

东北大学
硕士学位论文
Snort检测引擎的改进与实现
姓名：胡德华
申请学位级别：硕士
专业：计算机应用技术
指导教师：刘春雨
20050101

# Snort 检测引擎的改进与实现

## 摘 要

入侵检测是一种主动保护自己免受攻击的网络安全技术，是继“防火墙”、“数据加密”等传统安全保护措施后的新一代技术。检测引擎作为入侵检测系统(IDS)的核心模块，基本上采用基于模式匹配的检测方法，所以选择设计一个好的模式匹配算法对入侵检测系统的性能至关重要。

Snort 是一个强大的轻量级的网络入侵检测系统，它具有实时数据流量分析和日志 IP 网络数据包的能力，能够进行协议分析，对内容搜索或者匹配。它能够检测各种不同的攻击方式，并对攻击进行实时警报。此外，Snort 具有很好的扩展性和可移植性。

本文主要分析了 Snort 系统的架构、工作流程和三维规则链表，并着重分析了该系统的检测引擎及其采用的模式匹配算法。针对原系统模式匹配算法的不足，本人运用了新的多模式匹配算法对 Snort 检测引擎进行了改进，并应用到 Snort 的检测引擎模块中。通过实验结果证明所采用的算法是有效的，比原系统检测引擎的匹配速度有较大提高。最后提出了经过改进后的系统的优点和缺点，以及进一步的优化改进建议。

**关键词** 检测引擎 模式匹配 散列 Snort IDS

# The Improvement and Implementation of Snort Detection Engine

## ABSTRACT

Intrusion Detection is one kind of network security technologies that actively protects oneself from attack; continue the security technologies of new generation after the traditional safe protective measures, such as "fire wall", "the data encrypted ", etc.. The Detection Engine, as the core module of the Intrusion Detection System (IDS), primarily adopts the detection methods that based on pattern matching. So it is important for IDS's performance to select or design an excellent pattern matching methods.

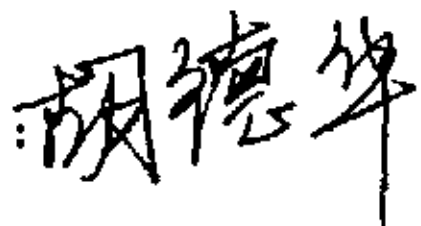
Snort is one powerful lightweight network IDS. It has the ability of realtime data analyzing and recording IP network data packets, and it can be able to process protocol analyzing, definite content searching or matching. Snort also can detect many different attack ways, and then give a realtime alarm. Furthermore, Snort has good expansibility and transability.

In this paper, I firstly describe the Snort's architecture, working flow and three-dimensional linked list, and then especially analyze the Detection Engine of the Snort and the Detection Engine's pattern matching algorithms. Because of the shortcomings of the original pattern matching algorithms the Snort used, I choose a new improved algorithm to improve the Snort's Detection Engine, and then apply it into the Snort's Detection Engine. Through several experiments' results, I prove that the new improved algorithm is efficient; moreover the speed of the improved Detection Engine is faster than original system's. The advantage and disadvantage of the newly implemented system as well as a suggestion of further improvement are given at last.

**Key words**    Detection Engine, Pattern Matching, hashing, Snort, IDS

# 独创性声明



本人声明所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：  
日期：2005.1.28

# 学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

（如作者和导师同意网上交流，请在下方签名；否则视为不同意。）

学位论文作者签名： 	导师签名： 
签字日期：2005.1.28	签字日期：2005.1.28

# 第一章 引言

## 1.1 计算机网络安全现状

近年来信息革命席卷全球，网络化浪潮汹涌而至，特别是互联网的爆炸式发展，给人类社会、经济、文化等带来了无限的机遇，但也给信息安全带来了严峻挑战，各种不安全因素的大量涌入致使计算机网络安全问题日益突出。2000年2月在短短的3天时间里美国数家顶级互联网站受到黑客攻击，陷入瘫痪。据估算，袭击所造成的经济损失大约在12亿美元以上，其中受害攻击3天内的损失高达10多亿美元的市场价值，营销和广告收入损失1亿美元以上。可见，保障计算机网络系统及整个信息基础设施的安全非常重要，它甚至关系到国家的安全和社会的稳定。

目前虽然使用了防火墙、访问控制、加密产品、虚拟专用网(VPN)、防病毒软件等技术来提高信息系统的安全，但由于如下原因它们仍不足以完全抵御入侵：第一，防火墙虽充当了外部网和内部网之间的屏障，但并不是所有的外部访问都要通过防火墙，此外安全威胁也可能来自内部，而防火墙本身也极易被外部黑客攻破；第二，软件开发商为了抢占市场，过早推出了不很稳定的产品，致使许多大型软件（如服务器守护程序、客户端应用程序、操作系统等）都存在很多致命的缺陷，它们极易被黑客利用；第三，由于用户的口令设置过于简单或管理不善，易被攻破[1]，许多基于口令的认证系统并不安全；第四，TCP/IP协议标准IPV4在设计上对安全考虑不足[2]，存在许多漏洞，使得路由攻击、ARP欺骗、IP欺骗和多种拒绝服务攻击不可避免，不少应用协议（如FTP，Telnet等）的认证口令采用明文传输，极易被窃听；第五，授权用户滥用权限，危害信息安全。调查表明，80%的网络安全事件来自内部人员。因此，为保证计算机系统的安全，需要一种能及时发现入侵，成功阻止网络黑客入侵，并在事后对入侵进行分析，查明系统漏洞并及时修补的网络安全技术，即入侵检测。

国际上信息安全研究起步早，力度大，积累多，应用广。20世纪80年代，美国国防部基于军事计算机系统的保密需要，在70年代的基础理论研究成果——计算机保密模型（Bell & La padula 模型[3]）的基础上，制定了“可信计算机系统安全评价准则”（TCSEC），其后又制定了关于网络系统、数据库等方面的一系列安全标准，形成了安全信息系统体系结构的最早原则。90年代初，英、法、德、荷四国针对TCSEC准则只考虑保密性的局限性，联合提出包括保密性、完整性、

# 第一章 引言

## 1.1 计算机网络安全现状

近年来信息革命席卷全球，网络化浪潮汹涌而至，特别是互联网的爆炸式发展，给人类社会、经济、文化等带来了无限的机遇，但也给信息安全带来了严峻挑战，各种不安全因素的大量涌入致使计算机网络安全问题日益突出。2000年2月在短短的3天时间里美国数家顶级互联网站受到黑客攻击，陷入瘫痪。据估算，袭击所造成的经济损失大约在12亿美元以上，其中受害攻击3天内的损失高达10多亿美元的市场价值，营销和广告收入损失1亿美元以上。可见，保障计算机网络系统及整个信息基础设施的安全非常重要，它甚至关系到国家的安全和社会的稳定。

目前虽然使用了防火墙、访问控制、加密产品、虚拟专用网(VPN)、防病毒软件等技术来提高信息系统的安全，但由于如下原因它们仍不足以完全抵御入侵：第一，防火墙虽充当了外部网和内部网之间的屏障，但并不是所有的外部访问都要通过防火墙，此外安全威胁也可能来自内部，而防火墙本身也极易被外部黑客攻破；第二，软件开发商为了抢占市场，过早推出了不很稳定的产品，致使许多大型软件（如服务器守护程序、客户端应用程序、操作系统等）都存在很多致命的缺陷，它们极易被黑客利用；第三，由于用户的口令设置过于简单或管理不善，易被攻破[1]，许多基于口令的认证系统并不安全；第四，TCP/IP协议标准IPV4在设计上对安全考虑不足[2]，存在许多漏洞，使得路由攻击、ARP欺骗、IP欺骗和多种拒绝服务攻击不可避免，不少应用协议（如FTP，Telnet等）的认证口令采用明文传输，极易被窃听；第五，授权用户滥用权限，危害信息安全。调查表明，80%的网络安全事件来自内部人员。因此，为保证计算机系统的安全，需要一种能及时发现入侵，成功阻止网络黑客入侵，并在事后对入侵进行分析，查明系统漏洞并及时修补的网络安全技术，即入侵检测。

国际上信息安全研究起步早，力度大，积累多，应用广。20世纪80年代，美国国防部基于军事计算机系统的保密需要，在70年代的基础理论研究成果——计算机保密模型（Bell & La padula 模型[3]）的基础上，制定了“可信计算机系统安全评价准则”（TCSEC），其后又制定了关于网络系统、数据库等方面的一系列安全标准，形成了安全信息系统体系结构的最早原则。90年代初，英、法、德、荷四国针对TCSEC准则只考虑保密性的局限性，联合提出包括保密性、完整性、



可用性概念的“信息技术安全评价准则”(ITSEC[3])。近年来六国七方共同提出了“信息技术安全评价通用准则”(CC for IT SEC[3])。CC 综合了国际上已有的评审准则和技术标准的精华,给出了框架和原则要求。然而,作为将取代 TCSEC 用于系统安全评测的国际标准,它仍然缺少综合解决信息的多种安全属性的理论模型依据。同时,某些高安全级别的产品对我国是封锁的。

我国的信息安全研究经历了通信保密、计算机数据保护两个发展阶段,现已进入网络信息安全的研究阶段。在安全体系的构建和评估方面,我国通过学习、吸收、消化 TCSEC 标准进行了安全操作系统、多级安全数据库的研制。在学习借鉴国外技术的基础上,国内一些部门也开发研制了一些防火墙、安全路由器、安全网关、黑客入侵检测、系统脆弱性扫描软件等。但是,这些产品安全技术的规范性、多接口的满足性方面与国外先进产品存在很大差距,理论基础和自主的技术手段也亟待发展和强化。总的来说,我国的网络信息安全研究起步晚,投入少,研究力量分散,与技术先进国家相比还有不小的差距。

## 1.2 入侵和入侵检测

入侵检测的研究最早可追溯到 1980 年,James Aderson[4]首先在文章“Computer Security Threat Monitoring and Surveillance”里引入了入侵检测这个概念,他的想法为入侵检测的设计和发展奠定了基础。1987 年 Doronthy.E.Denning[5]提出入侵检测系统(Intrusion Detection System, IDS)的抽象模型,首次将入侵检测的概念作为一种计算机系统安全防御问题的措施提出。1988 年的 Morris Internet 蠕虫事件导致许多 IDS 系统的开发研制,Teresa Lunt[6]等人进一步改进了 Denning 提出的入侵检测模型,并创建了 IDES(Intrusion Detection Expert System),该系统用于检测对单一主机的入侵尝试,提出了与系统平台无关的实时检测思想,1995 年开发了 IDES 完善后的版本—NIDES(Next-Generation Intrusion Detection System)[7],可以检测多个主机上的入侵,近年来还有人把遗传算法运用在入侵检测中。

入侵(Intrusion)是个广义的概念,不仅包括攻击者(如恶意的黑客)取得超出合法范围的系统控制权,也包括信息泄露,拒绝访问(Denial of Service)等对计算机系统造成危害的行为。按照最终实施攻击的手段可将网络入侵技术分为获取技术和破坏技术两大类:

- 获取技术

网络信息获取技术就是利用非正常、非授权的途径获取目标网络系统中的程

可用性概念的“信息技术安全评价准则”(ITSEC[3])。近年来六国七方共同提出了“信息技术安全评价通用准则”(CC for IT SEC[3])。CC 综合了国际上已有的评审准则和技术标准的精华,给出了框架和原则要求。然而,作为将取代 TCSEC 用于系统安全评测的国际标准,它仍然缺少综合解决信息的多种安全属性的理论模型依据。同时,某些高安全级别的产品对我国是封锁的。

我国的信息安全研究经历了通信保密、计算机数据保护两个发展阶段,现已进入网络信息安全的研究阶段。在安全体系的构建和评估方面,我国通过学习、吸收、消化 TCSEC 标准进行了安全操作系统、多级安全数据库的研制。在学习借鉴国外技术的基础上,国内一些部门也开发研制了一些防火墙、安全路由器、安全网关、黑客入侵检测、系统脆弱性扫描软件等。但是,这些产品安全技术的规范性、多接口的满足性方面与国外先进产品存在很大差距,理论基础和自主的技术手段也亟待发展和强化。总的来说,我国的网络信息安全研究起步晚,投入少,研究力量分散,与技术先进国家相比还有不小的差距。

## 1.2 入侵和入侵检测

入侵检测的研究最早可追溯到 1980 年,James Aderson[4]首先在文章“Computer Security Threat Monitoring and Surveillance”里引入了入侵检测这个概念,他的想法为入侵检测的设计和发展奠定了基础。1987 年 Doronthy.E.Denning[5]提出入侵检测系统(Intrusion Detection System, IDS)的抽象模型,首次将入侵检测的概念作为一种计算机系统安全防御问题的措施提出。1988 年的 Morris Internet 蠕虫事件导致许多 IDS 系统的开发研制,Teresa Lunt[6]等人进一步改进了 Denning 提出的入侵检测模型,并创建了 IDES(Intrusion Detection Expert System),该系统用于检测对单一主机的入侵尝试,提出了与系统平台无关的实时检测思想,1995 年开发了 IDES 完善后的版本—NIDES(Next-Generation Intrusion Detection System)[7],可以检测多个主机上的入侵,近年来还有人把遗传算法运用在入侵检测中。

入侵(Intrusion)是个广义的概念,不仅包括攻击者(如恶意的黑客)取得超出合法范围的系统控制权,也包括信息泄露,拒绝访问(Denial of Service)等对计算机系统造成危害的行为。按照最终实施攻击的手段可将网络入侵技术分为获取技术和破坏技术两大类:

- 获取技术

网络信息获取技术就是利用非正常、非授权的途径获取目标网络系统中的程



序、数据、运行结果甚至运行过程，常用技术包括：

### 1. 嗅探器技术

嗅探器使网络接口处于广播状态，从网络上截获传输的内容，其中可能包括网络上传输的秘密信息（如电子邮件、用户注册的 ID 及口令等）。嗅探器既可以用作进攻武器，也可以用作防御工具。当作为进攻武器时，它可以截获流经本地的数据，然后利用它们进入未授权系统。当作为防御工具时，它可以发现部分对本机的进攻信息，以便做出诊断和对策。嗅探器对信息的“窃取”是被动的，在网络上不会留下明显的痕迹，但其嗅探范围相当有限，仅对本网段流经嗅探器的数据包有效。

### 2. 扫描器技术

扫描器是一种重要的信息获取工具，根据其扫描的范围可分为网络扫描器（Network Scanner）和系统扫描器（System Scanner）。网络扫描器能自动监测任意规模的网络，发现其安全弱点，分析易损性条件，提供一系列的纠正措施、趋势分析、配置数据等，网络扫描器是在网络级通过对网络设备进行扫描来完成任务。系统扫描器与网络扫描器不同，它在系统级对系统进行扫描以发现其易损性，并提供详细的扫描结果。一般而言，网络扫描器主要用于获取对方网络的配置信息和设备信息，辅助对其实施攻击，而系统扫描器主要用作防御工具。

### 3. 特洛伊木马

特洛伊木马（Trojan Horse）是一个“伪装友善的、恶意攻击安全系统的程序”，在运行这些程序时，隐藏于其中的程序会发起攻击，如窃取帐户和口令，修改磁盘上的文件，利用受害计算机对其他系统实施进一步的攻击等。特洛伊木马与病毒类似，都是一旦受到“感染”，攻击效果非常明显，而且可以在宿主计算机毫无知觉的情况下传播给其他计算机。

#### ● 破坏技术

网络信息破坏技术旨在导致系统瘫痪、数据完整性失效或者服务可用性降级等，以致系统无法为用户提供服务。常用技术包括：

#### 1. 病毒和蠕虫

病毒和蠕虫在原理上可以说是同一类技术，都是可以自我复制的恶意代码。但是在实际使用上它们采用的技术和需要的宿主环境并不完全相同。随着个人计算机和多任务工作站的发展，计算环境的差别在不断缩小，病毒和蠕虫技术也在相辅相成，共同发展。

#### 2. 欺骗技术

欺骗技术主要指 IP 欺骗，它通过伪造可信任 IP 地址的数据分组，利用主机

间的自动认证机制,使对方主机信任攻击者,从而使攻击者有机会破坏目标系统。由于 IP 的身份认证仅仅通过地址来完成,所以伪造 IP 分组比较容易,而猜测 TCP 序列号则相对比较困难。由此防御 IP 欺骗可以用随机产生序列号,数据加密等措施。IPv6 中采用的 IPSec 机制也可以抵御 IP 欺骗的攻击。与 IP 欺骗类似的还有 DNS 欺骗、Email 欺骗、Web 欺骗等。Web 欺骗是一种新的入侵手段,攻击者可以监视和修改所有传给目标主机的 Web 页面,监视所有填入表格的信息,所以采用所谓安全的连接无法阻止攻击,而且攻击者很难被发现。

### 3. 拒绝服务攻击技术

用户对计算机系统安全的最低要求就是“可用性”,软硬件必须能够协调地工作并提供一定的服务。拒绝服务攻击正是破坏系统的可用性,有意阻塞、降低计算机或网络资源的服务,以达到攻击目的。这种攻击方式并不一定是直接、永久地破坏数据本身,而是破坏资源的可用性。

有矛就有盾,有攻击就有防御。目前,我们通过使用防火墙、用户认证和授权(I&A)、访问控制、信息加密、虚拟专用网(VPN)等安全技术来提高信息系统的安全性能,但由于如下原因它们仍不足以完全抵御入侵:首先,仅有防火墙的防护措施是不够的,因为防火墙完全不能阻止内部攻击,对于别有用心的内部人员来说防火墙形同虚设;其次,由于用户的口令设置过于简单或管理不善,很容易被破坏,所以许多基于口令的认证系统并不安全;第三,TCP/IP 协议标准 IPv4 在设计上对安全问题考虑不足,导致存在许多协议实现漏洞,使得路由攻击、ARP 攻击、IP 欺骗和多种拒绝式服务攻击不可避免,不少应用协议(如 FTP、Telnet 等)的认证口令采用明文传输,极易被窃听。虽然现在 IPSec 协议可以克服上述设计缺陷,但目前还没有得到广泛应用;第四,授权用户滥用权限,危害信息安全。因此,为保证计算机系统的安全,一种能及时发现入侵,成功阻止入侵,并在事后对入侵进行分析,查明系统漏洞并及时修补的网络安全技术—入侵检测系统应运而生。

入侵检测技术是继“防火墙”、“数据加密”等传统安全保护措施后提出的新一代安全保障技术。计算机系统各部分在设计、实现和部署中会给系统带来漏洞,因为没有经济可行的手段完全消除这些隐患,所以有效的入侵检测手段对于保证系统安全是必不可少的。即使一个系统中不存在某个特定的漏洞,入侵检测系统仍然可以检测到相应的攻击事件,并调整系统状态对未来可能发生的入侵发出警告。

入侵检测是防火墙的合理补充,能够帮助系统对付网络攻击,扩展了系统管理员的安全管理能力(包括安全审计、监视、进攻识别和响应),提高了信息安全

基础结构的完整性。入侵检测被认为是防火墙之后的第二道安全闸门，在尽量不影响网络性能的情况下对网络进行监测，从而提供内部攻击、外部攻击和误操作的实时保护。这些都通过入侵检测系统执行以下任务来实现：

- 监视、分析用户及系统活动；
- 系统构造和弱点的审计；
- 识别反映已知进攻的活动模式并向相关人士报警；
- 异常行为模式的统计分析；
- 评估重要系统和数据文件的完整性；
- 操作系统的审计跟踪管理，并识别用户违反安全策略的行为。

对一个成功的入侵检测系统来讲，它不但可使系统管理员时刻了解网络系统（包括程序、文件和硬件设备等）的任何变更，还能给网络安全策略的制定提供指南。更为重要的一点是，它应该易于管理、配置，从而使非专业人员也能非常容易地获得网络安全。而且，入侵检测的规模还应根据网络威胁、系统构造和安全需求的改变而改变。入侵检测系统在发现入侵后，会及时做出响应，包括切断网络连接、记录事件和报警等。

### 1.3 入侵检测的研究现状

国外机构早在 20 世纪 80 年代就开展了相关基础理论研究工作，随着计算机系统软、硬件的飞速发展，以及网络技术、分布式计算、系统工程、人工智能等计算机新兴技术与理论的不断发展与完善，入侵检测理论本身也处于发展变化中，但至今还未形成一个比较完整的成熟的理论体系。

1987 年，Dorothy Denning 提出的理论构架启发了很多研究者，并奠定了近期商业产品的基础。Dorothy Denning 在文中介绍了一个不依赖于特殊系统、应用环境、系统缺陷和入侵类型的通用型入侵检测模型，如图 1.1[8]所示：

基础结构的完整性。入侵检测被认为是防火墙之后的第二道安全闸门，在尽量不影响网络性能的情况下对网络进行监测，从而提供内部攻击、外部攻击和误操作的实时保护。这些都通过入侵检测系统执行以下任务来实现：

- 监视、分析用户及系统活动；
- 系统构造和弱点的审计；
- 识别反映已知进攻的活动模式并向相关人士报警；
- 异常行为模式的统计分析；
- 评估重要系统和数据文件的完整性；
- 操作系统的审计跟踪管理，并识别用户违反安全策略的行为。

对一个成功的入侵检测系统来讲，它不但可使系统管理员时刻了解网络系统（包括程序、文件和硬件设备等）的任何变更，还能给网络安全策略的制定提供指南。更为重要的一点是，它应该易于管理、配置，从而使非专业人员也能非常容易地获得网络安全。而且，入侵检测的规模还应根据网络威胁、系统构造和安全需求的改变而改变。入侵检测系统在发现入侵后，会及时做出响应，包括切断网络连接、记录事件和报警等。

### 1.3 入侵检测的研究现状

国外机构早在 20 世纪 80 年代就开展了相关基础理论研究工作，随着计算机系统软、硬件的飞速发展，以及网络技术、分布式计算、系统工程、人工智能等计算机新兴技术与理论的不断发展与完善，入侵检测理论本身也处于发展变化中，但至今还未形成一个比较完整的成熟的理论体系。

1987 年，Dorothy Denning 提出的理论构架启发了很多研究者，并奠定了近期商业产品的基础。Dorothy Denning 在文中介绍了一个不依赖于特殊系统、应用环境、系统缺陷和入侵类型的通用型入侵检测模型，如图 1.1[8]所示：



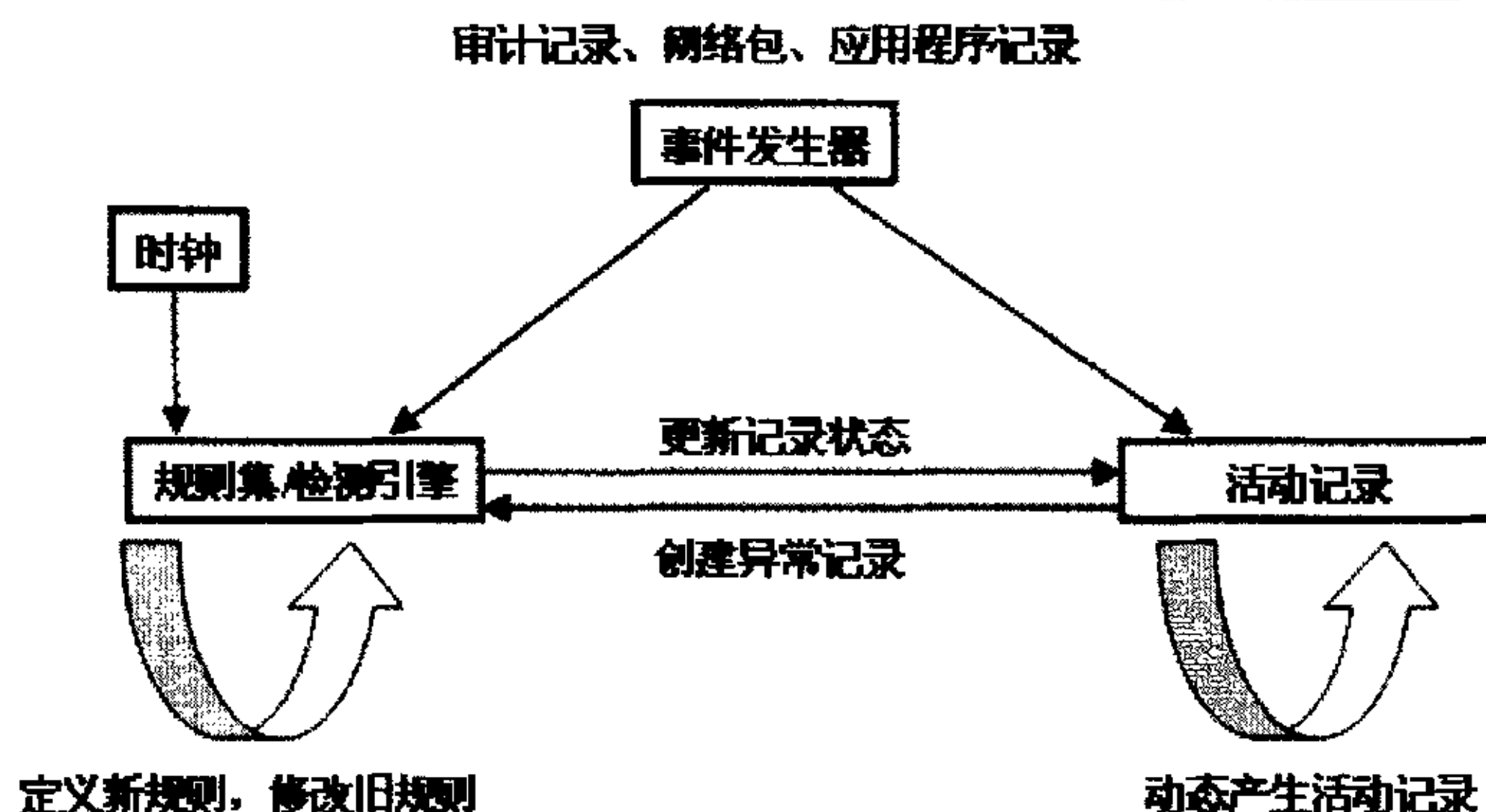


图 1.1 通用入侵检测模型

Fig. 1.1 Common Intrusion Detection Framework

其基本思路是入侵者的行为和合法用户的异常行为是可以从系统合法用户的正常行为区别出来的。为了定义一个用户的正常行为就必须为这个用户建立和维护一系列的行为轮廓配置，这些配置描述了用户正常使用系统的行为特征。IDS 可以利用这些配置监控当前用户活动，并与以前的用户活动进行比较，当用户的当前活动与以往活动的差别超出某些预定义的“边界”条件（轮廓配置的各项门限值）时，这个当前活动行为就认为是异常的，并且它很可能就是一个入侵行为。

随着人工智能、分布式技术等引入，特别是 Internet 的膨胀，入侵、攻击事件的频繁发生，网络用户迫切要求实时的、智能的入侵检测系统，这些需求进一步推动了入侵检测的发展。在国内，随着上网的关键部门、关键业务越来越多，更需要具有自主知识产权的入侵检测产品。但是我国的入侵检测研究仅仅停留在实验室和实验样品（缺乏升级和服务）阶段，或者是防火墙集成较为初级的入侵检测模块，可见入侵检测技术仍具有较大的发展空间。

## 1.4 课题来源及论文贡献

本课题来源主要有两点：首先，近几年来互联网得到了迅猛发展，网络干线基本都采用光纤作为传输设备，导致网络带宽较以前有显著提高，网络速度也越来越快，所以，对入侵检测系统来说就需要提高它的检测效率，而提高系统效率的根本就是要对入侵检测系统的核心：检测引擎，进行优化改进。另外，目前的大多数商业运营的入侵检测系统基本上采用了模式匹配的检测方法，经过在商业

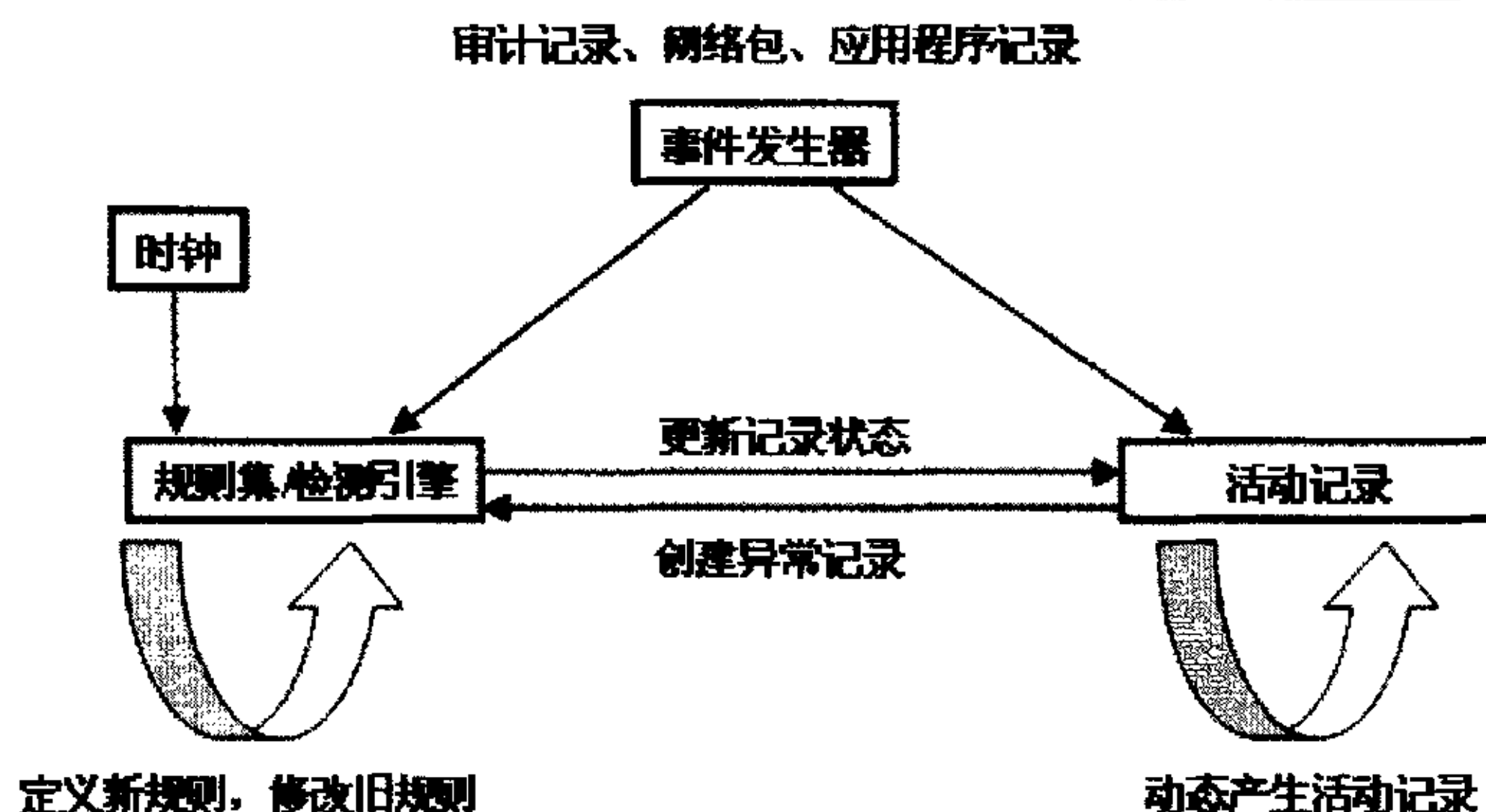


图 1.1 通用入侵检测模型

Fig. 1.1 Common Intrusion Detection Framework

其基本思路是入侵者的行为和合法用户的异常行为是可以从系统合法用户的正常行为区别出来的。为了定义一个用户的正常行为就必须为这个用户建立和维护一系列的行为轮廓配置，这些配置描述了用户正常使用系统的行为特征。IDS 可以利用这些配置监控当前用户活动，并与以前的用户活动进行比较，当用户的当前活动与以往活动的差别超出某些预定义的“边界”条件（轮廓配置的各项门限值）时，这个当前活动行为就认为是异常的，并且它很可能就是一个入侵行为。

随着人工智能、分布式技术等引入，特别是 Internet 的膨胀，入侵、攻击事件的频繁发生，网络用户迫切要求实时的、智能的入侵检测系统，这些需求进一步推动了入侵检测的发展。在国内，随着上网的关键部门、关键业务越来越多，更需要具有自主知识产权的入侵检测产品。但是我国的入侵检测研究仅仅停留在实验室和实验样品（缺乏升级和服务）阶段，或者是防火墙集成较为初级的入侵检测模块，可见入侵检测技术仍具有较大的发展空间。

## 1.4 课题来源及论文贡献

本课题来源主要有两点：首先，近几年来互联网得到了迅猛发展，网络干线基本都采用光纤作为传输设备，导致网络带宽较以前有显著提高，网络速度也越来越快，所以，对入侵检测系统来说就需要提高它的检测效率，而提高系统效率的根本就是要对入侵检测系统的核心：检测引擎，进行优化改进。另外，目前的大多数商业运营的入侵检测系统基本上采用了模式匹配的检测方法，经过在商业



系统中的广泛使用，对该种检测方法研究已经相当成熟，并且基于模式匹配算法的入侵检测系统检测效果好、易于设计实现、而且扩展性能好。

Snort 是典型的基于规则匹配的入侵检测系统，为提高入侵检测系统的检测效率和性能，本人通过对 Snort 系统进行分析，重点对 IDS 中的模式匹配算法的选用进行了研究，运用了基于数值查找的多模式匹配算法——Long-Karp-Rabin 算法，并在此基础上，对 Long-Karp-Rabin 算法的哈希表中节点的单链表线性结构进行改进，采用了二次散列结合链表的方式存储产生哈希碰撞的节点，理论上优化了算法，并通过实验，证明了改进后的多模式匹配算法可以加快入侵检测系统的效率。本人希望可以对基于规则匹配的 IDS 发展提供借鉴。

## 第二章 入侵检测概述

### 2.1 入侵检测的工作流程

入侵检测系统是一个典型的“窥探设备”，它不跨接多个物理网段（通常只有一个监听端口），无需转发任何流量，而只需要在网络上被动的、无声息的收集它所关心的报文即可。对收集来的报文入侵检测系统提取相应的流量，统计特征值，并利用内置的入侵知识库与这些流量特征进行智能分析、比较匹配。根据预设的阈值，匹配耦合度较高的报文流量将被认为是入侵行为，系统将进行报警或有限度的反击，入侵检测的工作流程主要有以下四步。

#### 2.1.1 信息收集

入侵检测的第一步是信息收集，内容包括系统、网络、数据及用户活动的状态和行为，而且需要在计算机网络系统中的若干不同关键点（不同网段和不同主机）收集信息，这除了尽可能扩大检测范围的因素外，还有一个重要的因素就是从单一源来的信息有可能看不出疑点，但从几个源来的信息的不一致性却是可疑行为或入侵的最好标识。当然，入侵检测很大程度上依赖于收集信息的可靠性和正确性，因此很有必要只利用所知道的真正的和精确的软件来报告这些信息，因为黑客经常替换软件修改和移走这些信息，例如替换被程序调用的子程序、库和其它工具。黑客对系统的修改可能使系统功能失常并看起来跟正常的一样，而实际上不是，例如，UNIX 系统的 PS 指令可以被替换为一个不显示侵入过程的指令，或者是编辑器被替换成一个读取不同于指定文件的文件（黑客隐藏了初始文件并用另一版本代替），这需要保证用来检测网络系统的软件的完整性，特别是入侵检测系统软件本身应具有相当强的坚固性，防止被篡改而收集到错误的信息。

入侵检测利用的信息一般来自以下四个方面：

- 系统和网络日志文件

黑客经常在系统日志文件中留下他们的踪迹，因此，充分利用系统和网络日志文件信息是检测入侵的必要条件。日志中包含发生在系统和网络上的不寻常和不期望活动的证据，这些证据可以指出有人正在入侵或已成功入侵了系统。通过查看日志文件，能够发现成功的入侵或入侵企图，并能很快启动相应的应急响应程序。日志文件中记录了各种行为类型，每种类型又包含不同的信息，例如记录“用户活动”类型的日志，就包含登录、用户 ID 改变、用户对文件的访问、授

## 第二章 入侵检测概述

### 2.1 入侵检测的工作流程

入侵检测系统是一个典型的“窥探设备”，它不跨越多个物理网段（通常只有一个监听端口），无需转发任何流量，而只需要在网络上被动的、无声息的收集它所关心的报文即可。对收集来的报文入侵检测系统提取相应的流量，统计特征值，并利用内置的入侵知识库与这些流量特征进行智能分析、比较匹配。根据预设的阈值，匹配耦合度较高的报文流量将被认为是入侵行为，系统将进行报警或有限度的反击，入侵检测的工作流程主要有以下四步。

#### 2.1.1 信息收集

入侵检测的第一步是信息收集，内容包括系统、网络、数据及用户活动的状态和行为，而且需要在计算机网络系统中的若干不同关键点（不同网段和不同主机）收集信息，这除了尽可能扩大检测范围的因素外，还有一个重要的因素就是从单一源来的信息有可能看不出疑点，但从几个源来的信息的不一致性却是可疑行为或入侵的最好标识。当然，入侵检测很大程度上依赖于收集信息的可靠性和正确性，因此很有必要只利用所知道的真正的和精确的软件来报告这些信息，因为黑客经常替换软件修改和移走这些信息，例如替换被程序调用的子程序、库和其它工具。黑客对系统的修改可能使系统功能失常并看起来跟正常的一样，而实际上不是，例如，UNIX 系统的 PS 指令可以被替换为一个不显示侵入过程的指令，或者是编辑器被替换成一个读取不同于指定文件的文件（黑客隐藏了初始文件并用另一版本代替），这需要保证用来检测网络系统的软件的完整性，特别是入侵检测系统软件本身应具有相当强的坚固性，防止被篡改而收集到错误的信息。

入侵检测利用的信息一般来自以下四个方面：

- 系统和网络日志文件

黑客经常在系统日志文件中留下他们的踪迹，因此，充分利用系统和网络日志文件信息是检测入侵的必要条件。日志中包含发生在系统和网络上的不寻常和不期望活动的证据，这些证据可以指出有人正在入侵或已成功入侵了系统。通过查看日志文件，能够发现成功的入侵或入侵企图，并能很快启动相应的应急响应程序。日志文件中记录了各种行为类型，每种类型又包含不同的信息，例如记录“用户活动”类型的日志，就包含登录、用户 ID 改变、用户对文件的访问、授

权和认证信息等内容。很显然，对用户活动来讲，不正常的或不期望的行为就是重复登录失败、登录到不期望的位置以及非授权的企图访问重要文件等等。

- 目录和文件中的不期望的改变

网络环境中的文件系统包含很多软件和数据文件，包含重要信息的文件和私有数据文件经常是黑客修改或破坏的目标。目录和文件中的不期望的改变（包括修改、创建和删除），特别是那些正常情况下限制访问的，很可能就是一种入侵产生的指示和信号。黑客经常替换、修改和破坏获得访问权的系统上的文件，同时为了隐藏系统中的活动痕迹，他们都会尽力去替换系统程序或修改系统日志文件。

