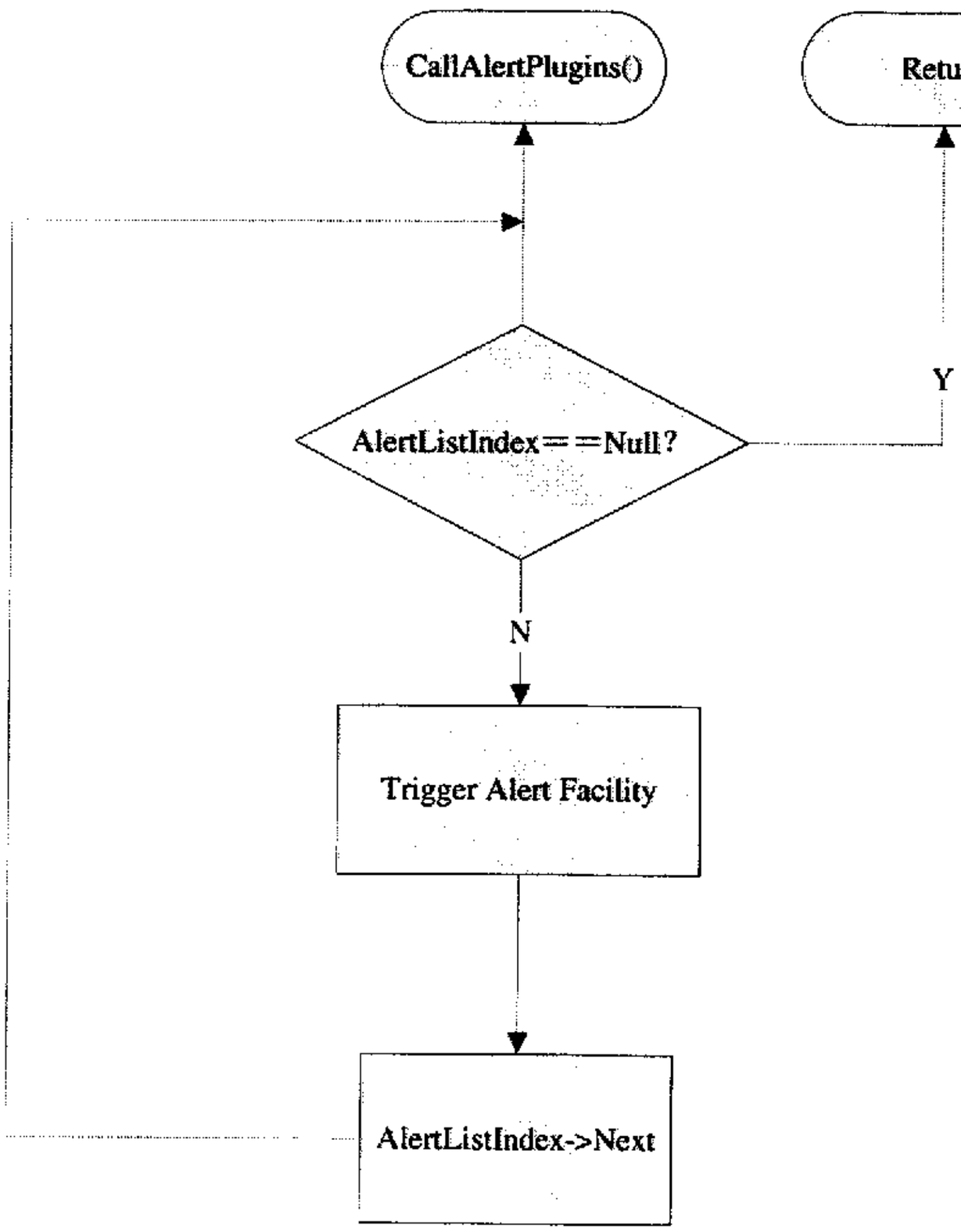



图 3-11 函数 CallAlertPlugins()的流程图

还有一些函数负责将数据包记录日志。所有这些函数都是被规则处理和检测引擎或者是数据包解码引擎激活的，并取决于以下一些参数，规则和规则选项的类型、命令行参数和输出插件设置等。

也有一些函数处理不同的报警类型：快速报警、全面报警、空报警以及 Unix 套接字报警。这些函数目前被认为是过时的并已经用一些输出插件函数来取代。但它们如今仍然被保存在 log.c 文件中。

释放不同格式的数据也采用不同的函数，这类函数的详细情况在 log.c 中都有说明。

# 第四章 MACIDS 的检测引擎剖析

## 4.1 基于模式匹配的误用检测机制

### 4.1.1 入侵检测的模式匹配历史简介

在早期，人们使用近乎暴力的模式匹配算法，这使得匹配非常慢，这部分的性能有很大的提升空间。暴力的模式匹配的改进首先是通过部分 Boyer-Moore 算法[25]实现的。几个月之后得以实现完全的 Boyer-Moore 算法。下一步又实现了一个“二维链表”，这使得基于模式匹配的 IDS 的性能提高了 200%到 500%。然后，IDS 的开发者们重新改写了检测引擎，使它包括了一个“函数指针的链表”，也称为一个“三维的连接链表”。

### 4.1.2 MACIDS 采用的模式匹配算法

当需要对数据包的有效负载部分进行内容的模式匹配的时候，MACIDS 使用 Boyer-Moore 算法。这个模式匹配算法是最有效的字符匹配算法，并经常用在文字编辑器的“搜索”和“替换”操作。

Boyer-Moore 算法综述

在介绍基础之前，先看看以下的术语：

(1) 要匹配的模式记为 P

(2) 要匹配的文本（网络有效负载）记为 T

(3) 模式（P）的长度记为 LP

(4) 文本（T）的长度记为 LT

(5) 模式（P）的第一和最后一个字符分别被记为 $P_1$ 和 $P_{LP}$

(6) 文本（T）的第一和最后一个字符分别被记为 $T_1$ 和 $T_{LT}$

(7) 当最初 P 和 T 匹配时，P 中的最后一个字符 $P_{LP}$ 也要和 $T_{LT}$ 相匹配。

要找的模式：EXAMPLE

文本内容：HERE IS A SIMPLE EXAMPLE

简单的模式匹配算法是按下面这样做的：

(1) 将 P 的左端和 T 的左端对齐如图 4-1 所示：

$P_1$　　$P_{LP}$

**EXAMPLE**

**HERE IS A SIMPLE EXAMPLE**

$T_1$　　$T_{LP}$　　$T_{LT}$

图 4-1 简单模式匹配的左端对齐图示

(2) 在 T 中从左到右对 P 进行匹配，直到发现有不匹配的字符或者 P 已经耗尽。在上图的情况下 $P_1$=E，$T_1$=H，并且 E！=H。

(3) 如果发现某个字符不匹配，P 向右移动一位并再次进行匹配（图 4-5）。这一次 $P_1$=E 和 $T_2$=E 对齐，由于“E”=“E”，那么继续检查 $P_2$=“X“和 $T_2$=“R“。而”X“！=”R“，将模式 P 右移一位之后从新开始匹配。

$P_1$　　$P_{LP}$

**EXAMPLE**

**HERE IS A SIMPLE EXAMPLE**

$T_2$　　$T_{LT}$

图 4-2 简单模式匹配的移动一位后的图示

(4) 上述过程将一直进行下去，直到在 T 中发现完全匹配的 P 或者 $P_{LP}$ 移到最后并超过了 $T_{LT}$。

在上述过程中，该简单算法为了发现该模式进行了 28 次尝试。这是一种暴力的模式匹配算法，将消耗很长时间来确定一个模式或者测定该段文字中没有这个模式。这和 IDS 刚开始的模式匹配技术相似。

Boyer-Moore 算法包含三个简单算法所不具有的方面，这三方面使得它即使在最坏情况下仍然十分高效。[26]

(1) 从右向左扫描。这和简单算法的从左向右扫描正好相反。P 的左端仍然和 T 的左端对齐，但匹配工作是从 P 的右端开始的，并向左移动直到发生不匹配的情况，如图 4-3 所示。

$P_1$　　$P_{LP}$

EXAMPLE
HERE IS A SIMPLE EXAMPLE
$T_I$ $T_{LP}$ $T_{LT}$

图 4-3 左端对齐图示

(2) 坏字符移动。第一步匹配工作是 $P_{LP}$＝“E”并且 $T_{LT}$＝“S”，如图 4-4 所示。就象很多先进的模式匹配算法一样 Boyer-Moore 算法对模式进行预处理并获得启发式信息。它将用这些信息来计算要将 P 向右移动几位。该算法必须决定在 T 中可以和 P 的最右端的字符匹配的字符。在这个例子中是字符“E”，如图 4-4 所示。

