

西安理工大学

硕士学位论文

Snort入侵检测系统的研究及其性能改进

姓名：谢少春

申请学位级别：硕士

专业：计算机软件与理论

指导教师：张亚玲

20080301

论文题目: Snort 入侵检测系统的研究及其性能改进¹

学科专业: 计算机软件与理论

研 究 生: 谢少春

签名: 谢少春

指导教师: 张亚玲 副教授

签名: 张亚玲

摘 要

计算机和网络技术的快速发展给人类生产和生活带来了革命性的变化,这也使得人类面临着网络安全这种新的威胁。传统的加密和防火墙技术已经不能完全满足信息安全的需求,入侵检测技术作为一种必要的安全手段,在网络安全领域发挥着其独到的作用。

Snort作为典型的轻量级网络入侵检测系统(NIDS),是一个免费的开源项目。对Snort设计原理和实现特点的研究,可以作为其他商用入侵检测系统的研发基石,有较强的学术意义和较高的商业价值。

本文以Snort系统为研究对象,通过剖析其源代码,系统地研究了Snort整体体系架构,以及详细分析了其中的多模式快速匹配模式集、快速检测引擎、字符匹配算法等,然后,本文围绕模式匹配部分展开对提高Snort性能的关键技术研究。

以Snort新特性的分析和现存多种规则匹配方法研究为基础,考虑到大量Snort规则在一定时间内只有一小部分规则是活跃的,提出了基于活跃规则集的Snort规则匹配方法,通过把每个端口下的规则分成活跃规则集与不活跃规则集,结合反馈规则匹配频度的思想,实时更新规则匹配顺序和控制活跃规则集大小,从而提高规则匹配速度。根据本文提出的改进方法,针对Snort 2.4进行了规则匹配算法的改进。经过采用来自林肯实验室的国际化入侵检测样本数据对改进系统进行对比测试分析,实验结果表明,改进后的算法规则匹配效率提高了6%~21%。最后,本文说明了Snort匹配性能改进的进一步工作,并对Snort和IDS技术的发展作了展望。


关键词: 基于网络的入侵检测系统(NIDS); 规则树; 活跃规则集; 规则匹配; 匹配频度

¹本研究得到陕西省教育厅 2006 年计划资助项目(项目编号:06JK231)的资助

Title: Research and Performance Improvement on Snort Intrusion Detection System

Major: Computer Software and Theory

Name: Shao-Chun Xie

Signature: 

Supervisor: Associate prof. Ya-Ling Zhang **Signature:** 

Abstract

With the rapid development in the technology of computer and the network technology, the security problem in network is increasingly outstanding. But the traditional Encryption and Firewall techniques can't fully meet the expectations. As one of new methods, the Intrusion Detection System (IDS) plays an important role in network security today. With its characteristic, IDS should contribute more to the information security.

Snort is a free, open-source, "lightweight" Network-based Intrusion Detection System (NIDS) that has many capabilities. By studying Snort's characteristics and implementation techniques, the people can learn about IDS's knowledge and Snort's advantage so that they can do something for IDS and Snort. Studying Snort is of academic significance and commercial value.

In this thesis, the author looks into the Snort's code, analyzes the global architecture, multi-rule inspection engine of Snort, the fast detection engine, and the algorithms of Snort string matching. In the following we can obtain the key technology of improving Snort performance by studying pattern matching.

After the analysis of new characteristics and several improvement methods of snort rule matching, and by considering of which only a small part of rules in massive snort rules are active in certain time period, a new snort rule matching method based on active rule sets is proposed in this paper. By dividing rule sets under the each port into the active rule sets and the inactive rule sets, combining with feedback of the rule matching frequency, rule matching order is renewed in real time and the size of active rule sets are limited, so the rule matching speed can be improved. According the method proposed, we have improved the rule matching algorithm for snort 2.4, and test the improved system using International standardized intrusion detection sample data from MIT Lincon Lab. The contrast result of the test show efficiency of the new system is enhanced for 6%~21%. At the end, this article has pointed out the further work of Snort match performance improvement, and explained the prospect of Snort and IDS technology.

Key words: NIDS; active rule sets; rule tree; rule matching; matching frequency

独创性声明

秉承祖国优良道德传统和学校的严谨学风郑重申明：本人所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的成果。尽我所知，除特别加以标注和致谢的地方外，论文中不包含其他人的研究成果。与我一同工作的同志对本文所研究的工作和成果的任何贡献均已在论文中作了明确的说明并已致谢。

本论文及其相关资料若有不实之处，由本人承担一切相关责任

论文作者签名：谢少春 08年3月31日

学位论文使用授权声明

本人谢少春在导师的指导下创作完成毕业论文。本人已通过论文的答辩，并已经在西安理工大学申请博士 / 硕士学位。本人作为学位论文著作权拥有者，同意授权西安理工大学拥有学位论文的部分使用权，即：1) 已获学位的研究生按学校规定提交印刷版和电子版学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，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；2) 为教学和科研目的，学校可以将公开的学位论文或解密后的学位论文作为资料在图书馆、资料室等场所或在校园网上供校内师生阅读、浏览。

本人学位论文全部或部分内容的公布（包括刊登）授权西安理工大学研究生部办理。

（保密的学位论文在解密后，适用本授权说明）

论文作者签名：谢少春 导师签名：张亚平 08年3月31日

1 前言

1.1 研究背景

计算机和互联网技术正在飞速地发展,网络应用日益普及且更加复杂,网络安全问题也成为互联网和网络应用发展中面临的重要问题。中国国家计算机网络应急技术处理协调中心(CNCERT/CC)公布的“2004年网络安全工作报告”^[1]显示,CNCERT/CC在2004年共收到报告的网络安全事件64000多件,同2003年相比,网络安全事件报告数量大大增加。

从调查报告公布的数据和状况可以看出,各种网络安全漏洞大量存在和不断地被发现,网络攻击行为日趋复杂,各种方法相互融合,网络安全问题变得错综复杂,使网络安全防御更加困难。然而由于现代计算机和网络自身设计的一些问题,当前还不具有一种能保证计算机和网络绝对安全的技术^[2]。

为了尽量保障计算机和网络系统特别是关键部门的信息安全,市场上涌现出了各种安全技术和产品,包括防火墙、安全路由器、身份认证系统、VPN设备等,这些技术和产品对系统有一定的保护作用,但都属于静态安全技术范畴,不能主动跟踪入侵者^[3],也不能积极有效地防止来自网络内部的非法行为。为了确保网络的安全,网络系统中拥有的网络安全性分析系统应该能对系统进行漏洞扫描,同时还要能对网络安全进行实时监控、攻击与反攻击。入侵检测系统(Intrusion Detection System IDS)应用而生。

入侵检测系统是一种通过收集和分析计算机系统或网络中关键点的信息,以检查计算机或网络中是否存在违反安全策略的行为和被攻击的迹象,并对此做出反应,从而保护网络和主机安全的系统。入侵检测系统能识别外部对计算机或者网络资源的恶意企图和行为,以及内部合法用户超越使用权限的非法行为^[4]。

近年来,IDS在理论研究、实际应用上都有很大发展,如各种检测模型、检测方法的提出,信息安全公司提供的各种IDS产品。

Snort是一个用C语言编写的开放源代码、免费的轻量级入侵检测工具,它在共享的网络上捕获网络传输数据包,分析捕获到的数据包,匹配入侵行为的特征或者从网络活动的角度检测异常行为,完成对入侵的预警或记录。根据GPL(GNU Public License)协议,只要遵循GPL,任何组织和个人都可以自由使用或者修改Snort。本文采用Snort2.4版本,作为论文的分析、实验的基础。

Snort是免费开源的,相对于其他IDS昂贵的费用,这是最大的优势。众多爱好者对Snort的支持,方便的插件机制可以保证Snort及时地对新入侵和病毒做出迅速地反应。Snort设计结构简洁,具有一般入侵检测系统的特征,开发者通过Snort能比较容易地掌握IDS技术,以便对更专业化的IDS做出贡献,在Snort中实现、通过检验的方法也可以移植到其他的IDS中。

IDS发展迅速，同时也面临着一些问题，如检测速度问题、误报、漏报、系统自身安全问题等。Snort从1998年开始作为一种免费软件发布以来，根据网络安全的需要，及时地融入了各种先进检测技术，如IP分片重组，多模式匹配算法，加强了Snort的检测功能。然而Snort仍面临着IDS普遍的问题，检测速度、误报、漏报，也是Snort迫切要求解决的问题。

基于以上原因，对Snort系统的研究存在着较强的学术意义和较高的商业价值。

检测引擎是IDS的关键部分，在Snort中，检测引擎采用字符匹配的方法来判断入侵，检测引擎的速度决定Snort的整体性能，Snort社区一直在探索着高性能有效的匹配方法，也提出了一些解决方法如快速匹配引擎。本文将首先介绍Snort最新的研究进展，研究和分析影响Snort检测效率的关键因素，在此基础上提出和实现了基于活跃规则集的Snort高效规则匹配方法，并进行了性能对比测试。

1.2 国内外研究现状及进展

入侵检测系统根据检测引擎采用检测技术的不同，可以将入侵检测系统分为异常入侵检测系统和误用入侵检测系统。虽然异常检测技术能够发现一些未知的攻击类型，但是其误报率较高的问题还没得到很好解决，所以目前实际应用的入侵检测系统的检测引擎都是基于误用检测技术的。一般来讲，常用的误用检测技术有：模式匹配技术、专家系统和状态迁移技术等。其中模式匹配技术由于其有分析速度快，准确率高等优点，在实际中最为常用。在实际研究和应用中，人们也开发出很多的基于规则的误用入侵检测系统。著名的开源入侵检测系统 Snort^[5]，Bro^[6]，Firestorm^[7]和 Prelude IDS^[8]等都采用了模式匹配技术。其中 Snort 是目前最著名、最活跃的开放源码网络入侵检测系统项目。Snort 定位于轻量级的入侵检测系统，已经实现了网络探测器和许多第三方的管理及日志分析工具，是目前世界上使用最广泛的开源入侵检测系统之一。Snort 可以完成实时流量分析和记录网络 IP 数据包的能力，能够进行协议分析、内容查找/匹配，能够检测到缓冲区溢出，端口扫描、CGI (common gateway interface) 攻击、SMB (server message block) 探测、操作系统指纹探测企图等。Snort 可以运行于 Linux/Unix 系列，Windows 等操作系统，具有良好的跨平台性，并提供丰富的报警机制。Snort 遵循 GPL (general public license)，所以任何企业、个人、组织都可以免费使用它作为自己的网络入侵检测系统。Snort 具有简洁明了的规则描述设计及已经成一定规模的攻击检测规则集，它的规则集已经被很多其他开放源码 IDS 项目所兼容。历经数年的发展，Snort 已经发展到了 2.x 版本。Snort 基本架构在 Snort 1.6 版本时确立，2.0 版本开始采用了新型的入侵检测引擎，功能更加完善和强大。Snort 已经成为学习和研究入侵检测系统的经典实例。

目前，许多商业入侵检测系统都是在 Snort 入侵检测系统的基础上发展来的，有些甚至是 Snort 的翻版。国内也开发了不少商用入侵检测系统，例如启明星辰公司的“天阗入