- 程序执行中的不期望行为

网络系统上的程序执行一般包括操作系统、网络服务、用户起动的程序和特定目的的应用，例如数据库服务器。每个在系统上执行的程序由一到多个进程来实现。每个进程执行在具有不同权限的环境中，这种环境控制着进程可访问的系统资源、程序和数据文件等。一个进程的执行行为由它运行时执行的操作来表现，操作执行的方式不同，它利用的系统资源也就不同，操作包括计算、文件传输、设备和其它进程，以及与网络间其它进程的通讯。一个进程出现了不期望的行为，可能表明黑客正在入侵你的系统，黑客可能会将程序或服务的运行分解，从而导致它失败，或者是以非用户或管理员意图的方式操作。

- 物理形式的入侵信息

物理形式的入侵信息包括两个方面的内容，一是未授权的对网络硬件连接；二是对物理资源的未授权访问。黑客会想方设法去突破网络的周边防卫，如果他们能够在物理上访问内部网，就能安装他们自己的设备和软件。依此，黑客就可以知道网上的由用户加上不安全（未授权）设备，然后利用这些设备访问网络。例如，用户在家里可能安装 Modem 以访问远程办公室，同时黑客正在利用自动工具来识别在公共电话线上的 Modem，如果一个拨号访问流量经过了这些自动工具，那么这个拨号访问就成为了威胁网络安全后门。黑客就会利用这个后门来访问内部网，从而越过内部网络原有的防护措施，然后捕获网络流量，进而攻击其它系统，并偷取敏感的私有信息等等。

## 2.1.2 信息分析

信息分析是指对上述四类收集到的有关系统、网络、数据及用户活动的状态和行为等信息，一般通过模式匹配、统计分析和完整性分析三种技术手段进行分析。其中前两种方法用于实时的入侵检测，而完整性分析则用于事后分析。

- 模式匹配



权和认证信息等内容。很显然，对用户活动来讲，不正常的或不期望的行为就是重复登录失败、登录到不期望的位置以及非授权的企图访问重要文件等等。

### ● 目录和文件中的不期望的改变

网络环境中的文件系统包含很多软件和数据文件，包含重要信息的文件和私有数据文件经常是黑客修改或破坏的目标。目录和文件中的不期望的改变（包括修改、创建和删除），特别是那些正常情况下限制访问的，很可能就是一种入侵产生的指示和信号。黑客经常替换、修改和破坏获得访问权的系统上的文件，同时为了隐藏系统中的活动痕迹，他们都会尽力去替换系统程序或修改系统日志文件。

### ● 程序执行中的不期望行为

网络系统上的程序执行一般包括操作系统、网络服务、用户起动的程序和特定目的的应用，例如数据库服务器。每个在系统上执行的程序由一到多个进程来实现。每个进程执行在具有不同权限的环境中，这种环境控制着进程可访问的系统资源、程序和数据文件等。一个进程的执行行为由它运行时执行的操作来表现，操作执行的方式不同，它利用的系统资源也就不同，操作包括计算、文件传输、设备和其它进程，以及与网络间其它进程的通讯。一个进程出现了不期望的行为，可能表明黑客正在入侵你的系统，黑客可能会将程序或服务的运行分解，从而导致它失败，或者是以非用户或管理员意图的方式操作。

### ● 物理形式的入侵信息

物理形式的入侵信息包括两个方面的内容，一是未授权的对网络硬件连接；二是对物理资源的未授权访问。黑客会想方设法去突破网络的周边防卫，如果他们能够在物理上访问内部网，就能安装他们自己的设备和软件。依此，黑客就可以知道网上的由用户加上不安全（未授权）设备，然后利用这些设备访问网络。例如，用户在家里可能安装 Modem 以访问远程办公室，同时黑客正在利用自动工具来识别在公共电话线上的 Modem，如果一个拨号访问流量经过了这些自动工具，那么这个拨号访问就成为了威胁网络安全后门。黑客就会利用这个后门来访问内部网，从而越过内部网络原有的防护措施，然后捕获网络流量，进而攻击其它系统，并偷取敏感的私有信息等等。

## 2.1.2 信息分析

信息分析是指对上述四类收集到的有关系统、网络、数据及用户活动的状态和行为等信息，一般通过模式匹配、统计分析和完整性分析三种技术手段进行分析。其中前两种方法用于实时的入侵检测，而完整性分析则用于事后分析。

### ● 模式匹配

模式匹配就是将收集到的信息与已知的网络入侵和系统误用模式数据库进行比较,从而发现违背安全策略的行为。该过程可以很简单(如通过字符串匹配以寻找一个简单的条目或指令),也可以很复杂(如利用正规的数学表达式来表示安全状态的变化)。一般来讲,一种进攻模式可以用一个过程(如执行一条指令)或一个输出(如获得权限)来表示。该方法的一大优点是只需收集相关的数据集合,显著减少了系统负担,且技术相当成熟。它与病毒防火墙采用的方法一样,检测准确率和效率都相当高。但是,该方法存在的弱点是需要不断的升级以对付不断出现的黑客攻击手法,不能检测到从未出现过的黑客攻击手段。

### ● 统计分析

统计分析方法首先给系统对象(如用户、文件、目录和设备等)创建一个统计描述,统计正常使用时的一些测量属性(如访问次数、操作失败次数和延时等)。测量属性的平均值将被用来与网络、系统的行为进行比较,任何观察值在正常值范围之外时,就认为有入侵发生。例如,统计分析可能标识一个不正常行为,因为它发现一个在晚八点至早六点未登录的账户却在凌晨两点试图登录。其优点是可检测到未知的入侵和更为复杂的入侵,缺点是误报、漏报率高,且不适应用户正常行为的突然改变。具体的统计分析方法如基于专家系统的、基于模型推理的和基于神经网络的分析方法,目前正处于研究热点和迅速发展之中。

### ● 完整性分析

完整性分析主要关注某个文件或对象是否被更改,这经常包括文件和目录的内容及属性,它在发现被更改的、被特洛伊化的应用程序方面特别有效。完整性分析利用强有力的加密机制,称为消息摘要函数(例如 MD5),它能识别哪怕是微小的变化。其优点是不管模式匹配方法和统计分析方法能否发现入侵,只要是成功的攻击导致了文件或其它对象的任何改变,它都能够发现。缺点是一般以批处理方式实现,不用于实时响应。尽管如此,完整性检测方法还应该是网络安全产品的必要手段之一。例如,可以在每一天的某个特定时间内开启完整性分析模块,对网络系统进行全面扫描检查。

## 2.1.3 信息存储

当入侵检测系统捕获到有攻击发生时,为了便于系统管理人员对攻击信息进行查看和对攻击行为进行分析,还需要将入侵检测系统收集到的信息进行保存,这些信息通常存储到用户指定的日志文件中,同时存储的信息也为攻击保留了数字证据。



模式匹配就是将收集到的信息与已知的网络入侵和系统误用模式数据库进行比较,从而发现违背安全策略的行为。该过程可以很简单(如通过字符串匹配以寻找一个简单的条目或指令),也可以很复杂(如利用正规的数学表达式来表示安全状态的变化)。一般来讲,一种进攻模式可以用一个过程(如执行一条指令)或一个输出(如获得权限)来表示。该方法的一大优点是只需收集相关的数据集合,显著减少了系统负担,且技术相当成熟。它与病毒防火墙采用的方法一样,检测准确率和效率都相当高。但是,该方法存在的弱点是需要不断的升级以对付不断出现的黑客攻击手法,不能检测到从未出现过的黑客攻击手段。

### ● 统计分析

统计分析方法首先给系统对象(如用户、文件、目录和设备等)创建一个统计描述,统计正常使用时的一些测量属性(如访问次数、操作失败次数和延时等)。测量属性的平均值将被用来与网络、系统的行为进行比较,任何观察值在正常值范围之外时,就认为有入侵发生。例如,统计分析可能标识一个不正常行为,因为它发现一个在晚八点至早六点未登录的账户却在凌晨两点试图登录。其优点是可检测到未知的入侵和更为复杂的入侵,缺点是误报、漏报率高,且不适应用户正常行为的突然改变。具体的统计分析方法如基于专家系统的、基于模型推理的和基于神经网络的分析方法,目前正处于研究热点和迅速发展之中。

### ● 完整性分析

完整性分析主要关注某个文件或对象是否被更改,这经常包括文件和目录的内容及属性,它在发现被更改的、被特洛伊化的应用程序方面特别有效。完整性分析利用强有力的加密机制,称为消息摘要函数(例如 MD5),它能识别哪怕是微小的变化。其优点是不管模式匹配方法和统计分析方法能否发现入侵,只要是成功的攻击导致了文件或其它对象的任何改变,它都能够发现。缺点是一般以批处理方式实现,不用于实时响应。尽管如此,完整性检测方法还应该是网络安全产品的必要手段之一。例如,可以在每一天的某个特定时间内开启完整性分析模块,对网络系统进行全面扫描检查。

## 2.1.3 信息存储

当入侵检测系统捕获到有攻击发生时,为了便于系统管理人员对攻击信息进行查看和对攻击行为进行分析,还需要将入侵检测系统收集到的信息进行保存,这些信息通常存储到用户指定的日志文件中,同时存储的信息也为攻击保留了数字证据。

### 2.1.4 攻击响应

对攻击信息进行了分析并确定攻击的类型后，入侵检测系统会根据用户的设置，对攻击行为进行相应的处理：如发出警报、给系统管理员发邮件等方式提醒用户，或者利用自动装置直接进行处理：如切断连接，过滤攻击者的 IP 地址等，从而使系统能够较早的避开或阻断攻击。

## 2.2 入侵检测采取的策略

入侵检测的目标是从信息特征中鉴别出攻击但不误报，入侵检测可以看作是传统的信号检测问题。入侵相当于被检测的信号，而“正常情况”相当于噪声。决策过程就是要分辨出观察到的是只有噪声还是噪声中有信号。典型的入侵检测器的判断是基于信号特征（基于特征的检测器）或是噪声（基于异常的检测器）。

### ● 基于特征的入侵检测

基于特征的入侵检测系统（Signature-based IDS）一般都有一个特征库，库中存储着对已知攻击行为的特征描述，当从一个或多个网络数据包中提取出特定模式时，如果发现这些模式中有与已知攻击模式相匹配的，入侵检测系统就知道有攻击行为发生。攻击行为特征可以通过多种方法取得：手工提取、利用传感器进行自动训练或学习等。因为特征是从已知攻击行为抽象而来的，所以系统只能鉴别出已知攻击，不能检测出新型的攻击，同时对特征的错误理解会产生误报。

### ● 基于异常的入侵检测

基于异常的入侵检测系统（Anomaly-based IDS），是基于以下思想：正常工作下的系统有一“基准”，例如 CPU 利用率、磁盘活动、用户注册、文件活动等等，一旦偏离这一“基准”检测系统就被触发，基于异常的检测器把正常的系统环境当作没受干扰的噪声。异常检测的主要优点是能够识别新的攻击，但经常出现对正常操作行为产生变化时误报，对伪装正常的入侵行为反而漏报，并且基于异常检测的系统很难确定入侵攻击的类型。

## 2.3 入侵检测的分类

根据检测方法来分析，现在的入侵检测产品基本可以分为以下三类：事后审计追踪分析，实时包分析和实时行为监测。

### 2.3.1 事后审计追踪分析

早期的入侵检测基本上采用审计追踪分析,如 SAIC 的 CMDS 就是分析 UNIX 的审计记录中的可疑行为。如果这类产品同时实现自动审计追踪和管理的话就更有价值。这样就可以将信息提取或存档,而去掉旧的和不再需要的审计数据。

这类产品有两个主要的优点。一是减少了检查和管理大量的审计数据的巨大困难。另一个是检测人员可以及时地回顾并做历史事件分析。更复杂的产品把结果用图表来表示,显示出入侵的趋势,这对在较长时间内发生的入侵特别有用。

从网络安全的观点来看,纯粹的在“事后”的产品缺点是:在检测到安全问题时再做响应和保护往往已经太迟,入侵可能已经造成了进一步的破坏。另外,大多入侵者懂得如何掩盖他们的行动,可以篡改审计数据,因此事后分析常常会漏掉攻击事件。

### 2.3.2 实时包分析

最初,在网络范围实现实时检测似乎是不可行的,系统做不到既能分析又能保持可接受的系统性能。现在已经有了不少实时检测及响应的入侵检测产品。实时入侵检测采用的一种做法是应用嗅探器(Sniffer),将网卡设置成所谓的混杂模式(通过将网卡设置成混杂模式,可以让网卡不必检查目的 MAC 地址,监听连接在集线器上的所有流量。),就可以捕获一个网段中传输的所有数据包。系统有一个分析机,通过分析数据包的报头和包的内容查找特定的网络攻击的特征,如 IP 欺骗、数据报淹没等等。当检测到可能的危险时,系统马上通过呼叫提醒、发送 Email 等方式向控制台报警,甚至切断网络会话。

实时包分析技术的优点在于:对于基于网络的攻击,最好的办法是通过数据包检查,而且不用在网络中的每一个主机上都放置检测软件。

当然,基于实时包分析的入侵检测也有不足之处:

- 基于数据包分析的入侵检测远离关键应用和它要保护的数据,这样难以清楚入侵在这些地方到底会做些什么动作。
- 数据包分析检测不到(或较难做到)一些攻击,如,在 UNIX 上用缓冲区溢出取得 Root 权限;浏览用户不应当接触的文件;通过拨号攻击关键任务服务器;在系统中插入特洛伊木马;非法使用关键应用;篡改网页内容;非法访问数据库等等。
- 嗅探器需要硬件开销,并且硬件的开销的代价依赖于网络的速度。
- 嗅探器对加密的数据包的分析用处不大。

### 2.3.3 实时行为监测

实时入侵监测的一种有效的方法是监视不同系统上的与安全相关的行为。这种方法距离要保护的资源更近了，除了监视操作系统的审计数据外，还有：

- 跟踪应用、数据库、WEB 服务器、路由器、防火墙等的审计数据；
- 监视关键文件，以发现特洛伊木马、非授权变化等；
- 监视 TCP 和 UDP 端口的活动。

实时行为监测可检测的攻击诸如：非授权访问敏感文件的企图，替代登录程序等。与包嗅探不同的是，它能检测到用户非法取得 Root 或管理员访问权限，当发现可疑行为的时候，在造成危害之前系统就会采取行动，响应的方法包括：在控制台提醒、发送 Email、冻结账户、中断入侵进程和中断入侵者的会话、关闭系统或者执行响应命令过程。

实时行为监测的优点在于：其工作特点更接近关键任务数据和应用，而且所有的设备在监视之中，从网络内外监视攻击的发生变的更容易。

与实时包分析的做法相比，实时行为检测是在较高的层次来实施的，对于在网络层实施的攻击（如拒绝服务、协议栈攻击）就显得力不从心。

## 2.4 入侵检测体系结构

当前，在网络环境中主要存在三种入侵检测体系结构[9]：基于主机的入侵检测系统(HIDS)、基于网络的入侵检测系统(NIDS)和分布式入侵检测系统(DIDS)。

### 2.4.1 基于主机的入侵检测系统（HIDS）

基于主机的入侵检测通常从主机的审计记录和日志文件中获得所需的主要数据源，并辅之以主机上的其他信息，例如文件系统属性、进程状态、CPU 和内存的使用情况等，在此基础上完成检测攻击行为的任务。从技术发展的历程来看，入侵检测是从主机审计的基础上开始发展的，因而早期的入侵检测系统都是基于主机的入侵检测技术。

基于主机的入侵检测能够较为准确地监测到发生在主机系统高层的复杂攻击行为，例如对文件系统所进行的具有潜在安全风险的访问操作序列、对系统配置的修改以及应用程序的异常运行情况等等。同时基于主机的入侵检测系统也有若干显而易见的缺点：首先，由于它严重依赖于特定的操作系统平台，所以，对不同的平台系统而言，它是无法移植的。其次，它在所保护主机上运行，将影响到宿主机的运行性能，特别是当宿主机是服务器的情况。另为，它通常无法对网络

### 2.3.3 实时行为监测

实时入侵监测的一种有效的方法是监视不同系统上的与安全相关的行为。这种方法距离要保护的资源更近了，除了监视操作系统的审计数据外，还有：

- 跟踪应用、数据库、WEB 服务器、路由器、防火墙等的审计数据；
- 监视关键文件，以发现特洛伊木马、非授权变化等；
- 监视 TCP 和 UDP 端口的活动。

实时行为监测可检测的攻击诸如：非授权访问敏感文件的企图，替代登录程序等。与包嗅探不同的是，它能检测到用户非法取得 Root 或管理员访问权限，当发现可疑行为的时候，在造成危害之前系统就会采取行动，响应的方法包括：在控制台提醒、发送 Email、冻结账户、中断入侵进程和中断入侵者的会话、关闭系统或者执行响应命令过程。

实时行为监测的优点在于：其工作特点更接近关键任务数据和应用，而且所有的设备在监视之中，从网络内外监视攻击的发生变的更容易。

与实时包分析的做法相比，实时行为检测是在较高的层次来实施的，对于在网络层实施的攻击（如拒绝服务、协议栈攻击）就显得力不从心。

## 2.4 入侵检测体系结构

当前，在网络环境中主要存在三种入侵检测体系结构[9]：基于主机的入侵检测系统(HIDS)、基于网络的入侵检测系统(NIDS)和分布式入侵检测系统(DIDS)。

### 2.4.1 基于主机的入侵检测系统（HIDS）

基于主机的入侵检测通常从主机的审计记录和日志文件中获得所需的主要数据源，并辅之以主机上的其他信息，例如文件系统属性、进程状态、CPU 和内存的使用情况等，在此基础上完成检测攻击行为的任务。从技术发展的历程来看，入侵检测是从主机审计的基础上开始发展的，因而早期的入侵检测系统都是基于主机的入侵检测技术。

基于主机的入侵检测能够较为准确地监测到发生在主机系统高层的复杂攻击行为，例如对文件系统所进行的具有潜在安全风险的访问操作序列、对系统配置的修改以及应用程序的异常运行情况等等。同时基于主机的入侵检测系统也有若干显而易见的缺点：首先，由于它严重依赖于特定的操作系统平台，所以，对不同的平台系统而言，它是无法移植的。其次，它在所保护主机上运行，将影响到宿主机的运行性能，特别是当宿主机是服务器的情况。另为，它通常无法对网络

环境下发生的大量攻击行为做出及时的反应。

HIDS 的典型结构如图 2.1 所示：

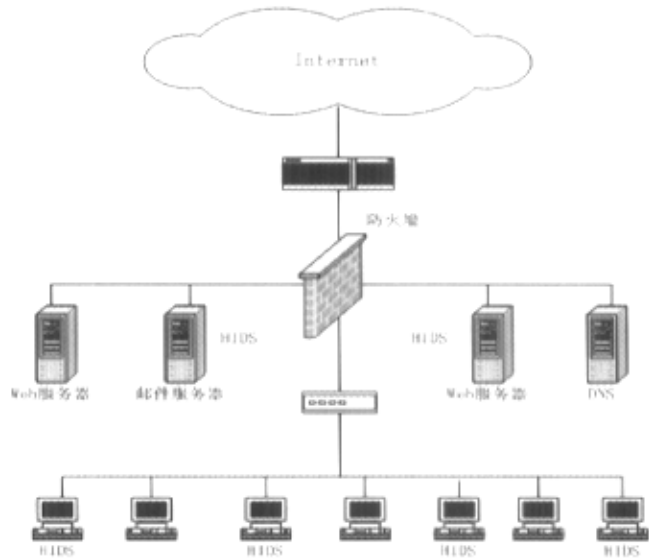


图 2.1 HIDS 网络

Fig. 2.1 HIDS Network

2.4.2 基于网络的入侵检测系统（NIDS）

随着网络环境的普及，出现了大量的基于网络的入侵检测系统。基于网络的入侵检测系统通过监听网络中的数据包来获得必要的数据来源，并通过协议分析、特征匹配、统计分析等手段发现当前攻击行为。

基于网络的入侵检测能够实时监控网络中的数据流量，发现潜在的攻击行为，并做出迅速响应。另外，它的分析对象是网络协议，通常而言是标准化的，独立于主机的操作系统类型，因此 NIDS 一般没有移植性的问题。同时它的运行丝毫不影响主机或服务器的自身运行，因为基于网络的入侵检测系统经常采取独立主机和被动监听的工作模式。基于网络的入侵检测系统，早期的开山之作是 UC Davis 研制的 NSM 系统，其基本架构仍然影响着现在的许多实际系统。

NIDS 的典型结构如图 2.2 所示：



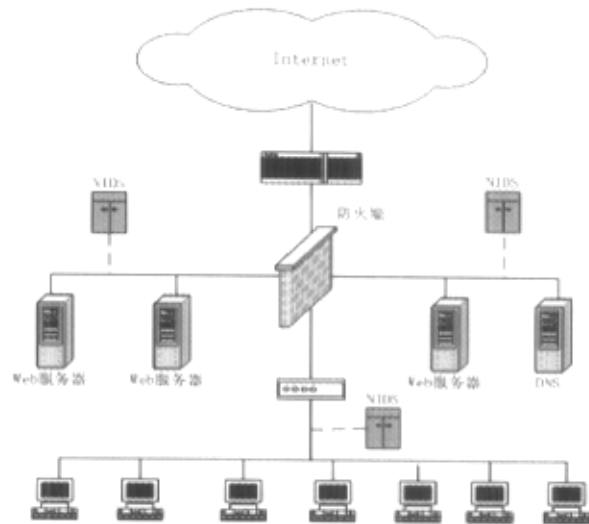


图 2.2 NIDS 网络

Fig. 2.2 NIDS Network

2.4.3 分布式入侵检测系统（DIDS）

进入 20 世纪 90 年代后，出现了把基于主机和基于网络的入侵检测结合起来的早期尝试，最早实现此种集成能力的原型系统是分布式入侵检测系统 DIDS，它将 NSM 和 Haystack 组件集成到一起，并采用中央控制台来解决关联处理和用户接口的问题。

典型的 DIDS 是管理端/探测器结构，NIDS 作为探测器放置在网络的各个地方，并向中央管理平台汇报情况。攻击日志定时地传送到管理平台，并保存在中央数据库中，新的攻击特征库能发送到各个探测器上。每个探测器能根据所在网络的实际需要配置不同的规则集。报警信息能发到管理平台的消息系统，用各种方式通知 IDS 管理员。现在有人根据此种数据来源混合的方式，称此类系统为“混合型”（Hybrid）系统。最著名的明确体现分布式架构的早期系统为 SRI 的 EMERALD 系统，它明确将分布式检测架构进行层次化的处理，并实现了不同层次上的分析单元，同时提供了开放的 API 接口，实现基本架构下的组件互换功能，之后，UC Davis 设计了 GrIDS（Graph-based IDS）系统，这也是处理可扩展性问题的一次有益尝试。后来的 Purdue 大学设计并原型实现的 AAFID 系统体现了基于自治代理的分布式架构思想。

DIDS 的典型结构如图 2.3 所示：

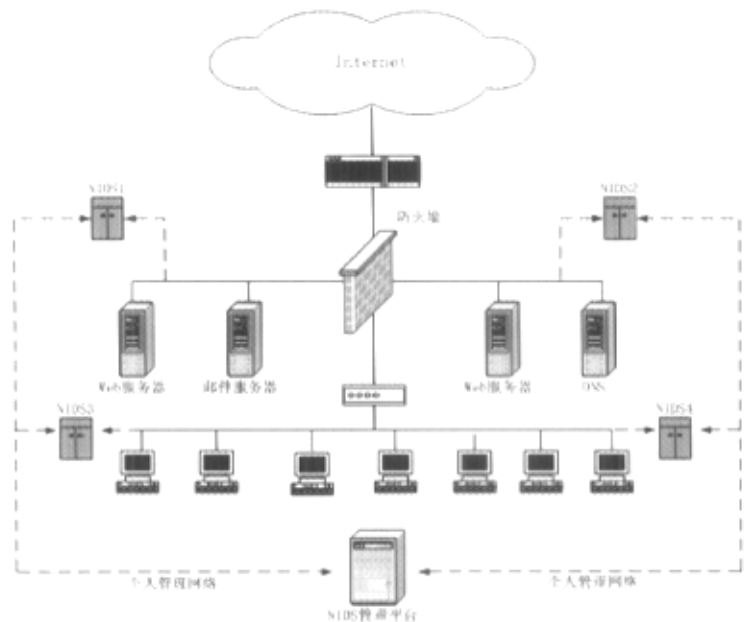


图 2.3 DIDS 网络

Fig. 2.3 DIDS Network

## 2.5 入侵检测技术

入侵检测系统首先是为保护关键信息基础设施的可用性、保密性和完整性而设计，以防止其遭受拒绝服务攻击、非授权获取信息、改动或破坏数据。当网络和计算机受到信息攻击时，入侵检测系统能自动检测并迅速报告这些事件。概括的说，目前入侵检测系统采用的技术为：

- 用户行为概率统计模型

这种入侵检测方法是基于对用户历史行为建模以及在早期的证据或模型的基础上，审计系统实时地检测用户对系统的使用情况，根据系统内部保存的用户行为的概率统计模型进行检测，当发现有可疑的用户行为发生时，保持跟踪并监测、记录该用户的行为。系统要根据每个用户以前的历史行为，生成用户的历史行为记录库，当用户改变他们的行为习惯时，这种异常就会被检测出来。

统计手段的主要优点是可以自适应学习用户的行为，主要问题是其可能被入侵者逐渐训练以至最终将入侵事件误认为正常。并且阈值设置不当会导致大比例的“误报”和“漏报”。

### ● 模式匹配技术

模式匹配检查对照一系列已有的攻击特征比较用户活动,从而发现入侵。这种想法的先进之处在于定义已知的问题模式,然后观察能与模式匹配的事件数据。独立的模式可以由独立事件、事件序列、事件临界值或者允许与、或操作的通用规则表达式组成。模式匹配的研究已经相当成熟,入侵检测需要做的是如何选择和运用模式匹配引擎。在入侵检测系统中应用模式匹配技术的研究包括:

1. 选择和实施模式匹配引擎。
2. 攻击模式集的收集:攻击模式集的收集可以通过以下途径获得,如公开发表的攻击模式,专有的知识和实践经验。攻击模式应是通用的,能代表一种攻击类别。
3. 攻击模式库的更新:发现新的攻击类型时,需要扩充模式库。有必要引入自学习部件到模式匹配中,一旦发现新攻击,自动地更新模式匹配数据库的内容。

### ● 神经网络

神经网络具有自学习、自适应的能力,因而,在基于神经网络模型的入侵检测系统中,只要提供系统的审计数据,神经网络就可以通过自学习中提取正常的用户或系统活动的特征模式,而不需要获取描述用户行为特征的特征集以及用户行为特征测度的统计分布[10]。因此,避开了选择统计特征的困难问题,使如何选择一个好的主体属性子集的问题成了一个不相关的事。从而使其在入侵检测中也得到了很好的应用。首先,用用户正常行为的样本模式对神经网络进行学习训练,然后神经网络接受用户活动的数据,并判断它与经训练产生的正常模式的偏离程度。

### ● 专家系统

早期的入侵检测系统多数采用专家系统来检测系统中的入侵行为[11]。NIDES、W&S、NADIR 等系统的异常性检测器中都有一个专家系统模块。在这些系统中,入侵行为编码成专家系统的规则。每个规则具有“IF 条件 THEN 动作”的形式;其中条件为审计记录中某个域上的限制条件;动作表示规则被触发时入侵检测系统所采取的处理动作,可以是一些新事实的断言或者是提高某个用户行为的可疑度。这些规则可以识别单个审计事件也可以识别表示一个入侵行为的一系列事件。专家系统可以自动地解释系统的审计记录并判断他们是否满足描述入侵行为的规则。但是,使用专家系统规则表示一系列的活动不具有直观性;除非由专业的知识库程序员来做专家系统的升级工作,否则规则的更新是很困难的;而且使用专家系统分析系统的审计数据也是很低效的,另外使用专家系统规则很

难检测出对系统的协同攻击。

- 数据挖掘

入侵检测的数据挖掘是指离线的知识发现过程，将先前收集的大量的数据进行过滤、转换和组织成信息集。这些信息用来发现以前未检测到的隐藏的入侵模式[12]。基于数据挖掘的入侵检测系统基本构成有几个部分：数据收集，数据清理，数据选择和转换，发现模块，以及结果显示等。

- 状态迁移分析技术[13]

一个入侵行为就是由攻击者执行的一系列的操作，这些操作可以使系统从某些初始状态迁移到一个可以威胁系统安全的状态。这里的状态指系统某一时刻的特征（由一系列系统属性来描述）。初始状态对应于入侵开始时的系统状态，危及系统安全的状态对应于已成功入侵时刻的系统状态；在这两个状态之间则可能有一个或多个中间状态的迁移。在识别出初始状态、威胁系统安全的状态后，主要应分析在这两个状态之间进行状态迁移的关键活动，这些迁移信息可以用状态迁移图描述或者用于生成专家系统的规则，从而用于检测系统的入侵活动。

状态迁移分析主要考虑入侵行为的每一步对系统状态迁移的影响，因而，它可以检测出协同攻击者以及利用用户对话对系统的攻击行为。但是，这种模型只适用于那些多个步骤之间具有全序关系的入侵行为的检测。对于那些入侵序列具有偏序等更复杂关系的入侵行为则无能为力。

综上所述，可以看出各种技术均有其突出的一面，同时又不可避免的有不足之处。因此，多种技术的综合运用是当今入侵检测发展的主流方向。

## 2.6 入侵检测系统的发展趋势

为了应付不断发展的入侵手段，人们在完善原有技术的基础上，又在研究新的检测方法，如数据融合技术，主动的自主代理方法，智能技术以及免疫学原理的应用等，主要发展方向可概括为：

- 协作式入侵检测

分布式入侵检测中的一个关键技术是协作式入侵检测技术。协作包括：同一系统中不同入侵检测部件之间的协作，尤其是主机型和网络型入侵检测部件之间的协作，以及异构平台部件的协作；不同安全工具之间的协作；不同厂家的安全产品之间的协作；不同组织之间预警能力和信息的协作。

- 智能化入侵检测

入侵检测系统的核心问题是系统的分析能力。应该使入侵检测系统的分析更

难检测出对系统的协同攻击。

- 数据挖掘

入侵检测的数据挖掘是指离线的知识发现过程，将先前收集的大量的数据进行过滤、转换和组织成信息集。这些信息用来发现以前未检测到的隐藏的入侵模式[12]。基于数据挖掘的入侵检测系统基本构成有几个部分：数据收集，数据清理，数据选择和转换，发现模块，以及结果显示等。

- 状态迁移分析技术[13]

一个入侵行为就是由攻击者执行的一系列的操作，这些操作可以使系统从某些初始状态迁移到一个可以威胁系统安全的状态。这里的状态指系统某一时刻的特征（由一系列系统属性来描述）。初始状态对应于入侵开始时的系统状态，危及系统安全的状态对应于已成功入侵时刻的系统状态；在这两个状态之间则可能有一个或多个中间状态的迁移。在识别出初始状态、威胁系统安全的状态后，主要应分析在这两个状态之间进行状态迁移的关键活动，这些迁移信息可以用状态迁移图描述或者用于生成专家系统的规则，从而用于检测系统的入侵活动。

状态迁移分析主要考虑入侵行为的每一步对系统状态迁移的影响，因而，它可以检测出协同攻击者以及利用用户对话对系统的攻击行为。但是，这种模型只适用于那些多个步骤之间具有全序关系的入侵行为的检测。对于那些入侵序列具有偏序等更复杂关系的入侵行为则无能为力。

综上所述，可以看出各种技术均有其突出的一面，同时又不可避免的有不足之处。因此，多种技术的综合运用是当今入侵检测发展的主流方向。

## 2.6 入侵检测系统的发展趋势

为了应付不断发展的入侵手段，人们在完善原有技术的基础上，又在研究新的检测方法，如数据融合技术，主动的自主代理方法，智能技术以及免疫学原理的应用等，主要发展方向可概括为：

- 协作式入侵检测

分布式入侵检测中的一个关键技术是协作式入侵检测技术。协作包括：同一系统中不同入侵检测部件之间的协作，尤其是主机型和网络型入侵检测部件之间的协作，以及异构平台部件的协作；不同安全工具之间的协作；不同厂家的安全产品之间的协作；不同组织之间预警能力和信息的协作。

- 智能化入侵检测

入侵检测系统的核心问题是系统的分析能力。应该使入侵检测系统的分析更

智能化。未来的入侵检测系统应该能够进行基于事件语义而不是基于事件语法的检测。这种方法弥补了当前在安全政策和检测政策之间的差距。例如，给定一个语义检测引擎，你将告诉入侵检测系统去“检测所有的违反访问控制的访问”，那么入侵检测系统就会去做而不管被检测的系统使用的是什么平台、什么协议或什么数据类型。这种类型的操作将是对当前的检测状态的一个极大的改进，在当前的检测中，检测目标需要复杂的、特定的与操作系统相关的检测特征。

#### ● 宽带高速网络的实时入侵检测

大量高速网络技术如 ATM、千兆以太网等近年里相继出现，在此背景下的各种宽带接入手段层出不穷。如何实现高速网络下的实时入侵检测已经成为现实面临的问题。目前的千兆 IDS 产品其性能指标与实际要求相差很远。要提高其性能主要需考虑以下两个方面：首先，IDS 的软件结构和算法需要重新设计，以期适应高速网的环境，提高运行速度和效率；其次，随着高速网络技术的不断发展与成熟，新的高速网络协议的设计也必将成为未来发展的趋势，那么，现有 IDS 如何适应和利用未来的新网络协议将是一个全新的问题。

#### ● 数据融合入侵检测

目前的 IDS 还存在着很多缺陷。首先，目前的技术还不能对付训练有素的黑客的复杂的攻击。其次，系统的虚警率太高。最后，系统对大量的数据处理，非但无助于解决问题，还降低了处理能力。数据融合技术是解决这一系列问题的方法[14]。

#### ● 与网络安全技术相结合

在入侵检测早期发展中，它常被当作是一种独立的技术，但现在入侵检测系统和网络管理的综合使用变得越来越必要了。单一的技术很难构筑一道强有力的安全防线，这就需要和其他安全技术共同组成更完备的安全保障系统，如结合防火墙、PKIX、安全电子交易 SET 等新的网络安全与电子商务技术，提供完整的网络安全保障。



## 第三章 Snort 系统分析

Snort 是一个轻量级网络入侵检测系统，具有实时数据流量分析和日志 IP 网络数据包的能力，能够进行协议分析，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，该软件遵循通用公共许可证 GPL，所以只要遵守 GPL 的任何组织和个人都可以自由使用。

许多商业运营的入侵检测系统都是在 Snort 检测引擎的基础上进行二次开发而来。历经数年发展，Snort 已经发展到 2.x 版本，其基本架构在 1.6 版本时初步建立，2.0 版本中采用了新型的架构设计和入侵检测方法，功能更加完善，部署也变得更加方便，Snort 系统已经成为学习研究入侵检测的经典实例。

### 3.1 Snort 系统架构

Snort 入侵检测系统主要由四部分组成：

- 数据包嗅探器
- 预处理器
- 检测引擎
- 报警输出模块

系统体系结构如图 3.1[15]所示：

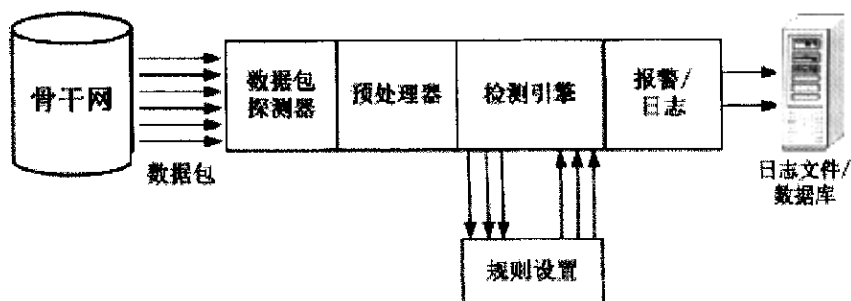


Fig. 3.1 Snort architecture

预处理器、检测引擎和报警模块都是插件结构，插件程序按照 Snort 提供的插件接口完成，使用时动态加载，在不用修改核心代码的前提下可以扩展 Snort 的功能和复杂性。这样既保障了插件程序和 Snort 核心代码的紧密相关性，又保

障了核心代码的良好结构。

Snort 系统主要有以下特点:

- 具有实时流量分析和日志 IP 网络数据包的能力,能够快速地检测网络攻击,及时地发出报警。Snort 的报警机制很丰富,例如:syslog、用户指定的文件、一个 UNIX 套接字,还有使用 SAMBA 协议向 Windows 客户程序发出 WinPopup 消息。利用 XML 插件,Snort 可以使用 SNML(Simple Network Markup Language,简单网络标记语言)把日志存放到一个文件或者适时报警[16]。
- 能够进行协议分析,内容的搜索/匹配。现在 Snort 能够分析的协议有 TCP、UDP 和 ICMP。将来可能提供对 ARP、ICRP、GRE、OSPF、RIP、IPX 等协议的支持。它能够检测多种方式的攻击和探测,例如:缓冲区溢出、秘密端口扫描、CGI 攻击、SMB 探测、探测操作系统指纹特征的企图等等。
- Snort 的日志格式既可以是 tcpdump 式的二进制格式,也可以解码成 ASCII 字符形式,更加便于用户尤其是新手检查。使用数据库输出插件,Snort 可以把日志记入数据库,当前支持的数据库包括:Postgresql、MySQL、任何 unixODBC 数据库,还有 Oracle。
- 使用 TCP 流插件(tcpstream),Snort 可以对 TCP 包进行重组。Snort 能够对 IP 包的内容进行匹配,但是对于 TCP 攻击,如果攻击者使用一个程序,每次发送只有一个字节的 TCP 包,完全可以避开 Snort 的模式匹配。而被攻击的主机的 TCP 协议栈会重组这些数据,将其送给在目标端口上监听的进程,从而使攻击包逃过 Snort 的监视。使用 TCP 流插件,可以对 TCP 包进行缓冲,然后进行匹配,使 Snort 具备了对付上面这种攻击的能力。
- 使用 SPADE(Statistical Packet Anomaly Detection Engine)插件,Snort 能够报告非正常的可疑包,从而对端口扫描进行有效的检测。
- 具有很强的系统防护能力。使用 FlexResp 功能,Snort 能够主动断开恶意连接。
- 使用一种简单的规则描述语言。最基本的规则只包含四个域:处理动作、协议、方向、注意的端口。例如:log tcp any any -> 10.1.1.0/24 79。还有一些功能选项可以组合使用,实现更为复杂的功能。
- 支持插件,可以使用具有特定功能的报告、检测子系统插件对其功能进行扩展。Snort 当前支持的插件包括:数据库日志输出插件、碎数据包检

测插件、端口扫描检测插件、HTTP URI normalization 插件、XML 插件等。

- 规则语言非常简单，能够对新的网络攻击做出很快的反应。发现新的攻击后，可以很快根据 Bugtraq 邮件列表，找出特征码，写出检测规则。因为其规则语言简单，所以很容易上手，节省人员的培训费用。

## 3.2 Snort 系统主要模块

### 3.2.1 数据包嗅探器

数据包嗅探器是一个并联在网络中的设备（可以是硬件，也可以是软件），它的工作原理和电话窃听很相似，不同的只是电话窃听的是语音网络，而数据包嗅探的是数据网。在互连网上通常指嗅探 IP 网络的流量，但是对不常用的网络协议如 IPX 和 AppleTalk 等也能嗅探。Snort 能够把捕获的数据包保存起来并可在事后查看。