$P_I$ $P_{LP}$
EXAMPLE
HERE IS A SIMPLE EXAMPLE
$T_I$ $T_{LP}$ $T_{LP+9}$ $T_{LT}$

图 4-4 Boyer-Moore 算法移动图示

(3) 好后缀移动。一旦进行这样的匹配，算法将首先从 P 的右端开始匹配。这时候 $P_{LP}$＝“E”将和 $T_{LT+9}$＝“E”，如图 4-4 所示。接下来它将匹配 $P_{LP-1}$＝“L”和 $T_{LT+8}$＝“L”。匹配成功继续 $P_{LP-2}$＝“P”和 $T_{LT+7}$＝“P”。又成功了，我们继续匹配 $P_{LP-3}$＝“M”和 $T_{LT+6}$＝“M”。由于又成功了，我们接着匹配 $P_{LP-4}$＝“A”和 $T_{LT+5}$＝“I”。这时发生了不匹配的情况。由于对模式的预处理，发现了 T 中包含字符串“MPLE”，那么将在 T 中寻找下一次出现该字符串的情况。然后把 P 向右移动并将 P 当中的“MPLE”和在 T 中下一次出现的“MPLE”对齐，如图 4-5 所示。

$P_I$ $P_{LP}$
EXAMPLE
HERE IS A SIMPLE EXAMPLE
$T_I$ $T_{LP}$ $T_{LT}$

图 4-5 右端对齐图示

(4) 一旦这个移动结束，字符串地匹配将从 P 的右端重新开始。这种情况下，P 的每个字符都从 $P_{LP}$＝“E”开始匹配，当匹配到 $T_{LT}$＝“E”时结束。

采用 Boyer—Moore 算法，上述查找过程只需要 12 步就成功匹配了。这比暴力方法比高了两倍多。

### 4.1.3 协议分析和模式匹配相结合的检测方式

模式匹配具有分析速度快、误报率小等优点是其它分析方法不可比拟的。单纯使用模式匹配的方法有很大的弊端，因此 snort 采用协议分析和模式匹配结合的方法来分析网络数据包。单纯的模式匹配方法的工作过程如下：

(1) 分析网络上的每一个数据包是否具有某种攻击特征。分析如下：

(2) 从网络数据包的包头开始和攻击特征比较；

(3) 如果比较结果相同，则检测到一个可能的攻击；

(4) 如果比较结果不同，从网络数据包中下一个位置重新开始比较。

(5) 直到检测到攻击或网络数据包中的所有字节匹配完毕，一个攻击特征匹配结束。

(6) 对于每一个攻击特征，重复步骤2开始的比较。

(7) 直到每一个攻击特征匹配完毕，对给定数据包的匹配完毕。

下面给出一个例子可以很形象地说明其工作原理。

以下的数据是监听到的网络数据包（实际上是一个HTTP的GET请求数据包）的前面部分。

```
00e04c90444300e04c939ff908004500018b225a4000800654b2c0a8
0017c0a800f904180050cb798df26ac5e62150184470a8fb000047455420
2f4f412f6b616f2e68746d20485454502f312e310d0a41636365
```

对于攻击模式“GET /cgi-bin/./phf”，首先从数据包头部开始[1]：

```
GET /cgi-bin/./phf---
00e04c90444300e04c939ff908004500018b225a4000800654b2c0a8
0017c0a800f904180050cb798df26ac5e62150184470a8fb00004745
54202f4f412f6b616f2e68746d20485454502f312e310d0a41636365
```

比较不成功，移动一个字节重新比较。

```
-GET /cgi-bin/./phf--
00e04c90444300e04c939ff908004500018b225a4000800654b2c0a8
```

[1]注意：在这里，我们直接用字符串表示以方便阅读，真正的应该是十六进制的字节流。－表示空格位。

0017c0a800f904180050cb798df26ac5e62150184470a8fb00004745

54202f4f412f6b616f2e68746d20485454502f312e310d0a41636365

结果还是不成功，再移动一次。

--GET /cgi-bin/./phf-

00e04c90444300e04c939ff908004500018b225a4000800654b2c0a8

0017c0a800f904180050cb798df26ac5e62150184470a8fb00004745

54202f4f412f6b616f2e68746d20485454502f312e310d0a41636365

重复比较，没有一次匹配成功。

传统模式匹配方法的问题：

计算量大。对于一个特定网络的每秒需要比较的最大次数为：

攻击特征字节数×网络数据包字节数×每秒数据包数量×攻击特征数量

如果所有攻击特征长度为20字节，网络数据包平均长度为30字节，每秒30,000数据包，供给特征库中有4000条特征，那么，每秒比较次数为：

20 x 300 x 30,000 x 4,000 = 720,000,000,000

检测准确性：传统的模式匹配只能检测特定类型的攻击。对攻击特征微小的变形都将使得检测失败。例如，对于 WEB 服务器，

GET /cgi-bin/phf

HEAD /cgi-bin/phf

GET //cgi-bin/phf

GET /cgi-bin/foobar/../phf

GET /cgi-bin/./phf

GET %00/cgi-bin/phf

GET /%63%67%69%2d%62%69%6e/phf

都是合法而且有效的。但以上的匹配方法却不能检测。

传统的模式匹配的检测方法的问题根本是它把网络数据包看作是无序的随意的字节流。他对该网络数据包的内部结构完全不了解。它对于网络中传输的图像或音频流同样进行匹配。可是网络通信协议是一个高度格式化的、具有明确含义和取值的数据流，如果将协议分析和模式匹配方法结合起来[27]，可以获得更好的效率、更精确的结果。

协议分析有效利用了网络协议的层次性和相关协议的知识快速地判断攻击特征是否存在。它的高效使得匹配的计算量大幅度减小。即使在 100M 的网络中，也可以充分地检测每一个数据包。

以下是基于协议分析的入侵检测系统如何处理上面例中的数据包的：

00e04c90444300e04c939ff90800**4500018b225a4000800654b2c0a80017**

**c0a800f904180050cb798df26ac5e62150184470a8fb0000474554202f4f**

412f6b616f2e68746d2048545450 2f312e310d0a41636365

协议规范指出以太网络数据包中第13字节处包含了两个字节的第三层协议标识。基于协议分析的入侵检测系统利用这个知识开始第一步检测：

(1) 跳过前面12个字节，读取13字节处的2字节协议标识：0800。根据协议规范可以判断这个网络数据包是IP包。

(2) IP协议规定IP包的第24字节处有一个1字节的第四层协议标识。因此系统跳过的15到24字节直接读取第四层协议标识：06，这个数据包是TCP协议。

(3) TCP协议在第35字节处有一个2字节的应用层协议标识（端口号）。于是系统跳过第25到36字节直接读取第37字节的端口号：80（十六进制的0050）。该数据包是一个HTTP协议的数据包。

HTTP 协议规定第 55 字节是 URL 开始处，我们要检测供给特征“GET /cgi-bin/./phf”，因此要仔细检测这个 URL。

可以看出，利用协议分析可以大大减小模式匹配的计算量，提高匹配的精确度，减少误报率。

因此，我们的检测算法是协议分析与模式匹配相结合的，这样减少了单纯模式匹配的计算机复杂量。

## 4.2　异常检测引擎的设计

### 4.2.1　异常检测模型的设计

#### 4.2.1.1　滑动窗口机制

本文在单一基于模式匹配的 IDS 中创建了异常检测引擎，使其与原来得模式匹配方法相结合，增强了原来系统的检测能力。

统计模型仿照 TCP 协议的流量控制协议[28]，采用滑动窗口的方式，如下图：

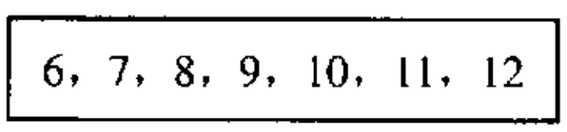



图 4-6 异常检测的滑动窗口机制

以时间段为窗口值，在特定的时间间隔内采样数据包，并将每一样本代入统计模型。

#### 4.2.1.2　异常检测模型的设计

中心极限定律认为，无论研究的统计总体服从什么样的分布，样本的平均值的分布接近一个正态分布，正态分布的均值等于总体分布的均值，标准偏差等于总体分布的标准偏差除以样本大小的平方根。[29]

在接下来进行的讨论当中，我们给定置信度为 $1-\alpha$，并且设 $X_1$，$X_2$，…$X_n$ 为总体 $N(\mu, \sigma^2)$ 的样本，且 $X$，$S^2$ 分别为样本均值和样本方差。