入侵检测系统”，中科网威的“天眼入侵检测系统”，绿盟科技的“冰之眼”入侵检测系统，联想也开发了“联想网御入侵检测系统”。虽然目前市面上的商用入侵检测系统种类繁多，但是分析其本质，这些入侵检测系统都借鉴了 Snort 的设计思想。Snort 的检测原理并不复杂，采用的是简单的模式匹配策略。准确性和快速性是衡量其性能的重要指标。前者主要取决于对入侵行为特征码的提炼的精确性和规则撰写的简洁实用性，由于网络入侵检测系统自身角色的被动性——只能被动的检测流经本网络的数据，而不能主动发送数据包去探测，所以只有将入侵行为的特征码归结为协议的不同字段的特征值，通过检测该特征值来决定入侵行为是否发生。后者主要取决于引擎的组织结构和采用的模式匹配算法。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。但随着网络带宽不断提高，以及网络攻击种类的急剧增加，致使 Snort 的检测任务越来越重。当 Snort 检测引擎过载时，就有可能漏掉一些造成严重后果的网络攻击行为。因此，模式匹配算法已经成为 Snort 系统瓶颈所在，所以设计高效的模式匹配算法对于提高 Snort 性能有很大意义。模式匹配算法，按照每次匹配模式的个数，可以分为单模式匹配算法和多模式匹配算法。在单模式匹配算法方面，目前广泛使用的主要有 KMP 算法^[9]，BM 算法^[10]和 KR 算法^[11]等。BM 算法是由 R.S.Booyer 和 J.S.Moore 两个人在 1977 年设计实现的一个字符串匹配算法，是目前实际应用中最为常用、效率较高的单模式匹配算法之一。它采用从后向前的比较方式，并利用当前尝试中的以匹配的信息和匹配失败的字符，查找预处理好的良好后缀跳转表（Good-Suffix Shift Table）和不良字符跳转表（Bad-Character Shift Table），依靠这些启发信息进行跳跃式的并行处理，忽略不必要的比较，提高了模式匹配的速度。实际上现有的单模算法都是在 BM 算法的基础上改进而来，或者受到了 BM 算法思想启发。在多模匹配算法方面，比较著名的算法有 AC 算法^[12]、Wu-Manber 算法^[13]，AC-BM 算法^[14]、Comments-Walter 算法^[15]，SBMH 算法^[16]和 E2XB 算法^[17]等。其中 AC 算法是多模匹配算法中的典型算法，是由 A.V.Aho 和 M.J.Corasick 联合提出的多模式匹配算法，最初用在贝尔实验室的图书检索系统里面，后来在其它领域得到了广泛应用。AC 算法在预处理阶段通过模式集合构建一个有限状态模式匹配自动机(Deterministic finite automaton, DFA)。其实质就是将多模式匹配问题转化成单模式匹配问题，然后将文本串作为自动机的参数输入进行匹配，对文本只需扫描一次。有限自动机由一系列的状态组成，每个状态可以用一个数字来表示。自动机处理文本的过程实质就是按序读入文本中的每个字符，并根据当前的状态进行跳转，假如跳转后的状态是终止状态，则输出匹配结果。自从 AC 算法采用有限状态自动机方法来解决多模匹配问题以来，AC 算法也成为其它多模式匹配算法的思想基础，比如 AC-BM 就采用了 AC 算法和 BM 算法结合思想。Comments-Walter 算法，类似于 AC-BM 算法，其思想是首先用模式集构建一棵搜索树，然后再采用类似 BM 算法的方法，在文本中用搜索树对模式进行跳跃搜索。Wu-Manber 算法继承了 BM 算法中不良字符跳转的思想，但与 BM 算法不同的是，该算法依据长度为 B（2 或 3）的字符块进行跳转。由于采用字符

块技术,降低了部分匹配的可能,增加了直接跳跃的机会,并且采用哈希技术和前缀表,进一步减少了匹配次数,提高了性能。国内对模式匹配算法的研究基本基于上述几种经典算法,做不同程度的改进工作。其中基于 BM 算法方面做的改进工作最多。除了这些改进算法,贺龙涛 2005 年在软件学报上提出一种全新的匹配算法 Linear DAWG Matching 算法(简称 LDM 算法)^[18]。该算法将正文分为 $[n/m]$ 个相互重叠、大小为 $2m-1$ 的扫描窗口。在每个扫描窗口内,算法批量地尝试 m 个可能位置,首先使用反向后缀自动机从窗口中间位置向前扫描模式前缀;若成功,则再使用正向有限状态自动机从中间位置向后扫描剩余的模式后缀。上海交通大学的王永成等人于 2002 年在上海交大学报上提出一种新的模式匹配算法^{[19][20]}。其思想是在有限自动机的多模式匹配算法(DFSA 算法)的基础上,结合 QS 算法的优点,提出一个快速的多模式字符串匹配算法,并基于算法的连续跳跃的思想,给出另一个更加有效的改进。在一般情况下,这两个算法不需要匹配目标文本串中的每个字符,并充分利用了匹配过程中本次匹配不成功的信息,跳过尽可能多的字符。在模式串较长和较短的情况下,算法都有很好的性能。电子科技大学的万国根、秦志光也在电子科技大学学报提出改进的 AC-BM 算法,将待匹配的字符串集合转换为一个类似于 Aho-Corasick 算法的有限状态自动机。匹配时,采取自后向前的方法,并借用 BM 算法的坏字符跳转和好前缀跳转技术。改进的 AC-BM 算法借助 BMH 算法思想,取消了原 AC-BM 算法的好前缀跳转,并对坏字符跳转部分的计算进行优化。新算法修改了 skip 的计算方法,不再保留每个节点的好前缀跳转参数及坏字符跳转参数,因此匹配只与当前匹配字符有关,而与当前节点无关,可以实现大小写正文的识别^[21]。在用先进模式匹配算法改进入侵检测引擎方面,北京大学文字信息处理技术国家重点实验室的张邈、徐辉等以 Snort 为蓝本,从规则库结构、串匹配算法及应用层协议分析等方面入手进行优化,设计并实现高效率的串匹配型入侵检测系统 SpeedIDS^[22];上海交通大学的陈一航、薛质基于 E2XB 的改进算法做检测引擎算法,并在 Snort 中实现^[23]。最后中科院高能物理研究所计算中心的李雪莹等基于几种常见的多模匹配模式算法改进了 Snort 检测引擎,并做了性能比较分析^[24]。

1.3 论文内容、方法和意义

基于网络的入侵检测技术在网络安全领域得到了广泛的应用。Snort作为一种免费开源的入侵检测工具,提供给了网络安全从业者一个研究平台。本论文以Snort为研究对象,来了解、研究一般的IDS。

本文的研究内容与成果包括以下几个方面:

- (1) 本文概述了入侵检测系统与Snort,总结了IDS及Snort面临的问题和发展趋势,指出了Snort在入侵检测系统研究中的学术意义和研究价值。
- (2) 分析了较新的Snort2.40版本的系统结构,重点剖析了Snort相对2.0以前版本的两个

重大改进部分：规则索引和多模式匹配，对Snort2.4从源代码的角度有了一个透彻的理解。

(3) 总结了改进Snort性能的一系列方法，然后围绕规则匹配部分，提出了基于活跃规则集的Snort规则匹配方法来改进Snort的性能。

(4) 根据本文提出的改进方法，我们针对Snort2.4进行了规则匹配算法改进，经过采用林肯实验室的样本数据对改进系统的测试，结果表明，改进后的算法规则匹配效率提高了6%~21%。

本论文的意义在于详细分析了Snort2.4的体系结构及Snort2.0后采用的全新的规则索引和多模式匹配引擎的实现，在windows下通过实验来测试Snort处理数据包的性能，并提出了基于活跃规则集来改进Snort的性能的方法，最后完成了Snort代码的修改。性能测试实验证明，本改进方法能显著提高Snort的规则匹配性能。

对作者来说，通过论文工作，深入了解了网络入侵检测系统的工作原理和具体实现。为作者以后从事计算机及网络安全方面的工作奠定了深厚的理论基础，积累了丰富的工作经验。

1.4 论文组织结构

论文全文的章节安排如下：

第一章介绍了入侵检测与网络安全的研究背景、课题意义及论文的组织结构。

第二章对入侵检测和 Snort 作了简要的概述，分析了 IDS 及 Snort 面临的问题和发展趋势。

第三章对著名的入侵检测系统 Snort 进行了详细分析，重点阐述了最新版本 Snort 的改进：包分类机制和快速匹配引擎。

第四章重点阐述了基于活跃规则集的 Snort 高效规则匹配方法。

第五章给出了基于活跃规则集的 Snort 高效规则匹配方法思想的实现方法，以及改进前后 Snort 的性能对比测试结果。

第六章对全文进行了总结，并对本课题相关的未来发展进行了展望。

2 入侵检测系统与 Snort

2.1 入侵检测系统

入侵检测(Intrusion Detection, ID), 顾名思义, 是对入侵行为的检测。它通过收集和分析计算机系统或网络中若干关键点的信息, 以检查计算机系统或网络中是否存在违反安全策略的行为和被攻击的迹象, 识别外部对计算机或者网络资源的恶意企图和行为, 以及内部合法用户超越使用权限的非法行为, 并对此做出反应, 从而保护网络和主机的安全^[25]。

入侵检测技术是一种主动保护自己免受攻击的网络安全技术。被认为是防火墙之后的第二道安全闸门, 它在不影响网络性能的情况下能对网络进行监测, 是一种积极主动的安全防护技术。

入侵检测系统(Intrusion Detection System IDS)是完成上面功能的独立系统。IDS提供了对内部攻击、外部攻击和误操作的检测, 在被保护系统的安全性受到侵害时发出报警并采取适当的行动来阻止入侵行为, 从而起到保护系统安全的作用^[26]。

2.2 IDS 的标准与结构

美国国防高级研究计划署(DARPA)工作组(IDWG)发起制订了一系列IDS和互联网工程任务组(IETF)的入侵检测建议草案, 从体系结构、API、通信机制、语言格式等方面规范IDS的标准。DARPA提出了公共入侵检测框架(CIDF)^[27], 如图2-1, CIDF主要包括四部分: IDS体系结构、通信机制、描述语言和应用编程接口API。

CIDF将一个入侵检测系统分为四个组件: 事件产生器(Event generators); 事件分析器(Event analyzers); 响应单元(Response units); 事件数据库(Event databases)。四个组件间或各个IDS间采用GIDO (General Intrusion Detection Object统一入侵检测对象)格式进行数据交换。

在CIDF模型中, 事件产生器、事件分析器和响应单元通常以应用程序的形式出现, 而事件数据库则采用文件数据流的形式。CIDF将IDS需要分析的数据统称为事件(Event)。

事件产生器, 从整个计算环境中获得事件, 转化成一定格式, 以向系统的其他部分提供此事件。

事件分析器, 分析得到的数据, 并产生分析结果。分析器是IDS的核心部分, 可以以不同的方法来分析数据, 以判断入侵。

响应单元, 则是对分析结果做出反应的功能单元, 它可以做出切断连接、改变文件属性等反应, 也可以只做简单的报警。

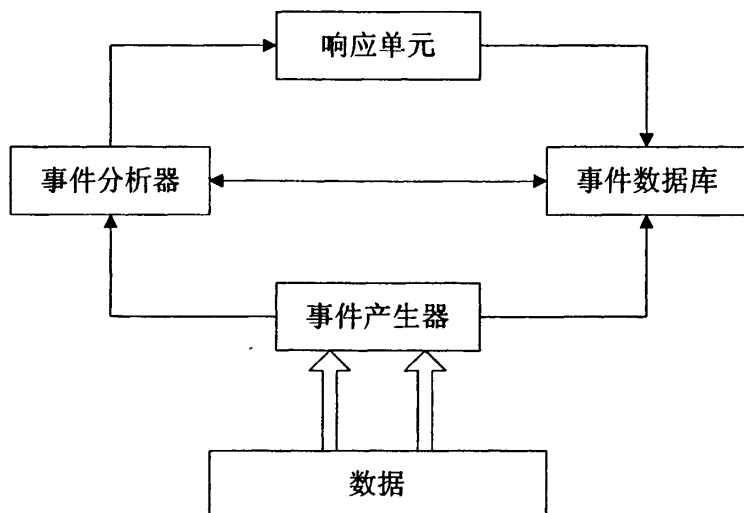


图2-1 CIDF基本模型

Fig.2-1 Fundamental model of CIDF

事件数据库，存放各种中间和最终数据的地方，可以是复杂的数据库，也可以是简单的文本文件。

IDS通信机制，为了保证各个组件之间安全、高效的通信能力，CIDF将通信机制构成一个三层模型：GIDO层、消息层、协议传输层。

IDS描述语言，CIDF使用一种被称为S表达式的入侵说明语言(CISL) 对事件分析结果、响应指示等过程进行表示说明，以达到IDS之间语法互操作。该语言类似于Lisp语言。