Snort 主要通过两种机制[15]来满足捕获网络流量的需要：

- 将网卡设置为混杂模式。网卡的默认方式将忽略所有不是以自己的 MAC 地址为目的地址的流量。
- 利用 libpcap（用于捕获网络数据包的必备工具，它是独立于系统的 API 接口，为底层网络监控提供了一个可移植的框架，可用于网络统计收集、安全监控、网络调试等应用[17]）从网卡捕获网络封包。Libpcap 库允许开发人员在不同的 UNIX 平台上从数据链路层（TCP/IP 模型中的第二层，如图 3.2 所示）接收数据包，而不必考虑网卡以及驱动程序的不同。更重要的是 libpcap 库直接从网卡取得数据包，它允许开发人员自己写程序解码、显示和记录数据报文。

测插件、端口扫描检测插件、HTTP URI normalization 插件、XML 插件等。

- 规则语言非常简单，能够对新的网络攻击做出很快的反应。发现新的攻击后，可以很快根据 Bugtraq 邮件列表，找出特征码，写出检测规则。因为其规则语言简单，所以很容易上手，节省人员的培训费用。

## 3.2 Snort 系统主要模块

### 3.2.1 数据包嗅探器

数据包嗅探器是一个并联在网络中的设备（可以是硬件，也可以是软件），它的工作原理和电话窃听很相似，不同的只是电话窃听的是语音网络，而数据包嗅探的是数据网。在互连网上通常指嗅探 IP 网络的流量，但是对不常用的网络协议如 IPX 和 AppleTalk 等也能嗅探。Snort 能够把捕获的数据包保存起来并可在事后查看。

Snort 主要通过两种机制[15]来满足捕获网络流量的需要：

- 将网卡设置为混杂模式。网卡的默认方式将忽略所有不是以自己的 MAC 地址为目的地址的流量。
- 利用 libpcap（用于捕获网络数据包的必备工具，它是独立于系统的 API 接口，为底层网络监控提供了一个可移植的框架，可用于网络统计收集、安全监控、网络调试等应用[17]）从网卡捕获网络封包。Libpcap 库允许开发人员在不同的 UNIX 平台上从数据链路层（TCP/IP 模型中的第二层，如图 3.2 所示）接收数据包，而不必考虑网卡以及驱动程序的不同。更重要的是 libpcap 库直接从网卡取得数据包，它允许开发人员自己写程序解码、显示和记录数据报文。

测插件、端口扫描检测插件、HTTP URI normalization 插件、XML 插件等。

- 规则语言非常简单，能够对新的网络攻击做出很快的反应。发现新的攻击后，可以很快根据 Bugtraq 邮件列表，找出特征码，写出检测规则。因为其规则语言简单，所以很容易上手，节省人员的培训费用。

## 3.2 Snort 系统主要模块

### 3.2.1 数据包嗅探器

数据包嗅探器是一个并联在网络中的设备（可以是硬件，也可以是软件），它的工作原理和电话窃听很相似，不同的只是电话窃听的是语音网络，而数据包嗅探的是数据网。在互连网上通常指嗅探 IP 网络的流量，但是对不常用的网络协议如 IPX 和 AppleTalk 等也能嗅探。Snort 能够把捕获的数据包保存起来并可在事后查看。

Snort 主要通过两种机制[15]来满足捕获网络流量的需要：

- 将网卡设置为混杂模式。网卡的默认方式将忽略所有不是以自己的 MAC 地址为目的地址的流量。
- 利用 libpcap（用于捕获网络数据包的必备工具，它是独立于系统的 API 接口，为底层网络监控提供了一个可移植的框架，可用于网络统计收集、安全监控、网络调试等应用[17]）从网卡捕获网络封包。Libpcap 库允许开发人员在不同的 UNIX 平台上从数据链路层（TCP/IP 模型中的第二层，如图 3.2 所示）接收数据包，而不必考虑网卡以及驱动程序的不同。更重要的是 libpcap 库直接从网卡取得数据包，它允许开发人员自己写程序解码、显示和记录数据报文。

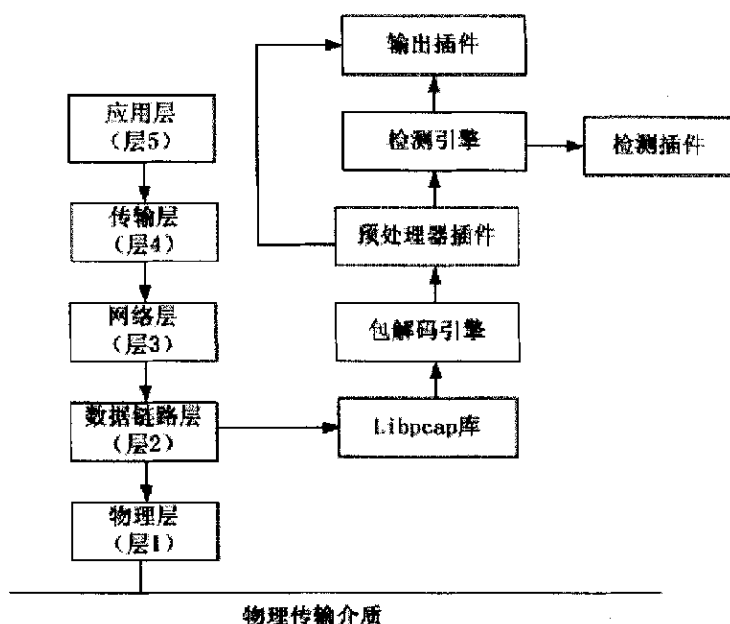


图 3.2 TCP/IP 模型和 Snort

Fig. 3.2 The TCP/IP model and Snort

Snort 数据包嗅探器模块通过封装 Libpcap 库，将网卡设置为混杂模式后，就实现了从网络数据链路层截获数据包的功能，而不管所截获的数据包采用何种网络协议。如图 3.3[15]描述了系统的数据包嗅探功能：

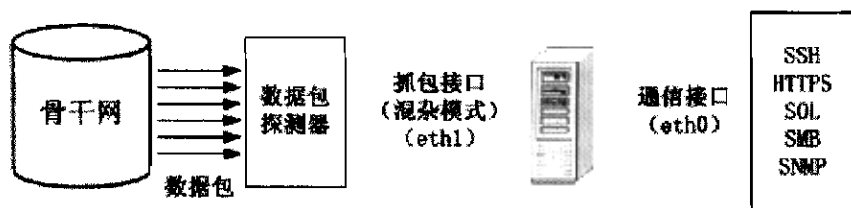


图 3.3 Snort 的数据包嗅探功能

Fig. 3.3 Snort's packet sniffing functionality

### 3.2.2 预处理器

因为从网络捕获来的数据包采用的协议有多种，而且数据包也比较原始，所以预处理器模块的作用就是针对当前截获的数据包进行预处理，其功能类似硬币分拣过程中的硬币检查工作，当所有的硬币已经收集来以后（从网络中捕获了数



据包)，这些硬币要通过一个传送带，在硬币打包（检测引擎）以前，硬币分拣机要判断收集来的是否都是真正的硬币，这个判断就是预处理的功能。预处理器是如何检查数据包，如图 3.4[15]所示：

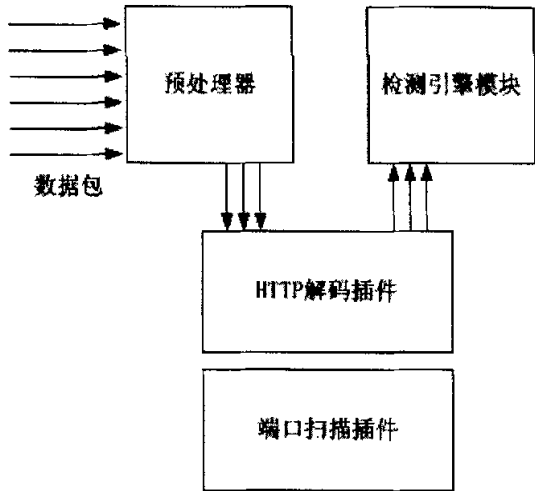


图 3.4 Snort 的预处理器

Fig. 3.4 Snort's Preprocessor

预处理器由多个模块构成，每个模块完成具体功能，按照协议类型对数据包进行预处理，可以根据实际环境的需要启动或停止预处理器插件。例如，如果认为网络中没有 RPC 服务，就可以方便地停用 RPC 的预处理插件，这样可以提高 Snort 的效率，当然也可以根据需要编写特定的预处理插件来实现特殊功能。

3.2.3 检测引擎

检测引擎是 Snort 的核心模块。当数据包从预处理器送过来后，检测引擎依据预先设置的规则检查数据包，一旦发现数据包中的内容和某条规则相匹配，就通知报警模块。Snort 是基于特征的 IDS，基于特征的 IDS 的功能实现依赖于各种不同的规则设置，检测引擎根据规则来匹配数据包。Snort 的规则很多，并根据不同的类型（木马、缓冲区溢出、权限滥用等）作了分组，规则要经常升级。

以硬币分拣机为例，检测引擎的工作就是硬币分拣机根据不同币值的硬币分类打包。如大多数的硬币都是一元、五角、一角、五分和一分的，但也会遇到一些特殊的硬币，如一元的纪念币、游戏机的代币币等等，这种硬币就要丢弃了。

如图 3.5[15]描述了检测引擎的工作流程：

据包)，这些硬币要通过一个传送带，在硬币打包（检测引擎）以前，硬币分拣机要判断收集来的是否都是真正的硬币，这个判断就是预处理的功能。预处理器是如何检查数据包，如图 3.4[15]所示：

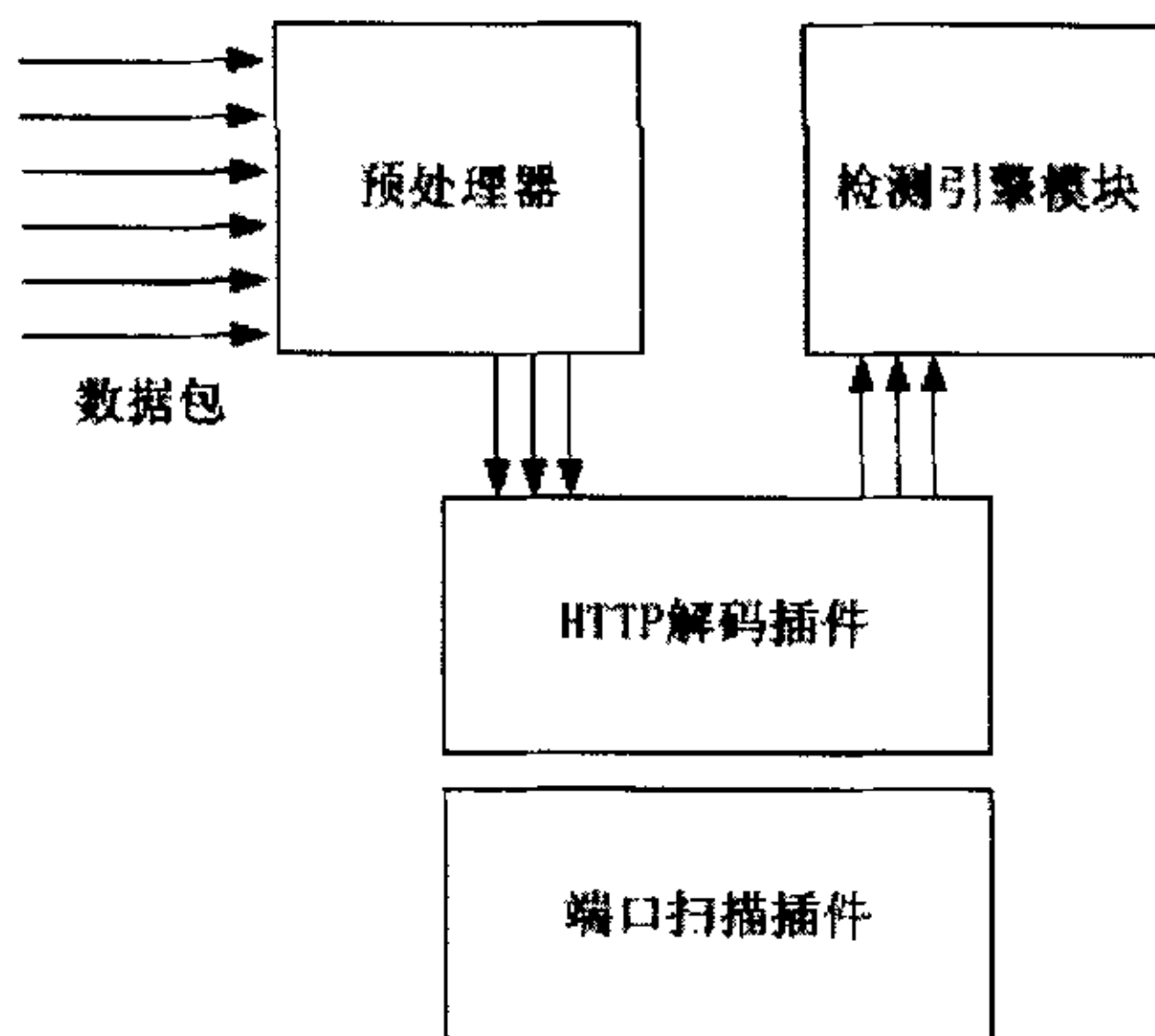


图 3.4 Snort 的预处理器

Fig. 3.4 Snort's Preprocessor

预处理器由多个模块构成，每个模块完成具体功能，按照协议类型对数据包进行预处理，可以根据实际环境的需要启动或停止预处理器插件。例如，如果认为网络中没有 RPC 服务，就可以方便地停用 RPC 的预处理插件，这样可以提高 Snort 的效率，当然也可以根据需要编写特定的预处理插件来实现特殊功能。

### 3.2.3 检测引擎

检测引擎是 Snort 的核心模块。当数据包从预处理器送过来后，检测引擎依据预先设置的规则检查数据包，一旦发现数据包中的内容和某条规则相匹配，就通知报警模块。Snort 是基于特征的 IDS，基于特征的 IDS 的功能实现依赖于各种不同的规则设置，检测引擎根据规则来匹配数据包。Snort 的规则很多，并根据不同的类型（木马、缓冲区溢出、权限滥用等）作了分组，规则要经常升级。

以硬币分拣机为例，检测引擎的工作就是硬币分拣机根据不同币值的硬币分类打包。如大多数的硬币都是一元、五角、一角、五分和一分的，但也会遇到一些特殊的硬币，如一元的纪念币、游戏机的代钱币等等，这种硬币就要丢弃了。

如图 3.5[15]描述了检测引擎的工作流程：

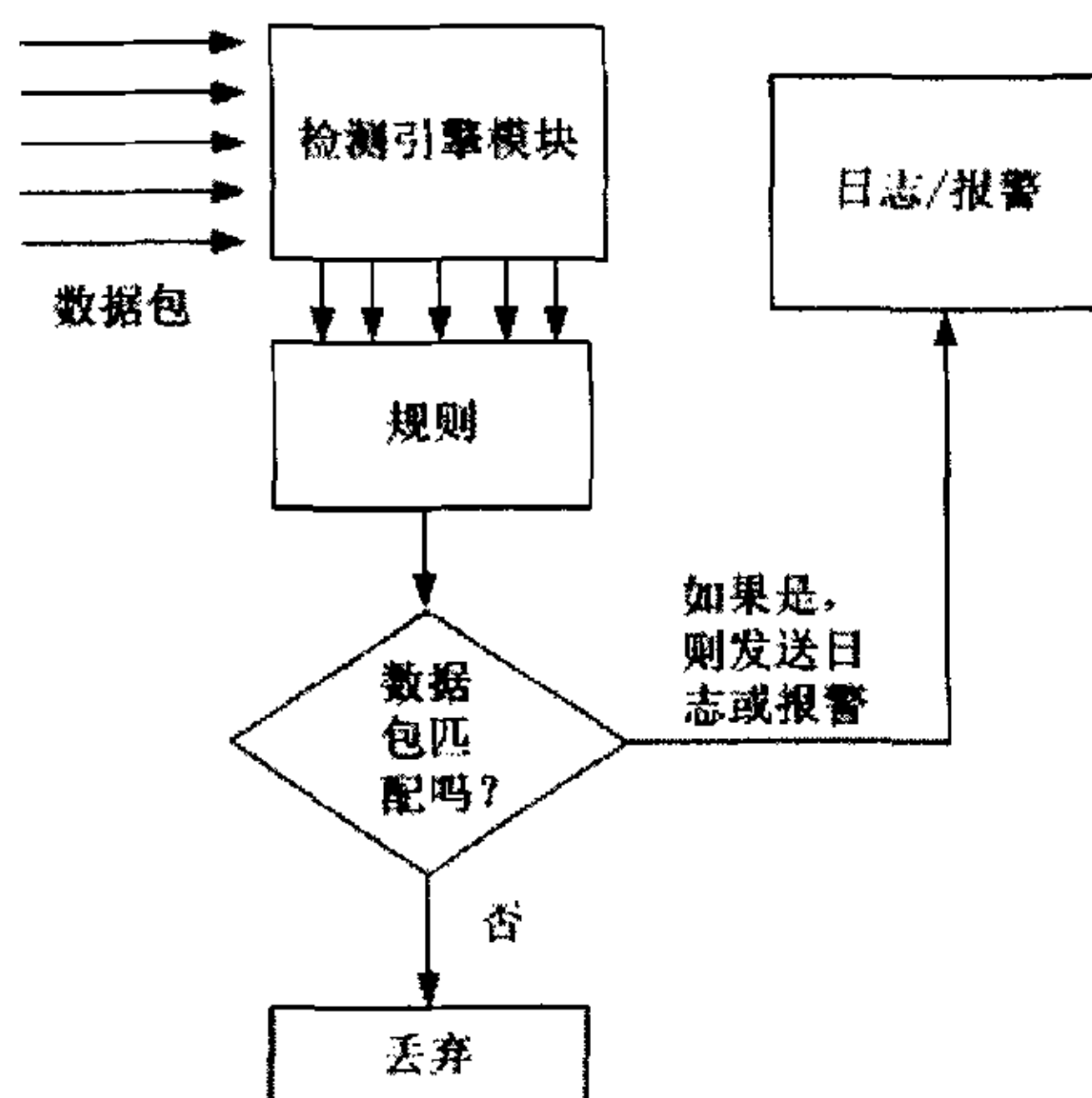


图 3.5 Snort 的检测引擎模块

Fig. 3.5 Snort's Detection Engine module

Snort 在初始化的时候会根据用户的设置情况,根据所采用的模式匹配方法采用不同的数据结构将规则文件加载到内存中,并进行优化处理。在检测引擎方面,Snort2.0 版本较以前版本有了显著改进,这将在下章详细分析。

### 3.2.4 报警/日志模块

检测引擎检查后的 Snort 数据需要以某种方式输出。如检测引擎中的某条规则被匹配,则会触发一条报警,这条报警信息会通过网络、UNIX socket、Windows Popup(SMB)或 SNMP 协议的 trap 命令送给日志文件。报警信息也可以记入 SQL 数据库,如 MySQL 或 Postgres 等。另外,还有各种专为 Snort 开发的辅助工具,如各种各样基于 WEB 的报警信息显示插件。这些插件用 Perl 或 PHP 开发,目的是为了更直观地显示报警信息。报警信息以文本文件的形式保存或保存在 MySQL 或 Postgres 等数据库中。

与预处理器和检测引擎模块一样,报警输出模块也使用插件实现多种输出功能,如把报警信息送入数据库或通过 SNMP、WinPopup 等网络协议传送。

报警输出模块的结构如图 3.6[15]所示:

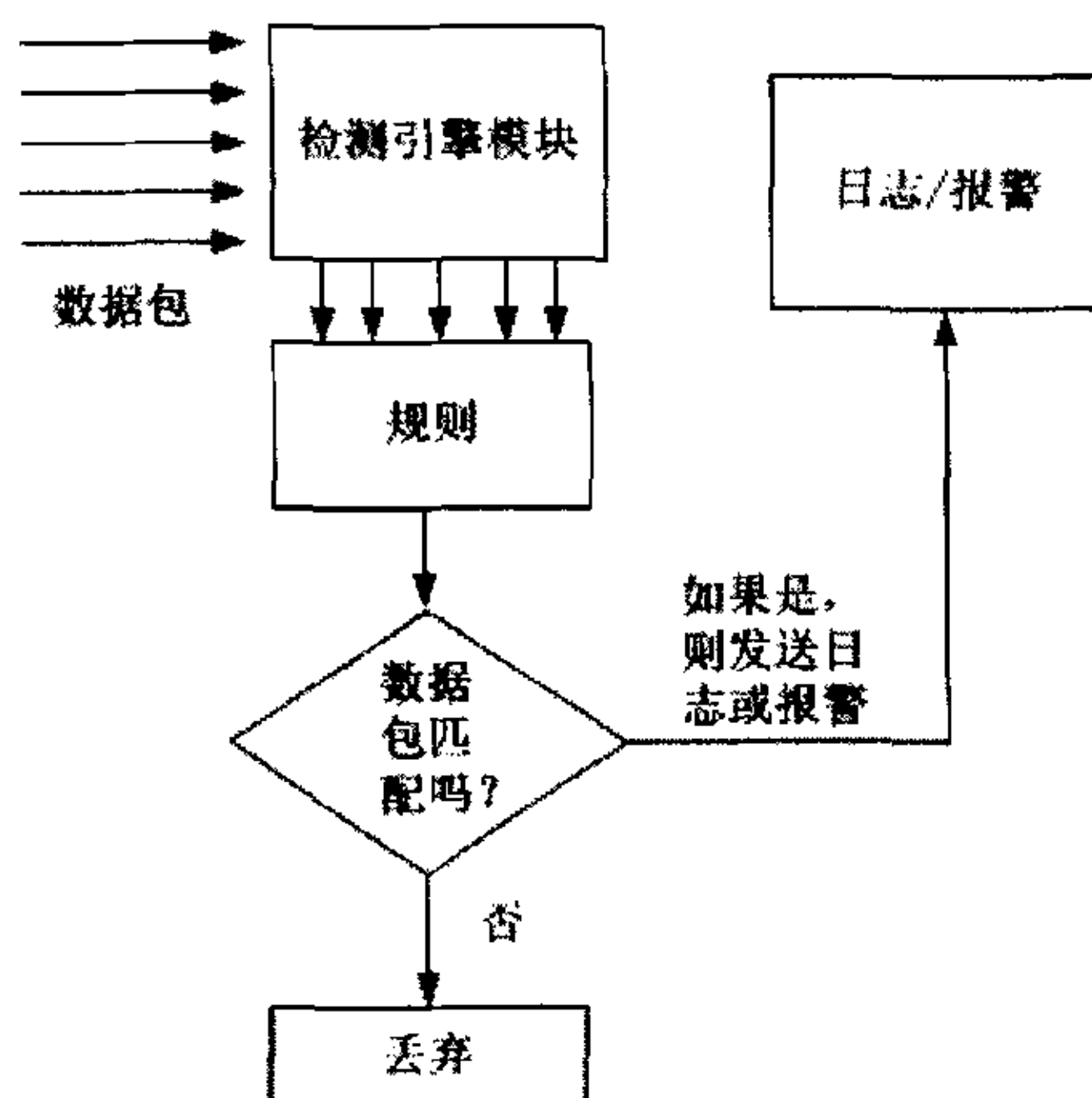


图 3.5 Snort 的检测引擎模块

Fig. 3.5 Snort's Detection Engine module

Snort 在初始化的时候会根据用户的设置情况,根据所采用的模式匹配方法采用不同的数据结构将规则文件加载到内存中,并进行优化处理。在检测引擎方面,Snort2.0 版本较以前版本有了显著改进,这将在下章详细分析。

### 3.2.4 报警/日志模块

检测引擎检查后的 Snort 数据需要以某种方式输出。如检测引擎中的某条规则被匹配,则会触发一条报警,这条报警信息会通过网络、UNIX socket、Windows Popup(SMB)或 SNMP 协议的 trap 命令送给日志文件。报警信息也可以记入 SQL 数据库,如 MySQL 或 Postgres 等。另外,还有各种专为 Snort 开发的辅助工具,如各种各样基于 WEB 的报警信息显示插件。这些插件用 Perl 或 PHP 开发,目的是为了更直观地显示报警信息。报警信息以文本文件的形式保存或保存在 MySQL 或 Postgres 等数据库中。

与预处理器和检测引擎模块一样,报警输出模块也使用插件实现多种输出功能,如把报警信息送入数据库或通过 SNMP、WinPopup 等网络协议传送。

报警输出模块的结构如图 3.6[15]所示:

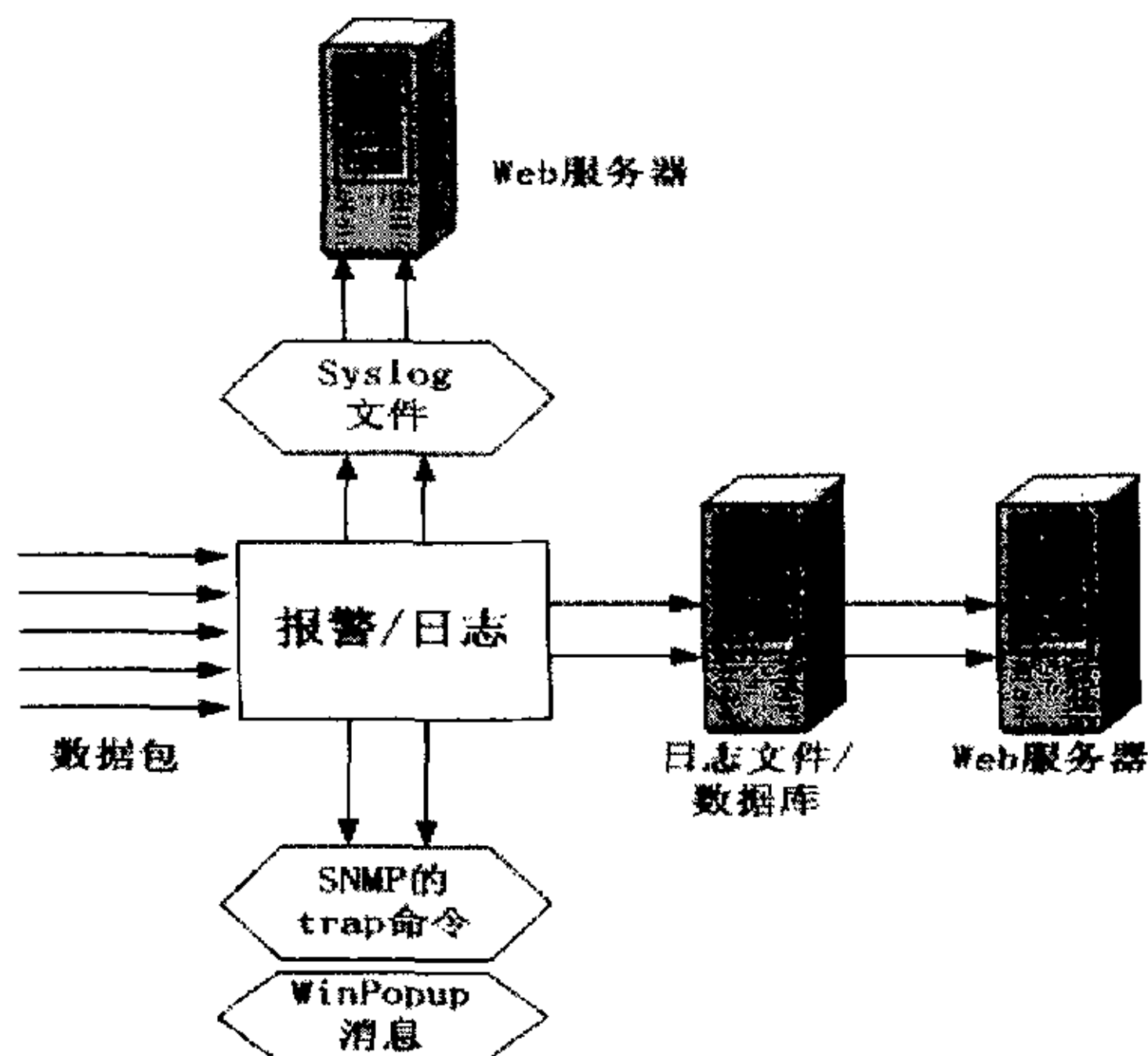


图 3.6 Snort 的报警模块

Fig. 3.6 Snort's alerting component

### 3.3 Snort 系统工作流程

Snort 的基本功能是数据包嗅探器，然而数据包嗅探只是 Snort 工作的开始，Snort 取得数据包后先用预处理插件处理，然后经过检测引擎中的所有规则链，如果检测到有符合规则链的数据包，则系统就会根据输出设置把该信息记录到文件并报警。Snort 系统内的工作流程如图 3.7[15]所示：

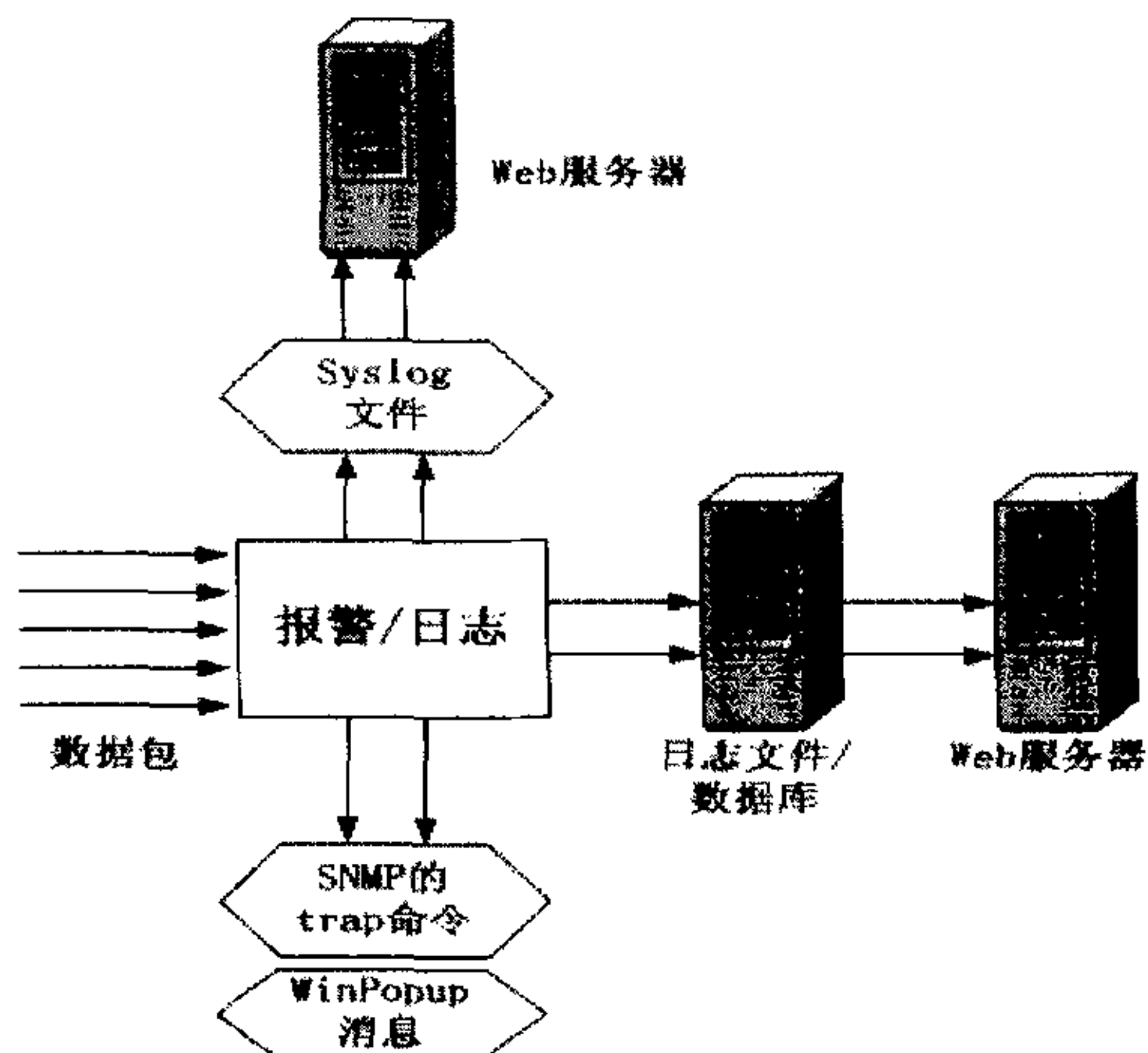


图 3.6 Snort 的报警模块

Fig. 3.6 Snort's alerting component

### 3.3 Snort 系统工作流程

Snort 的基本功能是数据包嗅探器，然而数据包嗅探只是 Snort 工作的开始，Snort 取得数据包后先用预处理插件处理，然后经过检测引擎中的所有规则链，如果检测到有符合规则链的数据包，则系统就会根据输出设置把该信息记录到文件并报警。Snort 系统内的工作流程如图 3.7[15]所示：



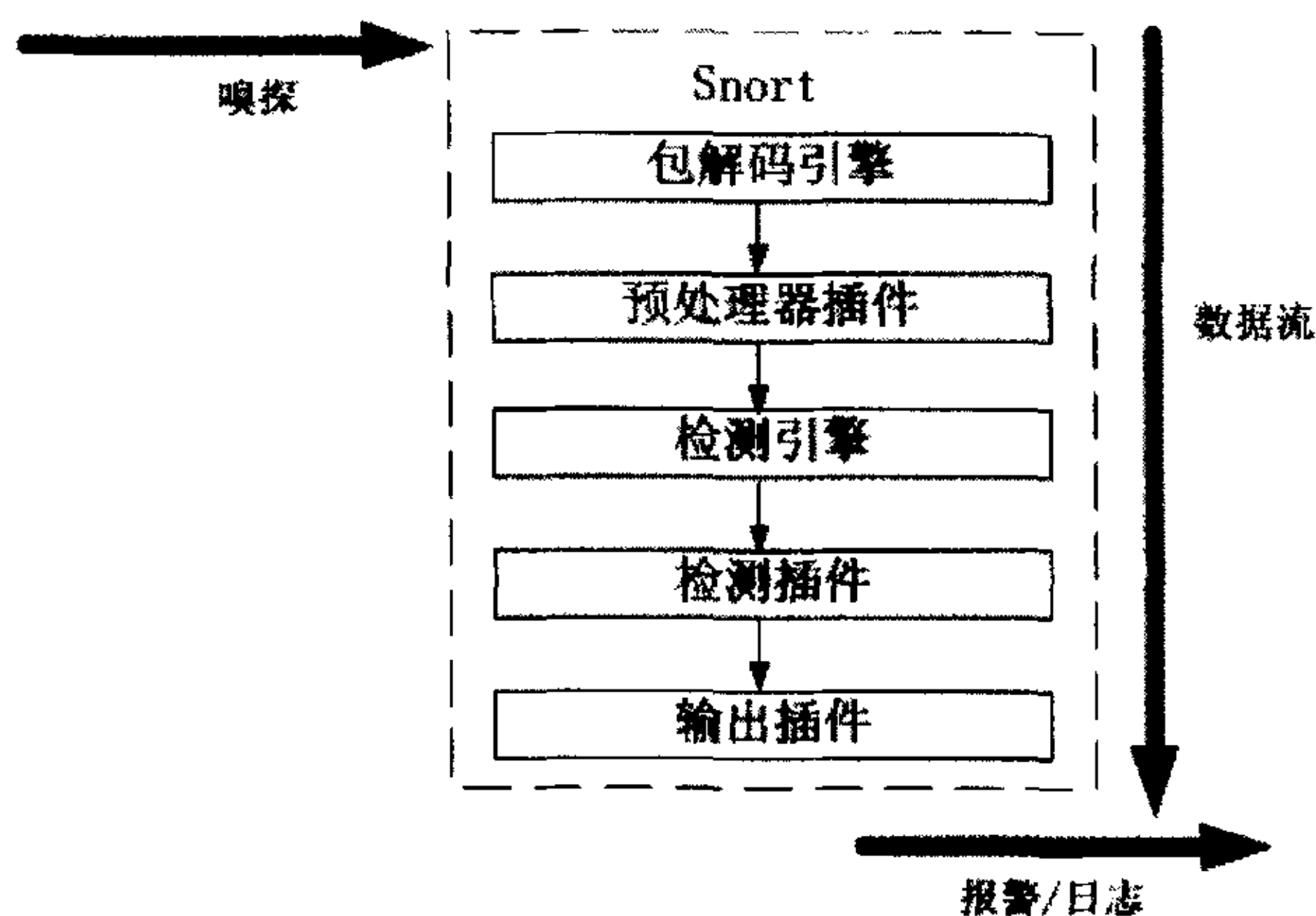


图 3.7 Snort 部件流程

Fig. 3.7 Snort's component working flow

图 3.7 提供了 Snort 处理过程的内部流程，四个主要部件处理过程如下：

- 包捕获/解码引擎。利用 libpcap 从网卡捕获网络上的数据包，然后数据包经过解码引擎填入到链路层协议的包结构体中，以便对高层次的协议进行解码，如 TCP 和 UDP 端口。
- 预处理器插件。数据包被送到各种各样的预处理器中，在检测引擎处理之前进行检查和操作。每个预处理器检查数据包是否应该注意、报警或者修改某些东西。
- 检测引擎。包被送到检测引擎，检测引擎通过各种规则文件中的不同选项来对每个包的特征和包信息进行单一、简单的检测。检测引擎插件对包提供额外的检测功能。规则中的每个关键字选项对应于检测引擎插件，能够提供不同的检测功能。
- 输出插件。Snort 通过检测引擎、预处理器和解码引擎输出报警。

### 3.4 Snort 规则

Snort 规则是基于文本的，它通常存在于 Snort 程序目录或者子目录中。规则文件按照不同的组进行分类，比如文件 ftp.rules 包含了 FTP 攻击内容。在启动的时候，Snort 读取所有的规则文件，并且建立一个三维的链表。Snort 使用该三维