如测量平均吞吐量，平均吞吐量可以用流量随机抽样样本的平均吞吐量来估计。随着不停地测量报文，网络流量的平均吞吐量也随之不断更新。最简化的形式可以表示为：

$$X_{n+1} \leftarrow X_n + x_{n+1}, \qquad (4-1)$$

其中，$X_n$ 是前 n 个单位时间内平均抽样的累加和（$X_0=0$），$x_{n+1}$ 是第 $n+1$ 个单位时间内平均抽样吞吐量，使用 $X_{n+1}$ 代替 $X_n$ 是存储信息，因此新的平均抽样吞吐量 $\overline{X_{n+1}}$ 为：

$$\overline{X_{n+1}} = \frac{X_{n+1}}{n+1}。 \qquad (4-2)$$

标准方差是测量数据的偏差，如果数据离平均值近，则置信区间较窄，对于 n+1 个数据值，样本标准差对总体标准差的无偏估计定义为：

$$S_{n+1} = \sqrt{\frac{[\sum_{i=1}^{n+1}(x_i - \overline{X_{n+1}})^2]}{n}} = \sqrt{\frac{\sum_{i=1}^{n+1} x_i^2 - (n+1)\overline{X^2_{n+1}}}{n}}。 \qquad (4-3)$$

对于每个测度，系统只要维护三个值：样本平均值 $\overline{X_n}$、样本累加和 $X_n$、样本平方累加和 $\overline{\omega_n}$，标准差可以通过以下运算得到：

$$\overline{\omega_n} = \sum_{i=1}^{n} x_i^2,$$

$$\overline{\omega_0} = 0,$$

$\overline{\omega_{n+1}} \leftarrow \overline{\omega_n} + x^2_{n+1}$，则：

$$S_{n+1} = \sqrt{\frac{\overline{\omega_{n+1} - X_{n+1}\,\overline{X_{n+1}}}}{n}} \qquad (4-4)$$

样本均值和样本方差能为流量特性的总体均值构造一个置信区间（如：平均吞吐量）。样本均值的标准差 $S_{\overline{X_n}} = \frac{S_n}{\sqrt{n}}$。如果从相同的流量中重复选择样本，并计算

每个样本的均值，则这个统计量表明期望的变化量。中心极限定理称对于大于 n 的样本，其均值服从均值等于流量总体的均值，标准差为 $S_{\overline{X_n}}$ 的正态分布。因此可以构造置信度为 1－α 的总体均值的置信区间 μ 为：

$$\overline{X_n} - Z_{\alpha/2} * \frac{S_n}{\sqrt{n}} \leqslant \mu \leqslant \overline{X_n} + Z_{\alpha/2} * \frac{S_n}{\sqrt{n}}. \qquad (4-5)$$

其中 $Z_{\alpha/2}$ 可以通过查正态分布表求得，样本中元素数目 n 越大，样本均值的偏差越小，其总体均值的偏差也就越小。如果当前时间范围内测度值满足式（4－5）的要求，说明当前流量行为正常，系统将更新队列数据，如果当前时间范围测度不满足式（4－5）的要求，则说明当前流量行为异常，不断更新历史队列统计数据记录，报告异常错误[30]。

### 4.2.2　流量的实时更新

即使对于稳定的网络行为，随着时间的推移，由于用户行为和网络环境的变化，其正常行为也会发生变化，因此我们只取一段时间内的流量信息来进行检测，而不是采用所有历史测量数据来进行检测[31, 32]。

为了维护一个固定大小的滑动窗口，需要在窗口头部抛弃旧的流量数据，在窗口的尾部增加新到的数据。head 指向滑动窗口的头部，tail 指向滑动窗口的尾部，同时滑动 head 和 tail，使得窗口的数据得到不断的更新。

## 4.3　开发平台

MACIDS 的原型是在 RedHat Linux 9.0（内核版本是 2.4.18-14）下使用 C 语言来开发的。异常检测引擎的加入使用 signal 和 alarm 函数配合[33,34]完成，每隔一段时间抽取一定统计值，并将其带入统计模型，进行分析，异常检测引擎和误用检测引擎在结构上采用并发执行。

# 第五章 MACIDS 的测试分析

## 5.1 测试分析概述

攻击测试是检验 IDS 检测性能的一个最直观有效的方法。通过在一个小型的网络环境中模拟各种攻击，观察 IDS 的检测效率、报警情况等，可得知 IDS 的整体性能。

除了 IDS 本身之外，还需要一些其他的网络配置工具来共同完成对 IDS 的测试，比如需要在一个网络环境下安装 Apache[35]、ACID[36]、ADODB[37]、Webmin[38]、MySQL[39]和 TCPAMP[40]等软件工具。

## 5.2 测试环境

测试平台如下图所示：

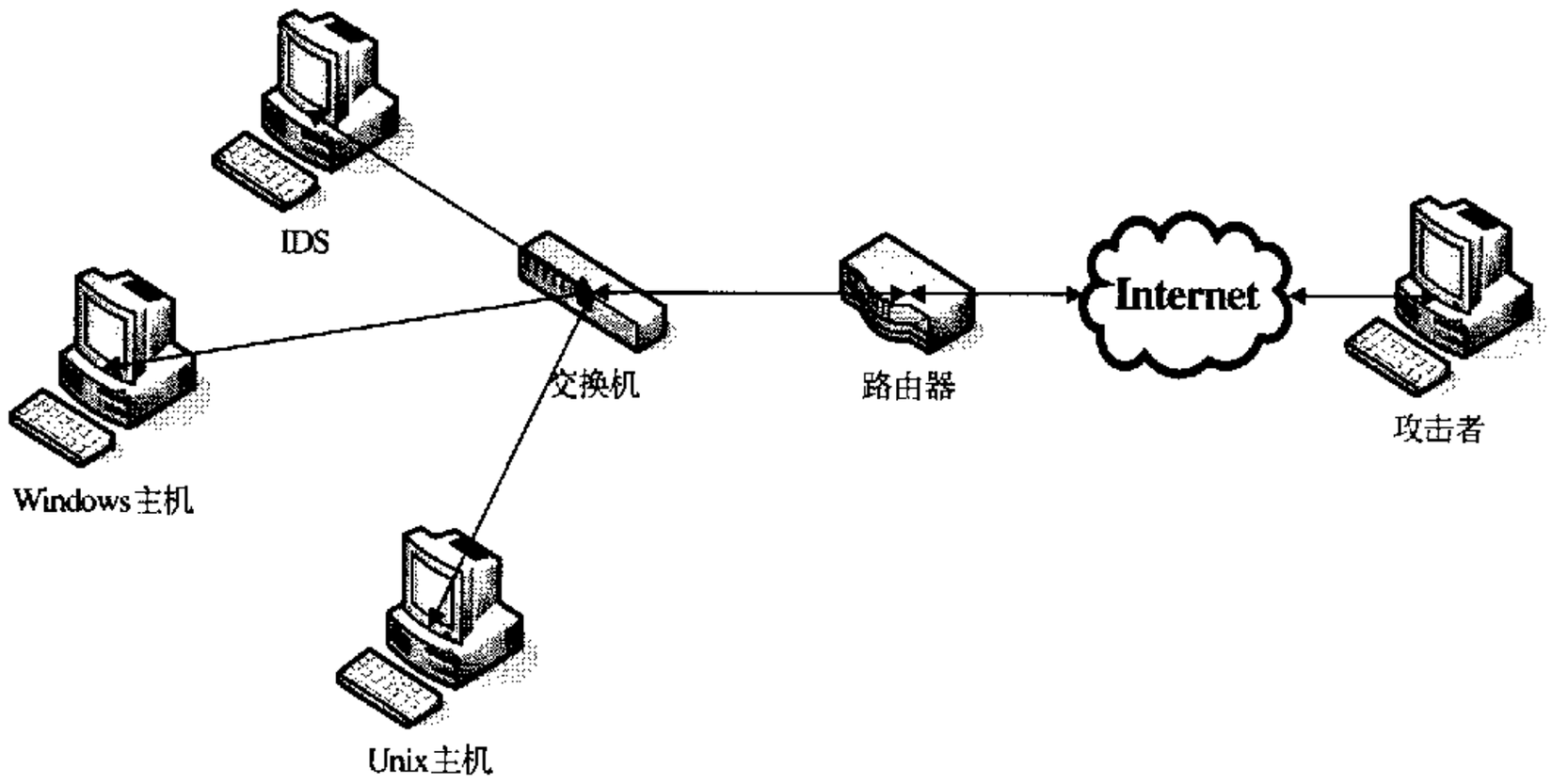



图 5－1 测试环境

## 5.3 测试方法

在以上测试环境中，安装好相应的软件。受攻击主机包括 Windows 系统和 UNIX 系统各一部，另一主机上安装 IDS，三者在同一子网内，发起攻击的主机位于另一子网，其上安装各种攻击工具，对目标发起各种攻击。主要包括端口扫描，漏洞刺探，暴力密码穷举，缓冲区溢出，拒绝服务（DoS）几种典型攻击方法进行。通过对比安装本检测模块前的 snort 系统检测率与安装本模块后的系统检测率进行对照，