IDSAPI接口，负责GIDO的编码、解码和传递，使得程序员可以以一种很简单的方式构建和传递GIDO。

2.3 IDS 的发展与分类

1980年，Anderson J P 在报告“Computer Security Thread Monitoring and Surveillance”^[28]第一次详细阐述了入侵检测的概念。提出了利用审计跟踪数据监视入侵活动的思想。这份报告被公认为是有关入侵探测的最早论述。

1984年~1987年，Dorothy Denning与Peter Neumann联合开发了一个实时入侵检测系统IDES，并提出了一种通用的入侵检测模型^[29]。该模型根据主机系统审计记录数据，生成有关系统的若干轮廓，并监测轮廓的变化差异发现系统的入侵行为，如图2-2所示。

随着入侵行为的种类不断增多，涉及的范围不断扩大。入侵检测系统的不同功能组件之间、不同IDS之间共享这类攻击信息是十分重要的。为此，Chen等提出一种通用的入侵检测框架模型，简称CIDF。

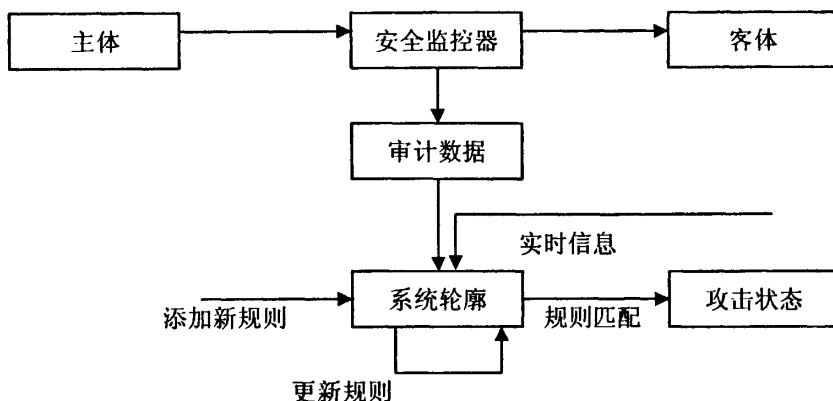


图 2-2 IDes 入侵检测模型

Fig.2-2 IDes Intrusion Detection Model

1988年的莫里斯蠕虫事件发生之后,美国军方、学术界和企业开始了对分布式入侵检测系统(DIDS)的研究,将基于主机和基于网络的检测方法集成到一起。它的检测模型采用了分层结构,包括数据、事件、主体、上下文、威胁、安全状态等6层。

直到20世纪90年代商业化的IDS才真正出现,如:国外ISS公司的RealSecure, Cisco公司的Secure IDS, IBM公司的IERS(Internet Emergency Response Service)等^[30]。

从20世纪90年代到现在,入侵检测系统的研发呈现出百家争鸣的繁荣局面,并在智能化和分布式两个方向取得了一些进展。

计算机和网络技术的继续发展,使得入侵检测技术仍然面临着巨大的挑战,IDS技术也必须通过和其他计算机技术的结合,探索新的方法,才能在网络安全方面发挥更大的功效。

IDS可以按照不同的方法来分类:

2.3.1 按检测方法分类

按照检测方法的不同,可以把IDS分为特征检测(Misuse Detection)的IDS和异常检测(Anomaly detection)的IDS^[31]。

特征检测又称为误用检测,是基于知识(或规则)的检测^[32]。它通过对已知的入侵方式特征做出确定性的描述,形成事件模式,当用户行为或系统状态同已知的事件模式相匹配时,则认为发生了入侵。通常采用专家系统法、模型推理法、状态转移法、模式匹配法、Petri网、信息反演法等方法。其优点是准确率较高。但是不能发现未知的入侵行为,比较难以检测到入侵变种。

异常检测又称为行为检测。首先它对系统对象的一些测量属性(比如访问次数、CPU使用频率、操作失败次数等)进行统计,找出一个阈值作为基准,如果用户的行为或资源使用的状况超出了这个基准值,则认为发生了入侵。通常采用基于概率统计的、基于

免疫学的、基于数据挖掘的、基于专家系统的、基于神经网络的方法等。这种方法的优点是通用性强，对具体系统的依赖性较少，并且能够发现未知的入侵行为。缺点是如何描述正常的行为模式，确定异常的基准值比较困难，导致误报率较高。

2.3.2 按检测所用数据来源分类

按检测所用数据来源通常分为基于主机的HIDS、基于网络的NIDS和分布式的DIDS三大类。

HIDS一般安装在重点保护的主机上，通过对系统日志、审计数据的分析和端口活动的监听发现入侵行为。其检测目标主要是主机系统和系统本地用户，通过监视操作系统和各种服务生成的日志文件，检测是否有入侵。NIDS 通常置于重要的网段内，监听网络数据包，发现攻击事件，一旦入侵特征匹配或者超过正常的阀门值就做出反应。

DIDS将信息采集、分析、响应等构件分布在网络的各个监控点上，他们之间通过通信传输构件联系，由管理中心提供统一管理。网络上的多个检测器共同收集信息，协同工作，并行的进行检测分析工作。

2.3.3 按实现方法分类

按实现方法可分为基于概率统计模型的检测、基于神经网络的检测、基于专家系统的检测、基于模型推理的检测和基于免疫的检测等技术。

2.4 Snort 系统概述

2.4.1 Snort 的功能

从检测模式而言，Snort属于网络入侵检测(IDS)的误用检测。它通过Libpcap库函数从网络中抓取数据包，对数据包进行解析，接着启动检测引擎，将解释好的数据包和规则模式集进行比较。如果匹配规则，则认为该入侵行为成立，使用规定的方式进行响应，然后结束一个数据包的处理过程，再抓取下一个数据包。如果没有匹配的规则，则是正常行为，直接返回，抓取下一个包进行处理。

Snort是基于规则检测的入侵检测工具，即针对每一种入侵行为，都提炼出它的特征值，并按照规范写成检测规则，形成一个规则数据库。利用此规则库和捕获的数据包进行比较，来判断是否为入侵。目前，Snort的检测规则库主要针对缓冲区溢出、端口扫描和CGI攻击等。最新数据表明Snort共有51类4300条检测规则。

Snort集成了多种告警机制来提供实时告警功能，包括：syslog、用户指定文件、

UNIXSocket、通过SMBClient使用WinPopup对Windows客户端告警。

Snort的插件机制使得它具有很好的扩展性和可移植性，用户可以根据自己的需要及在短时间内调整检测策略，对于新的攻击威胁做出迅速反应。

Snort有三种工作模式：嗅探器、数据包记录器、网络入侵检测系统。

Snort作为开源软件填补了只有商业入侵检测系统的空白，可以帮助中小网络的系统管理员有效地监视网络流量和检测入侵行为。Snort一开始就作为一种免费开源软件发布，只要遵循GPL协议，任何人可以得到它的版本，安装使用。在Sourcefire上也可以获得Snort的商业解决方案。

2.4.2 Snort 的发展

Snort作为一种开源免费的入侵检测工具，它的发展得到了全世界很多爱好者的支持，在www.snort.org和www.sourcefire.com，有最新的发布版本和有关资料，也可以通过论坛，新闻组等交换最新信息。

1998年12月，Marty Roesch发布了开源软件Snort的第一版本，它开始是作为嗅探器，随后在Snort中加入基于特征分析(signature-based rules-based)的功能，开始被用做简单的IDS。

1999年10月，Snort1.5发布，加入了重新写过的基于Boyer-Moore算法的模式匹配引擎，插件机制，预处理机制等。规则解释采用二维链表，从此一直沿用该体系结构，直到2.0版本。

2000年，Snort加入端口扫描插件，Ip重组插件，数据库输出等功能。

随着网络的发展和网络攻击的增多，Snort核心规则集也不断增加，Snort必须通过改进方法，来提高检测速度。在2.0里，Snort采用了全新的体系结构，并重新设计了Snort的检测引擎。删除了一些多余的插件和规则。但Snort的插件和预处理机制没有改变。

在2.0后的版本中，通过试用和测试，不断有人报告错误和修正错误，也根据入侵种类的变化加入了一些相应的处理插件和预处理插件。

Snort的规则极容易修改和调整，能及时对外部入侵做出反映。用户可以迅速及时地通过www.snort.org下载最新规则。由于规则格式逐渐成为了标准，用户也可以自己写一些规则来加强网络的安全性。

今天，Snort的各个方面仍然经过一些协作组织及个人的努力，在不断改进。

2.5 Snort 的安装和使用

Snort的跨平台性能极佳，目前已经支持Linux系列，Solaris，BSD系列，IRIX，HP-UX，Windows系列，ScoOpenserver，Unixware等。

2.5.1 Snort 的安装

在本文中选用 windows server 2003 作为操作系统。下面简单介绍在 windows server 2003 下 Snort 及相关软件的安装:

1) 安装 winpcap, winpcap 是一个针对 windows 系统的包捕获库, 它向 Snort 提供捕获包的功能, 是 Snort 运行的基础。

2) 安装 mysql 数据库, Snort 的输出方式有很多种, 本文利用 mysql 数据库建立存储系统。同时安装 mysql 数据库的相关软件, apache, php, phpMyAdmin 等, 以方便操作 mysql。最后导入 Snort 安装文件下的建库脚本, 生成储存 Snort 警告的数据库。

3) 安装 Snort, 加入规则和配置文件, 修改配置文件。

4) 安装 ACID, ACID 是一个数据浏览和分析工具, ACID 能处理 Snort 数据。通过 ACID, 可以很方便的查看, 管理各种报警数据, 能根据指定的参数生成各种图表和统计数据。如图 2-3 和图 2-4 为安装成功的界面。

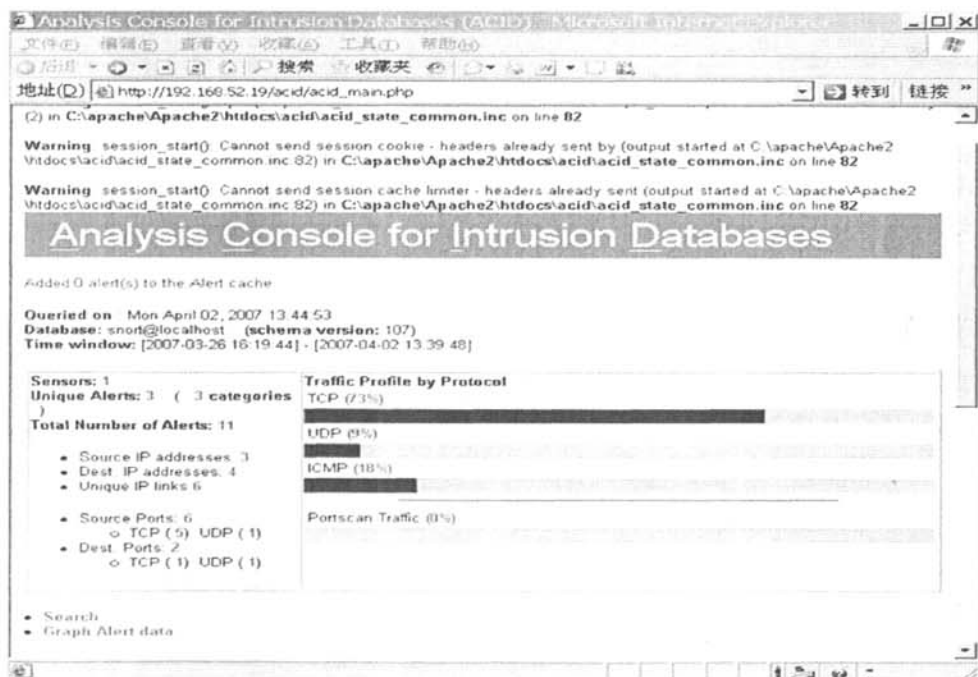


图2-3 ACID界面

Fig.2-3 ACID interface

ACID: Query Results - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 前进 搜索 收藏夹 转到 链接

地址(A) http://192.168.52.19/acid/acid_qry_main.php?&num_result_rows=1&submit=Query+DB&cur