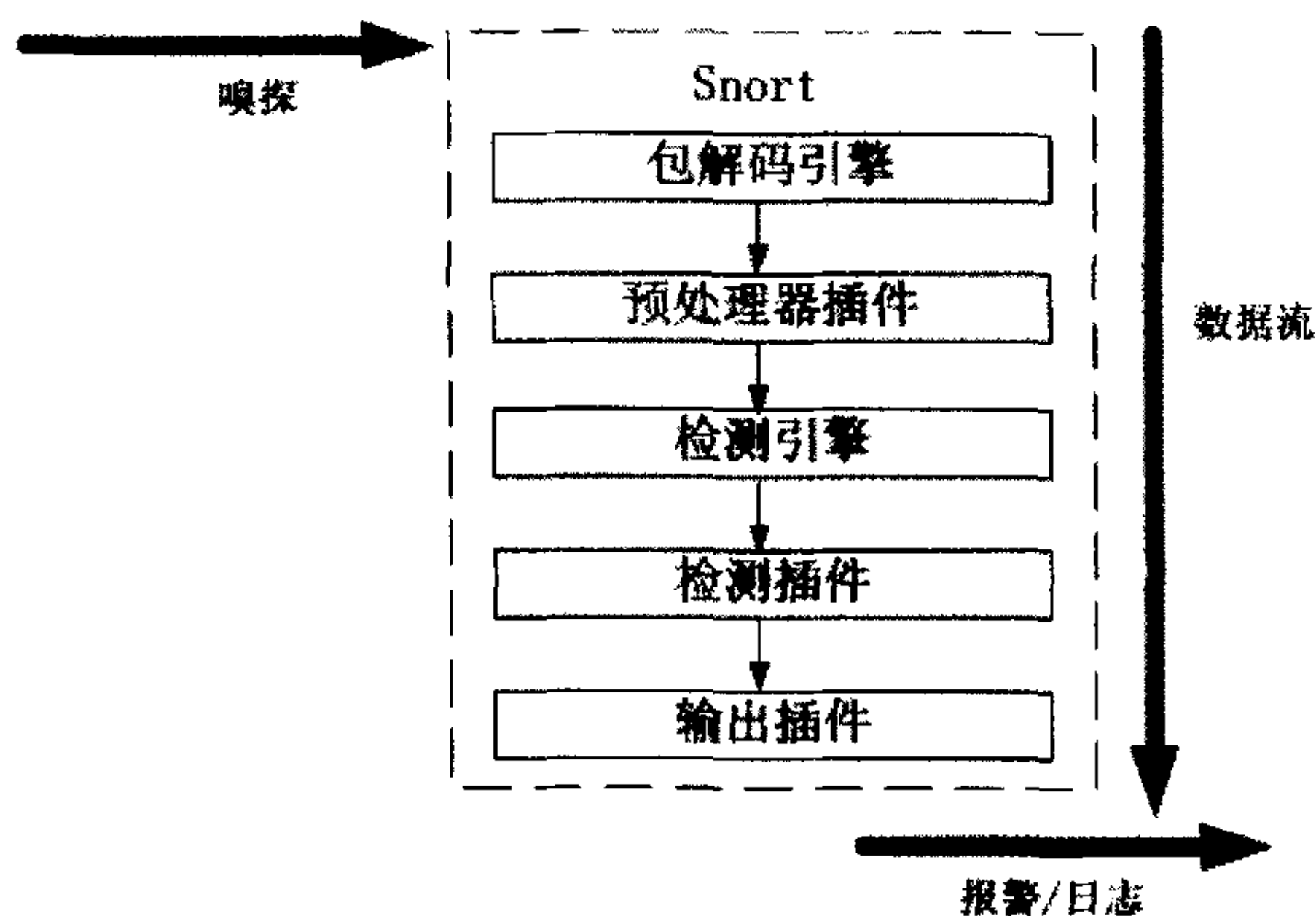


图 3.7 Snort 部件流程

Fig. 3.7 Snort's component working flow

图 3.7 提供了 Snort 处理过程的内部流程，四个主要部件处理过程如下：

- 包捕获/解码引擎。利用 libpcap 从网卡捕获网络上的数据包，然后数据包经过解码引擎填入到链路层协议的包结构体中，以便对高层次的协议进行解码，如 TCP 和 UDP 端口。
- 预处理器插件。数据包被送到各种各样的预处理器中，在检测引擎处理之前进行检查和操作。每个预处理器检查数据包是否应该注意、报警或者修改某些东西。
- 检测引擎。包被送到检测引擎，检测引擎通过各种规则文件中的不同选项来对每个包的特征和包信息进行单一、简单的检测。检测引擎插件对包提供额外的检测功能。规则中的每个关键字选项对应于检测引擎插件，能够提供不同的检测功能。
- 输出插件。Snort 通过检测引擎、预处理器和解码引擎输出报警。

### 3.4 Snort 规则

Snort 规则是基于文本的，它通常存在于 Snort 程序目录或者子目录中。规则文件按照不同的组进行分类，比如文件 ftp.rules 包含了 FTP 攻击内容。在启动的时候，Snort 读取所有的规则文件，并且建立一个三维的链表。Snort 使用该三维

链表匹配包和实现检测。输入和读取文件都在 parser.c 中的 ParseRulesFile() 函数中实现，parser.c 从主程序 snort.c 中引出。

主文件 snort.c 进行初始化时建立三维链表，启动时 Snort 系统读取 snort.conf 配置文件，在 snort.conf 文件的“Step 4——Customize your rule set”中链接特定的规则文件，如 snort.conf 文件的部分内容：

```
#####
# Step #4: Customize your rule set
#
.....
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
.....
```

主函数将每个规则文件中的 Snort 规则解析，在内存中建立用来进行模式匹配的数据结构。

### 3.4.1 规则格式

Snort 规则可以划分为两个逻辑部分[18]：

- 规则头(Rule Header)。
- 规则选项(Rule Options)。

规则头如图 3.8[18]所示：

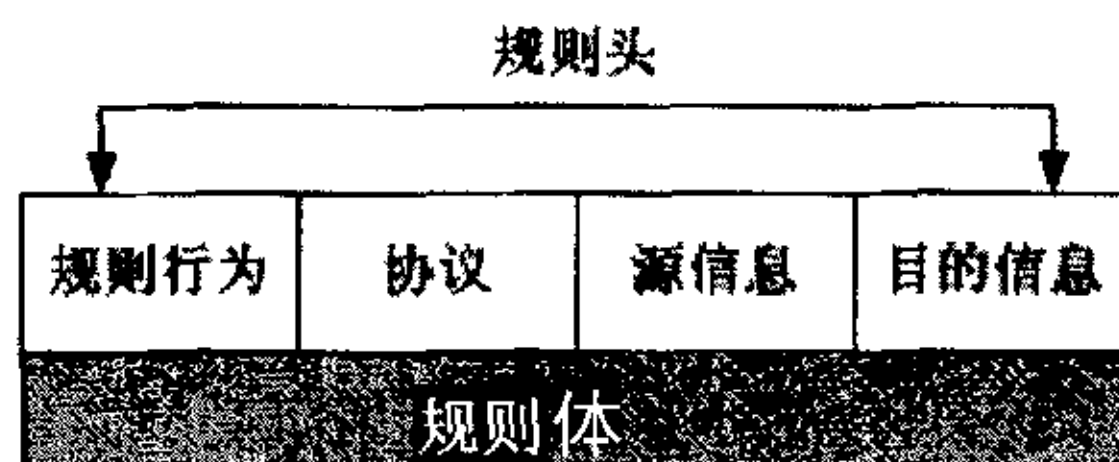


图 3.8 Snort 规则头

Fig. 3.8 Snort's Rule Head

链表匹配包和实现检测。输入和读取文件都在 parser.c 中的 ParseRulesFile() 函数中实现，parser.c 从主程序 snort.c 中引出。

主文件 snort.c 进行初始化时建立三维链表，启动时 Snort 系统读取 snort.conf 配置文件，在 snort.conf 文件的“Step 4——Customize your rule set”中链接特定的规则文件，如 snort.conf 文件的部分内容：

```
#####
# Step #4: Customize your rule set
#
.....
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
.....
```

主函数将每个规则文件中的 Snort 规则解析，在内存中建立用来进行模式匹配的数据结构。

### 3.4.1 规则格式

Snort 规则可以划分为两个逻辑部分[18]：

- 规则头(Rule Header)。
- 规则选项(Rule Options)。

规则头如图 3.8[18]所示：

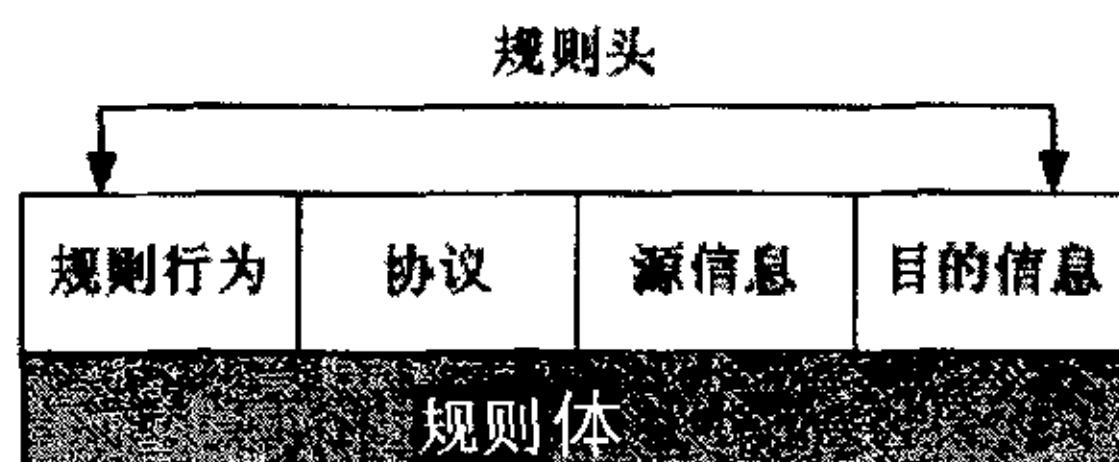


图 3.8 Snort 规则头

Fig. 3.8 Snort's Rule Head

规则头包含了规则动作、协议、IP 源地址和目的地址，子网掩码以及源端口和目标端口值等信息。而规则选项则包含警报信息以及用于确定是否触发响应规则动作而需检查的数据包区域位置的相关信息。

例如，Snort 系统的检测规则格式：

```
Alert tcp any any → 192.168.1.0/24 111 (content:"|00 01 86 a5|";msg:"mounted access";)
```

从规则的开头直到圆括号为止的部分称为规则头，而包含在圆括号之中的部分称为规则选项。在规则选项中，位于冒号“:”之前的词称为选项关键字 (Option keywords)。值得注意的是规则选项并不是对每一个规则而言都是必需的部分，它们只是用来更好地定义所要进行某种处理 (记录、警报等) 的数据包类型。只有当规则中的每一个元素都为真时，才能触发对应的规则动作。从此意义上讲，规则元素之间形成的是一种逻辑“与”的关系。与此同时，在每个规则库文件中的各种规则集合之间形成的是一种更大范围上的逻辑“或”关系。

### 3.4.2 Snort 规则树

规则树[18]的组成：

- 链表头

Snort 规则树有 5 个单独的规则链，这些链作为“树”顶部的链表头

1. Activation 报警并且开启另外一个动态规则。
2. Dynamic 当被上层的激活规则调用时记录网络流量的日志。
3. Alert 产生报警并记录这个数据包。
4. Pass 忽略这个数据包。
5. Log 记录网络的流量 (不报警)。

- 规则树节点 (RTN) 和选项树节点 (OTN)

对于五个规则链中的每一个，都有单独的被协议关闭的链表，树中的这一层被称为规则树节点 (RTN)。规则树节点支持以下四个协议：

1. TCP TCP 协议，如 SMTP, HTTP, FTP。
2. UDP UDP 协议 如 DNS lookups。
3. ICMP ICMP 协议 如 ping, traceroute。
4. IP IP 协议 如 IPsec, IGMP。

每一个协议链表中的是规则选项，称为选项树节点 (OTN)

- Content 中的内容，是具体采用 BM 或 AC 模式匹配算法进行查找的特征字符串。

规则头包含了规则动作、协议、IP 源地址和目的地址，子网掩码以及源端口和目标端口值等信息。而规则选项则包含警报信息以及用于确定是否触发响应规则动作而需检查的数据包区域位置的相关信息。

例如，Snort 系统的检测规则格式：

```
Alert tcp any any → 192.168.1.0/24 111 (content:"|00 01 86 a5|";msg:"mounted access";)
```

从规则的开头直到圆括号为止的部分称为规则头，而包含在圆括号之中的部分称为规则选项。在规则选项中，位于冒号“:”之前的词称为选项关键字 (Option keywords)。值得注意的是规则选项并不是对每一个规则而言都是必需的部分，它们只是用来更好地定义所要进行某种处理 (记录、警报等) 的数据包类型。只有当规则中的每一个元素都为真时，才能触发对应的规则动作。从此意义上讲，规则元素之间形成的是一种逻辑“与”的关系。与此同时，在每个规则库文件中的各种规则集合之间形成的是一种更大范围上的逻辑“或”关系。

### 3.4.2 Snort 规则树

规则树[18]的组成：

- 链表头

Snort 规则树有 5 个单独的规则链，这些链作为“树”顶部的链表头

1. Activation 报警并且开启另外一个动态规则。
2. Dynamic 当被上层的激活规则调用时记录网络流量的日志。
3. Alert 产生报警并记录这个数据包。
4. Pass 忽略这个数据包。
5. Log 记录网络的流量 (不报警)。

- 规则树节点 (RTN) 和选项树节点 (OTN)

对于五个规则链中的每一个，都有单独的被协议关闭的链表，树中的这一层被称为规则树节点 (RTN)。规则树节点支持以下四个协议：

1. TCP TCP 协议，如 SMTP, HTTP, FTP。
2. UDP UDP 协议 如 DNS lookups。
3. ICMP ICMP 协议 如 ping, traceroute。
4. IP IP 协议 如 IPsec, IGMP。

每一个协议链表中的是规则选项，称为选项树节点 (OTN)

- Content 中的内容，是具体采用 BM 或 AC 模式匹配算法进行查找的特征字符串。



- Flow 中的内容是系统根据设置, 指定要链接到的检测插件。

初始化时, Snort 读入规则文件, 把规则集合组织成一个二维的链表结构, 这个链表结构包含: 规则树节点(Rule Tree Nodes, RTN)和选项树节点(Option Tree Nodes, OTN)。图 3.9[15]显示了一个 TCP 报警链规则节点组装 OTN 的过程, 并展示了 Snort 系统利用三维链表进行匹配的过程:

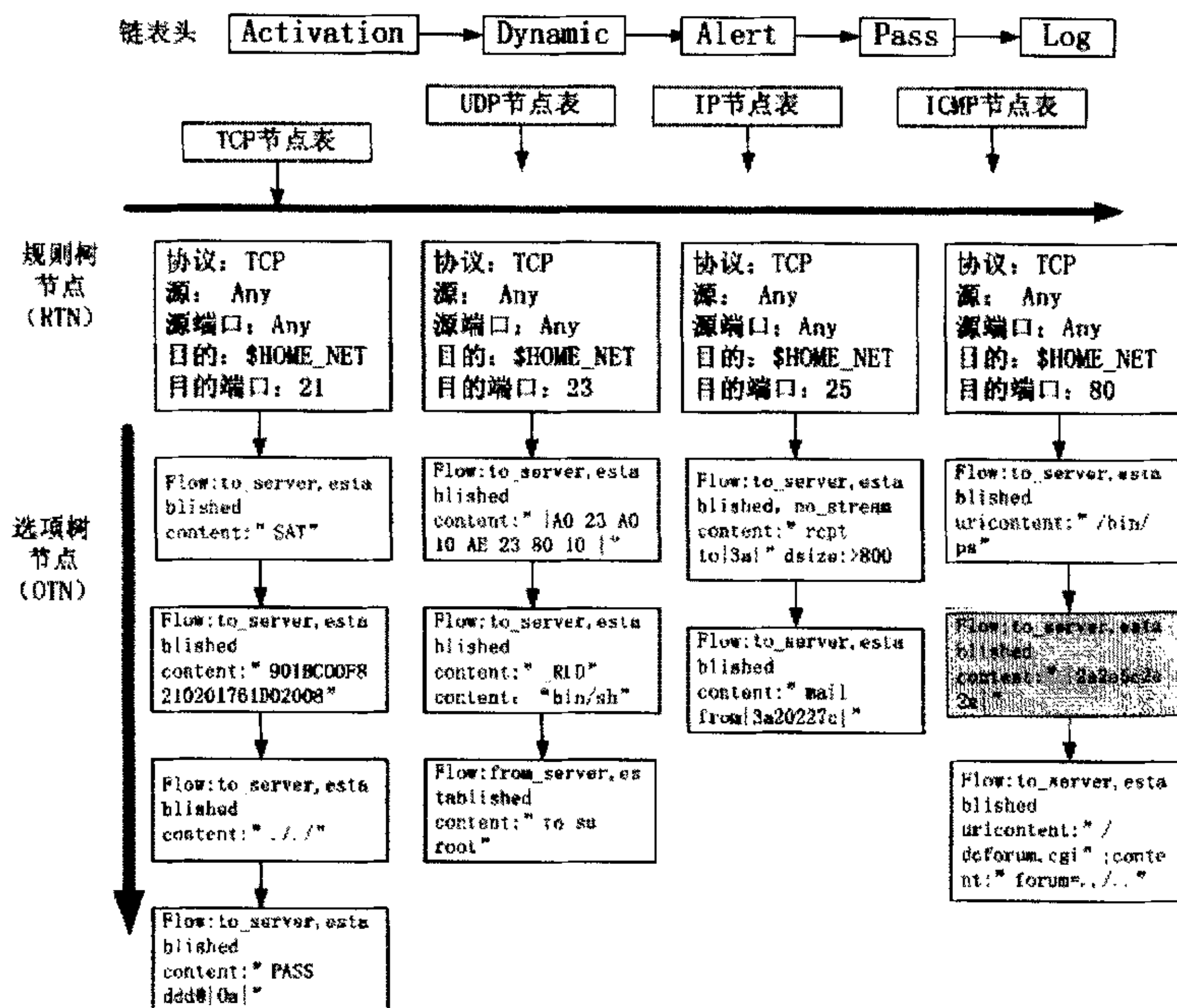


图 3.9 Snort 规则树

Fig. 3.9 Snort's Rule Tree

规则树节点 (RTN) 中包含规则的通用属性, 例如, 源 IP 地址、源端口号、目的 IP 地址、目的端口号、协议类型(TCP、ICMP、UDP)等。选项树节点 (OTN) 中包含一些可被添加到每条规则中的各种各样的信息, 例如, TCP 标志、ICMP 代码、类型、包负载的大小, 影响效率的主要瓶颈, 要查找的内容等。

RTN 节点从左到右组成一个链, 并作为各个 OTN 链的链头, 即 OTN 链是链在与之相关的 RTN 节点的下面的。按照给定的规则集对包进行检查的时, 首先沿着 RTN 链从左向右进行匹配直到找到一个匹配的 RTN 节点, 当要检查的包与某

一个 RTN 节点匹配时，沿着链在它下面的 OTN 链继续向下查找，对每个 OTN 中的选项的检查采用相应的插件函数进行，这些插件函数也同样被组织成链表的形式，当 OTN 节点中的一个选项与包匹配时，当前的插件函数调用链表中的下一个插件函数对该 OTN 节点中下一个选项进行检查。如果一个选项检查失败，则跳出该 OTN 节点，对 OTN 链表中的下一个 OTN 节点进行检查。为了提高效率，先对不需要对包内容进行检查的选项进行检查，然后再对需要对包内容进行检查的选项进行检查，以减少不必要的匹配所需的计算量。如果需要对包的内容进行检查，则使用著名的 BM 算法或经过优化的 AC 算法，将 OTN 节点选项所要求检查的模式串与包的内容进行精确的模式匹配。如果包中没有包含要找的串，继续与链表中下一个 OTN 节点中的选项所要查找的模式串进行匹配，直到在包中找到所要查找的内容或所要查找的串全部查找一遍为止。

### 3.5 小结

本章主要分析了 Snort 系统的架构及其主要模块，通过此章可以对 Snort 系统的体系结构、主要模块和 workflow 有一个详细了解，并且介绍了 Snort 特有的三维规则集合及其工作过程。下章对 Snort 系统的的关键模块—检测引擎进行了详细介绍，并着重对其模式匹配算法进行了分析改进。

一个 RTN 节点匹配时，沿着链在它下面的 OTN 链继续向下查找，对每个 OTN 中的选项的检查采用相应的插件函数进行，这些插件函数也同样被组织成链表的形式，当 OTN 节点中的一个选项与包匹配时，当前的插件函数调用链表中的下一个插件函数对该 OTN 节点中下一个选项进行检查。如果一个选项检查失败，则跳出该 OTN 节点，对 OTN 链表中的下一个 OTN 节点进行检查。为了提高效率，先对不需要对包内容进行检查的选项进行检查，然后再对需要对包内容进行检查的选项进行检查，以减少不必要的匹配所需的计算量。如果需要对包的内容进行检查，则使用著名的 BM 算法或经过优化的 AC 算法，将 OTN 节点选项所要求检查的模式串与包的内容进行精确的模式匹配。如果包中没有包含要找的串，继续与链表中下一个 OTN 节点中的选项所要查找的模式串进行匹配，直到在包中找到所要查找的内容或所要查找的串全部查找一遍为止。

### 3.5 小结

本章主要分析了 Snort 系统的架构及其主要模块，通过此章可以对 Snort 系统的体系结构、主要模块和 workflow 有一个详细了解，并且介绍了 Snort 特有的三维规则集合及其工作过程。下章对 Snort 系统的的关键模块—检测引擎进行了详细介绍，并着重对其模式匹配算法进行了分析改进。

## 第四章 Snort 检测引擎分析与改进

### 4.1 Snort 检测引擎工作原理

检测引擎是 Snort 系统中最重要的模块，其设计的好坏直接影响到系统的性能，Snort 检测引擎采用典型的特征匹配算法，在初始规则链表的基础上，2.0 版本重新构造了用于快速规则匹配的数据结构[19]，其工作流程可以分为以下三步。

#### 4.1.1 构造模式匹配链表

规则解析的主要接口函数是 ParseRulesFile()，在 SnortMain()中调用。但是，ParseRulesFile()只是一个过渡接口，其主要功能就是读取规则配置文件的每一行，并送到实际的规则解析模块 ParseRule()进行解析。ParseRule()的功能是解析所有的系统配置规则，包括插件配置、检测规则配置、变量定义、类型定义等[20]。

ParseRule()例程的基本算法流程如图 4.1 所示：

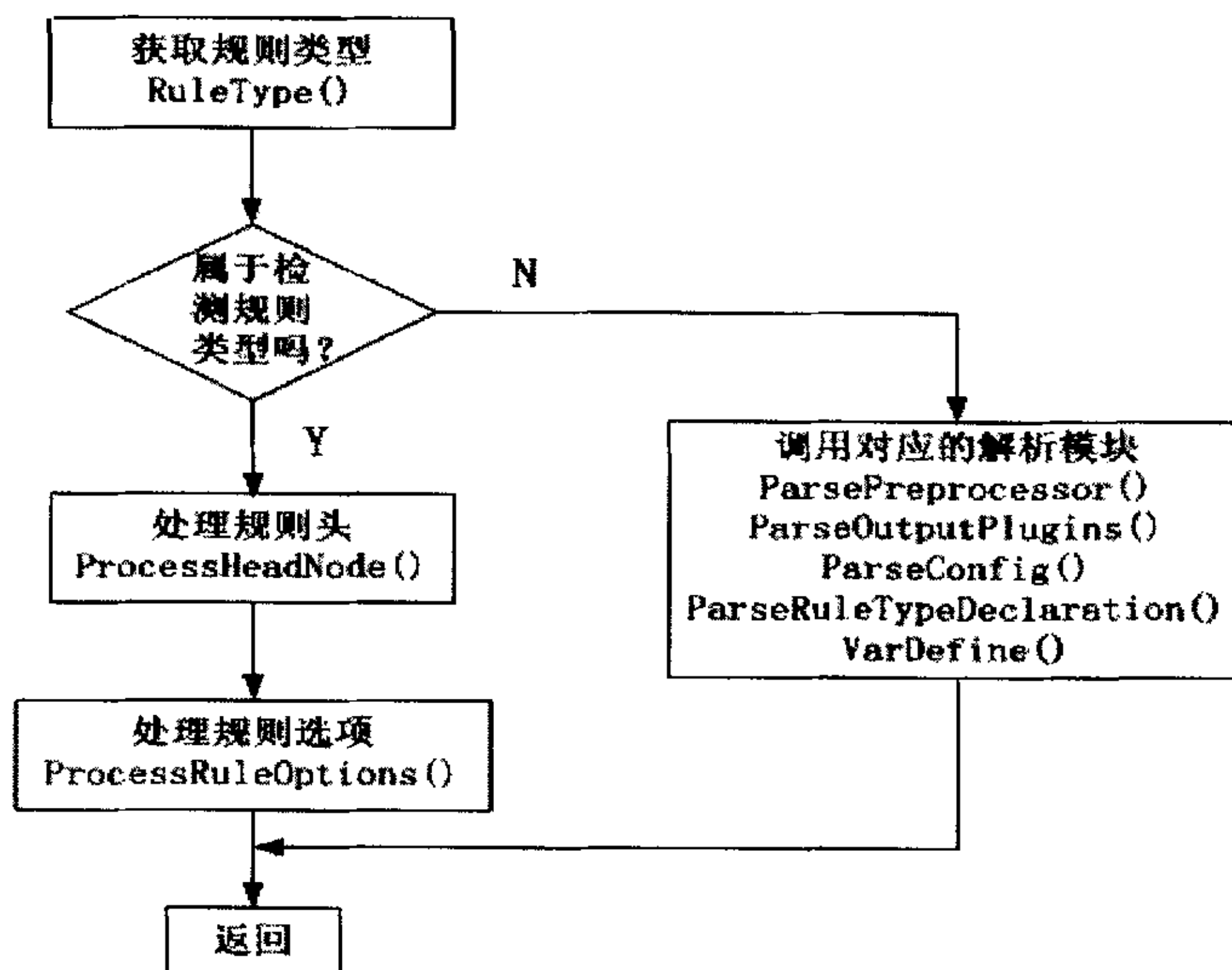


图 4.1 规则解析的工作流程

Fig. 4.1 Rule parsing flow

经过 ParseRuleFile()和 ParseRule()的处理后，系统生成如 3.4 节所述的三维规

## 第四章 Snort 检测引擎分析与改进

### 4.1 Snort 检测引擎工作原理

检测引擎是 Snort 系统中最重要模块，其设计的好坏直接影响到系统的性能，Snort 检测引擎采用典型的特征匹配算法，在初始规则链表的基础上，2.0 版本重新构造了用于快速规则匹配的数据结构[19]，其工作流程可以分为以下三步。

#### 4.1.1 构造模式匹配链表

规则解析的主要接口函数是 ParseRulesFile()，在 SnortMain()中调用。但是，ParseRulesFile()只是一个过渡接口，其主要功能就是读取规则配置文件的每一行，并送到实际的规则解析模块 ParseRule()进行解析。ParseRule()的功能是解析所有的系统配置规则，包括插件配置、检测规则配置、变量定义、类型定义等[20]。

ParseRule()例程的基本算法流程如图 4.1 所示：

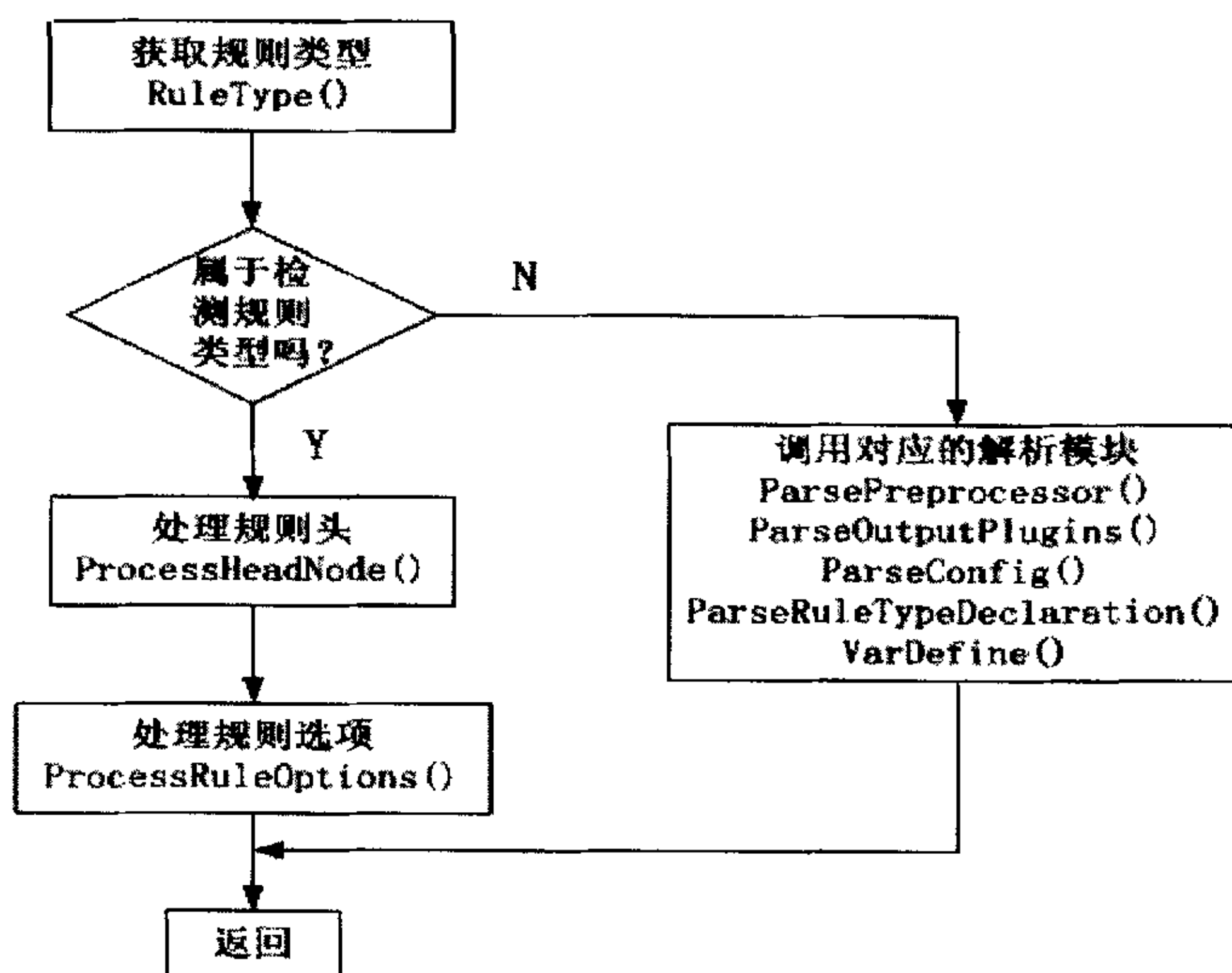


图 4.1 规则解析的工作流程

Fig. 4.1 Rule parsing flow

经过 ParseRuleFile()和 ParseRule()的处理后，系统生成如 3.4 节所述的三维规

则链表结构，接下去就是在这个基本的规则链表上，进一步划分，便于进行快速匹配。

#### 4.1.2 划分模式子集合

基本思想是通过规则中的目的端口和源端口值来划分类别，其要点如下：

- 如果源端口值为特定值，目的端口为任意值 (ANY)，则该规则加入到源端口值对应的子集合中；如果目的端口也为特定值，则该规则同时加入到目的端口对应的子集合中。
- 如果目的端口为特定值，而源端口为任意值 (ANY)，则该规则加入到目的端口对应的子集合中；如果源端口也为任意值 (ANY)，则该规则同时加入到源端口对应的子集合中。
- 如果目的端口和源端口都为任意值 (ANY)，则该规则加入到通用子集合中。
- 对于规则中端口值求反操作或者指定值范围的情况，等同于端口值为 ANY 情况加以处理。

此外，系统对于不含端口值的规则类型 (ICMP 或 IP 协议类型)，采用如下处理方法：

- 对于 ICMP 协议规则，如果规则中指定了 ICMP 类型值，则目的端口值指定为 ICMP 类型值；否则，规定为任意值 (ANY)。
- 对于 IP 协议类型，如果规则中指定了 IP 高层类型值，则目的端口值指定为 IP 高层协议类型值；否则，规定为任意值 (ANY)。
- 两者规则划分时，源端口值都指定为任意值 (ANY)。

在构造快速规则匹配引擎时，系统根据协议类型定义了 4 个 PORT\_RULE\_MAP 类型的全局变量：

```
Static PORT_RULE_MAP *prmTcpRTNX=NULL;
Static PORT_RULE_MAP *prmUdpRTNX=NULL;
Static PORT_RULE_MAP *prmIpRTNX=NULL;
Static PORT_RULE_MAP *prmIcmpRTNX=NULL;
```

fpCreateFastPacketDetection()例程进行构建操作时，利用上述四个变量来存放规则子集合的地址。PORT\_RULE\_MAP 数据结构中所涉及的数据结构类型 PORT\_GROUP，其负责存放根据特定端口值划分的规则子集合。在 PORT\_GROUP 结构中，定义了三组不同类型的规则节点链表，分别表示包含 content 内容匹配选项的规则、包含 uricontent 匹配选项的规则和没有内容匹配选项的规则。



则链表结构，接下去就是在这个基本的规则链表上，进一步划分，便于进行快速匹配。

#### 4.1.2 划分模式子集合

基本思想是通过规则中的目的端口和源端口值来划分类别，其要点如下：

- 如果源端口值为特定值，目的端口为任意值 (ANY)，则该规则加入到源端口值对应的子集合中；如果目的端口也为特定值，则该规则同时加入到目的端口对应的子集合中。
- 如果目的端口为特定值，而源端口为任意值 (ANY)，则该规则加入到目的端口对应的子集合中；如果源端口也为任意值 (ANY)，则该规则同时加入到源端口对应的子集合中。
- 如果目的端口和源端口都为任意值 (ANY)，则该规则加入到通用子集合中。
- 对于规则中端口值求反操作或者指定值范围的情况，等同于端口值为 ANY 情况加以处理。

此外，系统对于不含端口值的规则类型 (ICMP 或 IP 协议类型)，采用如下处理方法：

- 对于 ICMP 协议规则，如果规则中指定了 ICMP 类型值，则目的端口值指定为 ICMP 类型值；否则，规定为任意值 (ANY)。
- 对于 IP 协议类型，如果规则中指定了 IP 高层类型值，则目的端口值指定为 IP 高层协议类型值；否则，规定为任意值 (ANY)。
- 两者规则划分时，源端口值都指定为任意值 (ANY)。

在构造快速规则匹配引擎时，系统根据协议类型定义了 4 个 PORT\_RULE\_MAP 类型的全局变量：

```
Static PORT_RULE_MAP *prmTcpRTNX=NULL;
Static PORT_RULE_MAP *prmUdpRTNX=NULL;
Static PORT_RULE_MAP *prmIpRTNX=NULL;
Static PORT_RULE_MAP *prmIcmpRTNX=NULL;
```

fpCreateFastPacketDetection()例程进行构建操作时，利用上述四个变量来存放规则子集合的地址。PORT\_RULE\_MAP 数据结构中所涉及的数据结构类型 PORT\_GROUP，其负责存放根据特定端口值划分的规则子集合。在 PORT\_GROUP 结构中，定义了三组不同类型的规则节点链表，分别表示包含 content 内容匹配选项的规则、包含 uricontent 匹配选项的规则和没有内容匹配选项的规则。

根据划分标准而得到的各个规则子集合中，实际包含的都是 OTNX 结构类型的指针值，所指向的是初始规则链表结构中的对应节点。如此，将快速规则匹配引擎中的新构建的数据结构与原始的规则链表数据结构联系起来。其中，OTNX 数据结构是关键的桥梁。

划分规则子集合的函数 `fpCreateFastPacketDetection()` 的主流程算法如图 4.2[20]所示：

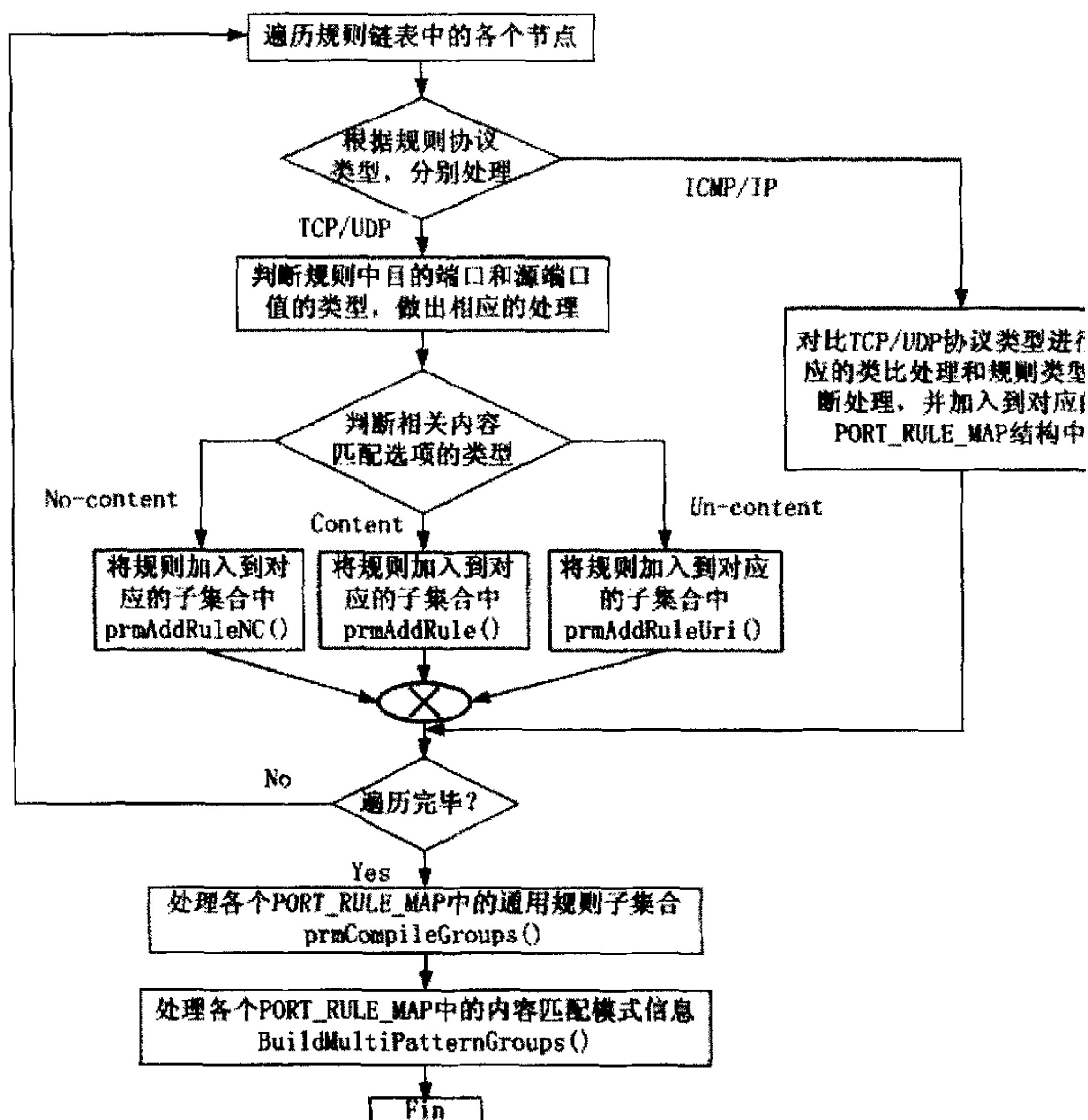


图 4.2 快速规则匹配引擎的构建

Fig. 4.2 The establishment of Fast Rule Matching Engine

函数 `fpCreateFastPacketDetection()` 在完成规则链表的遍历操作后，即完成了

各个规则子集合的初步划分,所有的规则节点都加入到如下三种类型的子集合中:

- 对应特定源端口值的 PORT\_GROUP 结构的规则链表。
- 对应特定目的端口值的 PORT\_GROUP 结构的规则链表。
- 通用规则节点链表。

对于通用规则节点链表而言,在构建快速规则匹配引擎中,它仅是作为一个过渡性的数据结构,对于未来的检测任务而言,每个数据包都有特定的源/目的端口值,为了达成根据端口值进行快速规则匹配的任务,函数 `fpCreateFastPacketDetection()` 在完成遍历操作后,对每个 `PORT_RULE_MAP` 结构中的通用规则链表进行了进一步的处理,调用例程 `prmCompileGroups()`,该例程的主要功能就是将通用规则链表中的各个规则节点加入到另外对应于特定端口值的两个规则链表中,即在 `PORT_RULE_MAP` 结构中的 `PORT_GROUP` 数组中每个非空元素中加入。最后,为了适应多模式搜索引擎算法的需要 `fpCreateFastPacketDetection()` 调用 `BuildMultiPatternGroup()` 在每个 `PORT_GROUP` 结构中构建多模式搜索引擎所需要的数据结构。