验证本检测模块的检测效率。

(1) 端口扫描：采用功能强大的扫描软件 NMAP[41]进行扫描，端口范围为 1－1024。扫描方式包括 TCP 扫描，UDP 扫描，系统指纹识别等。并且每种扫描方式分别进行以正常速度与慢速各实施若干次，以便比较扫描速度对检测效率的影响。

(2) 漏洞刺探：采用 nessus[42]作为主要工具，并以其他一些流行的漏洞审计工具作为补充。两台目标主机分别为 windows2000 及 RedHat Linux(内核版本 2.4.20-8)，并未打上任何补丁，目的是为了暴露尽量多的漏洞，让攻击者进行攻击，以衡量 IDS 的性能。

(3) 暴力密码穷举：由于不少应用程序对密码验证次数作了检查，因此随着密码校验失败次数的不断增加，每次验证的间隔会变得越来越长，这本身已经有很好的保护措施。但另一方面也使得 IDS 更难检测出来。而有些应用程序由于不作限制，则使得暴力破解更加简单。我们的测试对这两种情况都分别进行。

(4) DOS, DDOS 是危害越来越严重的攻击类型，同时也是 IDS 的一个薄弱环节，因为很可能 IDS 本身就会被这些杂乱而又庞大的攻击信息所湮没而发觉不出隐蔽在内的攻击，甚至导致本身丧失功能。

## 5.4 对误用检测引擎的测试

按照上面描述的测试方法进行测试，得出如下实验现象。

### 5.4.1 扫描结果。NMAP 扫描实验过程：

(1) 基本 TCP 端口扫描　#nmap -sT -v targetIP
　　产生如下报告：
　　SCAN SOCKS Proxy attempt
　　SCAN Proxy (8080) attempt
　　SCAN Squid Proxy attempt
(2) 网络主机探测扫描　#nmap -sP 192.168.100.1/24
　　结果：没有产生任何 alert
(3) 基本 UDP 端口扫描　#nmap -sU -v targetIP
　　结果：没有产生任何 alert
(4) 操作系统类型扫描　#nmap -O -v targetIP
　　结果：产生以下 alerts：
　　SCAN Squid Proxy attempt
　　SCAN Proxy (8080) attempt
　　(spp_stream4) NMAP FINGERPRINT (stateful) detection

SCAN nmap TCP
(spp_stream4) STEALTH ACTIVITY (XMAS scan) detection

误用检测引擎测试分析：

从上面的扫描检测结果中可以看出，snort 对于某些类型的扫描（如操作系统指纹识别）有比较好的发现性能，然而，由于 snort 是基于模式匹配的检测，因此，在对于某些主机发现的网段扫描，及 UDP 协议相关的扫描，其检测能力比较薄弱，而对于这些试探性的扫描，往往需要统计分析等技术去完成，因此基于异常分析的技术更加适合。

### 5.4.2　网络漏洞刺探过程的检测

在漏洞刺探的过程中，我们使用的软件是 nessus2.0.8，并启用所有漏洞探测选项，因为该软件作为一个免费的漏洞检查工具，在很多方面都相当优秀，也是黑客经常使用的工具之一。对目标进行扫描后，检查 snort 的警报数据库，其产生的警报如下：

表 5-1 漏洞刺探过程中 snort 所产生的报警

| ID | ≤ Signature ≥ | ≤ Timestamp ≥ | ≤ Source Address ≥ | ≤ Dest. Address ≥ | ≤ Lay 4 Proto |
|---|---|---|---|---|---|
| #0-(1-58) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:52:21 | 192.168.100.234:36507 | 192.168.100.233:8080 | TCI |
| #1-(1-57) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:51:38 | 192.168.100.234:36240 | 192.168.100.233:8080 | TCI |
| #2-(1-56) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:51:38 | 192.168.100.234:36239 | 192.168.100.233:8080 | TCI |
| #3-(1-55) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:51:11 | 192.168.100.234:36119 | 192.168.100.233:8080 | TCI |
| #4-(1-54) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:51:11 | 192.168.100.234:36118 | 192.168.100.233:8080 | TCI |
| #5-(1-53) | [snort] (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection | 2004-02-26 00:51:00 | 192.168.100.234:10004 | 192.168.100.233:25 | TCI |
| #6-(1-52) | [snort] (snort_decoder) WARNING: TCP Data Offset is | 2004-02-26 00:50:51 | 192.168.100.234:999 | 192.168.100.233:1900 | TCI |

| | | | | | |
|---|---|---|---|---|---|
| | less than 5! | | | | |
| #7-(1-51) | [snort] (snort_decoder) WARNING: TCP Data Offset is less than 5! | 2004-02-26 00:50:50 | 192.168.100.234:999 | 192.168.100.233:1900 | TCP |
| #8-(1-50) | [snort] (snort_decoder) WARNING: TCP Data Offset is less than 5! | 2004-02-26 00:50:49 | 192.168.100.234:999 | 192.168.100.233:1900 | TCP |
| #9-(1-49) | [snort] (snort_decoder) WARNING: TCP Data Offset is less than 5! | 2004-02-26 00:50:47 | 192.168.100.234:999 | 192.168.100.233:1900 | TCP |
| #10-(1-48) | [snort] (snort_decoder) WARNING: TCP Data Offset is less than 5! | 2004-02-26 00:50:46 | 192.168.100.234:999 | 192.168.100.233:1900 | TCP |
| #11-(1-47) | [snort] SCAN Squid Proxy attempt | 2004-02-26 00:50:45 | 192.168.100.234:36012 | 192.168.100.233:3128 | TCP |
| #12-(1-46) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:50:38 | 192.168.100.234:35960 | 192.168.100.233:8080 | TCP |
| #13-(1-45) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:49:41 | 192.168.100.234:35722 | 192.168.100.233:8080 | TCP |
| #14-(1-44) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:49:40 | 192.168.100.234:35721 | 192.168.100.233:8080 | TCP |
| #15-(1-43) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:49:15 | 192.168.100.234:35613 | 192.168.100.233:8080 | TCP |
| #16-(1-42) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:48:37 | 192.168.100.234:35442 | 192.168.100.233:8080 | TCP |
| #17-(1-41) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:47:22 | 192.168.100.234:35023 | 192.168.100.233:8080 | TCP |
| #18-(1-40) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:47:22 | 192.168.100.234:35022 | 192.168.100.233:8080 | TCP |
| #19-(1-39) | [snort] SCAN Squid Proxy attempt | 2004-02-26 00:47:11 | 192.168.100.234:34976 | 192.168.100.233:3128 | TCP |
| #20-(1-38) | url[snort] SCAN SOCKS Proxy attempt | 2004-02-26 00:46:23 | 192.168.100.234:34618 | 192.168.100.233:1080 | TCP |
| #21-(1-37) | [snort] SCAN Proxy (8080) attempt | 2004-02-26 00:45:22 | 192.168.100.234:34140 | 192.168.100.233:8080 | TCP |
| #22-(1-36) | [snort] SCAN Squid Proxy attempt | 2004-02-26 00:45:22 | 192.168.100.234:34139 | 192.168.100.233:3128 | TCP |
| #23-(1-25) | [snort] SCAN Amanda client version request | 2004-02-26 00:44:21 | 192.168.100.234:32882 | 192.168.100.233:10081 | UDP |
| #24-(1-24) | [snort] SCAN Amanda client version request | 2004-02-26 00:44:21 | 192.168.100.234:32882 | 192.168.100.233:10080 | UDP |
| #25-(1-23) | [snort] spp_bo: Back Orifice | 2004-02-26 | 192.168.100.234:32791 | 192.168.100.233:31337 | UDP |

| | | | | | |
|---|---|---|---|---|---|
| | Traffic detected (key: 31337) | 00:44:16 | | | |
| #26-(1-35) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #27-(1-34) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #28-(1-33) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #29-(1-32) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #30-(1-31) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #31-(1-30) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |
| #32-(1-29) | [snort] (spp_stream4) STEALTH ACTIVITY (NULL scan) detection | 2004-02-26 00:44:25 | 192.168.100.234:25 | 192.168.100.233:25 | TC |