Displaying alerts 1-12 of 12 total

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0 (1-1)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-03-26 16:19:44	192.168.52.125:1471	192.168.52.19:139	TCP
#1 (1-2)	[nessus[snort]] MS-SQL ping attempt	2007-03-26 16:19:45	192.168.52.11:1056	255.255.255.255:1434	UDP
#2 (1-4)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-03-26 16:28:19	192.168.52.19:2116	192.168.52.12:139	TCP
#3 (1-5)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-03-26 16:28:19	192.168.52.19:2116	192.168.52.12:139	TCP
#4 (1-3)	[arachNIDS][snort] ICMP L3retreiver Ping	2007-03-26 16:28:19	192.168.52.19	192.168.52.12	ICMP
#5 (1-7)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-03-26 16:40:19	192.168.52.19:2123	192.168.52.12:139	TCP
#6 (1-8)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-03-26 16:40:19	192.168.52.19:2123	192.168.52.12:139	TCP
#7 (1-6)	[arachNIDS][snort] ICMP L3retreiver Ping	2007-03-26 16:40:19	192.168.52.19	192.168.52.12	ICMP
#8 (1-9)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-04-02 13:39:02	192.168.52.19:3885	192.168.52.13:139	TCP
#9 (1-10)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-04-02 13:39:02	192.168.52.19:3885	192.168.52.13:139	TCP
#10 (1-11)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-04-02 13:39:48	192.168.52.14:1033	192.168.52.19:139	TCP
#11 (1-12)	[snort] NETBIOS SMB IPC\$ unicode share access	2007-04-02 13:50:22	192.168.52.160:1295	192.168.52.19:139	TCP

Action Selected All on Screen Entire Query

图2-4 ACID界面

Fig.2-4 ACID interface

2.5.2 Snort 的使用

Snort有三种工作模式：嗅探器、数据包记录器、网络入侵检测系统。嗅探器模式仅仅从网络上读取数据包并显示在终端上。数据包记录器模式把数据包记录到硬盘上。网络入侵检测模式是最复杂的，可配置的，在此模式下Snort分析网络数据流以匹配用户定义的规则，并根据检测结果做出反应。本文主要介绍的是第三种功能。详细使用方法可见Snort官方的操作参考手册。

1) 嗅探器

嗅探器模式指Snort从网络上读出数据包，并作为连续不断的流显示在终端上。如命令

```
./snort-vde
```

2) 数据包记录器

如果要把所有的包记录到硬盘上，指定一个日志目录，Snort自动记录数据包：

```
./snort -dev -l ./log -h 192.168.1.0/24
```

这个命令使得Snort把进入C类网络192.168.1的所有包的数据链路、TCP/IP以及应用层的数据记录到目录./log中。

使用下面的命令可以把所有的包记录到一个单一的二进制文件中：


```
./snort -l ./log -b
```

3) 网络入侵检测系统

Snort最重要的用途还是作为网络入侵检测系统(NIDS)，使用下面命令行可以启动这种模式：

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

snort.conf是配置文件。本文在snort.conf中配置了输出数据库，把报警信息直接写入数据库，-l ./log也可以去掉。

在NIDS模式下，Snort有6种输出报警机制：full, fast, socket, syslog, Smb(winpopup)和none。其中有4个可以在命令行状态下使用-A选项设置。这4个是：fast, full, unsock, none。

在默认情况下，Snort以ASCII格式记录日志，使用full报警机制，并且把报警发给syslog：

```
./snort -c snort.conf -l ./log -s -h 192.168.1.0/24
```

-s选项使Snort把报警消息发送到syslog。

2.6 IDS 及 Snort 面临的问题和发展趋势

随着网络流量的增大,各种入侵行为的增加,然而人们越来越依赖于计算机和互联网,同时对网络安全提出了越来越高的要求。IDS也被赋予了很大的责任。

当今IDS也面临以下四个方面的问题：

1. 如何提高IDS的检测速度，以适应网络通讯发展的需要^[33,34,35,36]

网络安全设备的处理速度一直是影响网络性能的一大瓶颈，在IDS中，截获网络的每一个数据包，并分析、匹配其中是否具有某种攻击的特征需要花费大量的时间和系统资源，如果其检测速度跟不上网络数据的传输速度，那么检测系统就会漏掉其中的部分数据包，从而导致漏报而影响系统的准确性和有效性。大部分现有的IDS只有几十兆的检测速度，随着百兆、甚至千兆网络的大量应用，IDS技术发展的速度已经远远落后于网络速度的发展。

2. 如何减少IDS的误报和漏报，提高其安全性和准确性^[37]

误报就是把不是入侵的行为当成入侵，而漏报是没有报告真正的入侵行为。基于模式匹配分析方法的IDS，由于每天都有新的攻击方法产生和新漏洞发布，而攻击特征库不能及时更新，这容易造成IDS漏报。而基于异常发现的IDS，必须当系统运行时的数值超过正常阈值，才认为受到攻击，易导致其较高的漏报率。另外，大多IDS是基于单包检查的，如果没有充分的协议分析，则无法识别伪装或变形的网络攻击，易造成大量漏报和误报。

3. 如何提高IDS的互动性，从而提高整个系统的安全性能^[38]

在大型网络中，网络的不同部分可能使用了多种入侵检测系统，还有防火墙、漏洞扫描等其他类别的安全设备，这些入侵检测系统之间以及IDS和其他安全组件之间交换信

息，共同协作来发现攻击，做出响应并阻止攻击是关系整个系统安全性的重要因素。有效的整合各种资源来达到安全目的，也是IDS研究的重要课题。

4. IDS自身的安全^[39]

IDS程序本身的健壮性是IDS系统好坏的重要指标。IDS的健壮性体现在两个方面：一是程序本身在各种网络环境下都能正常工作。二是程序各个模块之间的通信能够不被破坏，不可仿冒。这需要在模块间的通信过程中引入加密和认证的机制，模块间的通信也需要有良好的恢复重传机制。

由以上面临的众多问题可知，IDS最需要解决的是检测的准确性、检测引擎和检测方法的性能问题，从而提高检测速度，解决误报和漏报问题。这都离不开新技术的支撑，IDS体系结构的研究、安全通信机制的研究、新检测技术的研究、响应策略与恢复的研究以及协作式入侵检测技术的研究将是未来的研究重点。

2.7 本章小结

本章首先概述了入侵检测系统的概念，IDS的标准与结构，IDS的发展与分类，然后介绍了Snort的功能、发展、安装与使用，最后指出了IDS及Snort面临的问题和发展趋势。

3 Snort 入侵检测系统分析

Snort 是一个成熟的、被广泛使用的入侵检测系统。它是开源、免费的轻量级入侵检测软件^[40]。它由 C 语言编写，目前（2008 年 1 月）其最高的稳定版本是 2.8。本章将对 Snort 2.40 的系统结构作详细分析，并对其性能展开讨论。

3.1 Snort 2.40 系统结构

Snort 使用 libpcap 作为数据包捕获工具来获取网络上所有的数据包，其对应的 Windows 版本称为 WinPcap。今后 Snort 还可能使用更直接的捕获方式以提高性能。

Snort 主要由两大部分组成，一是 Snort 可执行文件，二是 Snort 规则。

Snort 可执行文件能工作在两种模式下，一是 Sniffer（包嗅探）模式，二是 IDS（入侵检测）模式。当它工作在 Sniffer 模式下时，可以实时捕获网络中的所有数据，并将数据显示在屏幕或记录在日志文件中，当 Snort 工作在 IDS 模式下时，系统可以实时检测网络中存在的攻击，予以报警或通知防火墙。首先通过配置文件确定应该加载哪些预处理插件和规则库。通过“规则”判断数据包中是否含有恶意内容。Snort 规则是一种简单的描述语言，每一种攻击特征都可以使用一条或多条规则描述。Snort 采取了全新的方式组织这些规则，并且同时使用了多种匹配算法，使其能在较高速的网络环境下正常工作，后面将详细阐述这些内容。

Snort 工作在入侵检测模式下时，如图 3-1 所示，由主控模块、解码模块、预处理模块、检测模块、输出模块几部分组成。主控模块首先完成各类插件的初始化工作，设置运行参数，然后读取并解析规则文件，最终进入抓包和检测流程。解码模块完成对网络包的解码，并把解码后的内容保存在全局结构 Packet 中。预处理模块完成一些主控模块难以完成的解码和检测，主要有分片重组、代码转换、异常检测等工作。检测模块是 Snort 的核心，它包含了对数据包内容的各项检测。输出模块负责将检测结果报告给用户，它支持多种输出方式，如屏幕、文件、数据库和 SMB 信息。

当一个数据包被 Snort 接收时，首先进入预处理过程，完成基本的 IP 包格式校验、IP 分片重组和解码工作后，数据包进入核心检测流程，Snort 依次对包头和负载内容与预定义的入侵特征进行匹配。匹配成功意味着发现入侵，这时，输出模块将执行与之相关的记录和报警工作。

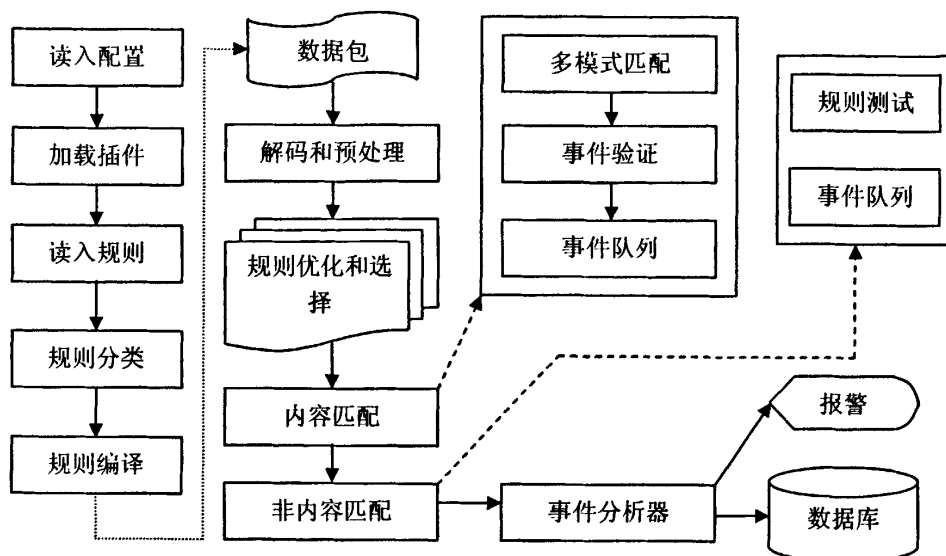


图 3-1 Snort 在 IDS 模式下的工作流程

Fig.3-1 Working Flow of Snort under IDS mode

3.1.1 Snort 规则

Snort 规则是 Snort 入侵检测系统的重要组成部分。规则集是 Snort 的攻击特征库，每条规则都对应一条攻击特征，Snort 通过它来识别攻击行为。每一条规则包括两个部分：规则头部和规则选项。

规则头部可以看作是一个七元组，由动作、协议、源 IP 地址和源端口号、方向操作符、目的 IP 地址和目的端口号构成。动作是指当 Snort 发现从网络中获取的数据包与事先定义好的规则相匹配时，下一步所要进行的处理方式。Snort 支持 alert、log、pass、activate、dynamic、drop、reject 等动作，最常见的动作是 alert（报警）。

规则选项由选项关键字和选项内容组成，关键字和选项间由冒号隔开，各关键字之间由分号隔开。规则选项包含了入侵特征串（content）、URI 请求特征串（URIContent）、告警内容（msg）、负载长度（dsize）、类型（classtype）、严重程度（priority）、版本（rev）及与模式匹配相关的重要信息（nocase、offset、depth 等），各选项间以分号分隔。每个选项由冒号分隔为选项关键字和选项值两部分。规则选项构成了 Snort 检测引擎的核心，它们非常容易使用，同时又很强大和容易扩展。

content 和 uricontent 是两个最重要的规则选项，分别表示在数据包负载和 URI 请求中搜索某个特征字符串。该字符串可以是一段普通的文本，也可以是一串 16 进制字符，前后用管道符“|”分隔，如 content: “|1AC9 1588 167E|”。这两个关键字都可以在规则中出现一次或多次。