### 4.1.3 进行模式匹配

例程 `fpEvalPacket()` 是主要的快速规则检测接口函数,它在 `Detect()` 中被调用,对当前数据包执行快速规则集匹配的任务。

从代码可以看出, `fpEvalPacket()` 的主体架构是根据当前数据包的高层协议类型,分别调用对应的规则快速匹配例程 `fpEvalHeaderTcp`、`fpEvalHeaderUdp`、`fpEvalHeaderIcmp` 等。如果当前的数据包仅仅是 IP 数据包,则调用 `fpEvalHeaderIp()`。

下面以 `fpEvalHeaderTcp()` 为例,分析说明具体的快速规则检测过程:函数 `fpEvalHeaderTcp()` 首先调用 `prmFindRuleGroupTcp()` 获取当前可用的规则子集合。然后根据返回值,分别以不同的规则子集合指针参数调用快速规则匹配历程 `fpEvalHeaderSW()`,执行具体的规则匹配任务。这就体现了规则集合划分的特点及按照规则子集合进行匹配的优势。执行具体的规则匹配任务的例程 `fpEvalHeaderSW()` 是 2.0 系统中对初始模块 `fpEvalHeader()` 进行了进一步优化后的检测模块。它比较旧模块的主要特点在于引入了多模式搜索匹配工作引擎。

通过以上分析可以看出,虽然 Snort2.0 比较老版本进行了很大变化,但其目的都是为了提高其基于特征匹配模块的效率。Snort 检测引擎的初始模块 `fpEvalHeader()` 的工作过程仅仅是简单的规则子集合遍历过程,所采用的模式匹配算法为 BM 串匹配算法,2.0 版本加入了的多模式搜索引擎,采用的算法为

各个规则子集合的初步划分,所有的规则节点都加入到如下三种类型的子集合中:

- 对应特定源端口值的 PORT\_GROUP 结构的规则链表。
- 对应特定目的端口值的 PORT\_GROUP 结构的规则链表。
- 通用规则节点链表。

对于通用规则节点链表而言,在构建快速规则匹配引擎中,它仅是作为一个过渡性的数据结构,对于未来的检测任务而言,每个数据包都有特定的源/目的端口值,为了达成根据端口值进行快速规则匹配的任务,函数 `fpCreateFastPacketDetection()` 在完成遍历操作后,对每个 `PORT_RULE_MAP` 结构中的通用规则链表进行了进一步的处理,调用例程 `prmCompileGroups()`,该例程的主要功能就是将通用规则链表中的各个规则节点加入到另外对应于特定端口值的两个规则链表中,即在 `PORT_RULE_MAP` 结构中的 `PORT_GROUP` 数组中每个非空元素中加入。最后,为了适应多模式搜索引擎算法的需要 `fpCreateFastPacketDetection()` 调用 `BuildMultiPatternGroup()` 在每个 `PORT_GROUP` 结构中构建多模式搜索引擎所需要的数据结构。

### 4.1.3 进行模式匹配

例程 `fpEvalPacket()` 是主要的快速规则检测接口函数,它在 `Detect()` 中被调用,对当前数据包执行快速规则集匹配的任务。

从代码可以看出, `fpEvalPacket()` 的主体架构是根据当前数据包的高层协议类型,分别调用对应的规则快速匹配例程 `fpEvalHeaderTcp`、`fpEvalHeaderUdp`、`fpEvalHeaderIcmp` 等。如果当前的数据包仅仅是 IP 数据包,则调用 `fpEvalHeaderIp()`。

下面以 `fpEvalHeaderTcp()` 为例,分析说明具体的快速规则检测过程:函数 `fpEvalHeaderTcp()` 首先调用 `prmFindRuleGroupTcp()` 获取当前可用的规则子集合。然后根据返回值,分别以不同的规则子集合指针参数调用快速规则匹配历程 `fpEvalHeaderSW()`,执行具体的规则匹配任务。这就体现了规则集合划分的特点及按照规则子集合进行匹配的优势。执行具体的规则匹配任务的例程 `fpEvalHeaderSW()` 是 2.0 系统中对初始模块 `fpEvalHeader()` 进行了进一步优化后的检测模块。它比较旧模块的主要特点在于引入了多模式搜索匹配工作引擎。

通过以上分析可以看出,虽然 Snort2.0 比较老版本进行了很大变化,但其目的都是为了提高其基于特征匹配模块的效率。Snort 检测引擎的初始模块 `fpEvalHeader()` 的工作过程仅仅是简单的规则子集合遍历过程,所采用的模式匹配算法为 BM 串匹配算法,2.0 版本加入了的多模式搜索引擎,采用的算法为

Aho-Corasick 多模式匹配算法。所以，对滥用入侵检测系统来说，采用更为快速的模式匹配算法是非常必要的，下节讨论 Snort 系统所采用的模式匹配算法。

## 4.2 Snort 检测引擎的模式匹配算法

Snort 系统提供了可选择的搜索匹配算法，2.0 版本之前的系统检测引擎主要采用 BM 模式匹配算法，实现此算法的模块为 mstring.c 文件。2.0 版本加入了基于 Aho-Corasick 多模式搜索引擎来加快系统的匹配速度，实现该算法的模块为 acsmx2.c 文件。下面对此两种算法分别进行分析。

### 4.2.1 Boyer-Moore 算法原理

1977 年 Boyer 和 Moore 提出了一个全新的 Boyer-Moore(BM)算法[21]，匹配过程中，模式从左向右移动，但字符的比较却从右向左进行，即按  $pattern[m-1]$ 、 $pattern[m-2]$ 、 $\dots$ 、 $pattern[0]$  的次序进行比较，在发现不匹配时，算法根据预先计算好的两个数组将模式向右移动尽可能远的距离，定义这两个数组为 shift 和 skip。

假定失配发生时的情况如图 4.3 所示，此时已经匹配的部分为 U，正文的 b 字符与模式的 a 字符发生失配。

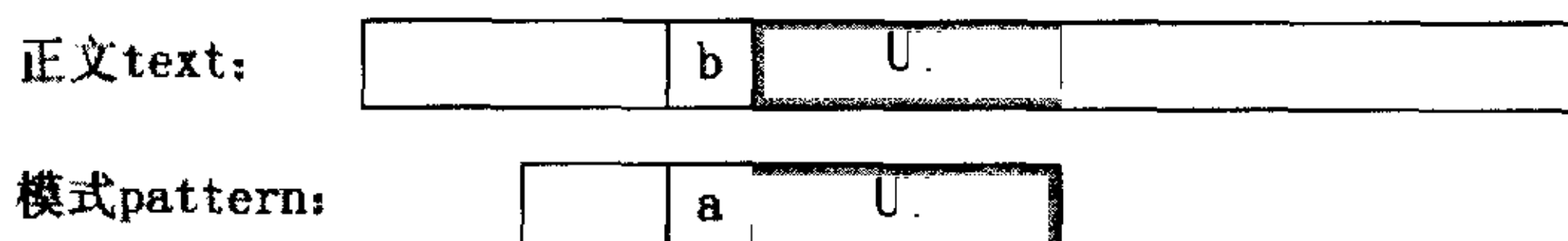


图 4.3 失配时的情况

Fig. 4.3 The case of mismatching

Shift 函数[22]通过对已经匹配部分的考查决定模式右移的长度，失配发生时，如果 U 在模式中再次出现且前缀不是 a 字符(例如是 c 字符)，则将模式串右移使得正文中的 U 与满足上述条件的最右边的 U 对齐。如图 4.4 所示：



Aho-Corasick 多模式匹配算法。所以，对滥用入侵检测系统来说，采用更为快速的模式匹配算法是非常必要的，下节讨论 Snort 系统所采用的模式匹配算法。

## 4.2 Snort 检测引擎的模式匹配算法

Snort 系统提供了可选择的搜索匹配算法，2.0 版本之前的系统检测引擎主要采用 BM 模式匹配算法，实现此算法的模块为 mstring.c 文件。2.0 版本加入了基于 Aho-Corasick 多模式搜索引擎来加快系统的匹配速度，实现该算法的模块为 acsmx2.c 文件。下面对此两种算法分别进行分析。

### 4.2.1 Boyer-Moore 算法原理

1977 年 Boyer 和 Moore 提出了一个全新的 Boyer-Moore(BM)算法[21]，匹配过程中，模式从左向右移动，但字符的比较却从右向左进行，即按  $pattern[m-1]$ 、 $pattern[m-2]$ 、 $\dots$ 、 $pattern[0]$  的次序进行比较，在发现不匹配时，算法根据预先计算好的两个数组将模式向右移动尽可能远的距离，定义这两个数组为 shift 和 skip。

假定失配发生时的情况如图 4.3 所示，此时已经匹配的部分为 U，正文的 b 字符与模式的 a 字符发生失配。

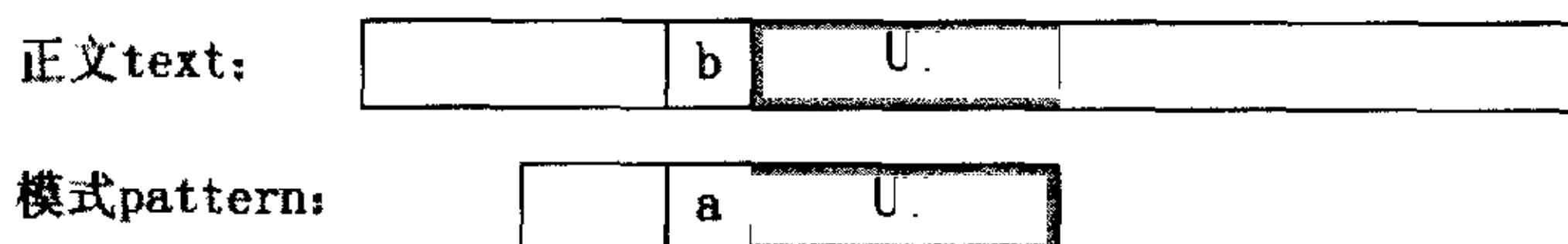


图 4.3 失配时的情况

Fig. 4.3 The case of mismatching

Shift 函数[22]通过对已经匹配部分的考查决定模式右移的长度，失配发生时，如果 U 在模式中再次出现且前缀不是 a 字符(例如是 c 字符)，则将模式串右移使得正文中的 U 与满足上述条件的最右边的 U 对齐。如图 4.4 所示：



Aho-Corasick 多模式匹配算法。所以，对滥用入侵检测系统来说，采用更为快速的模式匹配算法是非常必要的，下节讨论 Snort 系统所采用的模式匹配算法。

## 4.2 Snort 检测引擎的模式匹配算法

Snort 系统提供了可选择的搜索匹配算法，2.0 版本之前的系统检测引擎主要采用 BM 模式匹配算法，实现此算法的模块为 mstring.c 文件。2.0 版本加入了基于 Aho-Corasick 多模式搜索引擎来加快系统的匹配速度，实现该算法的模块为 acsmx2.c 文件。下面对此两种算法分别进行分析。

### 4.2.1 Boyer-Moore 算法原理

1977 年 Boyer 和 Moore 提出了一个全新的 Boyer-Moore(BM)算法[21]，匹配过程中，模式从左向右移动，但字符的比较却从右向左进行，即按  $pattern[m-1]$ 、 $pattern[m-2]$ 、 $\dots$ 、 $pattern[0]$  的次序进行比较，在发现不匹配时，算法根据预先计算好的两个数组将模式向右移动尽可能远的距离，定义这两个数组为 shift 和 skip。

假定失配发生时的情况如图 4.3 所示，此时已经匹配的部分为 U，正文的 b 字符与模式的 a 字符发生失配。

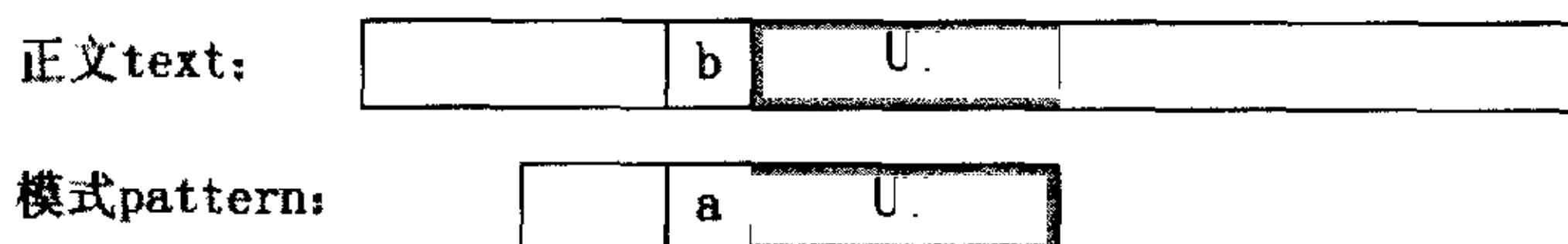


图 4.3 失配时的情况

Fig. 4.3 The case of mismatching

Shift 函数[22]通过对已经匹配部分的考查决定模式右移的长度，失配发生时，如果 U 在模式中再次出现且前缀不是 a 字符(例如是 c 字符)，则将模式串右移使得正文中的 U 与满足上述条件的最右边的 U 对齐。如图 4.4 所示：

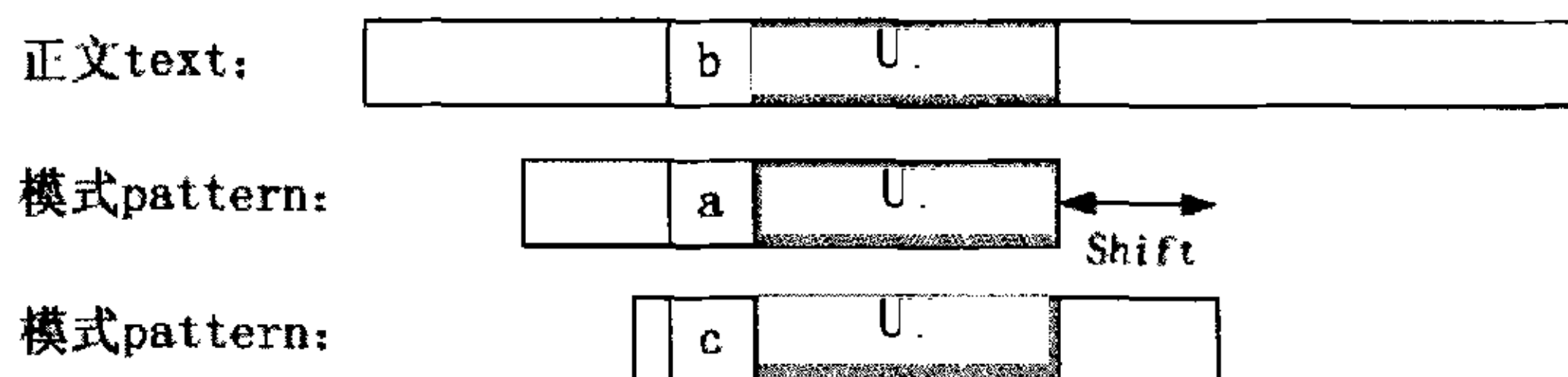


图 4.4 shift 的第一种情况

Fig. 4.4 Shift's first case

如果 U 在模式中没有再次出现或者再次出现但前缀是 a 字符, 则不能按照上述方法右移。如果模式存在一个前缀 V, 该 V 也是 U 的后缀, 那么找到这样一个最长的 V, 将模式右移, 使模式前缀 V 与正文中 U 的后缀 V 对齐, 如图 4.5 所示:

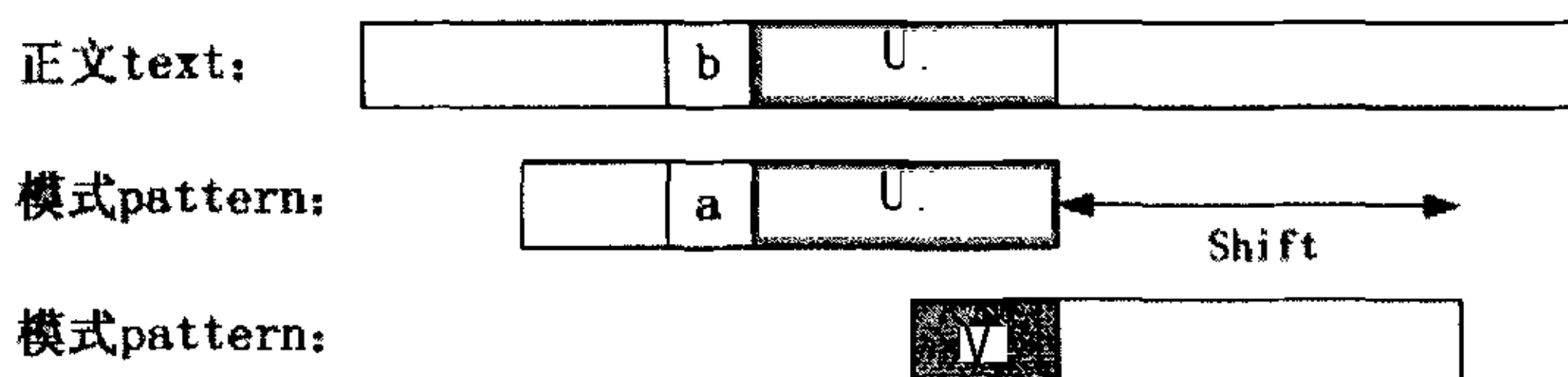


图 4.5 shift 的第二种情况

Fig. 4.5 Shift's second case

在极端情况下, 不存在这样的 V, 模式串将跳过 U, skip 函数[22]需要考查造成失配的正文字符(此时为 b) 在模式中出现的的情况。如果 b 在模式中存在, 则右移模式使正文 b 与模式中最右边的 b 对齐。如图 4.6 所示:

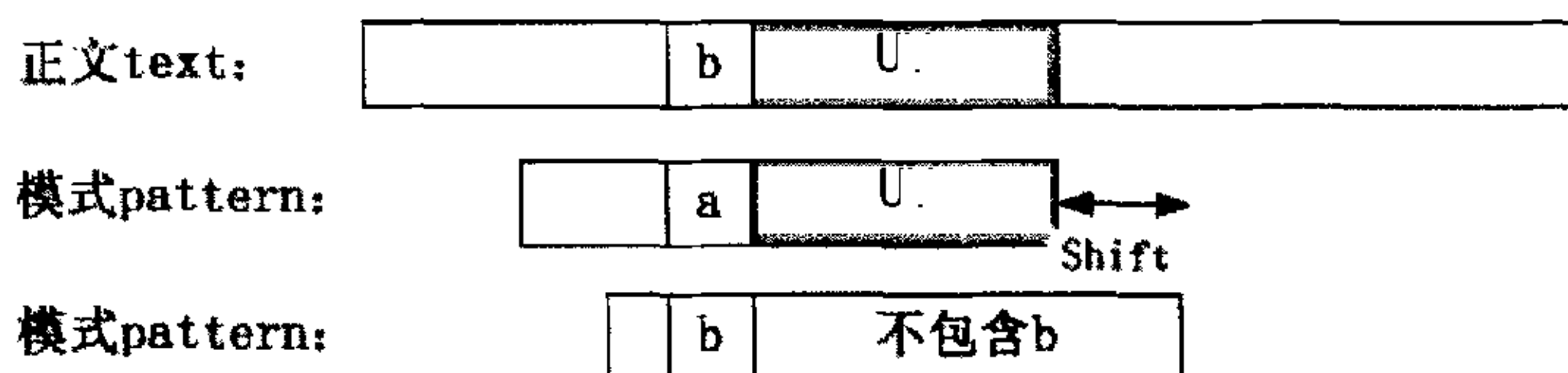


图 4.6 skip 的第一种情况

Fig. 4.6 Skip's first case

如果 b 在模式中不出现, 则将模式右移跳过字符 b, 如图 4.7 所示:

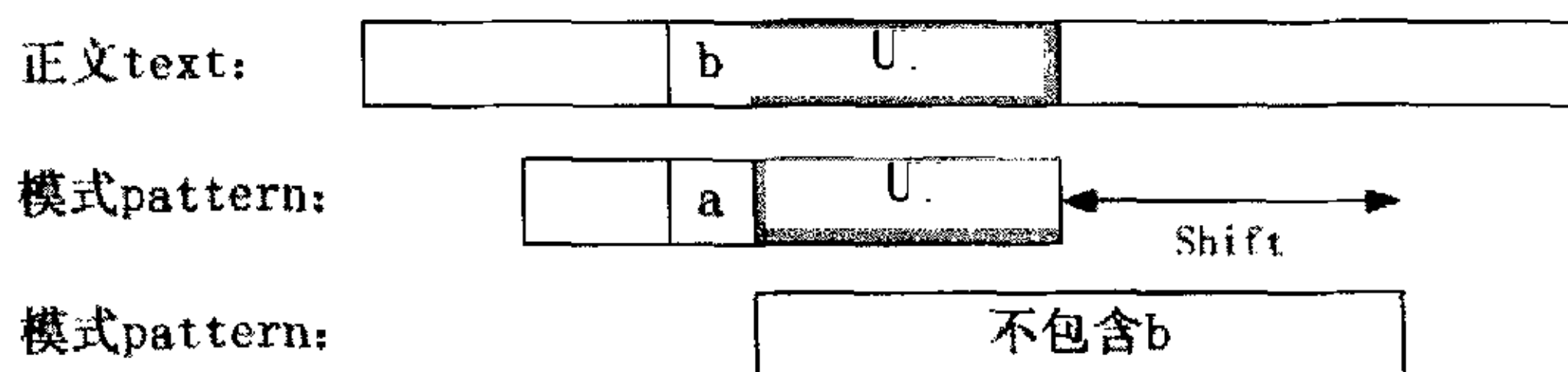


图 4.7 skip 的第二种情况

Fig. 4.7 Skip's second case

注意 skip 有可能是负值,因此在移动模式的时候,BM 算法应选取 shift 和 skip 的最大值。skip 数组对于字符空间较小的情况效果并不显著,但当字符空间很大而模式串较短时,例如字符空间为 ASC II 表时,将变得非常有效。BM 算法平均时间复杂度为  $O(n/m)$ 。

### 4.2.2 Boyer-Moore 模式匹配算法不足

入侵检测系统是在高网速和存在敌对环境的情况下具备实时匹配能力的专门应用程序。从算法分析可以看出,由于需要使用两个数组存储移动长度,需要在进行匹配之前对模式串计算装填数组,所以预处理时间花费比较大。其次,在入侵检测环境中,当数据包匹配定义的某条规则时,系统就报警并记录到日志中,然后会马上处理其它数据包,所以在匹配的过程中,应该让数据包尽可能的遍历完规则链表。尽管 BM 算法在单模式匹配中执行效率很高,但是在对多条规则进行匹配的情况下需要重复多遍调用,其效率并不是最佳的,尤其当前网络带宽不断增加,数据流量日益增大的环境中。所以在 snort2.0 系统中,设计者加入了多模式搜索匹配模块,采用的搜索算法为 Aho-Corasick 算法。

### 4.2.3 Aho-Corasick 多模式匹配算法原理

Aho-Corasick 状态机是多模式搜索算法之一,该算法 1975 年产生于贝尔实验室[23],最早被使用在图书馆的书目查询程序中,取得了很好的效果。有限状态机是系统所有可能状态的表示,以及该系统可接受的状态转换的信息。有限状态机的处理动作从最初状态开始,然后接受一个输入事件,根据输入事件从当前状态移动到下一个适当的状态。可以把状态机看作一个矩阵,行代表状态,列代表事件。根据输入的事件,可能要处理的动作以及进入或退出状态时的信息,矩阵元素提供了下一步要移动的适当状态。举例说明状态装换过程,如果当前状态是状态 10,并且下一输入事件为事件 6,则在矩阵中行为 10、列为 6 的元素值指

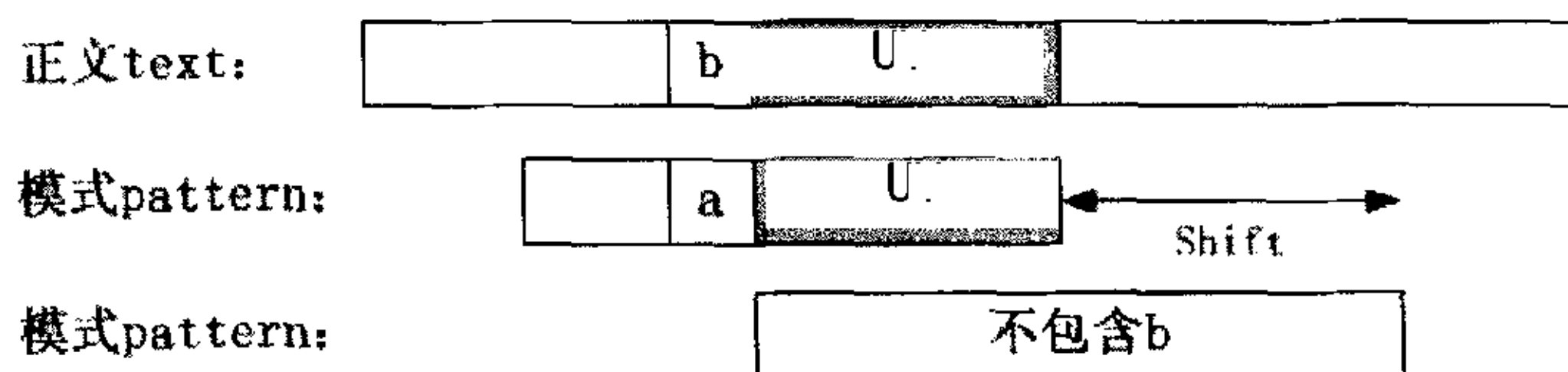


图 4.7 skip 的第二种情况

Fig. 4.7 Skip's second case

注意 skip 有可能是负值,因此在移动模式的时候,BM 算法应选取 shift 和 skip 的最大值。skip 数组对于字符空间较小的情况效果并不显著,但当字符空间很大而模式串较短时,例如字符空间为 ASC II 表时,将变得非常有效。BM 算法平均时间复杂度为  $O(n/m)$ 。

### 4.2.2 Boyer-Moore 模式匹配算法不足

入侵检测系统是在高网速和存在敌对环境的情况下具备实时匹配能力的专门应用程序。从算法分析可以看出,由于需要使用两个数组存储移动长度,需要在进行匹配之前对模式串计算装填数组,所以预处理时间花费比较大。其次,在入侵检测环境中,当数据包匹配定义的某条规则时,系统就报警并记录到日志中,然后会马上处理其它数据包,所以在匹配的过程中,应该让数据包尽可能的遍历完规则链表。尽管 BM 算法在单模式匹配中执行效率很高,但是在对多条规则进行匹配的情况下需要重复多遍调用,其效率并不是最佳的,尤其当前网络带宽不断增加,数据流量日益增大的环境中。所以在 snort2.0 系统中,设计者加入了多模式搜索匹配模块,采用的搜索算法为 Aho-Corasick 算法。

### 4.2.3 Aho-Corasick 多模式匹配算法原理

Aho-Corasick 状态机是多模式搜索算法之一,该算法 1975 年产生于贝尔实验室[23],最早被使用在图书馆的书目查询程序中,取得了很好的效果。有限状态机是系统所有可能状态的表示,以及该系统可接受的状态转换的信息。有限状态机的处理动作从最初状态开始,然后接受一个输入事件,根据输入事件从当前状态移动到下一个适当的状态。可以把状态机看作一个矩阵,行代表状态,列代表事件。根据输入的事件,可能要处理的动作以及进入或退出状态时的信息,矩阵元素提供了下一步要移动的适当状态。举例说明状态装换过程,如果当前状态是状态 10,并且下一输入事件为事件 6,则在矩阵中行为 10、列为 6 的元素值指

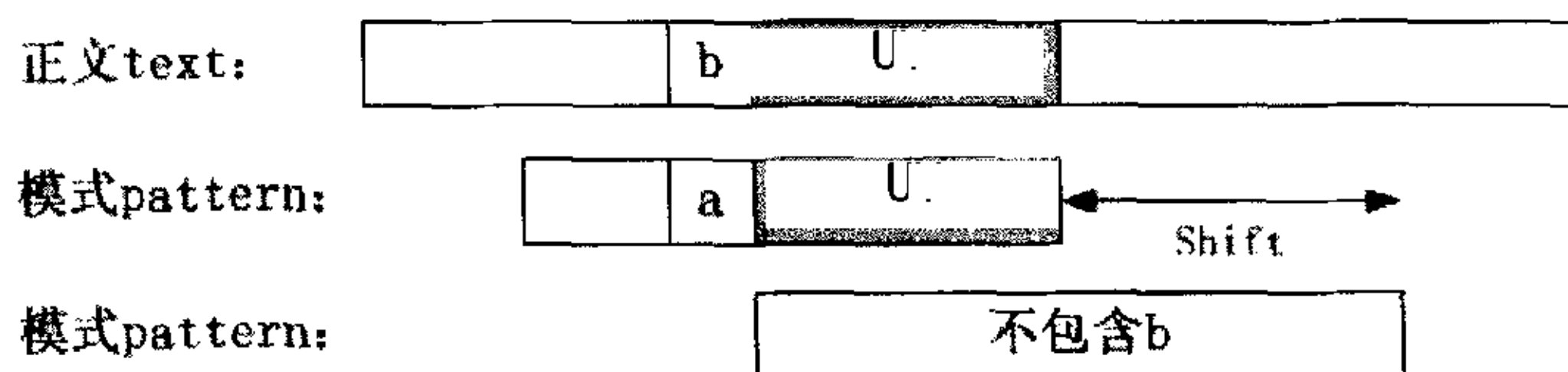


图 4.7 skip 的第二种情况

Fig. 4.7 Skip's second case

注意 skip 有可能是负值,因此在移动模式的时候,BM 算法应选取 shift 和 skip 的最大值。skip 数组对于字符空间较小的情况效果并不显著,但当字符空间很大而模式串较短时,例如字符空间为 ASC II 表时,将变得非常有效。BM 算法平均时间复杂度为  $O(n/m)$ 。

### 4.2.2 Boyer-Moore 模式匹配算法不足

入侵检测系统是在高网速和存在敌对环境的情况下具备实时匹配能力的专门应用程序。从算法分析可以看出,由于需要使用两个数组存储移动长度,需要在进行匹配之前对模式串计算装填数组,所以预处理时间花费比较大。其次,在入侵检测环境中,当数据包匹配定义的某条规则时,系统就报警并记录到日志中,然后会马上处理其它数据包,所以在匹配的过程中,应该让数据包尽可能的遍历完规则链表。尽管 BM 算法在单模式匹配中执行效率很高,但是在对多条规则进行匹配的情况下需要重复多遍调用,其效率并不是最佳的,尤其当前网络带宽不断增加,数据流量日益增大的环境中。所以在 snort2.0 系统中,设计者加入了多模式搜索匹配模块,采用的搜索算法为 Aho-Corasick 算法。

### 4.2.3 Aho-Corasick 多模式匹配算法原理

Aho-Corasick 状态机是多模式搜索算法之一,该算法 1975 年产生于贝尔实验室[23],最早被使用在图书馆的书目查询程序中,取得了很好的效果。有限状态机是系统所有可能状态的表示,以及该系统可接受的状态转换的信息。有限状态机的处理动作从最初状态开始,然后接受一个输入事件,根据输入事件从当前状态移动到下一个适当的状态。可以把状态机看作一个矩阵,行代表状态,列代表事件。根据输入的事件,可能要处理的动作以及进入或退出状态时的信息,矩阵元素提供了下一步要移动的适当状态。举例说明状态装换过程,如果当前状态是状态 10,并且下一输入事件为事件 6,则在矩阵中行为 10、列为 6 的元素值指



出了当前状态要改变的状态。

Aho-Corasick 算法的基本思想是：在预处理阶段，有限自动机算法建立三个函数，转向函数 goto，失效函数 failure 和输出函数 output，由此构造一个树型有限自动机。这样模式匹配的处理过程就变成了状态转换的处理过程，由“start”状态开始。在搜索查找阶段，通过这三个函数的交叉使用扫描文本，定位出关键字在文本中的所有出现位置。

下面举例说明 Aho-Corasick 算法的处理过程，例如用有限自动机，在目标串“antispam”中查找 { spam, stop, is} 中的模式串：

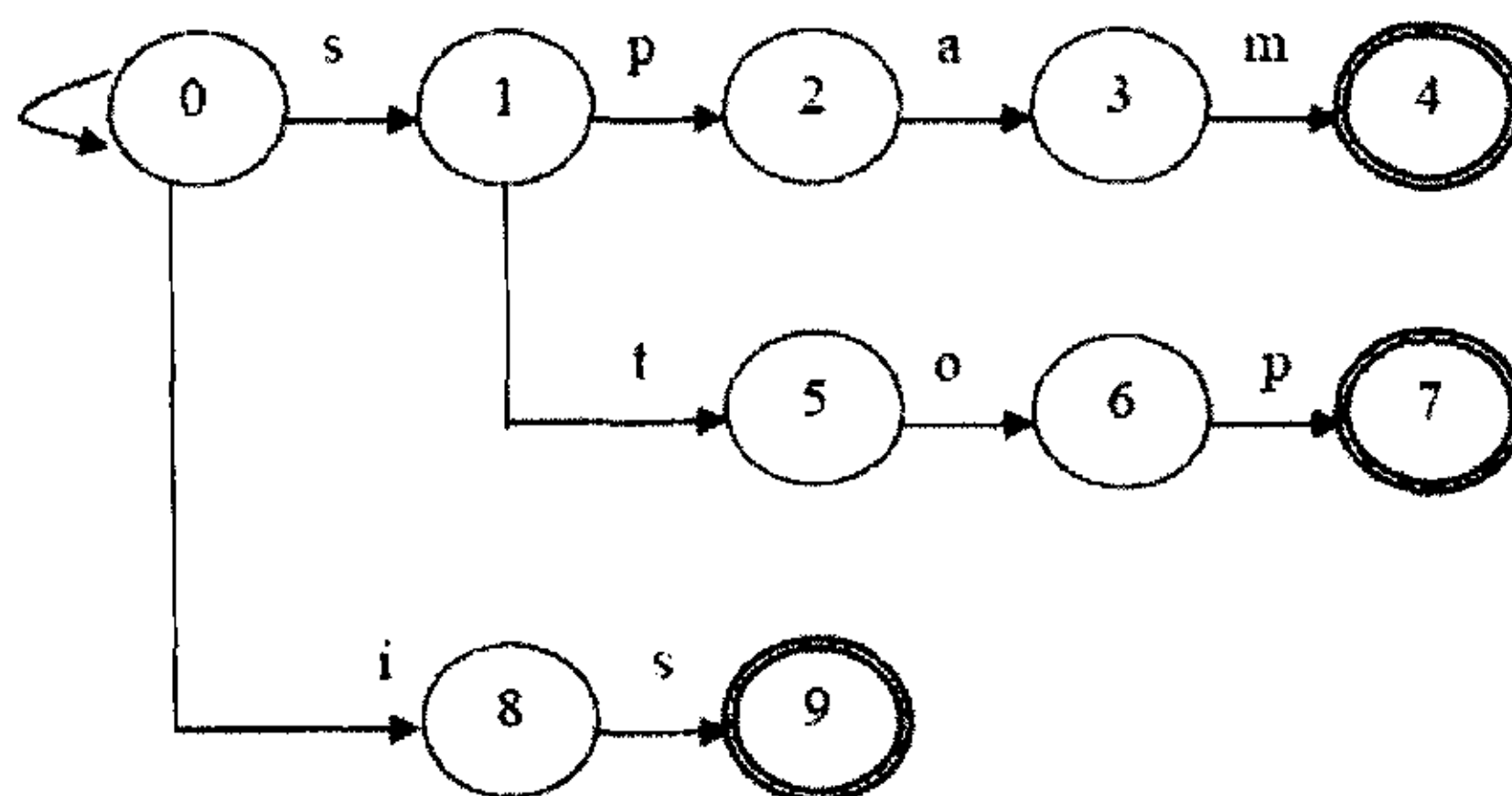


图 4.8 转向函数

Fig. 4.8 Goto function

有限自动机算法生成图 4.8 所示的树型的自动机，状态 0 是起始状态，状态的集合为 {0,1,2,...,9}，输入集合是中英文字符串，输出集合是模式串集合 { spam, stop, is}。

转向函数  $G(s,t)$ ，转向函数由一个文本字符串的字符匹配一个状态到另一个状态。如果到达了一个终态，由输出函数指定的模式就被成功匹配了。转向函数  $G(s,t)$  由图 4.8 中的箭头及上面的字符表示，定义了从箭头起始状态  $s$  时，如果“下一个”输入是箭头上的字符  $t$ ，则自动机的状态转到  $G(s, t)$ ，如  $G(0,i)=8$ 。

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	0	0	0	0	0	1

图 4.9 失效函数

Fig. 4.9 Failure function

失效函数  $F(s)$ ， $F(s)$  表示在状态  $s$  时，如果下一个输入  $t$  不在任何一个转向函数表示的箭头上，即一个输入字符导致了不匹配，那么自动机的状态将转到  $F(s)$ ，



失效函数被用来选择一个状态，由这个状态重新开始匹配。即如果  $G(s, t)$  不存在，则转到  $G(F(s), t)$  状态。  $F(s)$  的值如图 4.9 所示。

i	output(i)
4	{spam}
7	{stop}
9	{is}

图 4.10 输出函数

Fig. 4.10 Output function

输出函数  $P(s)$ ,  $P(s)$  表示自动机在状态  $s$  时可提供的输出，表示找到的目标串。图 4.8 中双圈的状态表示有输出， $P(s)$  的值如图 4.10 所示；其它状态的输出为空。

在算法中，有限自动机可用一个二维表描述，其中表的每一行代表自动机的一个状态，列坐标为 256 个 ASCII 码字符。这样，图 4.8 的自动机可描述为表 4.1 的形式：

表 4.1 有限自动机二维表

Table 4.1 Two-dimension table of Finite State Automata

	...	a	...	i	...	m	...	op	...	st	...
0		0		8		0		00		10	
1		0		8		0		02		15	
2		3		8		0		00		10	
3		0		8		4		00		10	
4		0		8		0		00		10	
5		0		8		0		60		10	
6		0		8		0		07		10	
7		0		8		0		00		10	
8		0		8		0		00		90	
9		0		8		0		02		15	

利用已构成的有限自动机进行多串一遍查找的过程如下：

- 从有限自动机的 0 状态出发，从目标字符串的第一个字符开始正向逐个取出的字符  $t$ ，并按转向函数  $G(s, t)$  进入下一状态，如果  $G(s, t)=fail$ ，则按  $G(F(s), t)$  进入下一状态。
- 当输出函数  $P(s)$  不为空时，输出  $P(s)$ 。

对于上面的例子，具体的匹配的过程见表 4.2 的状态转换序列，设文本  $T$  为

antispam。

表 4.2 模式匹配过程

Table 4.2 The process of pattern matching

目标串中的位置	1	2	3	4	5	6	7	8
输入字符	a	n	t	i	s	p	A	M
输入后的状态	0	0	0	8	9	2	3	4
输出	{}	{}	{}	{}	{is}	{}	{}	{spam}