关于漏洞攻击的分析：

上面的检测结果比较符合 snort 本身的系统特点，由于漏洞刺探是对操作系统本身或应用程序的漏洞尝试，因此 snort 的模式匹配技术可以很好的派上用场。而基于异常的检测技术往往会在难以确定检测到的是哪种攻击，所以引起的误报和漏报也较多。如果在这里实施，最好与模式匹配技术结合，一同定位。

### 5.4.3　暴力入侵攻击网络

拒绝服务攻击(DoS)与分布式拒绝服务攻击（DDoS）已经越来越成为网络安全的一个严重威胁，也是入侵检测系统一个十分薄弱的环节和入侵检测技术的一个重要的发展趋势。针对某一系统的特定服务（如 web）进行 DoS 攻击十分常见，通常会使得该服务甚至整个网络无法工作，造成损失相当大。SYN flood 作为一个十分典型的 DOS 攻击，他利用 TCP 协议 3 次握手的缺陷，通过大量的半连接，使得服务器大量消耗资源，从而难以为正常连接请求提供服务。因此，在 DoS 攻击测试过程中，我们采用典型的 SYN flood 攻击，对 UNIX 及 Windows 系统的 web 服务端口（80）进行攻击，实验过程发现：在没有攻击前，两个目标系统的 web 请求都能正

常进行，当开始攻击后，两个目标系统的主页无法连接成功。并且，只有在攻击停止后，才慢慢恢复。由此说明攻击收到了十分明显的效果。然而检查 snort 的警报日志，却发现没有留下任何关于 SYN flood 的警报信息。可见，起码对于这种十分常见的攻击，它并没有很好的检测出来。

### 5.5　对误用检测引擎的测试

我们将 SYN 和 ACK 的比值作为检测网络是否受到 SYN flood 攻击的测度，设测量流量的时间粒度为 15 分钟，第 i 个时间单位内抽样到的 SYN 位为 1 的数量和为 s，ACK 位为 1 的数量和为 a，那么他们的比值为 r＝s/a，在每个测量流量时间段内，根据滑动窗口内部的统计值计算出置信区间，同时判断该时间段内的统计值是否落在置信区间内，如果落在置信区间之外则产生报警，图 5-2 是从上午 8：30 到下午 17：30 分之间比值 r 随着时间的变化曲线，我们分别在上午 10：45 分到 11：45 分之间和下午 16：00 到 17：00 之间，连续用 SYN flood 工具对网络中的某台主机进行攻击，从图上我们业可以看出，在这两个时间段内 r 值产生较大的变化。

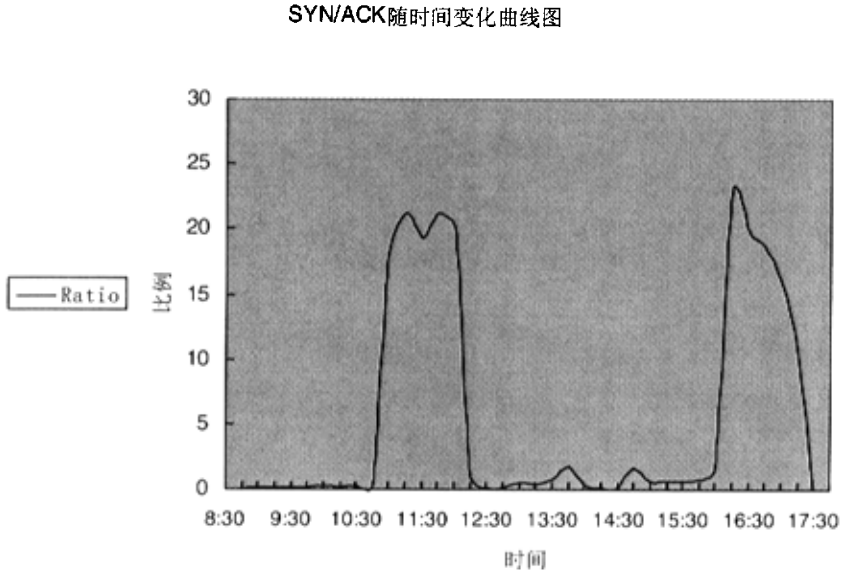



图 5-2　r 值随着时间的变化值

### 5.6　本入侵检测系统的测试结论

综合上面的测试结果，我们可以得出如下结论：snort 作为一个以模式匹配技术为主的入侵检测系统，它在对于大部分需要依靠字符串精确匹配的攻击检测方面性能较好，然而，对于需要进行统计以及监控连接情况的攻击，它缺少检测的能力。

此外，对于大流量的攻击，也有来不及分析的缺点。因此，在它上面增加一个基于异常分析的模块，是十分必要的。我们增加的基于异常的检测引擎用于检测 SYN flood 等通过建立大量连接进行的攻击方面具有一定的效果。

# 第六章 研究开发 IDS 的回顾

## 6.1 回顾与总结

本文在研究开源软件 snort 的基础上，提出了误用与异常相结合的入侵检测系统（MACIDS），采用模式匹配和统计分析相结合的检测方式，扩展了 snort 原来的功能和结构。

MACIDS 在结构上采用插件机制，简化了编码工作，系统本身可以很容易地增加插件，具有很大的灵活性，主要包括预处理插件，处理插件和输出插件。 MACIDS 整体上可以分为五大引擎，包括数据包获取和解码引擎、预处理引擎、基于误用的检测引擎、基于异常的检测引擎、记录日志引擎。

MACIDS 借鉴了 TCP 协议中流量控制的滑动窗口机制，采用滑动窗口的方法来实时获取数据包，动态更新，并定义了置信区间，来判断是否有异常情况发生，克服了 snort 原有系统先遭受攻击，再通过增加规则来增强检测能力的缺点，使系统的功能更强大。

同时，MACIDS 还有很多缺点，由于采用 libpcap 软件抓包的方式，使得系统不能适应大流量网络环境，当流量过大时，会丢失很多数据包；另外，虽然 MACIDS 具有异常检测能力，但却只是针对 SYN flood 等发送大量数据包的攻击，并不能发现所有类型的未知攻击；MACIDS 虽然部分地解决了漏报的问题，但同时又导致了误报率高的问题；分布式 IDS 是 IDS 的发展方向，也是检测大规模分布式攻击的有效手段，MACIDS 在针对分布式攻击的检测能力方面有很大不足。

## 6.2 入侵检测面临的挑战

目前为止，入侵检测技术仍在不断发展之中，入侵检测领域还面临着很多挑战，值得注意的是，这些挑战大多是目前入侵检测系统的结构所难以克服的，而且这些矛盾可能越来越尖锐。

以下便是对入侵检测系统提出挑战的主要因素：

(1) 攻击者不断增加的知识，日趋成熟多样自动化工具，以及越来越复杂细致的攻击手法。[44]