作为开源的基于网络的入侵检测系统，Snort 规则得到了人们广泛的认同，典型的 Snort 规则示例如下：

```
alert tcp any any -> $HTTP_SERVER 80 (msg:"WEB-IIS cmd.exe access";
flow:to_server, established; uricontent: "cmd.exe"; nocase; classtype: web-application-attack;
sid:1002; rev:7;)
```

示例中的规则对所有发往 HTTP 服务器 80 端口的 TCP 数据包进行检查，当发现 URL 请求中包含大小写无关的字符串“cmd.exe”时，给出“WEB-IIS cmd.exe access”的告警信息。

目前 Snort 的默认规则约有 4300 条，按照攻击方式不同，分为 backdoor、bad-traffic、cgi-bin、DDoS、dns、exploit、ftp、icmp、mysql、netbios、rpc、sql、web-attacks、web-iis 等近 60 类。

当 Snort 运行在 IDS 状态下时，通过配置文件决定使用哪些规则。Snort 运行机制如图 3-2 所示。用户还可以在配置文件中定义变量（如 IP 地址）、指定使用哪些插件、采取怎样的输出方式和报警方式等。典型的配置文件如下：

```
#定义变量
var HOME_NET 10.1.1.0/24

#定义预处理器
preprocessor stream4

#定义输出方式：记录到 MSSQL 的 snort 数据库
output database: log, mssql, dbname=snort user=snort password=test

#定义规则文件：
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/rpc.rules
...
```

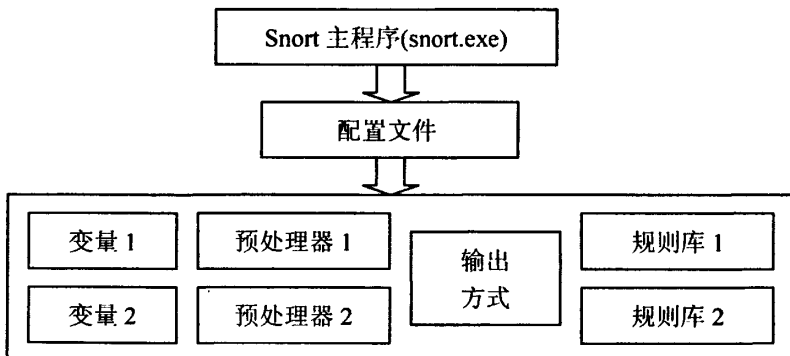


图 3-2 Snort 运行结构图

Fig.3-2 Architecture of Snort Main Program

3.1.2 Snort 的规则解析

规则是 Snort 系统重要的组成部分。一个入侵检测系统的检测性能，除了与检测方法密切相关外，还与检测规则的设置和数量密切相关。检测规则的数量越多，一方面意味着 IDS 能够检测出更多的入侵，另一方面意味着同一个检测对象，可能需要经过更多的处理步骤，才能确定是否与入侵特征相关。面对数千条入侵规则，Snort 首先要做的是把文本形式的规则文件读入内存中并以特定的结构存储。如图 3-3 所示。

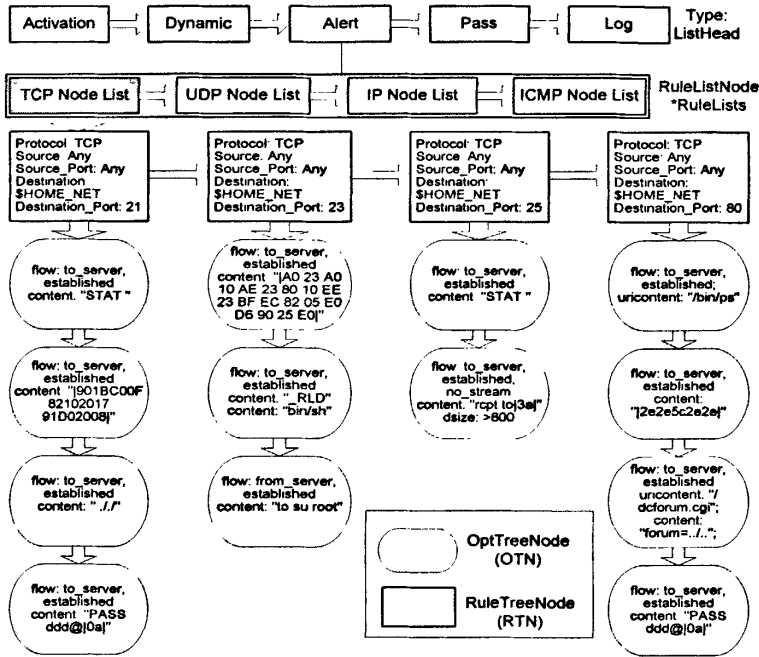


图 3-3 Snort 规则树

Fig.3-3 Rule Trees of Snort

规则树以如下方式产生：按照反应类型的不同，规则被分为 activation、dynamic、alert、pass、log 等几大类。每类中包含了 TCP、UDP、ICMP 等协议类型。

规则树中有两类结点：规则头结点（RuleTreeNode，简称 RTN）和规则选项结点（RuleOptionNode，简称 OTN）。RTN 中除了保存规则头的信息外，还分别保存了下一个 RTN（横向）和下一个 OTN 结点（纵向）的地址。OTN 中主要保存规则选项信息，一个 OTN 与一条规则是对应的，此外它还保存了一组用于处理和该 OTN 有关的函数列表，以及下一个 OTN 结点。

Snort 的 parser.c 负责规则解析。它根据配置文件打开相应的规则库文件，每次读取一条规则。首先提取规则头，并与已存在的 RTN 比较，如果未找到与当前规则头完全相符的 RTN，则在 RTN 链尾追加一个新的 RTN 结点，随后解析规则选项，形成 OTN 结点，并将该结点挂接到刚刚新建的 RTN 结点上。如果找到与当前规则头相符的 RTN，则将规

则选项解析后形成的 OTN 结点添加到当前 RTN 结点所对应 OTN 链的末尾。

当所有规则解析完毕后，各条规则便按照规则头分类，形成规则头节点（RTN）。所有规则头相同的规则，其规则选项被链接到相应的规则头结点下，形成规则选项链（OTN）。

存放规则头结点（RTN）的数据结构如下：

```
typedef struct _RuleTreeNode
{
    RuleFpList *rule_func;           /*用于匹配的函数*/
    int head_node_number;            /*规则头结点 ID*/
    int type;                         /*协议类型*/
    IpAddrSet *sip;                  /*源地址*/
    IpAddrSet *dip;                  /*目的地址*/
    int not_sp_flag;                 /*是否为“非”端口*/
    u_short hsp;                     /*源端口结束值*/
    u_short lsp;                     /*源端口起始值*/
    int not_dp_flag;                 /*是否为“非”端口*/
    u_short hdp;                     /*目的端口结束值*/
    u_short ldp;                     /*目的端口起始值*/
    u_int32_t flags;                 /*控制标识*/
    ...
    struct _RuleTreeNode *right;      /*指向下一个 RTN*/
    OptTreeNode *down;               /*指向 OTN 链*/
    struct _ListHead *listhead;      /*指向规则树的根结点*/
} RuleTreeNode;
```

早期版本的 Snort 是这样进行数据包匹配工作的：当数据包到达时，首先寻找与之匹配的规则头结点（沿图中矩形结点横向搜索），找到对应于数据包的结点后，与每一个规则选项结点匹配（沿图中圆角矩形结点纵向搜索）。

采用这样的方式进行数据匹配工作，效率是较为低下的。首先，每个数据包到达时，都必须找到相应的规则头结点。由于规则头结点在内存中以链表形式保存，查询的代价相对较大。Snort 中最多约有 280 个规则头结点，若采用该算法，每个数据包需要查询大约 140 次才能找到所属的规则头。其次，每个规则头结点下链接的规则选项结点数量差异较大。一些常用根结点（如 80，135，445 等端口）下的规则选项结点大约有 200 个。数据包与这么多 OTN 选项结点一一匹配，耗费了相当长的时间。

存放规则选项结点（OTN）的数据结构如下：

```
typedef struct _OptTreeNode
{
    OptFpList *opt_func;             /*检测函数列表*/
    RspFpList *rsp_func;             /*响应函数列表*/
    void *ds_list[64];               /*动态特征数据列表*/
    int type;                         /*报警类型*/
    int proto;                        /*解析规则时使用的协议*/
    struct _RuleTreeNode *proto_node; /*指向该选项的规则头*/
    SigInfo sigInfo;                 /*规则的签名信息*/
    struct _OptTreeNode *next;        /*下一条规则*/
    struct _RuleTreeNode *rtn;        /*本规则选项的头*/
} OptTreeNode;
```

3.1.3 Snort 的新特性：规则索引

Snort 自 2.0 版后引入了快速检测引擎 (FPDE)，重写了部分插件，使系统性能显著提升。FPDE 提出了一种新的创建规则索引的思路，大大提高了系统性能。

FPDE 事实上是对动态最优决策树的静态实现^[41]。入侵检测系统中，常有很多属性需要匹配，如地址、协议、端口、URI、选项、大小、负载内容等等。匹配这些属性所需要的时间是不同的，所以不同的匹配顺序可能带来性能的明显差异。例如检测端口仅仅需要一次运算，而检测负载的内容则需要复杂的比较。我们希望通过每一次检测都能减少需要继续比较的规则的数量，即尽量减少费时的操作。对 Snort 规则进行的统计分析表明，首先应该对端口进行分类，因为它可以迅速把规则集从数千个降到数十个甚至是几个。

某些入侵检测系统还可以根据接收到数据包中各属性的比例不同，动态地调整各属性的检测顺序，实现动态决策。但由于某些属性之间是有关联的，这会增加系统的复杂性，却无法带来性能的明显提升，还需要考虑某些攻击。Snort 采用固定的优化顺序对包检测。

a. 三层规则索引结构

Snort 初始化时，会根据配置文件的要求加载相应的规则并生成规则树。随后，Snort 对规则进行三层分类，分类层次从上至下依次为：协议映射类 (PORT_RULE_MAP)、源/目标端口集合类 (PORT_GROUPS)、内容集合类 (PORT_GROUP)。如图 3-4 所示。

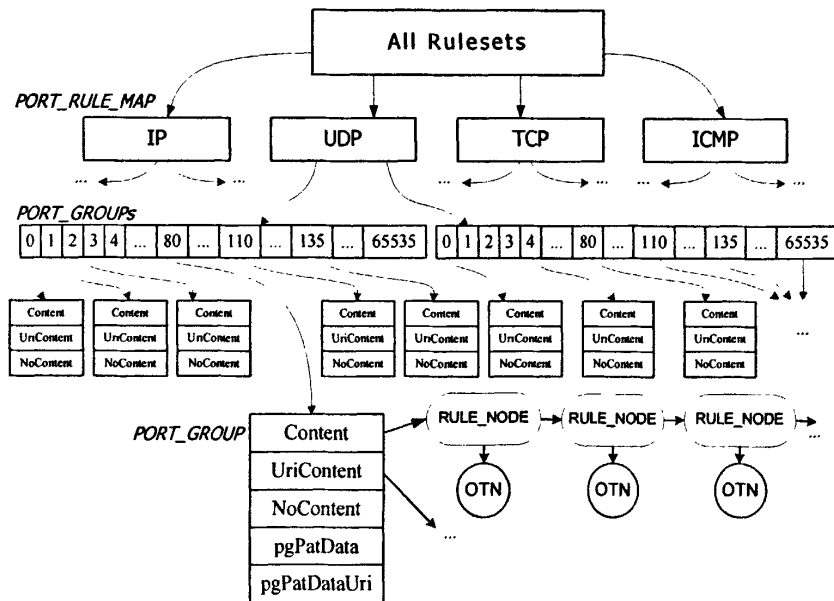


图 3-4 Snort 规则索引图

Fig.3-4 Index of Snort Rules

b. 端口集合类

对不同的协议，Snort 都为它们建立源/目标端口规则映射 PORT_RULE_MAP，它的数据结构如下：

```
typedef struct {
    int      prmNumDstRules;      /*目的规则数量*/
    int      prmNumSrcRules;      /*源规则数量*/
    int      prmNumGenericRules;  /*通用规则数量*/
    int      prmNumDstGroups;     /*目的规则(端口)集合数量*/
    int      prmNumSrcGroups;     /*源规则(端口)集合数量*/
    PORT_GROUP *prmSrcPort[MAX_PORTS]; /*源规则端口集合类*/
    PORT_GROUP *prmDstPort[MAX_PORTS]; /*目的规则端口集合类*/
    PORT_GROUP *prmGeneric;       /*通用规则集合类*/
} PORT_RULE_MAP;
```