当状态机 M 在状态 5 的时候并且当前的输入是 s，因为  $G(5,s)=9$ ，自动机进入状态 9，表明已经在输入文本字符串的第五个字符末尾处发现关键字“is”。在状态 9，当输入字符是 p 时，自动机完成两个状态转换。因为  $G(9,p)=fail$ ，所以 M 的输入状态为  $f(9)=1$ 。由于  $G(1,p)=2$ 。M 把 2 为输入状态继续下一个输入字符。在这个状态转换循环，没有输出状态。

#### 4.2.4 Aho-Corasick 多模式匹配算法不足

Aho-Corasick 算法(简称 AC 算法)是 Knuth-Morris-Pratt 算法和有限状态机的结合，是最早的一种多模式匹配的线性算法[24]，算法的第 1 部分由模式集 P 组成模式匹配自动机，第 2 部分利用模式匹配自动机进行模式匹配，将文本 y 作为输入，y 中模式字符串  $p_i$  出现的位置作为输出。模式匹配自动机由一系列的状态组成，每一个状态由一个数字代表，模式匹配自动机连续地读入 y 中的符号，进行状态转换，其中也会产生一些输出，输出的值就是模式字符串  $p_i$  出现的位置。模式匹配自动机的行为由 3 个函数来驱动：一个状态转换函数 g，一个状态失败函数 f 和一个输出函数 output。

实验证明在 1 万次搜索比较中，AC 算法的查找效率明显高于 BM 算法[25]。AC 算法的时间复杂度为  $O(n)$ ，预处理的时间随模式的大小呈线性变化，其时间复杂度为  $O(m)$ 。但是，和 KMP 算法类似，Aho-Corasick 算法在对文本进行搜索的过程中没有跳跃，而是按顺序输入文本 yi，无法跳过不必要的比较，因此在实际的搜索过程中，Aho-Corasick 算法不是性能最佳的搜索算法，并且有限自动机算法是以空间换时间的经典算法，当模式集较大时可能产生空间膨胀问题，这是因为对于一个总长度为 m 个字节的模式集，根据有限自动机的构造原理，在最坏的情况下具有 m 个状态，对于每个状态可能有 256 种不同的输入，因此描述有限自动机的二维表所需内存空间为  $256*m$  字节，当模式集较大时对于内存的要求将会变得很高。

antispam。

表 4.2 模式匹配过程

Table 4.2 The process of pattern matching

目标串中的位置	1	2	3	4	5	6	7	8
输入字符	a	n	t	i	s	p	A	M
输入后的状态	0	0	0	8	9	2	3	4
输出	{}	{}	{}	{}	{is}	{}	{}	{spam}

当状态机 M 在状态 5 的时候并且当前的输入是 s，因为  $G(5,s)=9$ ，自动机进入状态 9，表明已经在输入文本字符串的第五个字符末尾处发现关键字“is”。在状态 9，当输入字符是 p 时，自动机完成两个状态转换。因为  $G(9,p)=fail$ ，所以 M 的输入状态为  $f(9)=1$ 。由于  $G(1,p)=2$ 。M 把 2 为输入状态继续下一个输入字符。在这个状态转换循环，没有输出状态。

#### 4.2.4 Aho-Corasick 多模式匹配算法不足

Aho-Corasick 算法(简称 AC 算法)是 Knuth-Morris-Pratt 算法和有限状态机的结合，是最早的一种多模式匹配的线性算法[24]，算法的第 1 部分由模式集 P 组成模式匹配自动机，第 2 部分利用模式匹配自动机进行模式匹配，将文本 y 作为输入，y 中模式字符串  $p_i$  出现的位置作为输出。模式匹配自动机由一系列的状态组成，每一个状态由一个数字代表，模式匹配自动机连续地读入 y 中的符号，进行状态转换，其中也会产生一些输出，输出的值就是模式字符串  $p_i$  出现的位置。模式匹配自动机的行为由 3 个函数来驱动：一个状态转换函数 g，一个状态失败函数 f 和一个输出函数 output。

实验证明在 1 万次搜索比较中，AC 算法的查找效率明显高于 BM 算法[25]。AC 算法的时间复杂度为  $O(n)$ ，预处理的时间随模式的大小呈线性变化，其时间复杂度为  $O(m)$ 。但是，和 KMP 算法类似，Aho-Corasick 算法在对文本进行搜索的过程中没有跳跃，而是按顺序输入文本 yi，无法跳过不必要的比较，因此在实际的搜索过程中，Aho-Corasick 算法不是性能最佳的搜索算法，并且有限自动机算法是以空间换时间的经典算法，当模式集较大时可能产生空间膨胀问题，这是因为对于一个总长度为 m 个字节的模式集，根据有限自动机的构造原理，在最坏的情况下具有 m 个状态，对于每个状态可能有 256 种不同的输入，因此描述有限自动机的二维表所需内存空间为  $256*m$  字节，当模式集较大时对于内存的要求将会变得很高。

### 4.3 Snort 检测引擎模式匹配算法的改进

模式匹配是 IDS 本身的一个专门问题,需要对模式匹配算法的各个方面进行考虑,现今入侵检测系统基本都采用了多关键字查找算法来提高检测效率,主要的多模式匹配算法有 Aho-Corasick 自动机算法(1975 年),反向跳跃自动机算法(2000 年),SBOM 算法(1999 年),mgrep 算法(1994 年)等,实验数据表明使用哈希散列方法的算法在实时入侵检测的多模式匹配中具有显著优势[26],本文采用的 Long-Karp-Rabin2 多模式匹配算法就是基于此实验结果。

#### 4.3.1 Long-Karp-Rabin 多模式匹配算法原理

Long-Karp-Rabin[27]多模式匹配算法源于 Karp-Rabin (KR[28])串匹配算法的基本思想,是对 KR (Karp-Rabin)串匹配算法的扩展,是一种简洁的实时多关键词匹配算法,该算法主要使用数值处理来快速定位关键字位置。

KR 串匹配算法的基本思想是先定义一个指印函数,将模式串映射成一个比模式串短得多的指印(二进制位串数据),然后将正文中每一个长度为  $m$  的子串也映射成为一个比子串短得多的指印(二进制位串数据),算法的匹配比较过程是先比较模式串的指印函数值和正文子串的指印函数值,只有两者相等时才比较模式串与正文子串是否确实匹配。使用这种方法能以比较数字是否相等来代替费时较多的串比较过程,提高搜索速度的关键是指印函数值易于求出,并且该种算法可以用前一个指印函数值递推求出下一个指印函数值,递推公式如下:  $h[i+1] = ((h[i] - x \times \text{asc}(t[i])) \times d + \text{asc}(t[i+m])) \bmod q$ , 其中  $h[i]$  为前一个指印函数的值,  $t[i]$  为正文中第  $i$  个字符,  $m$  为子串长度。

进行关键词匹配的工作是固定的和最为耗时的,匹配入口点过多,进行关键词匹配的次数越多,算法消耗的时间就越长。K-R 串匹配算法为找到正确的关键字位置只进行数值计算,当搜索文本与关键字文本的哈希函数值相等时才进行耗时的字符串比较[29]。实际应用中, K-R 算法的效率低于 KMP, BM 等算法,但是 Karp-Rabin 算法基于数值计算比较的特点却可以成为多模式匹配搜索的算法基础[30]。

Long-Karp-Rabin 算法是一种跳跃型的算法,设计的主要思路是充分利用硬件多级流水线的效率,使用数值运算代替字节比较,对多关键字进行匹配,下面逐步介绍该算法思想:

第一步: 分解关键词

在多模式匹配中,模式是任意字符组合在一起构成的关键字,例如一个关键



### 4.3 Snort 检测引擎模式匹配算法的改进

模式匹配是 IDS 本身的一个专门问题,需要对模式匹配算法的各个方面进行考虑,现今入侵检测系统基本都采用了多关键字查找算法来提高检测效率,主要的多模式匹配算法有 Aho-Corasick 自动机算法(1975 年),反向跳跃自动机算法(2000 年),SBOM 算法(1999 年),mgrep 算法(1994 年)等,实验数据表明使用哈希散列方法的算法在实时入侵检测的多模式匹配中具有显著优势[26],本文采用的 Long-Karp-Rabin2 多模式匹配算法就是基于此实验结果。

#### 4.3.1 Long-Karp-Rabin 多模式匹配算法原理

Long-Karp-Rabin[27]多模式匹配算法源于 Karp-Rabin (KR[28])串匹配算法的基本思想,是对 KR (Karp-Rabin)串匹配算法的扩展,是一种简洁的实时多关键词匹配算法,该算法主要使用数值处理来快速定位关键字位置。

KR 串匹配算法的基本思想是先定义一个指印函数,将模式串映射成一个比模式串短得多的指印(二进制位串数据),然后将正文中每一个长度为  $m$  的子串也映射成为一个比子串短得多的指印(二进制位串数据),算法的匹配比较过程是先比较模式串的指印函数值和正文子串的指印函数值,只有两者相等时才比较模式串与正文子串是否确实匹配。使用这种方法能以比较数字是否相等来代替费时较多的串比较过程,提高搜索速度的关键是指印函数值易于求出,并且该种算法可以用前一个指印函数值递推求出下一个指印函数值,递推公式如下:  $h[i+1] = ((h[i] - x \times \text{asc}(t[i])) \times d + \text{asc}(t[i+m])) \bmod q$ , 其中  $h[i]$  为前一个指印函数的值,  $t[i]$  为正文中第  $i$  个字符,  $m$  为子串长度。

进行关键词匹配的工作是固定的和最为耗时的,匹配入口点过多,进行关键词匹配的次数越多,算法消耗的时间就越长。K-R 串匹配算法为找到正确的关键字位置只进行数值计算,当搜索文本与关键字文本的哈希函数值相等时才进行耗时的字符串比较[29]。实际应用中, K-R 算法的效率低于 KMP, BM 等算法,但是 Karp-Rabin 算法基于数值计算比较的特点却可以成为多模式匹配搜索的算法基础[30]。

Long-Karp-Rabin 算法是一种跳跃型的算法,设计的主要思路是充分利用硬件多级流水线的效率,使用数值运算代替字节比较,对多关键字进行匹配,下面逐步介绍该算法思想:

第一步: 分解关键词

在多模式匹配中,模式是任意字符组合在一起构成的关键字,例如一个关键

词  $y = "12345678"$ , ( $m = |y| = 8$ ,  $|y|$  表示关键字的长度),  $y$  可以分解成多个整数, 这些整数彼此相连, 同时后面一个整数前面部分和前个整数的后面部分相同。例如, 将  $|Y| = 8$  分解成 5 个整数的情况, 如图 4.11 所示:

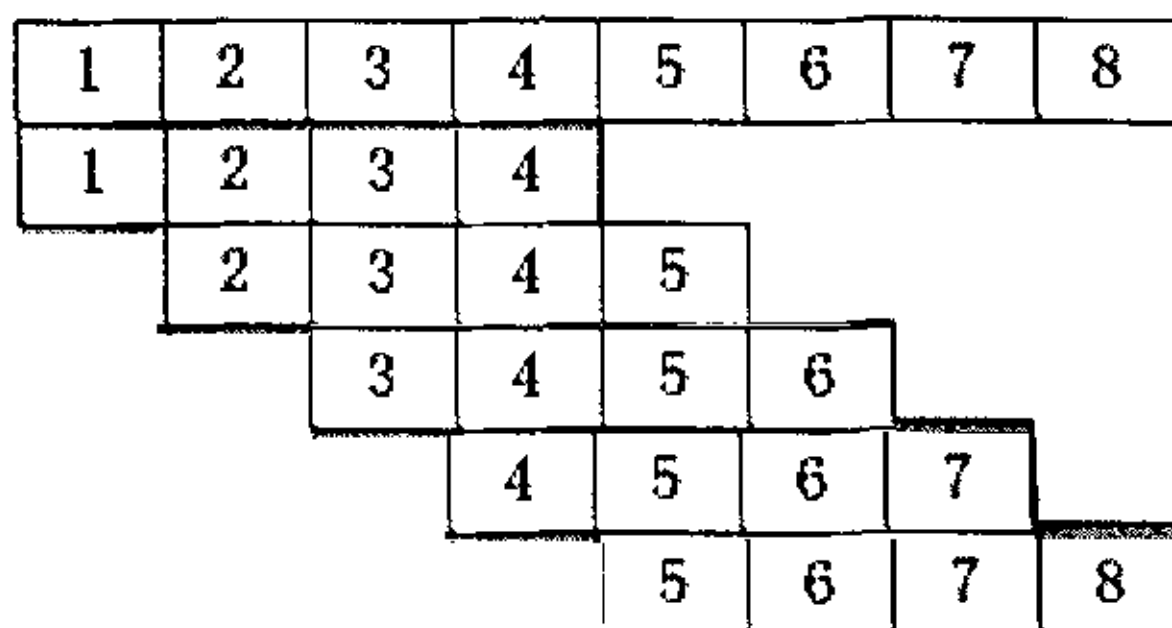


图 4.11  $|Y| = 8$  分解成 5 部分

Fig. 4.11  $|Y| = 8$  divide into 5 parts

第二步: 计算哈希表

通过第一步, 关键字分解成了多个整数, 然后把全部关键词建立的整数集合映射到一个表中, 称这个表为 CHECK 哈希表。如果字符片断  $z[i..i+w-1]$  表示的整数出现在任何一个关键词片断中, 那么  $CHECK[H(z[i..i+w-1])]$  ( $H()$  为哈希函数) 一定指向一个链表结构, 不是一个空指针。

第三步: 匹配过程:

1. 计算跳跃步长  $step = \minLen / w$ ;
2.  $size = m / w$ ;
3.  $p$  从  $step$  到  $size$  开始循环,  $p = p + step$ ;
4. 计算  $t = Buff[p] \bmod q$ ;
5. 如果  $CHECK[t]$  等于空, 则到第 3 步;
6. 利用  $CHECK[t]$  指向的链表结构, 进行完全匹配的比较, 如果发现了关键词则报告,  $p = p + 1$ ; 继续循环到第 3 步;

此算法中哈希函数的选择非常重要, 如果最短关键词长度  $n$ , 文本长度  $m$ , 关键词个数  $k$ ; 如要求 Long-Karp-Rabin 算法的第 5 步判断中,  $CHECK[t]$  不为空指针的概率低于 1%, 即要求 CHECK 哈希表的填充概率低于 1%, 公式  $q > 1 * k * (n - w + 1)$ , 算法的时间复杂度:

$$\frac{m/w}{(n/w - 1)} = \frac{m}{n - w} = \frac{m}{n} * \left( \frac{1}{1 - w/n} \right)$$



下面通过举例说明 Long-Karp-Rabin 算法的具体匹配过程:

关键字集合: {hero, where, redo};

文本为: Sregthayeurewherent;

哈希函数:  $\text{hash}(w[0 \dots m-1]) = (w[0] \cdot 2^{m-1} + w[1] \cdot 2^{m-2} + \dots + w[m-1] \cdot 2^0) \bmod q$   
 $q=37$ ;

最小关键字长度: 4,  $\text{step}=4/2=2$ ,  $\text{size}=19/2$  近似为 20。

● 首先进行关键字分解:

hero 分解为:  $\text{he} \rightarrow \{1, 0\}$ ,  $\text{er} \rightarrow \{1, 1\}$ ,  $\text{ro} \rightarrow \{1, 2\}$ ;

where 分解为:  $\text{wh} \rightarrow \{2, 0\}$ ,  $\text{he} \rightarrow \{2, 1\}$ ,  $\text{er} \rightarrow \{2, 2\}$ ,  $\text{re} \rightarrow \{2, 3\}$ ;

redo 分解为:  $\text{re} \rightarrow \{3, 0\}$ ,  $\text{ed} \rightarrow \{3, 1\}$ ,  $\text{do} \rightarrow \{3, 2\}$ ;

● 计算哈希表:

$\text{hash}(w[\text{'he'}]) \% 37 = 13 \rightarrow \{1, 0\}$

$\text{hash}(w[\text{'er'}]) \% 37 = 20 \rightarrow \{1, 1\}$

$\text{hash}(w[\text{'ro'}]) \% 37 = 6 \rightarrow \{1, 2\}$

$\text{hash}(w[\text{'wh'}]) \% 37 = 9 \rightarrow \{2, 0\}$

$\text{hash}(w[\text{'he'}]) \% 37 = 13 \rightarrow \{2, 1\}$

$\text{hash}(w[\text{'er'}]) \% 37 = 20 \rightarrow \{2, 2\}$

$\text{hash}(w[\text{'re'}]) \% 37 = 33 \rightarrow \{2, 3\}$

$\text{hash}(w[\text{'re'}]) \% 37 = 33 \rightarrow \{3, 0\}$

$\text{hash}(w[\text{'ed'}]) \% 37 = 6 \rightarrow \{3, 1\}$

$\text{hash}(w[\text{'do'}]) \% 37 = 15 \rightarrow \{3, 2\}$

当有碰撞发生时, 采用单链表方式, 将碰撞的节点链接, 将 CHECK 哈希表填充为:

CHECK[13]存储(1,0)→(2,1); CHECK[20]存储(1,1) →(2,2); CHECK[6]存储(1,2); CHECK[9]存储(2,0); CHECK[33]存储(2,3) →(3,0); CHECK[6]存储(3,1); CHECK[15]存储(3,2)。

● 进行匹配:

对文本 “Sregthayeurewherent” 从左到右进行查找, 如果经过计算的字符串的 CHECK 哈希表为空, 则无符合的关键字匹配, 指针向右移动 2 后, 继续查找。

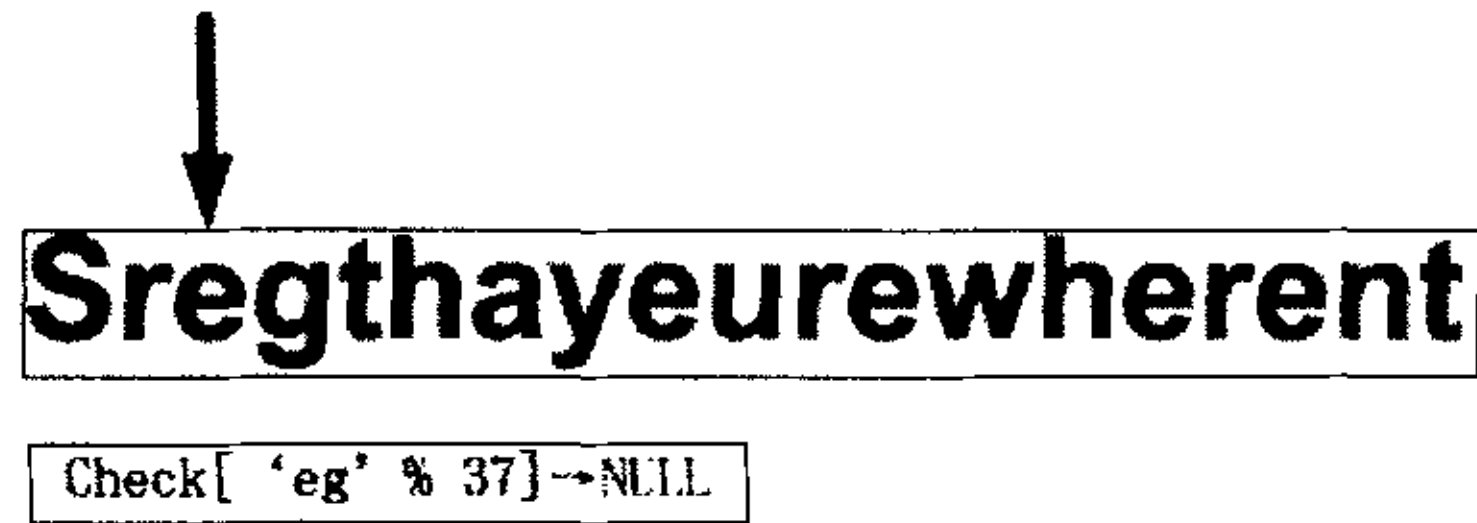


图 4.12 Long-Karp-Rabin 匹配过程(a)

Fig. 4.12 Long-Karp-Rabin matching process (a)

如图 4.12 所示，搜索到“eg”时，通过检查 CHECK 哈希表，发现该处值为空，所以没有关键字匹配，指针向右移动 2，继续搜索：

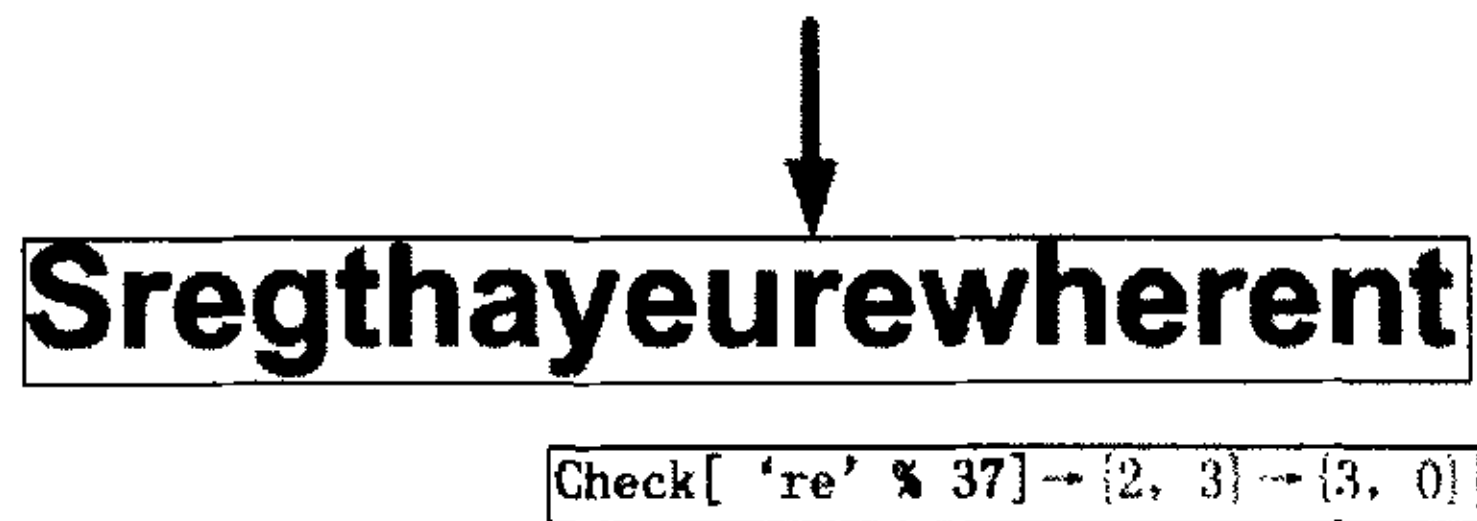


图 4.13 Long-Karp-Rabin 匹配过程(b)

Fig. 4.13 Long-Karp-Rabin matching process (b)

如图 4.13，当搜索到“re”时，发现“re”在 CHECK 哈希表中的值不空，所以有关键字可以匹配。根据 CHECK 哈希表中的值“{2, 3}和{3, 0}”，可以看出是关键字“where”和“redo”分解后的“re”匹配上了，所以根据链表值：{2, 3}, {3, 0}进行具体字符串匹配。根据位置{2, 3}，则关键字应该为“where”，往前搜索得字符为“y”，所以不匹配“where”，然后再根据{3, 0}，则关键字应该为“redo”，往后搜索得字符为“w”，所以不匹配“redo”，则在指针继续向右移动。

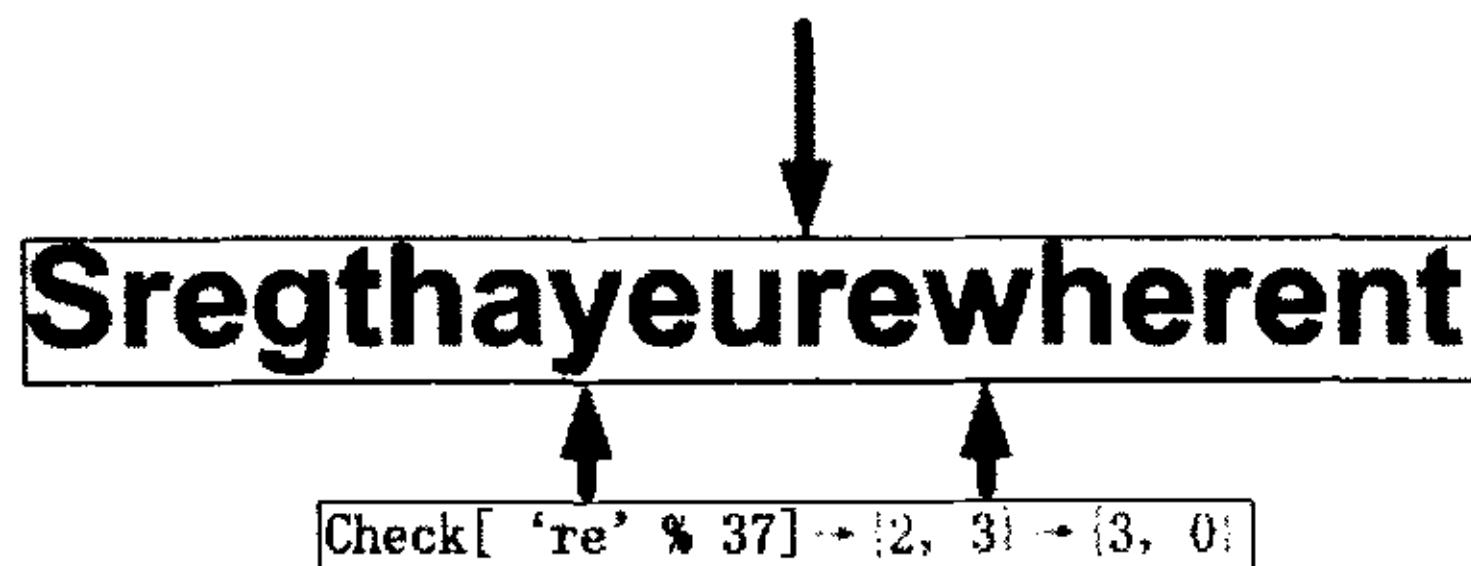


图 4.14 Long-Karp-Rabin 匹配过程(d)

Fig. 4.14 Long-Karp-Rabin matching process (d)

当指针移动到“wh”时，匹配过程如同前述，对照 CHECK 哈希表中“CHECK[‘wh’ % 37] → {2, 0}”，发现文本中的“where”关键字，如图 4.15：

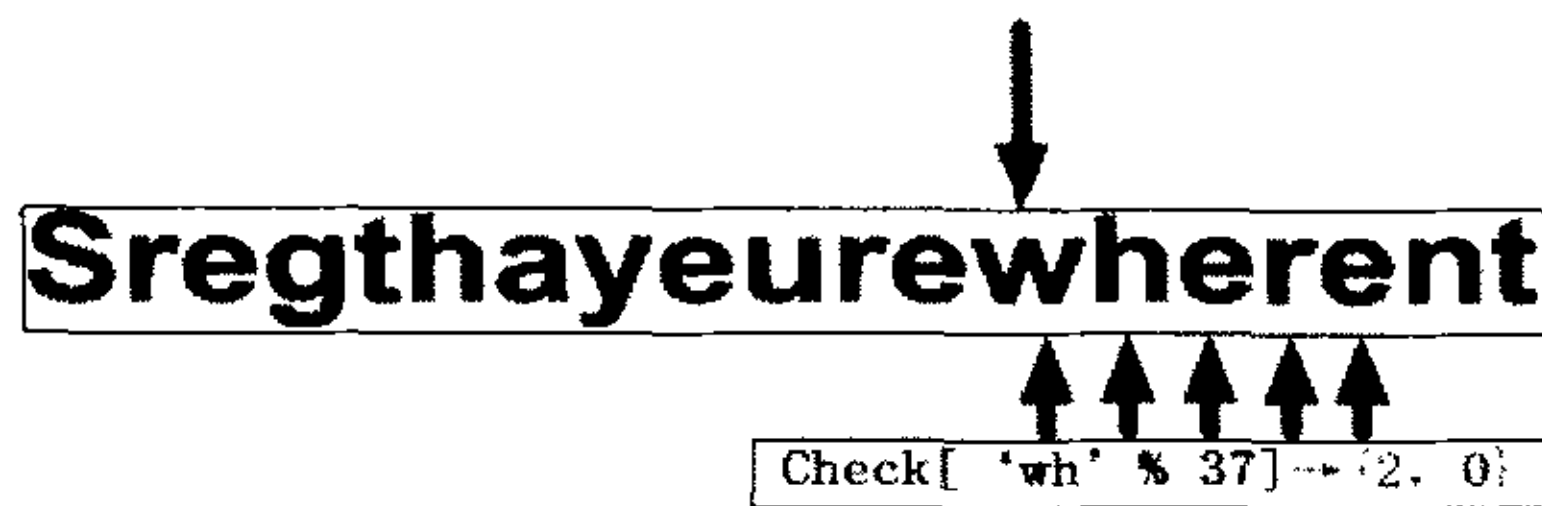


图 4.15 Long-Karp-Rabin 匹配过程(e)

Fig. 4.15 Long-Karp-Rabin matching process (e)

### 4.3.2 Long-Karp-Rabin 多模式匹配算法分析

Long-Karp-Rabin 多模式匹配算法利用数值快速查找关键字位置的优点，并实现了跳跃前进，所以查找更加快速。依照算法思想及参数设置，在 VC6.0 环境中本人用 c 语言实现了 Long-Karp-Rabin 多模式匹配算法，算法的流程图如下：

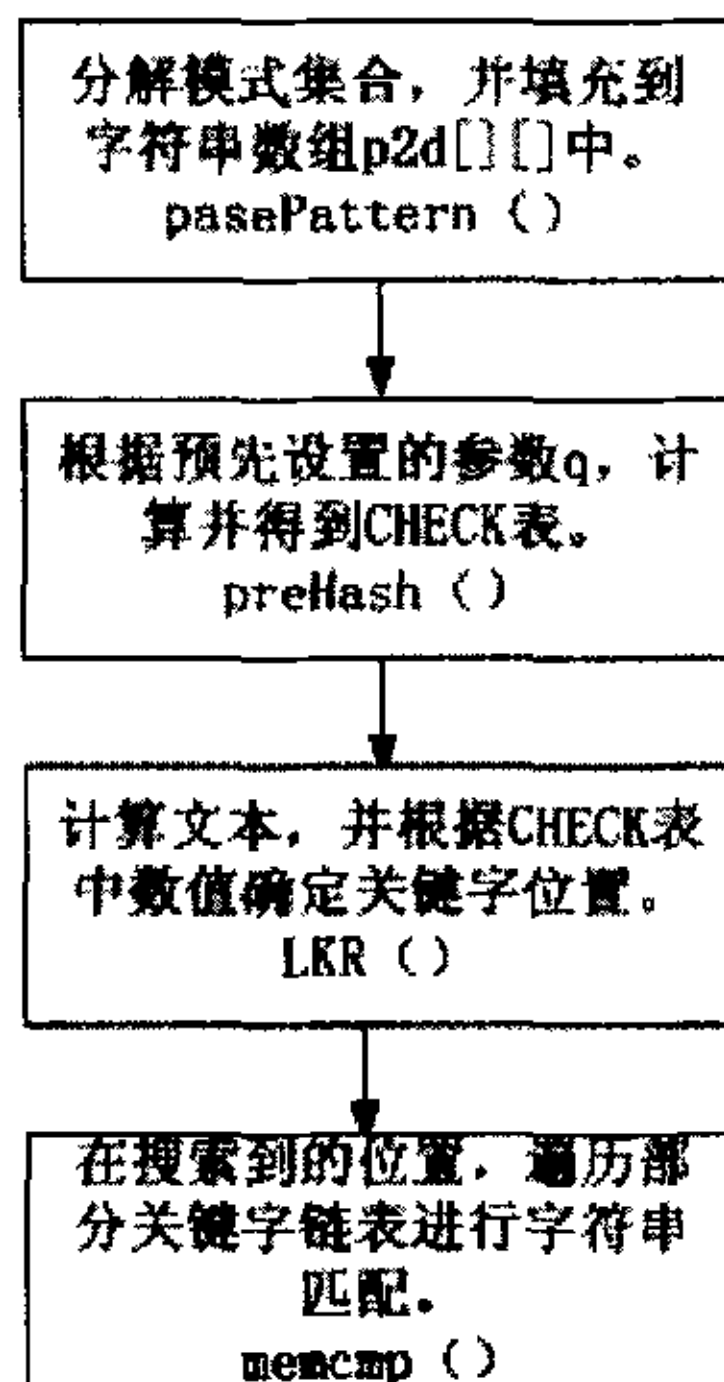


图 4.16 Long-Karp-Rabin 工作流程

Fig. 4.16 Long-Karp-Rabin working flow

从图 4.16 可以看出第一步分解模式集合后，得到多维 p2d 字符串数组：  
p2d[1][0]="he";p2d[1][1]="er";p2d[1][2]="ro";p2d[1][3]="null";……

当指针移动到“wh”时，匹配过程如同前述，对照 CHECK 哈希表中“CHECK[‘wh’ % 37] → {2, 0}”，发现文本中的“where”关键字，如图 4.15：

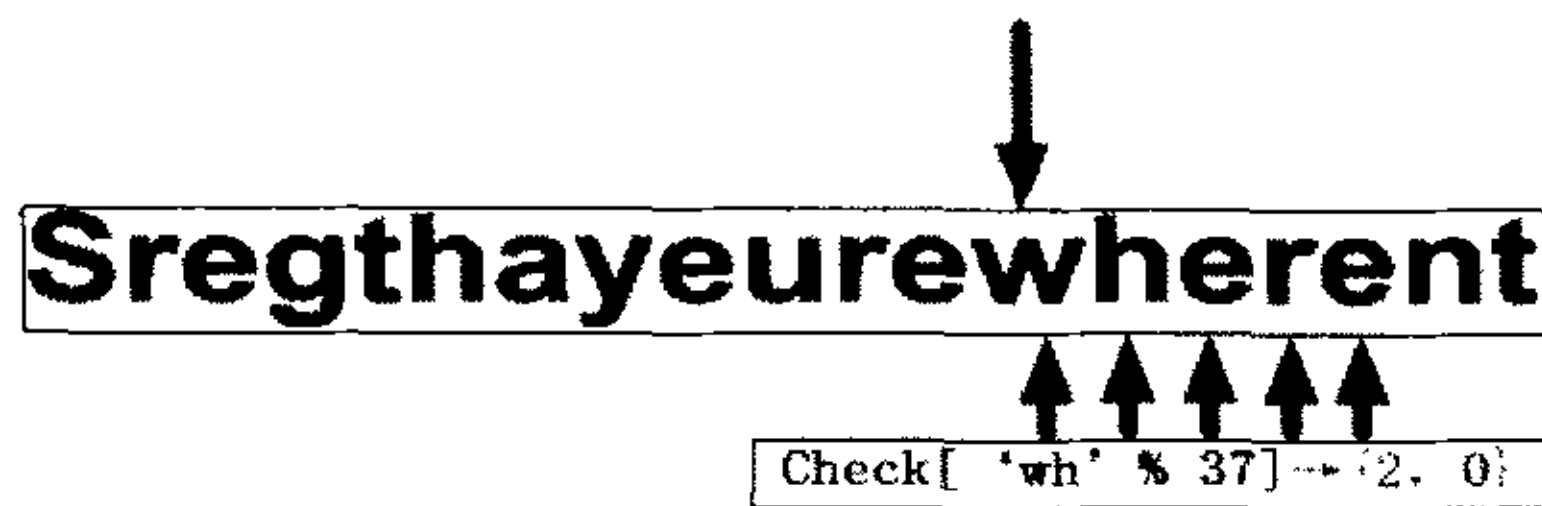


图 4.15 Long-Karp-Rabin 匹配过程(e)

Fig. 4.15 Long-Karp-Rabin matching process (e)

### 4.3.2 Long-Karp-Rabin 多模式匹配算法分析

Long-Karp-Rabin 多模式匹配算法利用数值快速查找关键字位置的优点，并实现了跳跃前进，所以查找更加快速。依照算法思想及参数设置，在 VC6.0 环境中本人用 c 语言实现了 Long-Karp-Rabin 多模式匹配算法，算法的流程图如下：

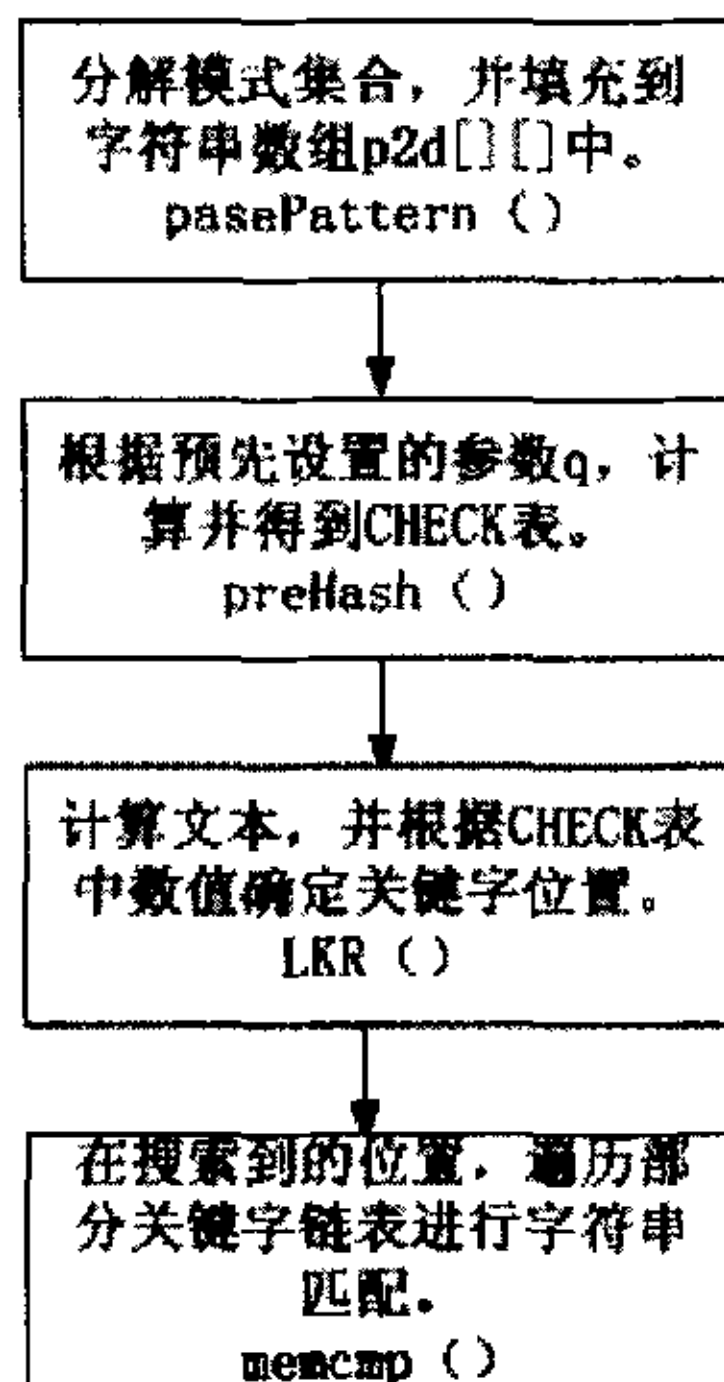


图 4.16 Long-Karp-Rabin 工作流程

Fig. 4.16 Long-Karp-Rabin working flow

从图 4.16 可以看出第一步分解模式集合后，得到多维 p2d 字符串数组：

p2d[1][0]="he";p2d[1][1]="er";p2d[1][2]="ro";p2d[1][3]="null";……

```
p2d[2][0]="wh";p2d[2][1]="he";p2d[2][2]="er";p2d[2][3]="re"; .....
p2d[3][0]="re";p2d[3][1]="ed";p2d[3][2]="do";p2d[3][3]="null" .....
|
```