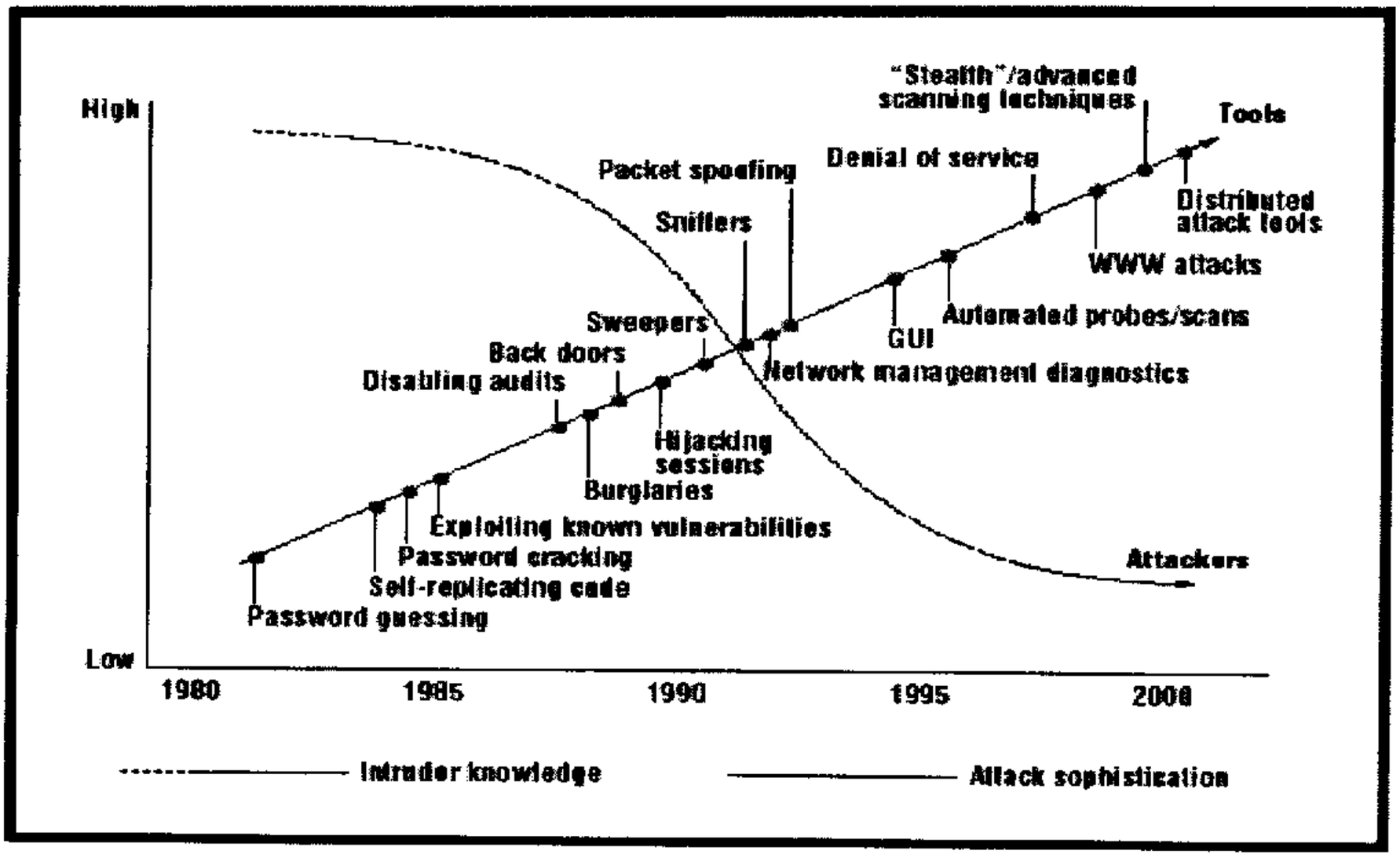



图 6－1 攻击的复杂性和入侵技术知识的对比

(2) 恶意信息采用加密的方法传输。

网络入侵检测系统通过匹配网络数据包发现攻击行为，IDS 往往假设攻击信息是通过明文传输的，因此对信息的稍加改变便可能骗过 IDS 的检测。TFN 现在便已经通过加密的方法传输控制信息。还有许多系统通过 VPN（虚拟专用网）进行网络之间的互联，如果 IDS 不了解其所用的隧道机制，会出现大量的误报和漏报。

(3) 不断增大的网络流量。

用户往往要求 IDS 尽可能快地报警，因此需要对获得的数据进行实时的分析，这导致对所在系统的要求越来越高，商业产品一般都建议采用当前最好的硬件环境。尽管如此，对千兆以上的流量，单一的 IDS 系统仍很难应付。可以想见，随着网络流量的进一步加大（许多大型 ICP 目前都有数百兆的带宽），对 IDS 将提出更大的挑战，在 PC 机上运行纯软件系统的方式需要突破。

(4) 广泛接受的术语和概念框架的缺乏。

入侵检测系统的厂家基本处于各自为战的情况，标准的缺乏使得其间的互通几乎不可能。

(5) 采用不恰当的自动反应所造成的风险。

入侵检测系统可以很容易的与防火墙结合，当发现有攻击行为时，过滤掉所有来自攻击者的 IP 的数据。但是，不恰当的反应很容易带来新的问题，一个典型的例子便是：攻击者假冒大量不同的 IP 进行模拟攻击，而 IDS 系统自动配置防火墙将这些实际上并没有进行任何攻击的地址都过滤掉，于是形成了新的拒绝访问攻击

（DoS）。

(6) 对 IDS 自身的攻击。

和其他系统一样，IDS 本身也往往存在安全漏洞。如果查询 bugtraq 的邮件列表，诸如 Axent、NetProwler、NFR、ISS Realsecure 等知名产品都有漏洞被发觉出来。若对 IDS 攻击成功，则直接导致其报警失灵，入侵者在其后所作的行为将无法被记录。

## 6.3 进一步工作的展望

由于本文的出发点是针对高速公路相关信息系统的进行入侵检测，虽然有规模小，灵活等特点，但在大范围联合检测攻击方面做的不够，随着入侵检测技术的发展，已有的 IDS，无论是 HIDS，还是 NIDS，都是针对单一网段的，对于一个庞大复杂的网络来说，传统的 IDS 就显得很单薄了。一方面，网络的异构性决定了单一 IDS 功能上的不足，即便功能上满足了，性能也不能适应高速网络的发展。另一方面，多个系统中运行多种独立的 IDS，它们之间并不能建立有效的沟通，这就使得入侵检测系统的功能不能很好的延伸。这种情况下，各种基于协同工作的分布式入侵检测系统方案是大势所趋。

# 参考文献


1. 《高速公路暂行技术要求》，中华人民共和国交通部，人民交通出版社，2000。
2. 韩东海、王超、李群编著.入侵检测系统实例剖析.清华大学出版社，2002。
3. Atkins, P.Buis, C.Hare, R.Kelley, C.Nachenberg, A.B.Nelson, P.Phillips, T.Ritchey, and W.Steen.Internet Security Professional Reference.New Riders Publishing, 1996.
4. S.M.Bellovin.Security problems in the TCP/IP protocol suite.Computer Communication Review, Vol.19, No. 2, pp. 32-48, April 1989.
5. 朱祖韬 基于 agent 的混合式 IDS 的设计与研究 中山大学硕士毕业论文，2003。
6. 《收费数据安全性应用技术研究》,《高速公路联网收费暂行技术要求》修订课题组，2003。
7. （美）Anonymous 等 朱鲁华 李志 李海波 译《最高安全机密》，机械工业出版社，2003。
8. B. Mukherjee, T.L. Heberlein, K.N. Levitt. Network intrusion detection, IEEE Network 8, 1994 (3), 26-41.
9. Eugene H. Spafford, Diego Zamboni. Intrusion detection using autonomous agents, Computer Networks, 2000 (34), 547-570.
10. R. Heady, G. Luger, A. Maccabe, M. Servilla. The architecture of a network level intrusion detection system, Technical Report, University of New Mexico, Department of Computer Science, August 1990.
11. ICSA, Inc. An Introduction to Intrusion Detection & Assessment.
12. http://www.ietf.org/html.charters/idwg-charter.html.
13. http://www.isi.edu/gost/cidf/.
14. James P. Anderson.Computer Security Threat Monitoring and Surveillance, February 26, 1980. http://csrc.nist.gov/publications/history/ande80.pdf.
15. D.E. Denning, D.L. Edwards, R. Jagannathan, T.F. Lunt, P.G. Neumann, A prototype IDES - a real-time intrusion detection expert system, Technical Report, Computer Science Laboratory, SRI International, 1987.
16. D.E. Denning, P.G. Neumann, Requirements and model for IDES - a real-time instrusion detection system, Technical Report, Computer Science Laboratory, SRI International, August 1985.
17. Steven R.Snapp 1, James Brentano, ... and Doug Mansur. "DIDS (Distributed Intrusion Detection System) –Motivation, Architecture, and an Early Prototype".Computer Security Laboratory, UC, Davis.
18. Rebecca Gurley Bace 著 陈明奇 吴秋新等译 《入侵检测》，人民邮电出版社，2001。
19. www.tcpdump.org, libpcap.
20. www.snort.org, snort
21. Brian Caswell, Jay Beale, James C. Foster, Jeffrey Posluns 著，宋劲松等 译 《Snort 2.0 入侵检测》，国防工业出版社。
22. 唐正军等.网络入侵检测系统的设计与实现.北京:电子工业出版社, 2002.

23. www.protocols.com, 对协议的理解。

24. http://staff.washington.edu/dittrich/misc/ddos/, SYN flood.

25. Boyer-Moore 算法 http://www-igm.univ-mlv.fr/~lecroq/string/node14.html.

26. Neil Desai Increasing Performance in High Speed NIDS. http://www.docshow.net

27. 3rd Generation Intrusion Detection Technology From Network ICE Protocol Analysis and Command Parsing vs Pattern Matching in Intrusion Detection System.

28. W. Richard Stevens 著 范建华等 译 TCP/IP 详解 卷 1：协议，机械工业出版社，2000。

29. 朱勇华 郜淑彩 等编 应用数理统计，武汉水里电力大学出版社。

30. Dorothy E. Denning SRI International 333 Ravenswood Ave.Menlo Park, CA 94025. AN INTRUSION-DETECTION MODEL.

31. 程光、龚俭、丁伟 基于抽样测量的高速网络实时异常检测模型 软件学报 2003 594－599。

32. Kaleton Internet Dept. 5364 Suite 145 269/2 Soi Potisarn Moo 6 Naklua Banglamung Chonburi 20150 Thailand Combination of Misuse and Anomaly Network Intrusion Detection System.