该映射中的 prmSrcPort、prmDstPort 和 prmGeneric 这三个数组分别用于存放源端口规则集、目标端口规则集与通用规则集。各规则依据下述的分类方法，加入不同的规则集合中：

- ① 如果源端口值为特定值 i ，目的端口为 ANY，则将该规则加入源端口集 prmSrcPort[i]中；如果目的端口值为特定值 i ，源端口为 ANY，则将该规则加入到目的端口集 prmDstPort[i]中。
- ② 如果源端口值为特定值 i ，目的端口值为特定值 j ，则将该规则同时加入源端口集 prmSrcPort[i]和目的端口集 prmDstPort[j]中。
- ③ 如果目的端口和源端口均为 ANY，则该规则加入通用集（prmGeneric）中。
- ④ 如果规则中出现端口值求反或指定范围的情况，视其端口值为 ANY。

将所有通用集中的规则加入每个非空的 prmDstPort[i]和 prmSrcPort[i]中，其中 $i \in [0, 65535]$ 。通用集合（prmGeneric）将不再参与匹配工作。

c. 内容集合类

为了能更有效地进行匹配操作，Snort 根据内容类别不同进行分类。分类方法如下：

- ① 规则中若含有 content 关键字，即需要对负载内容作匹配的规则，则加入内容规则集 pgHead；
- ② 规则中若含有 uricontent 关键字，即需要对 URI 请求作匹配的规则，则加入 URI 规则集 pgUriHead；
- ③ 规则中不含有上述关键字的，加入无内容规则集 pgHeadNC。

其它规则选项编译为规则选项（OTN）链表，保存在相应的 PORT_GROUP 中，它的数据结构如下：

```

typedef struct {
    RULE_NODE *pgHead, *pgTail, *pgCur;           /*内容规则集*/
    RULE_NODE *pgHeadNC, *pgTailNC, *pgCurNC;     /*无内容规则集*/
    RULE_NODE *pgUriHead, *pgUriTail, *pgUriCur;  /*URI 规则集*/
    /*规则编译后将结果存放在下面两块空间中*/
    void * pgPatData;                               /*内容规则集的编译结果*/
    void * pgPatDataUri;                           /*URI 规则集的编译结果*/
}PORT_GROUP;

```

其中, RULE_NODE 是 OTN 与 RTN 结构的简单封装。

3.1.4 Snort 的新特性: 多模式匹配

基于误用的入侵检测系统, 归根到底是搜索数据包中是否含有特定内容。因此, 模式匹配性能的优劣直接影响入侵检测系统的性能。目前最著名的快速模式匹配算法是 Boyer-Moore (BM) 算法。

然而, 与一般的单一模式匹配不同, IDS 中的模式数量众多, 内容相对固定。这要求 IDS 最好能一次完成多个模式匹配操作, 而不是逐一进行模式匹配。BM 算法一次仅能对一个字符串进行匹配, 显然这不能满足 Snort 对性能的要求。必须对 BM 算法进行改进, 使之能同时匹配一个或多个模式, 是入侵检测系统检测模块的难点。

Aho-Corasick(AC)、Karp-Rabin(KR)和 WuMan^[42]算法都是较为著名的多模式匹配算法, 由于 AC 和 KR 算法有一些弱点, WuMan 成为目前 Snort 的首选算法。

在 3.1.3 中构造的三层索引后, 能使每个包到达时都能找到一组对应的模式与之匹配。由于在系统运行时, 模式集合是固定的, 因此 Snort 在初始化时就编译了这些模式集。Snort 对每个端口下的模式进行编译, 将编译的结果放在内容集合类的 pgPatData 和 pgPatDataUri 中。

Snort 根据每个内容集合类中模式数量的不同选择不同的算法。Snort 对模式数量小于 5 的集合采用 BM 算法, 否则采用 WuMan 等算法。WuMan 算法继承了 BM 算法的某些思想, 但比 BM 算法复杂得多, 为节省空间, Snort 对它进行了适当的修改。算法如下:

a. 模式的移动: Hash 值与 Shift 表

WuMan 的基本思路是将多个模式同时移动尽可能大的距离。如图 3-5 所示, 设 P 为多个模式串, T 为欲匹配的文本, m 为最短模式的长度 (图中为 P_4), T 的最末端两个字节为 x_1x_2 。

首先将模式 P 的最短模式右端与 x_1x_2 对齐, 并将其与文本 T 的最后两个字符进行比较。①若这两个字符未出现在 P 中任何一个模式中, 则将 P 右移一段距离 $m-1$ 。②若这两个字符出现在 P 中, 则将 P 右移一段距离。设 q 为 P 中所有出现 x_1x_2 位置的最大值 (即

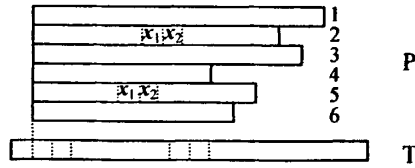


图 3-5 模式的移动

Fig.3-5 Movement of Patterns

最右端 x_1x_2 的位置，图中位于 P_5 ），则右移距离为 $m-q$ 。

对于已确定的多模式串 P ，针对每个不同的 x_1x_2 ，其移动距离是确定的。使用 (1) 中的方法预先计算每个可能的移动距离，即可在匹配过程中通过查表的方式得到 P 的移动距离。为此，Snort 将每一种可能的 x_1x_2 对应的移动距离预运算后保存在 Shift 数组中，数组的下标为 x_1x_2 的哈希值。每次移动操作时，计算 $\text{Hash}(x_1, x_2)$ 即可得到 x_1x_2 对应的移动距离 $\text{Shift}[\text{Hash}(x_1, x_2)]$ 。此处采用的 Hash 算法是十分简单的，不会增加额外的开销。

在原始的 WuMan 算法中，还可以将模式的比较长度增加至 3，即比较文本 T 和模式 P 的最后三个字符。由于算法需要的空间太大，Snort 中没有采用。另外，Snort 2.40 中将 P 的左端与 T 对齐，并与 T 左端的前两个字符进行比较，其结果是完全相同的。

b. 前缀索引：Hash 值与前缀索引 (Prefix Index) 表

如果 $\text{Shift}[\text{Hash}(x_1, x_2)]$ 不为 0，就意味着 x_1x_2 与某个 P_i 匹配，但无法得知到底是哪个 P_i 。为了避免 T 中的文本与 P 一个个地比较，需建立一张前缀索引表 $\text{Prefix}[\text{Hash}(x_1, x_2)]$ ，用于存放所有含有 x_1x_2 的 P_i 。 P_i 是按其前两个字符的 Hash 值排序的。这样，对于特定的 x_1x_2 ，计算 Hash 值，就可以直接找到对应的一组模式 P_i 。见表 3-1。

WuMan 算法还要求维护一张后缀表，作为发生前缀冲突时的索引，它与前缀索引表的建立方式完全相同。在 Snort 中，同一个端口集合下的模式数量一般不会超过数百条，前缀冲突的可能性并不大。因此，Snort 当发生前缀冲突时，Snort 使用一种类似 BM 的算法，从右向左的匹配算法对每个模式进行匹配。

c. 搜索算法

①根据文本 T 中后两个位置（图中的 x_1x_2 ）的值，通过 Shift 表确定模式应该移动的距离，直到移动到一个可能的匹配项出现。

②检查前缀索引中是否含有与当前文本 T 的前缀（图中的 t_1t_2 ）相同的模式，若没有跳转至④。

③使用从右向左的方式，比较每一个模式 P_i 是否与当前文本匹配。

④将模式 P 向右移动一个字符，并继续进行搜索操作。

上述过程循环往复，直至最终完成匹配操作。

表 3-1 Hash 值与 P 的对应关系

Table 3-1 The corresponding relationships of Hash Values and P

Hash(x_1, x_2)	Hash 值为 Hash(x_1, x_2)的 P_i
0	$P_0 P_1$
1	P_2
2	-
3	$P_3 P_4 P_5 P_6$
...	...
65536	-

3.2 Snort 插件

Snort 结构的可扩展性是非常强大的, 插件结构令系统具有极大的灵活性。Snort 的插件分为三类: 预处理插件、检测插件和输出插件。

预处理插件针对当前捕获的包进行预处理, 比如初步的解码、重组工作, 这些工作可以把数据包从原始形态转换成一种易于分析的模式。编写预处理插件有相应的模板, 编写者不需要对 Snort 系统的其它部分有额外的理解。

检测插件用于大多数关键字的检查, 如 ttl、pcrc、content、uricontent、byte_jump 等, 当 Snort 规则中出现新的关键字时, 只需在检测插件中添加相应的文件即可。

输出插件用于各类场合的输出, 如输出到屏幕、日志和不同的数据库, 甚至通知防火墙。

3.2.1 Snort 最新插件介绍

Portscan: 端口扫描是黑客攻击的前奏, 发现端口扫描应引起管理员的重视。端口扫描预处理器一方面可以记录来自某个 IP 地址的端口扫描的开始和结束, 另一方面可以把扫描类型、目的地址和端口等信息全部记录下来。端口扫描有多种方式, Snort 目前可以发现 TCP、UDP、FIN、SYNFIN 等多种扫描。使用第三方的 SPADE 异常检测引擎, 还可以发现更多种类的隐蔽扫描。

Stream4: 该预处理器提供了 TCP 流重组和状态分析的功能。在 2.30 之前的版本中, 该插件的流重组能力不够强大, 打开该插件后可能出现性能明显下降、丢包率剧增的情况,

成为黑客攻击的目标。现在，它能同时处理 256000 个并发连接，基本满足了应用的需要。

Frag3: 该预处理器用于 IP 包分片的重组。它使用了最新版本 Linux 中的部分实现。初步测试表明，它要比 Frag2 快 250%。它使用了操作系统对 IP 栈的处理模型，因此能最大程度地避免精心设计的 IP 碎片逃避 IDS 的检测。

HttpInspect (原 HTTP_Decode): 对 Http URL 请求中经编码的字符进行解码，这可以躲过 IDS 的检测，也可能逃过操作系统的检查，造成漏洞。IIS 和 apache 等著名 Web 服务器都出现过此类情况，它可能直接造成服务被攻陷。现在，这部分代码经过了严格的检查和详细分类，能完成 ASCII、UTF-8、Unicode、多次编码等类型的解码工作。

3.3 关于 Snort 性能的讨论

Snort 入侵检测系统由几个主要部分组成：当一个数据包到达时，首先由 libpcap 负责抓包。随后，包内容被解析，存放在 Packet 结构中。预处理插件将包作进一步处理，然后进入核心的规则匹配工作，最后事件日志模块将结果写入数据库或显示给用户。

负责抓包的 libpcap 是一个用户态驱动程序，当收到包时，需要将包从内核内存复制一份到用户内存区。这大大影响了抓包性能。目前大多数商业 IDS 都采用的内核级驱动程序抓包^[43]，如果把使用普通 libpcap 的 Snort 系统直接与这些系统比较，是极不公平的。Snort 可以使用 MMAPed pcap^[44]来代替 libpcap，这是一个特殊版本的 libpcap，其接口与普通的 libpcap 相同。但 MMAPed pcap 直接读取内核内存，减少了数据包在 Snort 读取之前被复制的次数。另外，提高抓包性能的另一个方法是采用网络处理能力较高的操作系统，如 FreeBSD。

包内容解析采用了与 Unix 系统相同的算法，应该说这个模块的性能已经发挥到极致，从最新发布的 beta 版中可以看出，今后 Snort 将改变原有的 Packet 结构，为支持 IPv6 做准备。