其中数组的行号代表模式集合中的第几个模式，列号代表分割的模式关键字的部分号：比如  $p2d[1][0]$ ="he"，指的是模式集合中的第一个关键字"hero"，存储的是该关键字的第 0 部分"he"；同理  $p2d[1][1]$ ="er"表示模式集合第一个关键字的第 1 部分……。CHECK 哈希表中存储的是模式链表的首指针，链表的节点为数据结构 `_LKR_pattern`，定义如下：

```
typedef struct _LKR_pattern {
    struct _LKR_pattern *next;
    int    flag;
    int    iRow;
    int    iCol;
} LKR_pattern ;
```

在进行搜索时，程序根据分割的关键字部分长度  $minLen$ ，每次从文本 `text` 截取  $minLen$  大小的字符串，根据公式  $hash(w[0 .. m-1])=(w[0]*2^{m-1}+w[1]*2^{m-2}+\dots+w[m-1]*2^0) \bmod q$  计算该字符串的数值  $n$ ，再计算  $CHECK[i=n\%q]$ ，如果  $CHECK[i]$  为空，则证明此字符串没有匹配的关键字；当不为空时，根据  $CHECK[i]$  中存储的链表指针进行的串比较。

Long-Karp-Rabin 算法在进行字符串匹配之前，需要把分解的模式集合构建成单链表的形式，所以当规则集合较为庞大时，Long-Karp-Rabin 算法的预处理过程比较花费时间，这也是 Long-Karp-Rabin 等多模式匹配算法进行预处理的必要操作。在不考虑此预处理时间消耗的情况下，本人对上述三个算法进行如下实验。

在 VC6.0 环境中，测试 BM 算法、AC 算法和 Long-Karp-Rabin 算法的匹配效率，设置如下：

实验环境：单 CPU，PIV 1.8G，内存 256M

模式集合为 3 个关键字 { "hero", "where", "redo" }，文本 `text[]={ "sregthayeurewherent" }`， $q=37$ ，通过对 `text` 文本的比较次数来考查它们处理所花费的时间（单位 s），结果如表 4.3 所示：

表 4.3 匹配花费的时间比较

Table 4.3 The comparison of time spending

匹配次数	BM算法	AC算法	Long-Karp-Rabin算法
100	0.000000 secs	0.000000 secs	0.000000 secs
10000	0.062000 secs	0.016000 secs	0.000000 secs
1000000	5.906000 secs	0.860000 secs	0.140000 secs

从表 4.3 可以明显看出,在搜索时间上 Long-Karp-Rabin 算法明显要比前两个 Snort 入侵检测系统中采用的算法所花费的时间少。

表 4.4 是三个算法在 1000000 次比较时, Windows 系统参数发生的变化:

表 4.4 系统资源消耗比较

Table 4.4 The comparison of system resources spending

	系统原始值	BM算法	AC算法	Long-Karp-Rabin算法
Cpu实用率	35%-41%	100%	69%左右	57%左右
内存使用率	35%	大约35%	大于35%	大于35%
系统缓存 (k)	122376	122156	122000左右	122100左右

通过比较以上数据,可以确定 Long-Karp-Rabin 算法除了查找快速的优点外,消耗占用的系统资源比 AC 多模式算法要少。

比较而言,采用的 Long-Karp-Rabin 多关键字匹配算法较 Snort 系统的 BM、AC 算法高效,所以可以应用到 Snort 入侵检测引擎中作为系统的多模式匹配算法。但是 Long-Karp-Rabin 多模式匹配算法在处理关键字集合比较大的情况时,其部分关键字链表过长,这是该算法的主要不足,针对此不足,本人对该算法做了进一步改进。

### 4.3.3 Long-Karp-Rabin 多模式匹配算法的改进

从 Long-Karp-Rabin 算法流程可以看出,当查找到“部分关键字”的位置时,由于哈希表的不均匀分布,节点会发生大量碰撞问题。比如,在上节的例子中, $305\%37=9$ ,  $379\%37=9$  ……., Long-Karp-Rabin 算法处理碰撞的方法是采用链表的形式,把碰撞的部分关键字链接成单链表的形式,如图 4.17 所示:



表 4.3 匹配花费的时间比较

Table 4.3 The comparison of time spending

匹配次数	BM算法	AC算法	Long-Karp-Rabin算法
100	0.000000 secs	0.000000 secs	0.000000 secs
10000	0.062000 secs	0.016000 secs	0.000000 secs
1000000	5.906000 secs	0.860000 secs	0.140000 secs

从表 4.3 可以明显看出,在搜索时间上 Long-Karp-Rabin 算法明显要比前两个 Snort 入侵检测系统中采用的算法所花费的时间少。

表 4.4 是三个算法在 1000000 次比较时, Windows 系统参数发生的变化:

表 4.4 系统资源消耗比较

Table 4.4 The comparison of system resources spending

	系统原始值	BM算法	AC算法	Long-Karp-Rabin算法
Cpu实用率	35%-41%	100%	69%左右	57%左右
内存使用率	35%	大约35%	大于35%	大于35%
系统缓存 (k)	122376	122156	122000左右	122100左右

通过比较以上数据,可以确定 Long-Karp-Rabin 算法除了查找快速的优点外,消耗占用的系统资源比 AC 多模式算法要少。

比较而言,采用的 Long-Karp-Rabin 多关键字匹配算法较 Snort 系统的 BM、AC 算法高效,所以可以应用到 Snort 入侵检测引擎中作为系统的多模式匹配算法。但是 Long-Karp-Rabin 多模式匹配算法在处理关键字集合比较大的情况时,其部分关键字链表过长,这是该算法的主要不足,针对此不足,本人对该算法做了进一步改进。

### 4.3.3 Long-Karp-Rabin 多模式匹配算法的改进

从 Long-Karp-Rabin 算法流程可以看出,当查找到“部分关键字”的位置时,由于哈希表的不均匀分布,节点会发生大量碰撞问题。比如,在上节的例子中, $305\%37=9$ ,  $379\%37=9$  ……., Long-Karp-Rabin 算法处理碰撞的方法是采用链表的形式,把碰撞的部分关键字链接成单链表的形式,如图 4.17 所示:

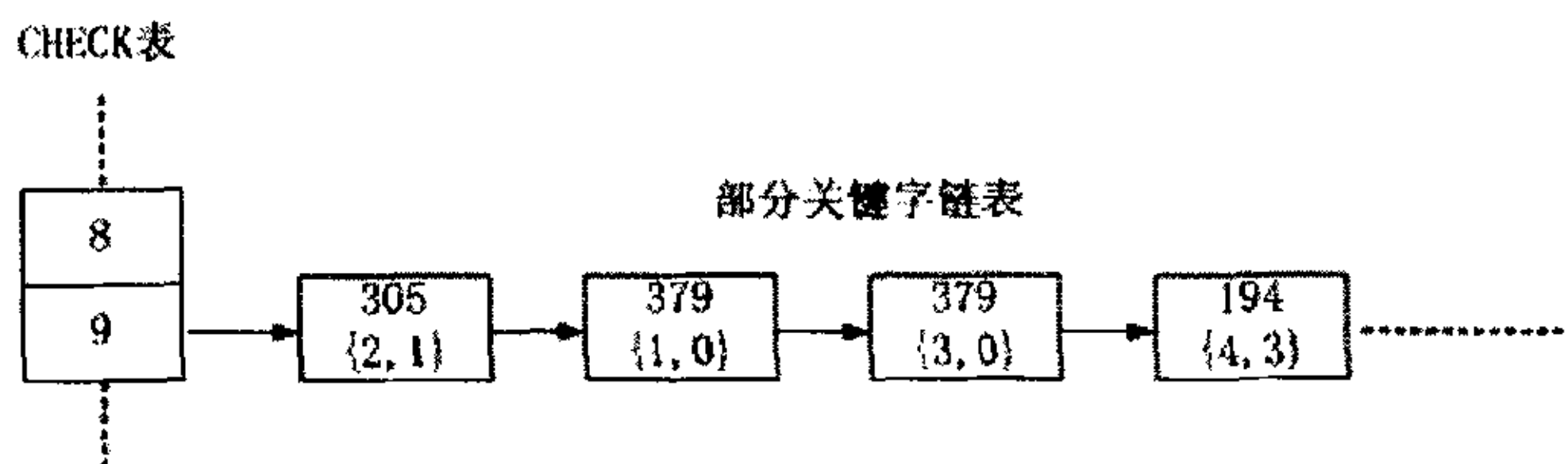


图 4.17 模式链表结构

Fig. 4.17 The structure of the patterns linked list

可以看出，哈希值相同的节点是无序的连接在一起的，所以，对搜索文本 text 经过哈希计算后，如果对应的 CHECK 哈希表中的值不为空，Long-Karp-Rabin 算法只有通过遍历整个单链表，对逐个节点进行查找才能确定字符串位置。

在入侵检测系统中，规则集合少则上百条，多则上千、上万条，无论采用什么样的哈希散列方式，都会产生哈希碰撞。所以，预处理构建 CHECK 哈希表时，有的节点发生碰撞的几率较高，该节点的部分关键字链表将会很长，那么在实时匹配进行到此位置时，将不能进行快速搜索。遍历单链表的平均时间复杂度为  $O(n/2)$ 。

从以上分析可以知道，Long-Karp-Rabin 算法的主要不足就是简单运用单链表的方式存储产生的节点，本人对此进行如下改进：选择一个不同于  $q$  的模数  $q_2$ ，采用二次散列的方式处理一次散列产生碰撞的节点，然后再采用单链表的方式对二次产生的碰撞节点进行存储。例如，如图 4.18 所示，当  $q=37$  时，第一次散列后哈希表 CHECK 的节点 9 处产生碰撞，然后取一个不同于  $q$  的质数  $q_2$ ，对节点 9 处产生碰撞的节点再进行模  $q_2$  的运算，将运算后的节点存储到另一个哈希表 CHECK2 中，构造后的结构如图 4.18 所示：

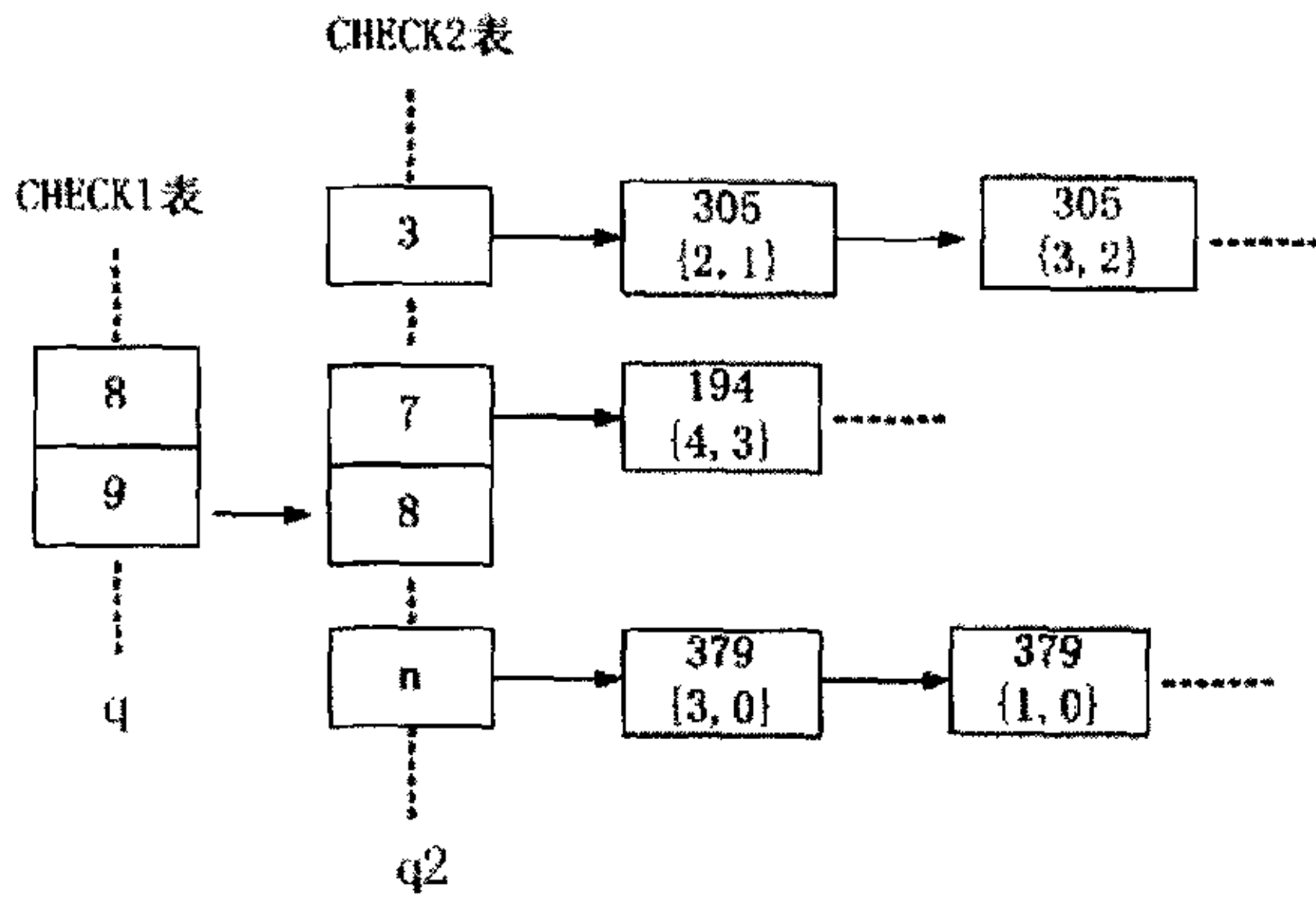


图 4.18 改进的存储结构

Fig. 4.18 The improved structure of the storage

可以看出，采用二次散列的方式存储发生碰撞的关键字片段，按计算后的关键字片段搭建哈希表，数值相同的节点按单链表的方式进行处理。当进行关键字匹配时，根据哈希表的特点，通过比较数值大小，可以快速定位到数值相同的关键字节点，再通过遍历单链表进行具体的字符串匹配。例如，如图 4.18，当要查找数值为“194”的节点时，需要 4 步才能找到该节点，过程如图 4.19 所示：

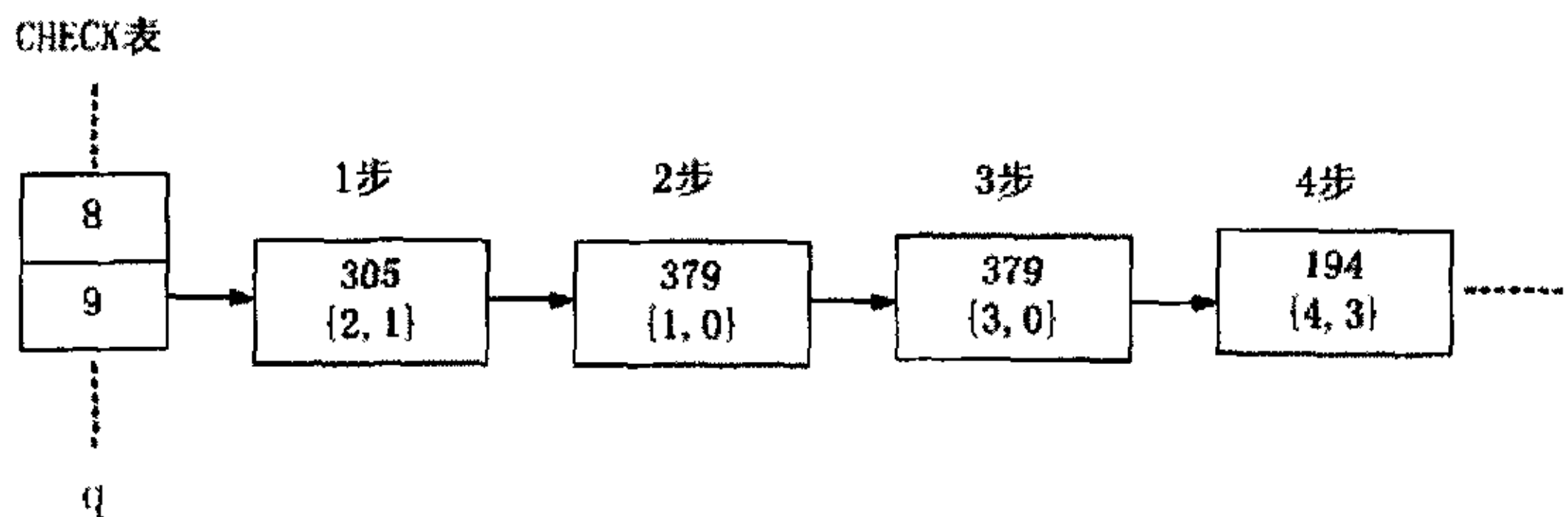


图 4.19 改进前的搜索步数

Fig. 4.19 The search steps before improved

当应用了新的结构后，再查找该节点时，步数缩减到 2 步就可以快速定位到该节点了，大大提高了搜索速度。查找过程如图 4.20 所示：

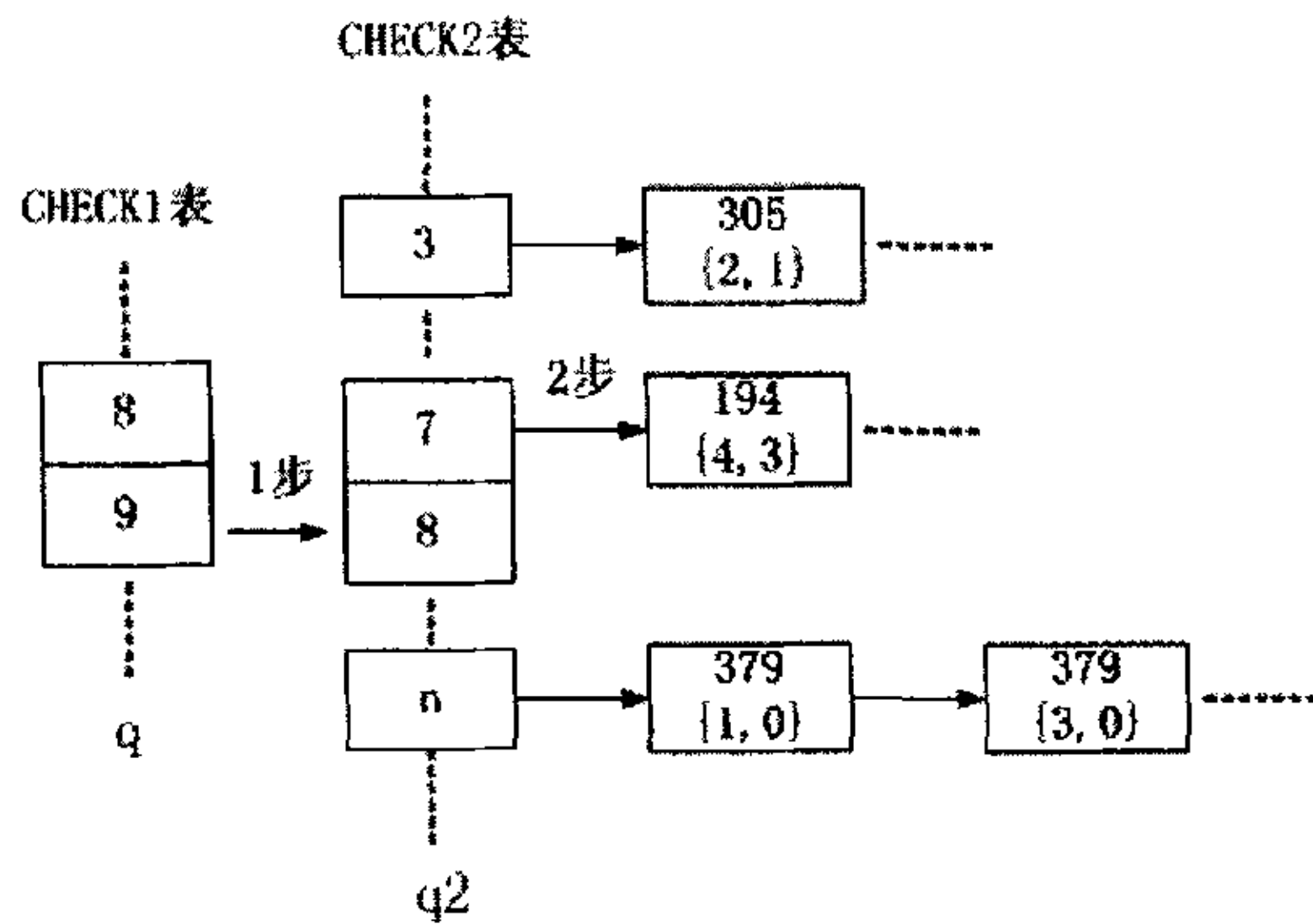


图 4.20 改进后的搜索步数

Fig. 4.20 The search steps after improved

理论上，线性探测再散列的哈希表查找成功时的平均查找长度为：

$$S_{nl} \approx \frac{1}{2} \left( 1 + \frac{1}{1 - \theta} \right) [31],$$

采用二次散列的方式明显减少了搜索关键字位置的时间。

#### 4.3.4 实验分析

根据上述改进的 Long-Karp-Rabin2 算法思想，在 VC6.0 环境中，测试 Long-Karp-Rabin 算法和 Long-Karp-Rabin2 算法的匹配效率，设置如下：

实验环境：单 CPU，PIV 1.8G，内存 256M

为达到测试目的，为达到增加 CHECK 哈希表中节点碰撞的几率，选择如下模式集合及文本，

文本：text[]={"sregthefaayeurelxxont"};

模式集合：pattern[]={"ceaka","lxxmxo","mvbid"};

预先设置的 p2d 数组初始化为：

p2d[1][0]="ce";p2d[1][1]="ea";p2d[1][2]="ak";p2d[1][3]="ka";

p2d[2][0]="lx";p2d[2][1]="xm";p2d[2][2]="mx";p2d[2][3]="xo";

p2d[3][0]="mv";p2d[3][1]="vb";p2d[3][2]="bi";p2d[3][3]="id";

下表是实验输出的数据：

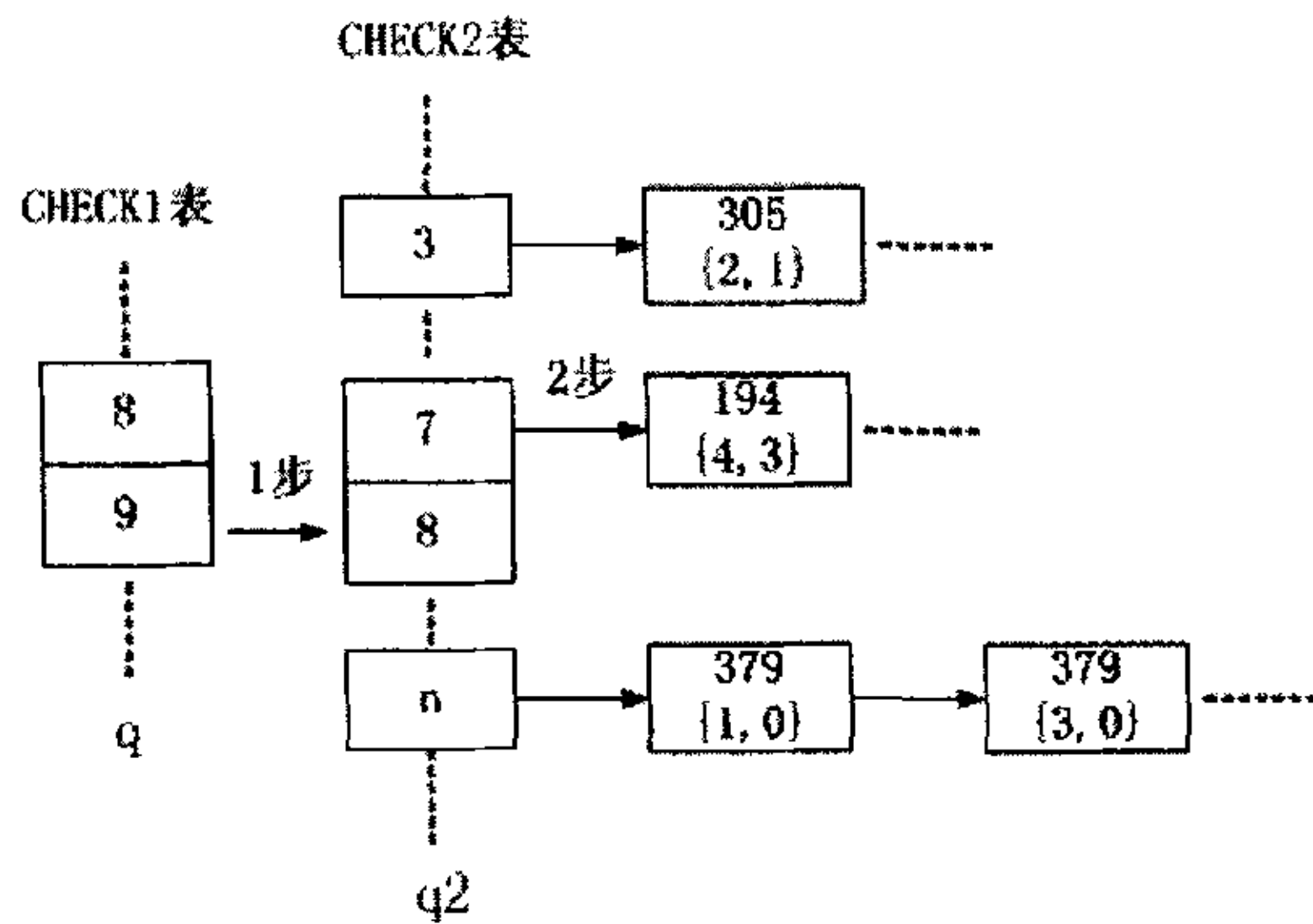


图 4.20 改进后的搜索步数

Fig. 4.20 The search steps after improved

理论上，线性探测再散列的哈希表查找成功时的平均查找长度为：

$$S_{nl} \approx \frac{1}{2} \left( 1 + \frac{1}{1 - \theta} \right) [31]$$
，采用二次散列的方式明显减少了搜索关键字位置的时间。

#### 4.3.4 实验分析

根据上述改进的 Long-Karp-Rabin2 算法思想，在 VC6.0 环境中，测试 Long-Karp-Rabin 算法和 Long-Karp-Rabin2 算法的匹配效率，设置如下：

实验环境：单 CPU，PIV 1.8G，内存 256M

为达到测试目的，为达到增加 CHECK 哈希表中节点碰撞的几率，选择如下模式集合及文本，

文本：text[]={"sregthefaayeurelxxont"};

模式集合：pattern[]={"ceaka","lxmxo","mvbid"};

预先设置的 p2d 数组初始化为：

p2d[1][0]="ce";p2d[1][1]="ea";p2d[1][2]="ak";p2d[1][3]="ka";

p2d[2][0]="lx";p2d[2][1]="xm";p2d[2][2]="mx";p2d[2][3]="xo";

p2d[3][0]="mv";p2d[3][1]="vb";p2d[3][2]="bi";p2d[3][3]="id";

下表是实验输出的数据：

表 4.5 改进后匹配花费时间比较

Table 4.5 The comparison of time spending after improved

匹配次数	Long-Karp-Rabin算法	Long-Karp-Rabin2算法
10000	0.015000 secs	0.016000 secs
1000000	1.187000 secs	1.156000 secs
10000000	114.546000 secs	114.313000 secs

从表 4.5 可以看出, 在匹配的次数小于 10000 次时, Long-Karp-Rabin2 算法搜索的速度基本上和 Long-Karp-Rabin 算法相当, 此时并没有体现出应用二次散列的优点。但是当比较次数逐渐增大时, 应用二次散列的优点则体现出来, 此时 Long-Karp-Rabin2 算法的速度快于 Long-Karp-Rabin 算法。

表 4.6 是两个算法在 1000000 次比较时, Windows 2003 系统参数发生的变化:

表 4.6 改进后系统资源消耗比较

Table 4.6 The comparison of system resources spending after improved

	系统原始值	Long-Karp-Rabin算法	Long-Karp-Rabin2算法
Cpu利用率	35%-41%	100%左右	100%左右
内存使用率	40%	大于40%	大于40%
系统缓存 (k)	91000左右	90880左右	90760左右

通过比较以上数据, 可以确定 Long-Karp-Rabin2 算法除了查找快速的优点外, 消耗占用的系统资源并不比 Long-Karp-Rabin 算法多。综合以上数据, 可以认为本人对 Long-Karp-Rabin 算法的进一步改进是可行的, 相比较而言 Long-Karp-Rabin2 算法更优。

## 4.4 小结

本章分析了 Snort 检测引擎的模式匹配算法, 指出了 BM 和 AC 模式匹配算法的不足, 同时提出多模式匹配算法—Long-Karp-Rabin 算法, 并且将该算法进一步改进为 Long-Karp-Rabin2 算法。通过实验, 证明 Long-Karp-Rabin2 算法进行字符串匹配的速度是高效的, 完全可以应用到入侵检测引擎中。下章将讨论 Snort 检测引擎应用 Long-Karp-Rabin2 算法的实现方案。



表 4.5 改进后匹配花费时间比较

Table 4.5 The comparison of time spending after improved

匹配次数	Long-Karp-Rabin算法	Long-Karp-Rabin2算法
10000	0.015000 secs	0.016000 secs
1000000	1.187000 secs	1.156000 secs
10000000	114.546000 secs	114.313000 secs

从表 4.5 可以看出，在匹配的次数小于 10000 次时，Long-Karp-Rabin2 算法搜索的速度基本上和 Long-Karp-Rabin 算法相当，此时并没有体现出应用二次散列的优点。但是当比较次数逐渐增大时，应用二次散列的优点则体现出来，此时 Long-Karp-Rabin2 算法的速度快于 Long-Karp-Rabin 算法。

表 4.6 是两个算法在 1000000 次比较时，Windows 2003 系统参数发生的变化：

表 4.6 改进后系统资源消耗比较

Table 4.6 The comparison of system resources spending after improved

	系统原始值	Long-Karp-Rabin算法	Long-Karp-Rabin2算法
Cpu利用率	35%-41%	100%左右	100%左右
内存使用率	40%	大于40%	大于40%
系统缓存 (k)	91000左右	90880左右	90760左右

通过比较以上数据，可以确定 Long-Karp-Rabin2 算法除了查找快速的优点外，消耗占用的系统资源并不比 Long-Karp-Rabin 算法多。综合以上数据，可以认为本人对 Long-Karp-Rabin 算法的进一步改进是可行的，相比较而言 Long-Karp-Rabin2 算法更优。

## 4.4 小结

本章分析了 Snort 检测引擎的模式匹配算法，指出了 BM 和 AC 模式匹配算法的不足，同时提出多模式匹配算法—Long-Karp-Rabin 算法，并且将该算法进一步改进为 Long-Karp-Rabin2 算法。通过实验，证明 Long-Karp-Rabin2 算法进行字符串匹配的速度是高效的，完全可以应用到入侵检测引擎中。下章将讨论 Snort 检测引擎应用 Long-Karp-Rabin2 算法的实现方案。

## 第五章 Snort 检测引擎改进方案

应用上章提出的 Long-Karp-Rabin2 多模式匹配算法，本章对 Snort 系统检测引擎提出改进方案，并通过实验，部分验证了算法的有效性。

### 5.1 Snort 主函数流程

Snort 主工作流程函数为 SnortMain，它是程序的入口，其内部机制如图 5.1 所示：

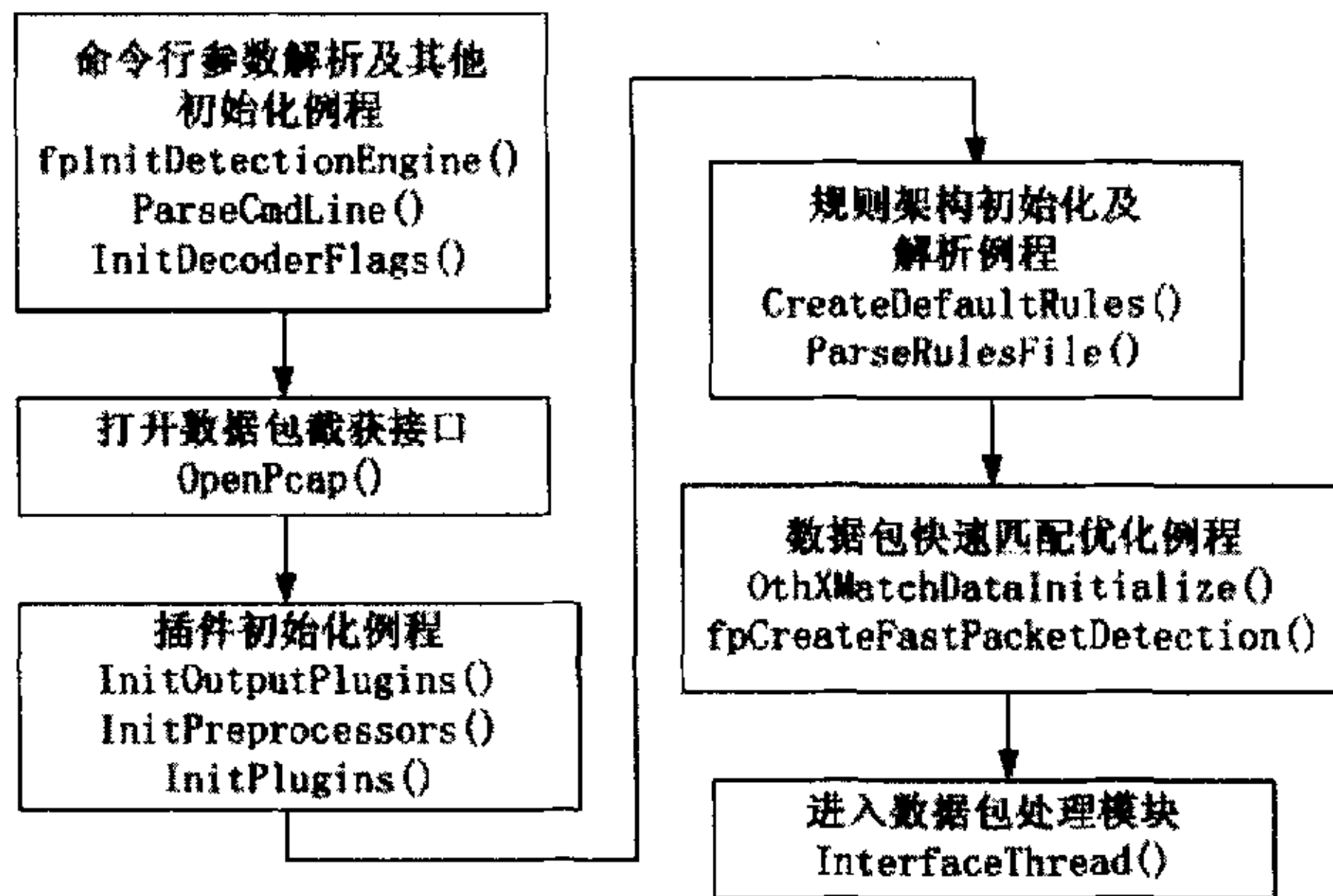


图 5.1 SnortMain 函数工作流程

Fig. 5.1 SnortMain function's working flow

下面对图中工作流程进行解释

- 命令行参数解析函数 ParseCmdLine() 在 Snort 系统各版本中都扮演着一个重要的角色，其任务就是解析指定的各种命令行参数，并将解析结果存放在全局 PV 类型的变量 pv 中。数据类型 PV 中包含各种标识字段，用来指示各种参数设置情况，包括例如规则文件、系统运行模式、显示模式、插件激活模式等；检测引擎初始化例程 fpInitDetectionEngin()，其实质是在 2.0 版本中引入的模块，用于指定快速规则匹配模块的缺省配置参数，包括缺省的搜索算法等；协议解析器初始化 InitDecoderFlags()，负责指定在协议解析过程中产生警报信息的错误类型。

## 第五章 Snort 检测引擎改进方案

应用上章提出的 Long-Karp-Rabin2 多模式匹配算法，本章对 Snort 系统检测引擎提出改进方案，并通过实验，部分验证了算法的有效性。

### 5.1 Snort 主函数流程

Snort 主工作流程函数为 SnortMain，它是程序的入口，其内部机制如图 5.1 所示：

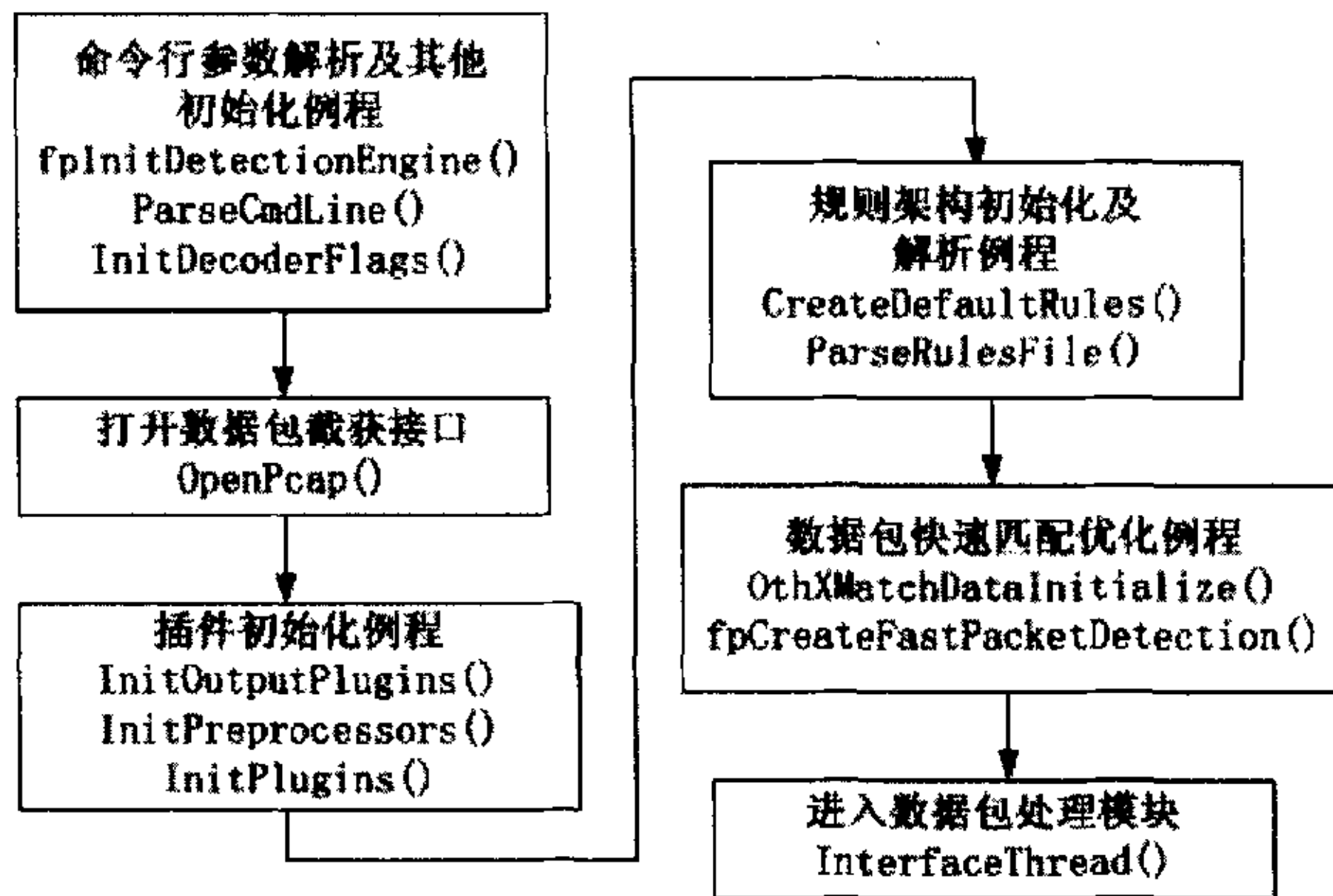


图 5.1 SnortMain 函数工作流程

Fig. 5.1 SnortMain function's working flow

下面对图中工作流程进行解释

- 命令行参数解析函数 ParseCmdLine() 在 Snort 系统各版本中都扮演着一个重要的角色，其任务就是解析指定的各种命令行参数，并将解析结果存放在全局 PV 类型的变量 pv 中。数据类型 PV 中包含各种标识字段，用来指示各种参数设置情况，包括例如规则文件、系统运行模式、显示模式、插件激活模式等；检测引擎初始化例程 fpInitDetectionEngin()，其实质是在 2.0 版本中引入的模块，用于指定快速规则匹配模块的缺省配置参数，包括缺省的搜索算法等；协议解析器初始化 InitDecoderFlags()，负责指定在协议解析过程中产生警报信息的错误类型。