33. W. Richard Stevens 著，尤晋元等译.UNIX 环境高级编程.机械工业出版社，2000。

34. Kurt Wall 等 著，张辉 译，GNU/Linux 编程指南，清华大学出版社，2003。

35. www.apache.org, Apache.

36. http://acidlab.sourceforge.net/, ACID

37. http://php.weblogs.com/adodb, ADODB

38. www.webmin.com, Webmin

39. www.mysql.com, MYSQL

40. www.sourceforge.org, TCMAMP

41. http://www.insecure.org/nmap, nmap.

42. http://www.nessus.com/, nessus.

43. John McHugh, Alan Christie, and Julia Allen Defending Yourself: The Role of Intrusion Detection System, Software Engineering Institute, CERT Coordination Centre.

# 附 录 RedHat Linux9 环境下 snort 与 ACID, Webmin 的整合配置方案

Snort 作为一个免费的轻量级入侵检测系统，以其出色的检测性能而或用户好评。然而与大部分商业 IDS 相比，snort 尽管轻便、免费，但其监控功能的集成性较差，分布式部署困难、不易管理等缺点，需要较多的配置。下面给出一个在 RedHat Linux9 环境下的 snort 与 ACID, Webmin, Apache 和 MySQL 的整合方案，通过这种整合，使得 snort 能更好的应用于分布式的部署和管理。

配置步骤：

0. 编译和安装 libpcap：

libpcap 的作用和下载安装方式

```
# tar -xvzf libpcap-0.7.2.tar.gz
# cd libpcap-0.7.2
#./configure
# make
# make install
```

1. 编译和安装 snort：

为了 snort 能使用 MySQL，首先应安装 MySQL 提供的依赖模块，包括 client 和开发接口包。从 www.mysql.com 上下载以下两个 rpm 包并安装：

```
# rpm  -ivh MySQL-client-*.**.**-*.rpm
# rpm  -ivh MySQL-devel-*.**.**-*.rpm
```

然后从www.snort.org上下载最新版本的snort源代码，以下面步骤编译和安装：

```
# cp snort-2.0.*.tar.gz /usr/src/redhat/SOURCES
# cd /usr/src/redhat/SOURCES
# tar  -zxvf snort-1.8.*.tar.gz
# cd /usr/src/redhat/SOURCES/snort-1.8.*
# ./configure --with-mysql      //请注意加上此编译选项
# make
# make install
```

从同一网站上下载最新的规则库snortrules.tar.gz，解压后放到/etc目录下，步骤如下：

```
# mkdir /etc/snort
# cp snortrules.tar.gz /etc/snort
```

```
# cd /etc/snort
# tar -zxvf snortrule.tar.gz
#cd /etc/snort/rules
# mv * ../
# cd ..
# rmdir rules
```

在此目录下编辑snort.conf，将下面一行：

```
#output database: log, mysql, user=root password=test dbname=db
host=localhost
```

改为

```
output database: log, mysql, user=snort password=snort dbname=snort
host=yourSQLserverIP
```

注意:host后面为MySQL数据库所在的服务器的IP地址或域名若本机则为localhost。并将里面的

```
var RULE_PATH ../rules
```

一行注释为

```
#var RULE_PATH ../rules
```

并将文件中所有引用到该变量的串作相应修改，把该串从原来行中去除。如原来文件中是：include $RULE_PATH/scan.rules

的一行改为：

```
include scan.rules
```

为 snort 新建一个日志文件：

```
# mkdir /var/log/snort
```

用编辑器创建一个名称为snortd的文件，并在文件里键入以下内容：

```
#!/bin/sh
#
# snortd          Start/Stop the snort IDS daemon.
#
# chkconfig: 2345 40 60
#description:  snort is a lightweight network intrusion detection tool that
#               currently detects more than 1100 host and network
#               vulnerabilities, portscans, backdoors, and more.
#
# June 10, 2000 -- Dave Wreski <dave@linuxsecurity.com>
#   - initial version
#
# July 08, 2000 Dave Wreski <dave@guardiandigital.com>
#   - added snort user/group
#   - support for 1.6.2

# Source function library.
. /etc/rc.d/init.d/functions
```

```
# Specify your network interface here
INTERFACE=eth0

# See how we were called.
case "$1" in
  start)
          echo -n "Starting snort: "
          ifconfig eth0 up
          daemon /usr/local/bin/snort -U -o -i $INTERFACE -d -D \
             -c /etc/snort/snort.conf
          touch /var/lock/subsys/snort
          sleep 3
          rm /var/log/snort/alert
          echo
          ;;
  stop)
          echo -n "Stopping snort: "
          killproc snort
          rm -f /var/lock/subsys/snort
          echo
          ;;
  restart)
          $0 stop
          $0 start
          ;;
  status)
          status snort
          ;;
  *)
          echo "Usage: $0 {start|stop|restart|status}"
          exit 1
esac
exit 0
```

然后用以下方式复制到相应目录下并修改起文件模式：

```
# cp snortd /etc/rc.d/init.d
# cd /etc/rc.d/init.d
# chmod 755 snortd
# chkconfig --level 2345 snortd on
```

2. 配置 Apache 服务器，MySQL 和 ACID：

安装最新版本的 Apache 服务器：

```
# rpm -ivh apache-1.3.X-X.i386.rpm
# chkconfig --level 2345 httpd on
# /etc/rc.d/init.d/httpd start
```

安装最新版本的MySQL数据库，并创建root用户：

```
# rpm -ivh MySQL-4.0.15-0.i386.rpm
# rpm -ivh MySQL-client-4.0.15-0.i386.rpm
# rpm -ivh MySQL-shared—4.0.15-0.i386.rpm
# mysql -u root
mysql>  set  password  for  'root'@'localhost'  =
password('yourpassword');
mysql> create database snort;
mysql>exit
# chkconfig - level 3 mysql on
```

到 http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/snort/snort/contrib/下载名字为create_mysql的脚本，放到当前目录，并以下面方法运行：

```
# mysql -u root -p
mysql> connect snort
mysql>source create_mysql
mysql>grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.* to snort;
mysql>grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.* to
snort@localhost;
```

然后为这些帐号设置密码：

```
mysql>connect mysql
mysql>  set  password  for  'snort'@'localhost'  =
password('yourpassword');
mysql> set password for 'snort'@'%' = password('yourpassword');
mysql> flush privileges;
mysql> exit
```

为ACID安装所需要的模块：

```
# rpm -ivh php-4.1.*-*.i386.rpm      //安装PHP
# rpm -ivh php-mysql-4.1.*-*.i386.rpm     //安装插件
```

下载以下文件：

| | |
|---|---|
| ACID | http://acidlab.sourceforge.net/ |
| ADODB | http://php.weblogs.com/adodb |
| PHPLOT | http://www.phplot.com/ |
| GD | http://www.boutell.com/gd/ |

以下面方法解压到/var/www/html/目录下：

```
# tar -zxvf acid-0.9.*.tar.gz -C /var/www/html
```

```
# tar -zxvf adodb231.tgz -C /var/www/html
# tar -zxvf gd-1.8.4.tar.gz -C /var/www/html
# tar -zxvf phplot-4.4.6.tar.gz -C /var/www/html
```

解压完后注意对以上目录名改名，使它不出现版本号。如phplot-4.4.6.tar.gz解压后目录名为phplot-4.4.6，应改名为phplot。

配置ACID：

```
# cd /var/www/html/acid
# vi acid_conf.php
```

将相应行改为：

```
$DBlib_path="../adodb";
$alert_dbname="snort";
$alert_user="snort";
$alert_password="yourpassword";
$Chartlib_path="../phplot";
```

为ACID目录访问建立密码文件：

```
# mkdir /usr/local/etc/apache/passwords
# htpasswd -c /usr/local/etc/apache/passwords/passwords admin
#
```