每个包到达后，都必须经过数个预处理插件，因此预处理插件的性能对 Snort 的整体性能至关重要。所有插件中最为复杂的是负责 TCP 流重组的 stream4 插件。在遭受某些攻击类型的攻击时下，stream4 插件成为系统瓶颈^[45]。目前 Snort 的流重组能力已经比过去强大了 1000 倍。

规则匹配一直是最受关注的部分。Snort 1.x 受到诟病最多的是规则匹配的效率较为低下，不能满足 100M 或更高速的网络环境。自 Snort 2.0 发布以后，引入了 WM 和 AC 算法，模式匹配的效率有了明显提高，在最好的情况下，其匹配性能是 1.x 版本的 18 倍。对 Snort 早期版本的研究表明，出现大量小数据包或非 TCP 包时，规则头匹配耗时严重；出现大量 HTTP 等 TCP 流量时，内容匹配成为系统瓶颈^[46]。这两个问题目前已有明显改善。在没有更优秀的模式匹配理论出现之前，IDS 在这方面的性能提高空间已经不大。

当遭受包含大量数据包（如 DDos）攻击时，事件和日志模块成为系统瓶颈。Snort

支持 Alert_fast、Alert_full 等多种不同输出方式，当有过多的报警信息需要写入磁盘或数据时，日志模块将不堪重负。使用 mucus、stick 这类攻击包生成工具可以在短时间内塞满事件数据库，使系统疲于应付大量 IO 操作，直至崩溃。

作为一个软件 IDS，某些方面的性能瓶颈是难以克服的。将 IDS 移植到嵌入式系统，使用 FPGA/NP/ASIC 结合的方式设计硬件入侵检测系统前景十分广阔。

3.4 本章小结

本章分析了著名的开源入侵检测系统 Snort。在简要介绍了 Snort 系统的原理后，对新版 Snort 中最为重要的规则解析模块和匹配算法作了详细的研究和分析。Snort 现在使用了全新的协议分类机制和规则索引，并采用了改进的 Wu-Man 多模式匹配算法和 BM 算法。最后，还对 Snort 的插件进行了介绍，对插件与 Snort 性能的关系进行了讨论。这些分析对今后修改和优化 Snort 系统奠定了基础。

4 基于活跃规则集的 Snort 高效规则匹配方法

根据改进 Snort 性能的方法, 围绕规则优化的思路, 本章提出了基于活跃规则集的 Snort 规则匹配方法, 下面将详细阐述这种方法。

4.1 Snort 规则匹配的机制

Snort规则包含两个逻辑部分内容: 规则头和规则选项。规则头对应于规则树结点RTN (Rule Tree Node), 包含执行的动作、使用的协议、源(目的) IP地址和掩码、端口信息及数据流向等。规则选项OTN (Optional Tree Node)包含一些报警信息(msg)、匹配内容(content)项。

在Snort 2.0版本以前, Snort初始化并解析规则时, 分别生成TCP、IP、ICMP、UDP四个不同的规则树, 每一个规则树包含独立的三维链表: RTN、OTN和指向匹配函数的指针。当Snort捕获到一个数据包时, 按照协议类型转入对应RTN; 然后与RTN结点依次进行串行匹配, 如果规则头不完全匹配, 则转向下一个RTN, 否则, 如果采用单模式匹配算法, 则向下与OTN结点进行串行匹配; Snort 2.0以后开始采用多模式匹配算法, 如果采用多模式匹配算法, 则向下与OTN 结点进行并行匹配。每个OTN结点包含一条规则所对应的全部选项, 同时包含一组函数指针, 用来实现对这些选项的匹配操作。当数据包头和OTN也完全匹配时, 就触发一个规则, 产生报警, 然后进行下一个数据包的检查^[47]。

4.2 提高 Snort 规则匹配速度的方法

提高 Snort 检测性能可以从下面三个角度来进行考虑:

(1)减少需要检测的网络流量, 即通过对流量数据进行分析, 只选择那些可能具有攻击特征网络流量进行匹配; 主要进行协议流分析, 协议流是指在应用层协议中的客户端和服务端端的通讯。客户到服务器的数据流称之为客户流; 服务器到客户的数据流称之为服务器流。协议流分析是指根据应用层协议的特征, 将网络上的数据流分成客户流和服务流。

(2)规则优化, 即通过一定的方式来对规则文件进行优化, 减少数据所要匹配的规则条数; 规则优化的主要思想是通过将 Snort 规则集进行一定的优化, 使得可以在数据包进行检测的时候, 迅速定位到一个较小的规则集合中进行检测。

(3)采用效率更高的模式匹配算法, 减少模式匹配所花费的时间。模式匹配算法比较成熟, 且相对于前面介绍的协议流分析和规则优化而言, 对提高 Snort 检测性能的效果不十分明显。

文献[48]基于构造出更小的规则集^[48]来提高Snort性能的思想, 提出了一种根据网络流

量和规则特点来优化组织Snort规则的方法，取得了良好的效果，但实现起来比较复杂；文献[49]用决策树来组织规则，提出了一种新的检测引擎Snort-ng^[49]，缺点是仍然是一种静态的改进方法；文献[50]则是根据实验指出随着规则数量的增加，Snort 2.0在处理时间和内存消耗方面都明显优于Snort-ng^[50]。文献[51]和[52]充分考虑了网络攻击的阶段集中性，提出根据动态选项索引排序规则的方法，从而尽快匹配当前活跃的攻击^[51,52]，缺点是这种实现方法没有考虑多模式匹配算法的情形。

4.3 基于活跃规则集的规则匹配方法

根据Snort的新特性，考虑到这样一个事实，即尽管Snort的规则数越来越多，但在一定时间内只有一小部分规则是活跃的，大部分规则都不会触发。基于这样的思想，我们提出一种新的规则分类组织排序方法。我们的方法通过一个反映规则被触发次数的入侵匹配频度变量把每个端口下的规则集分成活跃规则集与不活跃规则集，活跃规则集是匹配频度不为零的规则集合，不活跃规则集是匹配频度为零的规则集合。并充分考虑了网络攻击的阶段集中性。由于我们的方法构造出了更小更优的规则集，所以无论对单模式匹配还是多模式匹配，基于活跃规则集的规则匹配方法都是适应的，都会带来性能的提升。

4.3.1 OptGroupNode 选项分组结构的建立

为了划分活跃规则集与不活跃规则集，我们首先在原来Snort的每个RULE-NODE下增加一个OptGroupNode选项结点链(见图4-1)，选项链由2个结点组成，OptGroupNode选项结点包括三个内容：flags 的值(可赋值为0或1)，一个向右的指针(指向下一个OptGroupNode选项结点) 以及一个向下的指针(指向选项链)。

OptGroupNode选项结点数据结构描述如下：

```
typedef struct OptGroupNode
{
    Int  flags;
    struct OptGroupNode * right;
    /* ptr to the next OptGroupNode in the list */
    OptTreeNode * down;
    /* list of rule options to associate with this rule node */
} OptGroupNode;
```

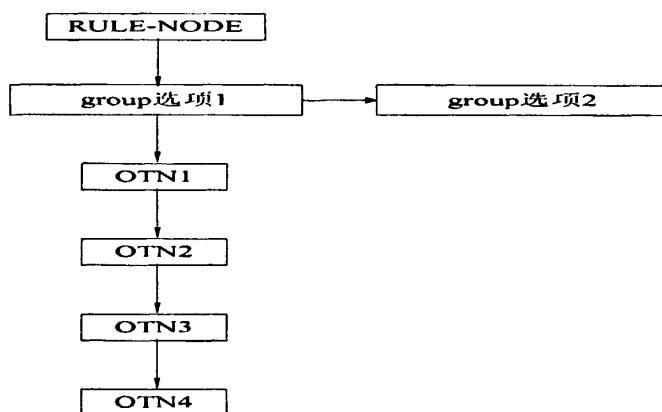



图4-1 增加OptGroupName选项链后的规则树示意图

Fig.4-1 Regular tree schematic drawing after introducing OptGroupName

4.3.2 OptIndexNode 选项索引结构的建立

我们为每一条规则选项建立一个索引Index，在Snort的规则解析函数ParseRulesFile() 中为每一个OTN生成一个指向该选项结点的指针OptIndexNode，并将它与规则头相连，形成“选项索引链表”，如图4-2所示。

选项索引结构为：

```
typedef struct OptIndexNode
```

```
{
```

```
int sum; /* match's frequentness */
```

```
struct OptTreeNode*otn_addr; /* point to the OTN node */
```

```
struct OptIndexNode*next; /* point to the next OIN node */
```

```
struct OptIndexNode*former; /* point to the former OIN node */
```

```
} OptIndexNode;
```

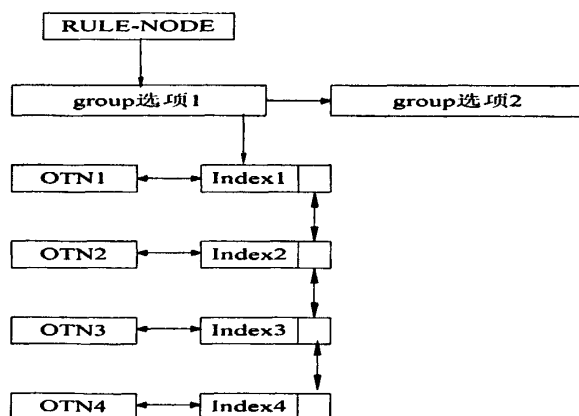


图4-2 增加OptIndexNode索引后的规则树示意图

Fig.4-2 Regular tree schematic drawing after introducing OptIndexNode

因为增加了Index选项链, OptTreeNode-> next 就成为冗余的指针, 考虑到初始化的方便, 我们仍然保留此指针, 它也仅在初始化时提供链接, 同时为了能够记录某规则的匹配频度(即修改OptIndexNode.sum), 需在OptTreeNode中增加OptTreeNode->oin_addr指针。

4.3.3 规则树的建立及规则匹配

我们把每个端口下的规则分成2组, 一组为活跃规则集(flags值为0), 另一组为不活跃规则集(flags值为1)。在Snort中, 函数ParseRuleOptions(char *rule, int rule_type, int protocol)完成规则中全部选项的解析。因为增加了OptGroupNode选项链, 程序需作如下改动:

(1) 在每个RULE-NODE结点下增加一个由2个OptGroupNode结点组成的选项链; 第一个结点的flags值为0, 第二个结点的flags值为1;

(2) Snort初始化时我们把相应端口下的全部规则通过OptIndexNode索引链接在一个flags值为0的OptGroupNode选项结点下;

(3) 为每一条规则选项建立一个OIN节点;

(4) 初始化OIN节点的左指针(OptIndexNode->otn_addr=OptTreeNode), 初始化OTN指向OIN的指针(OptTreeNode->oin_addr=OptIndexNode), 依次为具有相同规则头的规则选项的OTN建立双向链表。

(5) Snort初始化形成规则树;

(6) Snort在检测出攻击数据包之后, 则对与之匹配的规则的匹配频度进行更新, 同时按照匹配频度的高低对规则选项的索引链进行排序;

(7) 然后我们统计匹配频度, 把匹配频度不为零的规则选项仍然放在这个OptGroupNode选项结点下, 而把匹配频度为零的OptGroupNode结点链从原结点链中分离出来, 并把它链接到flags值为1的OptGroupNode选项结点下; 如图4-3所示, 为反馈匹配频度之后形成的规则树;

(8) 并且随着Snort的运行, 我们把flags值为1的OptGroupNode选项结点下出现的匹配频度不为零的规则选项重新链接到flags值为0的OptGroupNode选项结点下。Snort规则匹配时, 我们先匹配OptGroupNode选项1(flags值为0)下的活跃规则集, 当OptGroupNode选项1下OptIndexNode指向的所有OTN没有匹配的规则时, 我们然后才沿着OptGroupNode链匹配OptGroupNode选项2(flags值为1)下的所有OTN规则选项。至此我们对一个端口下的规则匹配完成。