- **OpenPcap()**是所有 Snort 版本中的一个关键例程，其主要作用是根据命令行参数解析结果，分别调用 Libpcap 库函数 `pcap_open_live()` 和 `pcap_open_offline()`，并获得对应的数据包截获接口数据结构。
- 插件初始化例程主要包括输出插件初始化例程、检测插件初始化例程和预处理器插件初始化例程等。各种插件的初始化过程都是类似的，主要任务就是将各种插件的关键字名称与对应的初始化处理函数对应起来，并注册到对应的关键字链表结构中，以便以后的规则解析例程使用。
- 规则架构初始化和解析例程。**CreateDefaultRules()**负责进行初始化的规则架构建设工作。Snort 较早版本的检测规则链表架构是个分散的二维链表结构。从 1.8 版本开始，引入了新的三维链表数据结构，将所有分散的规则链表结构都统一链接起来。2.0 版本也继承了这一架构的形式。关键的检测规则解析任务由函数 `ParseRulesFile()`负责完成，实质上该函数不仅是解析检测规则集合，而且对于所有的系统配置规则都进行解析，包括预处理器、输出插件、配置命令等。因此，`ParseRulesFile()`是 Snort 系统中的一个核心关键模块。
- 规则优化及快速匹配模块是 Snort2.0 版本中引入的最重要的设计特色。在 `SnortMain()`函数中，主要涉及调用了构造和初始化用于规则优化和快速匹配数据结构例程，包含 `OtnXMatchInfoInitialize()` 和 `fpCreateFastPacketDetection()`。`OtnXMatchInfoInitialize()`主要是针对 `OTNX_MATCH_DATA` 数据结构类型的全局变量 `omd`，为其申请指定的内存块空间。`OTNX_MATCH_DATA` 数据结构中包含了快速规则匹配时所用到的若干信息，特别包含了一个 `MATCH_INFO` 类型的链表指针字段。`MATCH_INFO` 数据结构中存放了在规则匹配中满足匹配条件所产生的所有相关信息。而 `fpCreateFastPacketDetection()`是构造规则快速匹配引擎的主要接口函数，它的基本原理是读入由规则解析模块构建的规则链表中的所有规则链表头（`RuleTreeNode` 类型）和规则选项节点（`OptTreeNode` 类型），然后在其基础上构建新一层的数据结构，以便用于快速规则匹配之用。
- 数据包处理模块 `InterfaceThread()`。该函数的命名其实是沿袭 1.8 版本后的习惯，在 2.0 版本中似乎并没有加入多线程的能力。`InterfaceThread()`的功能非常简单，主要是调用 Libpcap 库函数 `pcap_loop()`。

Snort 系统检测模块的主要接口函数为 `ProcessPacket()`和 `Preprocess()`。前者作为 libpcap 库函数 `pcap_loop()`的接口回调函数，在 `pcap_loop` 函数内部调用以处理

截获的数据包。后者在 ProcessPacket()中调用，具体执行入侵检测任务。

- 接口函数 ProcessPacket()主要功能包括：
- 调用数据包协议解码模块，并将解码结果存储在特定的 Packet 类型数据结构中。Packet 是 Snort 系统中一个非常重要的全局数据结构类型。
- 根据 Snort 的不同运行模式，调用不同的处理模块。
- 对入侵检测的运行模式，调用 Preprocess()模块。

## 5.2 改进的检测引擎设计

从第二章可以知道，Snort 把规则集合组织成一个二维的链表结构，这个链表结构包含：规则树节点(Rule Tree Nodes, RTN)和选项树节点(Option Tree Nodes, OTN)。规则树节点中包含规则的通用属性，例如：源 IP 地址、源端口号、目的 IP 地址、目的端口号、协议类型(TCP、ICMP、UDP)等。选项树节点中包含一些可被添加到每条规则中的各种各样的信息，例如：TCP 标志、ICMP 代码、类型、包负载的大小、影响效率的主要瓶颈、要查找的内容等。规则集合的二维链表结构如图 3.10 所示。

如果需要对包的内容进行检查，则将 OTN 节点中“content”选项的内容与数据包的内容进行精确的模式匹配，选项树（OTN）中的节点组织形式如图 5.2 所示：

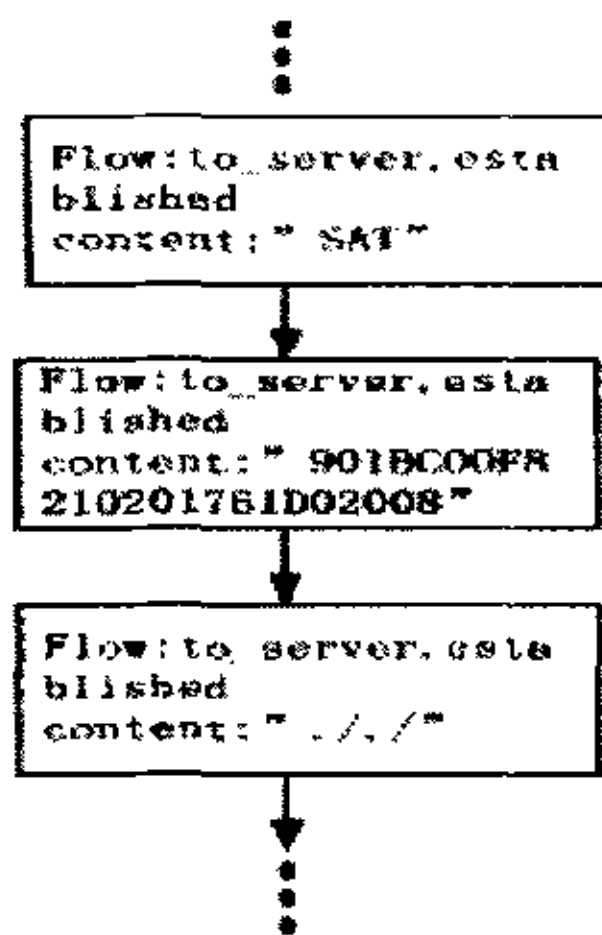


图 5.2 OTN 节点的链表结构

Fig. 5.2 The linked list structure of the OTN node

图 5.2 中节点的“content”字段的内容就是 BM 算法或 AC 算法进行匹配时所用到的模式串。在预处理阶段系统把 OTN 节点中的规则集合按所选用的算法，



截获的数据包。后者在 ProcessPacket()中调用，具体执行入侵检测任务。

- 接口函数 ProcessPacket()主要功能包括：
- 调用数据包协议解码模块，并将解码结果存储在特定的 Packet 类型数据结构中。Packet 是 Snort 系统中一个非常重要的全局数据结构类型。
- 根据 Snort 的不同运行模式，调用不同的处理模块。
- 对入侵检测的运行模式，调用 Preprocess()模块。

## 5.2 改进的检测引擎设计

从第二章可以知道，Snort 把规则集合组织成一个二维的链表结构，这个链表结构包含：规则树节点(Rule Tree Nodes, RTN)和选项树节点(Option Tree Nodes, OTN)。规则树节点中包含规则的通用属性，例如：源 IP 地址、源端口号、目的 IP 地址、目的端口号、协议类型(TCP、ICMP、UDP)等。选项树节点中包含一些可被添加到每条规则中的各种各样的信息，例如：TCP 标志、ICMP 代码、类型、包负载的大小、影响效率的主要瓶颈、要查找的内容等。规则集合的二维链表结构如图 3.10 所示。

如果需要对包的内容进行检查，则将 OTN 节点中“content”选项的内容与数据包的内容进行精确的模式匹配，选项树（OTN）中的节点组织形式如图 5.2 所示：

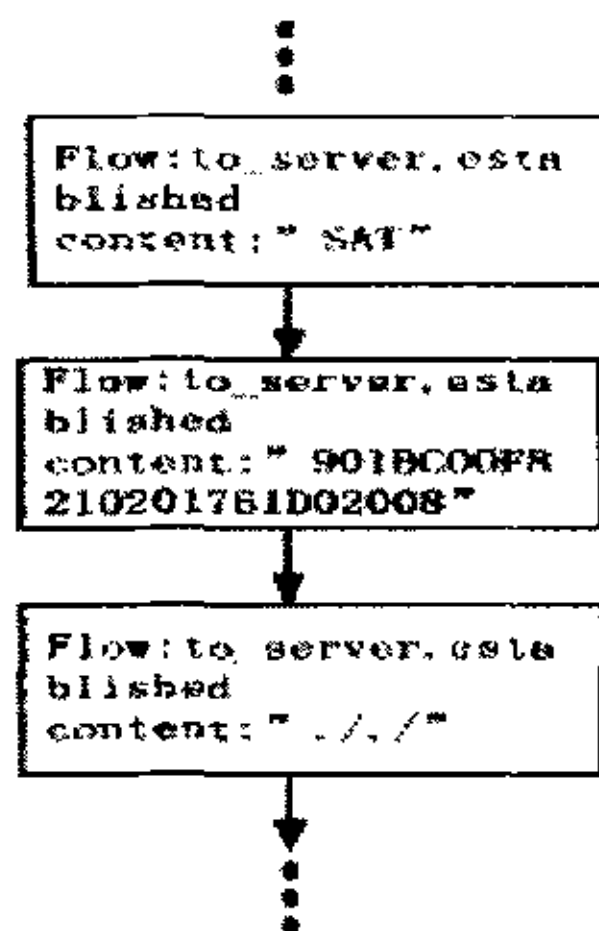


图 5.2 OTN 节点的链表结构

Fig. 5.2 The linked list structure of the OTN node

图 5.2 中节点的“content”字段的内容就是 BM 算法或 AC 算法进行匹配时所用到的模式串。在预处理阶段系统把 OTN 节点中的规则集合按所选用的算法，



组织成适当的形式，然后对捕获的数据包执行当前规则子集合所包含所有模式的匹配工作。

为了应用 Long-Karp-Rabin2 算法，根据 Long-Karp-Rabin2 算法的特点——基于部分关键字的哈希数值比较，所以在预处理阶段把 4.1.2 节划分后的模式子集合作进一步处理：在构造 Snort 规则树 OTN 节点的同时，建立与划分后的规则子集合相对应的 Pattern 字符串数组，用于存放该规则子集合内的“content”内容，然后再根据 Pattern 字符串数组创建 CHECK 哈希表，处理过程如图 5.3 所示：

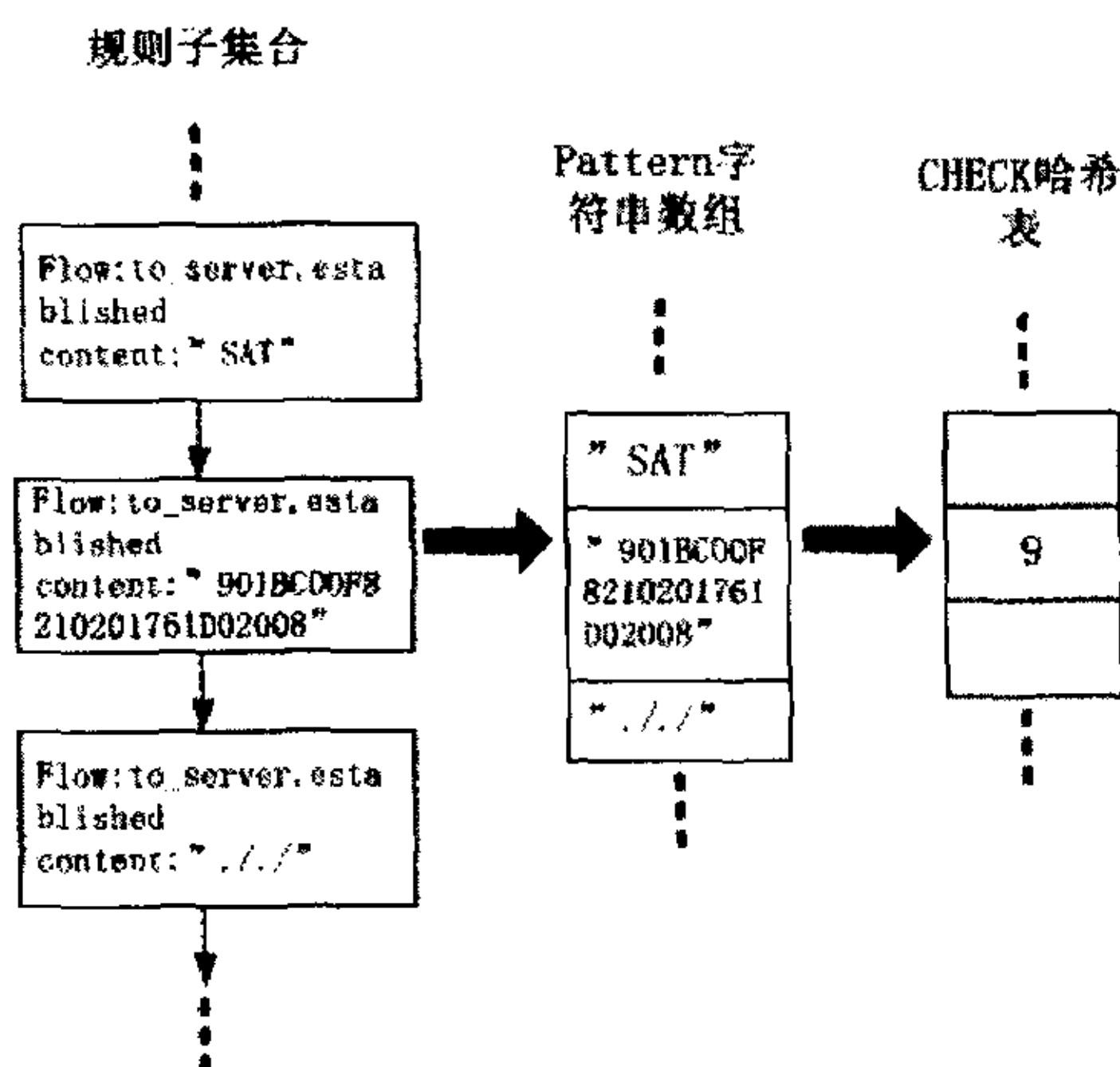


图 5.3 CHECK 哈希表构建过程

Fig. 5.3 The establishment process of the CHECK hash table

当对数据包内容进行检查时，将数据包内容作为搜索文本 text，根据 4.3 节所述，运用 Long-Karp-Rabin2 算法进行匹配，改进的 Snort 系统工作流程如图 5.4 所示：

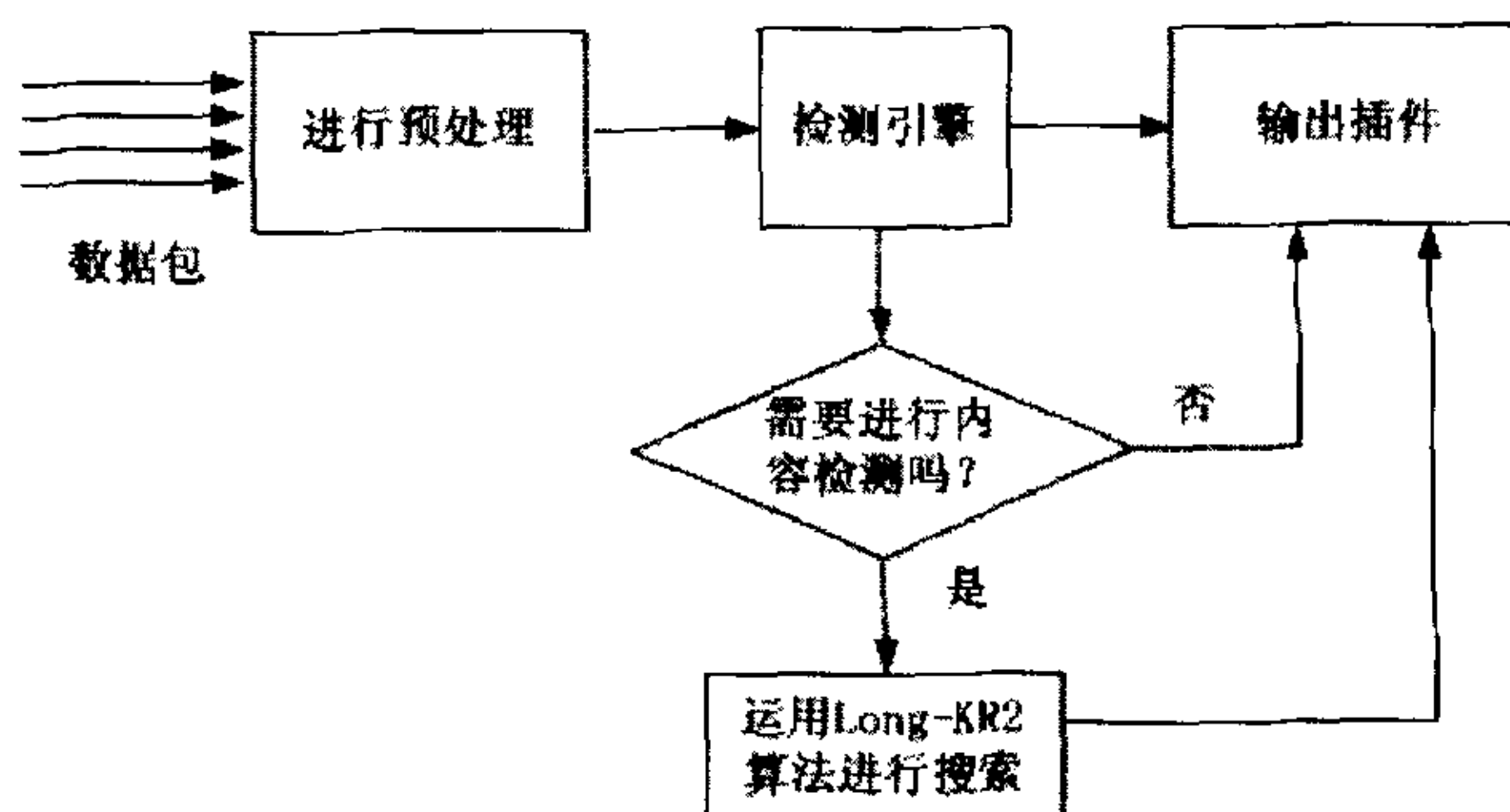


图 5.4 改进后的检测引擎工作流程

Fig. 5.4 The working flow of the Detection Engine after improved

数据包到达检测引擎后，Snort 将通过规则头以下面的顺序进行导航：Acitivation, Dynamic, Alert, Pass 和 log，在每一个规则头，检查 RTN 和 OTN，然后根据数据包的协议（TCP、UDP、ICMP 等）获取当前适用的规则子集合，

从 4.1 节可以知道，Snort 快速规则检测接口函数 `fpEvalPacket()` 根据数据包的高层协议类型，分别调用对应的规则快速匹配函数 `fpEvalHeaderTcp`、`fpEvalHeaderUdp`、`fpEvalHeaderIcmp` 等，然后快速匹配函数调用对应的 `prmFindRuleGroupTcp`、`prmFindRuleGroupIcmp`、`prmFindRuleGroupIcmp` 等函数，获取当前适用的规则子集合。根据返回值，调用快速规则匹配例程 `fpEvalHeaderSw()`。

函数 `fpEvalHeaderSw()` 对数据包是否需要内容进行匹配分别进行了处理，当需要进行内容匹配时调用函数 `mpseSearch`，该函数代码主要部分为：

```

|
switch( p->method )
{
    case MPSE_AC:
        return acsmSearch( (ACSM_STRUCT*) p->obj, T, n, action, data );
        break;
    |

```

从以上代码可以看出，Snort 检测引擎提供了扩展接口，为此把 Long-Karp-Rabin2 搜索模块加入到此函数中。

至此，对 Snort2.0 系统检测引擎的改进工作已经初步完成，下面将通过实验来测试新算法的有效性。

### 5.3 实验结果与分析

仅仅分析每个算法的时间复杂度并不能完全比较出算法的实际效率，因为理论假设和实际情况的差异以及算法本身实现带来的问题都有可能使得实际效果出现偏差。根据影响算法效率的因素和体现算法效率的指标，即模式集尺寸、最小关键字长度、预处理时间、内存占用四个方面进行实际使用效率对比测试。

测试系统环境：

平台：兼容机，单 Intel CPU 1.8GHz，内存 256MB

操作系统：Windows 2003

#### ● 查找速度与模式集尺寸的关系

模式集中的关键字数目是影响匹配效率的重要指标，有限自动机 AC 算法的理论时间复杂度与关键字数目无关，在许多跳跃类匹配算法中，当关键字数增大时，可能需要大量的字符比较操作，引起效率下降。本实验目的是观察各个算法的查找速度，以及算法受模式集尺寸和最小关键字长度影响情况。

设置如下：文本 text 长度 100；模式集合 pattern 模式模式集尺寸达到 1000 关键字以上时；匹配次数：10000 次。

下表为得到的实验数据：

表 5.1 模式集尺寸与查找效率比较

模式集合长度	AC算法	Long-Karp-Rabin2算法
1080	0.031000 secs	0.012000 secs
1200	0.047000 secs	0.015000 secs

在随模式集尺寸增大查找效率下降的幅度上，AC 算法和 Long-Karp-Rabin2 算法有 50%的波动下降。AC 算法和 Long-Karp-Rabin2 算法受模式尺寸影响最小，但和模式集尺寸仍有一定关系。Long-Karp-Rabin2 算法比 AC 算法效率大致有 250%—300%的提高。

#### ● 查找效率与模式集中最小关键字长度关系

在跳跃类匹配算法中，最大跳跃距离由模式集中关键字最小长度决定，所以模式集中最小关键字长度是影响算法匹配效率的重要因素。本实验目的是观察算法随模式集中的最小关键字长度增大时的查找效率变化情况。

至此，对 Snort2.0 系统检测引擎的改进工作已经初步完成，下面将通过实验来测试新算法的有效性。

### 5.3 实验结果与分析

仅仅分析每个算法的时间复杂度并不能完全比较出算法的实际效率，因为理论假设和实际情况的差异以及算法本身实现带来的问题都有可能使得实际效果出现偏差。根据影响算法效率的因素和体现算法效率的指标，即模式集尺寸、最小关键字长度、预处理时间、内存占用四个方面进行实际使用效率对比测试。

测试系统环境：

平台：兼容机，单 Intel CPU 1.8GHz，内存 256MB

操作系统：Windows 2003

#### ● 查找速度与模式集尺寸的关系

模式集中的关键字数目是影响匹配效率的重要指标，有限自动机 AC 算法的理论时间复杂度与关键字数目无关，在许多跳跃类匹配算法中，当关键字数增大时，可能需要大量的字符比较操作，引起效率下降。本实验目的是观察各个算法的查找速度，以及算法受模式集尺寸和最小关键字长度影响情况。

设置如下：文本 text 长度 100；模式集合 pattern 模式模式集尺寸达到 1000 关键字以上时；匹配次数：10000 次。

下表为得到的实验数据：

表 5.1 模式集尺寸与查找效率比较

Table 5.1 The comparison of patterns size and searching efficiency		
模式集合长度	AC算法	Long-Karp-Rabin2算法
1080	0.031000 secs	0.012000 secs
1200	0.047000 secs	0.015000 secs

在随模式集尺寸增大查找效率下降的幅度上，AC 算法和 Long-Karp-Rabin2 算法有 50%的波动下降。AC 算法和 Long-Karp-Rabin2 算法受模式尺寸影响最小，但和模式集尺寸仍有一定关系。Long-Karp-Rabin2 算法比 AC 算法效率大致有 250%—300%的提高。

#### ● 查找效率与模式集中最小关键字长度关系

在跳跃类匹配算法中，最大跳跃距离由模式集中关键字最小长度决定，所以模式集中最小关键字长度是影响算法匹配效率的重要因素。本实验目的是观察算法随模式集中的最小关键字长度增大时的查找效率变化情况。

最小关键字长度分别为 4, 10 的 7 个模式集, 对这 7 个模式集, 依次记录文本的搜索时间, 设置如下: 文本 text 长度 100; 匹配次数: 100000 次。

表 5.2 关键字长度与查找效率比较

Table 5.2 The comparison of pattern length and searching efficiency

最小关键字长度	AC算法	Long-Karp-Rabin2算法
4	0.328000 secs	0.130000 secs
10	0.359000 secs	0.298000 secs

AC 算法查找速度与关键字的最小长度无关, 可以看到, Long-Karp-Rabin2 算法受关键字长度影响最明显

#### ● 预处理时间对比

前述两个实验, 观察了各种算法的查找搜索时间情况, 本实验在于研究各种算法在不同模式集上的预处理花费时间

表 5.3 预处理时间对比

Table 5.3 The comparison of preprocessing spending

规则集合大小	AC算法	Long-Karp-Rabin2算法
1080条	0.078000 secs	0.000150 secs
3000条	0.109000 secs	0.000240 secs

在模式集尺寸大于 1000 时, AC 算法的处理时间较多, Long-Karp-Rabin2 算法则有明显的优势。随着模式集规模的增大, Long-Karp-Rabin2 算法的增幅小。

#### ● 算法内存占用情况

前述的三个实验集中研究了各种算法的时间效率, 在本次实验中, 将考虑空间效率, 即各个算法运行中的内存消耗情况 (Mbyte)。

表 5.4 内存消耗比较

Table 5.4 The comparison of memory spending

规则集合大小	AC算法	Long-Karp-Rabin2算法
1218条	10左右	0.7左右
2018条	23左右	0.9左右

在多模式匹配算法中, 随着关键字规模的增长, 其内存占用的空间越来越大。这是匹配算法在大规模关键字上进行应用相对较为关键的一个方面。因为在机器硬件相对固定的情况下, 如果随着关键字规模的增大, 内存占用出现膨胀, 那么对于不同的应用, 后继的处理则可能出现无内存资源而无法运行的情况。根据上表可以看出随着关键字总数的增长, Long-Karp-Rabin2 算法占用空间与增幅最小。



## 第六章 结束语

### 6.1 本文的主要工作

本文主要在分析开放源码 Snort 系统的基础上, 对其检测引擎采用的模式匹配算法进行了改进, 运用了基于数值查找的多模式匹配算法——Long-Karp-Rabin 算法, 并在此基础上, 对 Long-Karp-Rabin 算法的单链表线性结构进行改进, 采用了二次散列结合链表的方式存储节点, 理论上优化了算法, 使系统的检测效率得到进一步提高。最后通过实验证明经过改进后的 Long-Karp-Rabin2 多模式匹配算法是有效的。

### 6.2 进一步改进及展望

网络入侵检测已经成为当今的一个热点研究问题, 随着网络速度越来越快, 对网络入侵检测效率的要求也越来越高, 其归根到底就是模式匹配算法的效率问题。因此, 首先, 模式匹配算法对入侵检测系统的检测效率有很大影响, 所以多模式匹配算法的研究仍然是一个备受关注的课题, 另外入侵检测系统受模式串的影响很大, 所以如何选择合适有效的特征串, 也是提高算法查找速度需要考虑的问题。其次, 多模式匹配算法随着关键字总数增多时, 其内存占用空间会发生膨胀现象。由此, 可以研究相对较优的算法, 既可以最大限度的提高系统效率, 又尽量避免发生内存膨胀现象, 从而实现在大流量情况下的实时审计。最后, 可以对应用 Long-Karp-Rabin2 算法的检测引擎进一步改进, 如可以改为并行处理的方式, 因为 Long-Karp-Rabin2 算法基于哈希数值计算查找方式适合进行并行计算, 检测引擎的并行处理也是提高匹配效率的很重要的一个研究和应用方向。

总之, 多模式匹配算法可应用于多个方面, 不仅仅局限在网络安全应用中, 对网络流量分析、网络测量、模式压缩、数据挖掘、模式识别、手写识别、文件压缩和屏幕更新等方向的应用都可以起到很好的指导作用。



## 参考文献

1. Terry Escamilla. 入侵者检测[M]. 北京: 电子工业出版社, 1999
2. S M Bellovin. Security Problem in the TCP/IP Protocol Suite[J]. Computer Communication Review, 1989, 19(2): 32-48
3. 刘欣, 沈昌祥. 多级数据库安全模型[J], 信息安全, 2004, 10
4. J P Anderson. Computer Security Threat Monitoring and Surveillance[R]. Technical report, James P Anderson Co. Fort Washington, 1980, 4
5. Dorothy E Denning. An Intrusion-detection Model[J]. IEEE Transactions on Software Engineering, 1987, 13(2): 222-232
6. Teresa L, Jagannathan R, Lee Rosanna. IDES: The enhanced prototype, a real-time intrusion detection system[R]. SRI International, 1988, 8(12): 88
7. Crosbie M, Spafford G. Applying genetic programming to intrusion detection[R]. Purdue University: Department of Computer Sciences, Coast Laboratory, 1997
8. S. Staniford-Chen, B. Tung, D. Schnackenberg. The common intrusion detection framework (CIDF), The Information Survivability Workshop, Orlando, FL, 1998
9. Brian Caswell, Jay Beale, James C.Foster, Jeffrey Posluns. Snort2.0 Intrusion Detection[M], Syngress Publishing, Inc, 2003, 3-6
10. Cannady J. Artificial neural networks for misuse detection[R]. the 1998 National Information Systems Security Conference, Arlington,VA., October 5-8 1998
11. Debra Anderson, Thane Frivold, Alfonso Valdes. Next-generation Intrusion Detection Expert System (NIDES) A Summary[R], Computer Science Laboratory, 1995, SRI-CSL-95-07
12. Wenke Lee, Salvatore J Stolfo. Data Mining Approaches for Intrusion Detection[R], the 7th USENIX Security Symposium, 1998
13. P.A Porras. STAT—A state transition analysis tool for intrusion detection [D]. University of California SantaBarbara, 1992
14. Tim Bass. Intrusion Detection Systems and Multisensor Data Fusion [J]. Communication of the ACM, 2000, 43(4): 99-105
15. Brian Caswell, Jay Beale, James C.Foster, Jeffrey Posluns. Snort2.0 入侵检测 [M], 北京: 国防工业出版社, 2004, 27-31
16. 强大的轻量级网络入侵检测系统 Snort. <http://www.net110.net/aqjs/z02>
17. 求是科技, 谭思亮. 监听与隐藏—网络侦听揭密与数据保护技术[M], 北京,

- 人民邮电出版社, 2002, 131
18. Brian Caswell, Jay Beale, James C.Foster, Jeffrey Posluns. Snort2.0 Intrusion Detection[M], Syngress Publishing, Inc, 2003, 118-122
19. Marc Norton & Dan Roelker of Sourcefire. Snort 2.0 - Multi-Rule Inspection Engine. [http://www.sourcefire.com/technology/snort\\_engine.html](http://www.sourcefire.com/technology/snort_engine.html)
20. 唐正军等. 入侵检测技术导论[M], 北京: 机械工业出版社, 2004
21. Robert S.Boyer, J Strother Moore. A Fast String Searching Algorithm[J], Communications of the ACM, 1977, 20(10): 762-772
22. Cole,R. Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm[J]. SLAM Journal on Computing, 1994, 23(5): 1075-1091
23. Alfred V.Aho, Margaret J.Corasick. Efficient String Matching:An Aid to Bibliographic Search[J], Communications of the ACM, 1975, 18(6): 333-340
24. Alfred V.Aho, Jeffrey D.Ullman. Optimal Partial-Match Retrieval When Fields Are Independently Specified[J], TODS, 1979, 4(2): 168~179
25. 杨武, 方滨兴, 云晓春, 张宏莉. 入侵检测系统中高效模式匹配算法的研究[J], 计算机工程, 2004, 30(13): 92-94
26. 张冬艳, 殷丽华, 胡铭曾, 云晓春, 郑秀荣. 面向内容安全的多模精确匹配算法性能分析[J], 通讯学报, 2004, 25(7): 128-136
27. 中科院计算所软件研究室. <http://www.software.ict.ac.cn/seminar/lectures/20021121-ACBM%CB%E3%B7%A8.ppt>
28. Richard M.Karp, Michael O.Rabin. Efficient randomized pattern matching algorithms[J], IBM Research and Development, 1987, 31(2): 249-260
29. Christian Charras, Thierry Lecroq. Karp-Rabin algorithm.  
<http://www-igm.univ-mlv.fr/~lecroq/string/node5.html#SECTION0050>
30. Wikipedia. Rabin-Karp string search algorithm. [http://en.wikipedia.org/wiki/Rabin-Karp\\_string\\_search\\_algorithm](http://en.wikipedia.org/wiki/Rabin-Karp_string_search_algorithm)
31. Mark Allen Weiss. 数据结构与算法分析—C 语言描述[M], 北京: 机械工业出版社, 2004

## 致 谢

本课题的研究工作是在导师刘春雨教授的指导下完成的。导师渊博的专业知识，严谨的治学风格，求实的工作态度，给我留下了深刻的印象，也培养了我良好的学习和工作作风，在此向导师刘春雨教授致以衷心感谢。

在课题的研究和论文的撰写过程中，得到了东软信息技术学院研发中心老师的指导和帮助，在这就不一一列出了。他们对教学和科研工作的热情投入深深感染了我，在此，我向各位老师表示衷心的感谢。

同时，我还要感谢在研发中心一起实习的同学高志刚、高凌云、李兴隆、刘红旗、雷琼、花学周、王鹏，感谢他们在学习和生活上对我的关心和照顾，感谢他们给我带来的快乐。

我还要感谢我的父母，是他们多年来不遗余力地给我以鼓励和支持，不断地激励着我积极进取。

由于本人能力有限，论文中难免出现疏漏之处，敬请各位老师和同学批评指正，本人不胜感激。

参考文献(31条)

1. Terry Escamilla 入侵者检测 1999
2. S M Bellovin Security Problem in the TCP/IP Protocol Suite 1989(02)
3. 刘欣, 沈昌祥 多级数据库安全模型[期刊论文]-信息安全 2004(10)
4. J P Anderson Computer Security Threat Monitoring and Surveillance 1980
5. Dorothy E Denning An Intrusion-detection Model 1987(02)
6. Teresa L, Jagannathan R, Lee Rosanna IDES:The enhanced prototype,a real-time intrusion detection system 1988(12)
7. Crosbie M, Spafford G Applying genetic programming to intrusion detection 1997
8. S Staniford-Chen, B Tung, D Schnackenberg The common intrusion detection framework (CIDF) 1998
9. Brian Caswell, Jay Beale, James C Foster, Jeffrey Posluns Snort2.0 Intrusion Detection 2003
10. Cannady J Artificial neural networks for misuse detection 1998
11. Debra Anderson, Thane Frivold, Alfnsó Valdes Next-generation Intrusion Detection Expert System (NIDES) A Summary[Computer Science Laboratory, SRI-CSL-95-07] 1995
12. Wenke Lee, Salvatore J Stolfo Data Mining Approaches for Intrusion Detection 1998
13. P A Porras STAT-A state transition analysis tool for intrusion detection 1992
14. Tim Bass Intrusion Detection Systems and Multisensor Data Fusion 2000(04)
15. Brian Caswell, Jay Beale, James C Foster, Jeffrey Posluns Snort2.0 入侵检测 2004
16. 强大的轻量级网络入侵检测系统Snort
17. 求是科技, 谭思亮 监听与隐藏-网络侦听揭密与数据保护技术 2002
18. Brian Caswell, Jay Beale, James C Foster, Jeffrey Posluns Snort2.0 Intrusion Detection 2003
19. Marc Norton & Dan Roelker of Sourcefire Snort 2.0 - Multi-Rule Inspection Engine
20. 唐正军 入侵检测技术导论 2004
21. Robert S Boyer, J Strother Moore A Fast String Searching Algorithm 1977(10)
22. Cole R Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm 1994(05)
23. Alfred V Aho, Margaret J Corasick Efficient String Matching:An Aid to Bibliographic Search 1975(06)
24. Alfred V Aho, Jeffrey D Ullman Optimal Partial-Match Retrieval When Fields Are Independently Specified 1979(02)
25. 杨武, 方滨兴, 云晓春, 张宏莉 入侵检测系统中高效模式匹配算法的研究[期刊论文]-计算机工程 2004(13)
26. 张冬艳, 殷丽华, 胡铭曾, 云晓春, 郑秀荣 面向内容安全的多模精确匹配算法性能分析[期刊论文]-通信学报 2004(7)
27. 中国科学院计算所软件研究室 查看详情
28. Richard M Karp, Michael O Rabin Efficient randomized pattern matching algorithms 1987(02)
29. Christian Charras, Thierry Lecroq Karp-Rabin algorithm
30. Wikipedia Rabin-Karp string search algorithm
31. Mark Allen Weiss 数据结构与算法分析-C语言描述 2004

本文读者也读过(10条)

1. 戴宝鑫 一个分布式防火墙系统的设计与实现[学位论文]2006
2. 敖晓宏 一种UPS监控系统的Web管理技术的研究与实现[学位论文]2005
3. 陈海滨 基于Snort分布式入侵检测系统的研究[学位论文]2007
4. 鄂丰凯 自主式足球机器人控制系统研究[学位论文]2005
5. 庞博 基于代理结构的Web个性化推荐技术的研究与实现[学位论文]2005
6. 白宁 沈阳海关物流监控体系研究[学位论文]2008
7. 冯博 网络环境下的知识协同管理问题研究[学位论文]2006
8. 薛必春 多媒体网络教学系统的设计与开发[学位论文]2005
9. 凌晓明 基于Snort的分布式入侵检测系统[学位论文]2006
10. 贾永刚 基于SNORT的分布式入侵检测体系结构的研究[学位论文]2007

引证文献(2条)

1. 张宏宇, 蒋文保, 刘宝旭 基于邮政综合网的Snort规则库的优化设计[期刊论文]-信息安全与通信保密 2009(3)
2. 张伟 基于移动代理的分布式入侵检测系统的研究[学位论文]硕士 2006

引用本文格式：[胡德华](#) Snort检测引擎的改进与实现[学位论文]硕士 2005