编辑/etc/httpd/conf/httpd.conf，在适当的地方增加以下目录项：

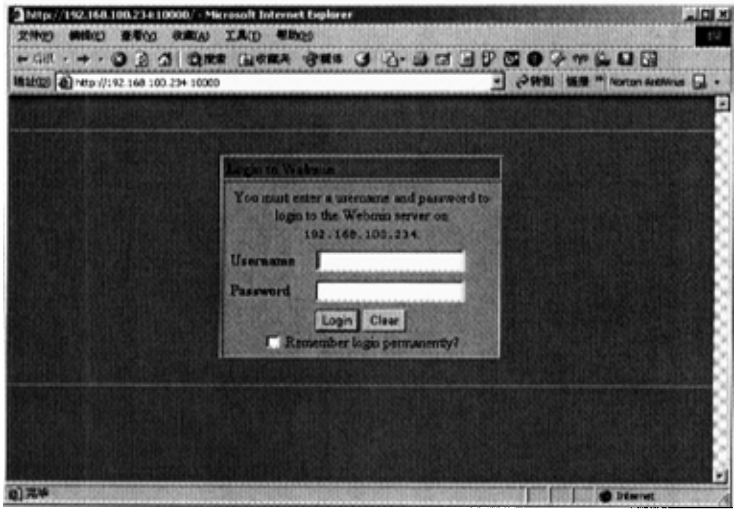
```
<Directory "/var/www/html/acid">
 AuthType Basic
 AuthName "yourcompany"
 AuthUserFile /usr/lib/apache/passwords/passwords
 Require user admin
 AllowOverride None
 </Directory>
```

3. 安装Webmin

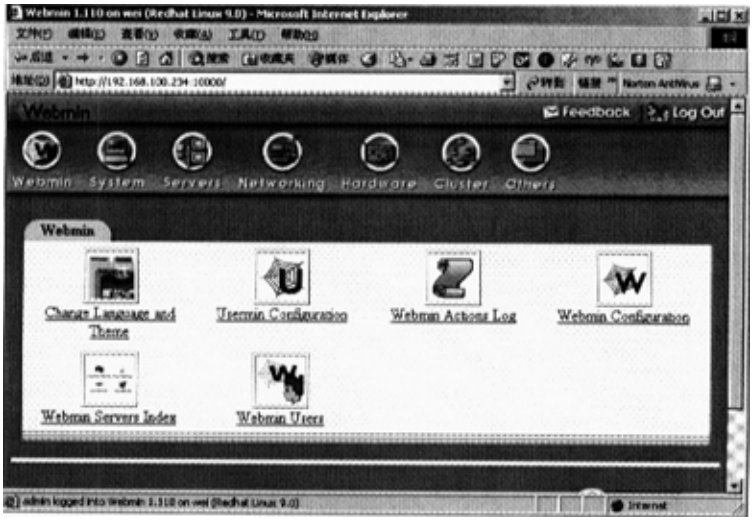
从webmin网站下载最新版本的webmin rpm包并安装：

```
# rpm -ivh webmin-1.x.-x.noarch.rpm
```

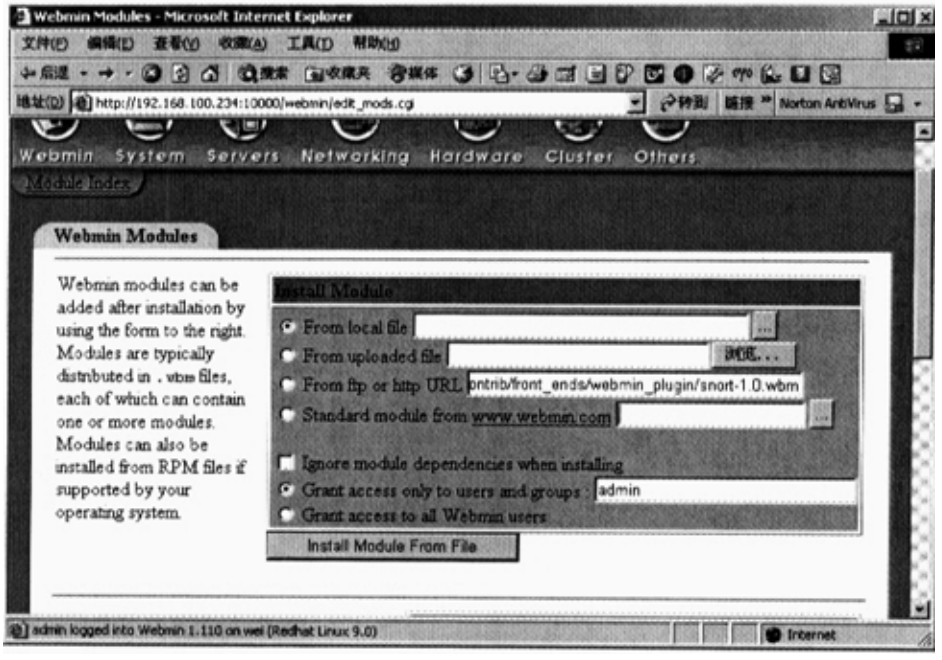
在本机上用浏览器登录到如下地址：http://127.0.0.1:10000，这时会出现webmin登录界面



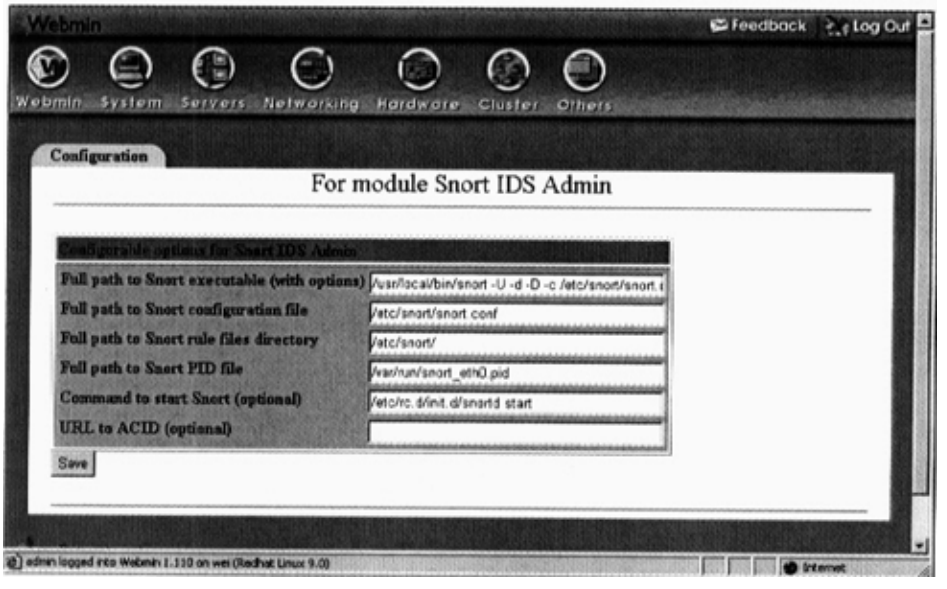

以 admin 登录后，界面显示如下：




选择 webmin configuration，并选择 webmin modules，进入如下界面：




在From ftp or URL栏打入：
http://www.snort.org/dl/contrib/front_ends/webmin_plugin/snort-1.0.wbm
并点击安装。
安装完毕后，选择server，Snort IDS Admin 进行配置，在module configure中进行如下配置：




Full path to Snort executable (with options) = /usr/local/bin/snort -U -d -D -c /etc/snort/snort.conf

Full path to Snort configuration file = /etc/snort/snort.conf

```
Full path to Snort rule files directory = /etc/snort
Full path to Snort PID file = /var/run/snort_eth1.pid
Command to start Snort (optional) = /etc/rc.d/init.d/snortd start
```

然后保存退出。

可用/etc/rc.d/init.d/snortd start 来启动 snort， 并用浏览器连到中心数据库中查看入侵检测的信息：
地址为 http://192.168.100.234/acid，用户名为上面所配置的 admin。

# 致 谢

衷心感谢导师梁华金老师在我研究生学习期间所给予的精心指导和关怀。梁老师在学习上对我严格要求，循循善诱，严厉而不失和蔼，在研究上教导我要踏踏实实，实事求是、勇于创新，而且想方设法为我提供良好的学习环境和生活环境。没有梁老师的鼓励和悉心关怀，我是不能完成研究生学业的。还有梁老师严谨的治学态度和精益求精的工作作风深深的影响了我，我从梁老师那里学到的不仅是学习、研究的方法，而且还有做人、做事的道理，这些将会影响我的一生。

感谢广东新粤机电公司的副总经理姚赤丹高工、邸磊经理，他们不但给予我充分的信任，而且为我创造了很好的研究和实验环境。

感谢我的师弟卫增权，为我的论文的测试等工作提出宝贵意见。还要感谢我众多的同学和朋友：感谢我的女朋友施晓霞，在我写论文期间给我很大支持；感谢张思彦、柯为、卢学东、懂丰华、魏硕、黄国萍、唐亮、王伟等，他们在学习上和生活上都给了我很大的帮助。同时，在此对所有给予我帮助和关心的各位老师和同学一并谢过。

# 原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：李伟

日期：2004 年 5 月 8 日