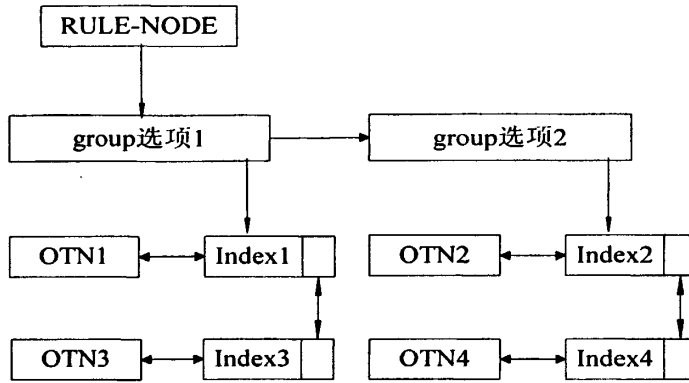


图4-3 反馈匹配频度后的Snort规则树

Fig.4-3 Snort rule tree after feedbacking match frequency

4.3.4 存在问题的解决

为了避免OptIndexNode->sum的溢出，同时又考虑到这样一个问题，即我们对匹配频度的更新滞后于攻击的变化，这样会增加无效匹配的次数。因此我们每隔一定时间 t 把排在OptIndexNode索引链最前面的（OptIndexNode.sum）乘以常数 $c(0 < c < 1)$ ，然后重新排序，这样可以大大减少无效匹配的次数。同时我们每隔时间 T 把每个规则选项的匹配次数减1，从而实时动态分离出不活跃规则集。

4.4 活跃规则集的形成和保持

我们对Snort性能的提高程度依赖于活跃规则集的大小，所以怎样形成和保持一个合适大小的活跃规则集是关键。

4.4.1 活跃规则集的形成

考虑一个端口下的全部规则集合 $\{r_1, r_2, \dots, r_n\}$ ，我们取一段时间 t ，假设网络流量为 p 个数据包/s，每条规则对应的攻击出现的概率分别为 $P_{r_1}, P_{r_2}, \dots, P_{r_n}$ 。则每秒钟与规则 r_k 对应攻击数据包的数量是 $p \times P_{r_k}$ 个攻击数据包/s。我们启动Snort一段时间，我们假设匹配频度不为零的规则序号依次为 $1, 2, \dots, i$ ， M_{r_i} 代表对应规则的匹配频度，此时活跃规则集下的规

则满足: $M_{r_s} \neq 0$, 其中, $s=1, \dots, i$
 $M_{r_s} = 0$, 其中, $s=i+1, \dots, n$ 。

我们的目标是分离出和保持一个这样的活跃规则集, 即活跃规则集中的最大序号 s 应当满足 $s \leq i$ 。

4.4.2 活跃规则集的保持

启动Snort一段时间后, 我们已经分离出活跃规则集。然后我们应当考虑怎样实时动态维持这样一个活跃规则集, 即活跃规则集中的最大序号 s 应当满足 $s \leq i$, 因为随着运行时间的增加, 新的攻击的出现, s 是递增的, 所以我们要确定一个时间段 t , 每隔时间 t 把活跃规则集中的老化规则分离到不活跃规则集。

我们假设在 t 这段时间里, 增加了 a 个攻击, 为使 $s \leq i$, 我们有不等式 $i-b+1 \geq a$, 即 $i-a+1 \geq b$, 我们这里可取 b 的上界, 即 $b=i-a+1$, 其中 b 为活跃规则集中的某个序号 ($1 < b < i$), 如果我们取每隔 $1s$ 把每个规则选项的匹配次数减1, 则 t 满足等式 $t = pP_b$ 。

这样每隔时间 t 把活跃规则集中序号大于等于 b 的规则重新归入不活跃规则集, 同时把不活跃规则集中匹配频度不为零的规则重新链接到活跃规则集。这样实时动态的更新和维持活跃规则集。

4.5 本章小结

本章在前人工作的基础上提出了基于活跃规则集的Snort规则匹配方法, 该方法通过引入OptGroupNode选项分组结构把同一个端口下的Snort规则分为活跃规则集和不活跃规则集。通过引入OptIndexNode选项索引结构来统计入侵频度, 以便为OptGroupNode选项分组结构提供划分活跃规则集的依据。本章中详述了引入规则集划分后规则树的建立和规则匹配的流程, 并且对存在的问题进行了分析和解决。最后对活跃规则集的形成和保持作了形式化分析。

5 算法改进实现与性能测试

本章在对 Snort 中函数的调用过程和检测引擎分析的基础上，阐述了基于活跃规则集的 Snort 高效规则匹配方法的实现技术，最后给出了改进系统的性能测试方法以及测试结果。

5.1 Snort 包处理流程

为了修改 Snort 规则匹配方法，需要对 Snort 处理数据包的流程作一个了解，Snort 对包处理的流程如图 5-1 所示。

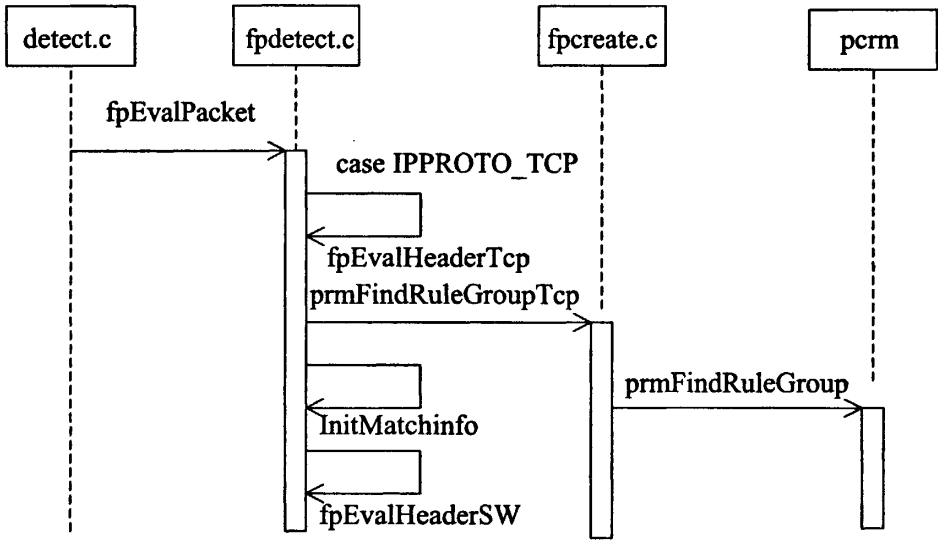


图 5-1 Snort 对包的处理过程
Fig.5-1 The process of Snort processing packet

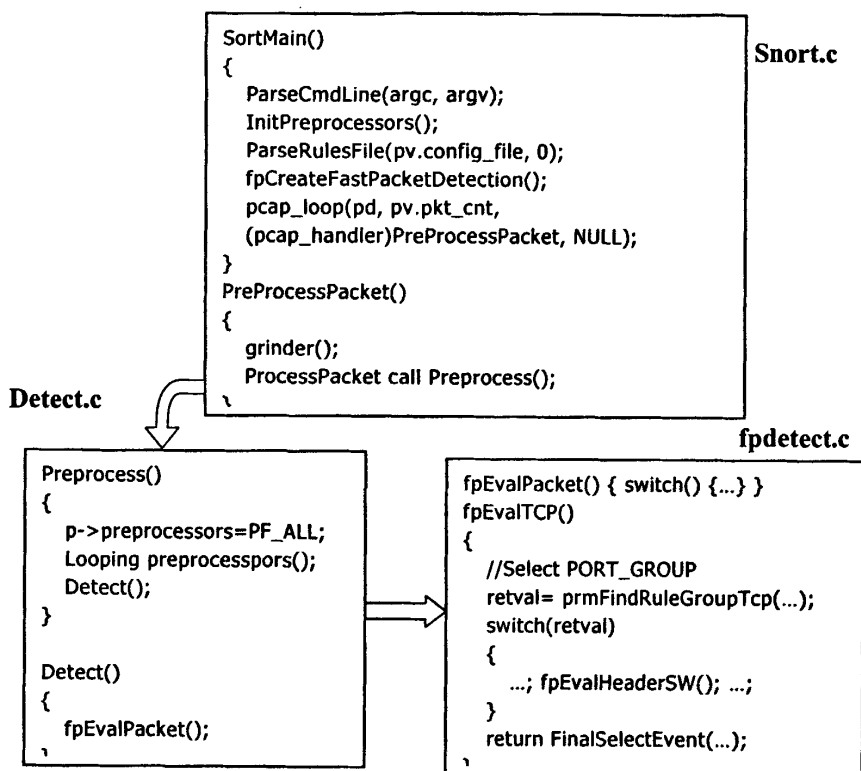
在处理完预处理模块后，调用 Detect 函数对捕获的 packet 结构类型参数的数据包内容进行特征规则匹配。该函数调用 fpEvalPacket 函数，进而根据捕获数据包的协议类型判断 Tcp/Udp/Icmp 协议，若不能规为这三种协议的就算作 Ip 协议，然后调用相应的处理函数。以 Tcp 协议为例，调用 fpEvalHeaderTcp 函数。然后 fpEvalHeaderTcp 调用一系列函数对包的具体类型进行分类。最后调用 fpEvalHeaderSW 将包与规则进行匹配。由上面的包处理流程可知，要改进规则匹配性能就是要对 Snort 规则匹配函数进行修改。

5.2 Snort 部分源代码分析

为了实现对原 Snort 系统的规则匹配部分代码的修改，本节将详细分析 Snort 2.40 系统的部分相关源代码。以便实现在较高版本基础上的规则匹配改进工作。

5.2.1 Snort 2.40 主程序

a. Snort.c 文件



Snort.c 包含了 Snort 系统的所有主控函数。Snort 启动后进行简单的参数检查后调用 SnortMain 函数。

SnortMain 是系统的真正入口。首先它调用 ParseCmdLine 函数对运行参数进行解析，并把这些信息保存在全局变量 pv 中，它包含了协议的校验模式、日志输出方式、运行模式等。InitPreprocessors 等函数完成了插件的初始化工作，所有解码、输入输出插件的初始化工作。随后，ParseRuleFile 函数的主体在 parser.c 中，它完成所有规则配置文件和规则的读取。它调用 ParseRule 和 ParseRuleOptions 分离规则头和规则选项中详细的内容，并把它们组织成规则树，保存在内存中。fpCreateFastPacketDetection 根据规则树完成规则索引的初始化和模式编译工作，这是 Snort 初始化中最为复杂的一步。接着，Snort 调用 InterfaceThread 函数启动抓包和检测过程，并由它调用 libpcap 库的 pcap_loop 函数，回调函数 PreProcessPacket 作为 pcap_loop 函数的参数，完成包的检测工作。

PreProcessPacket 函数完成简单的计数等工作后，调用 ProcessPacket 对抓到的数据进行解码，并保存在全局结构 Packet 中。Packet 是一个重要的全局结构。随后，ProcessPacket 调用 Preprocess 函数，代码转入 detect.c 中。

b. Detect.c 文件

Preprocess 首先调用所有预处理插件，如果在预处理过程中发现攻击，可能会取消后续的检查直接报警。接着，该函数调用 Detect 函数对数据包进行检测，它是真正的数据包检测函数。Detect 函数做的仅仅是调用 fpEvalPacket，进行包头快速检测。如果发现攻击，则返回 1，否则返回 0。所有具体的检查模块都包含在此文件中，在此不再赘述。

c. fpDetect.c 文件

该文件是快速检测引擎（FPDE）的核心。它根据包的类型和端口号，由 prmFindRuleGroup 从索引中查找对应的端口集合和内容集合。找到对应的规则集合后，调用 fpEvalHeaderSW 等模块分别进行头匹配和模式匹配。

5.2.2 快速匹配引擎的创建（FPDE）

快速检测引擎为规则树创建索引时调用 fpCreateFastPacketDetection。创建索引的过程是对规则树的遍历。遍历规则树时以 RTN 为主链、以 OTN 为次链，依次读出每条规则的内容，并根据规则的源端口和目的端口分类，将规则加入对应的内容规则集中。CheckPorts 函数用来提取规则的端口，OtnHasContent 和 OtnHasUriContent 函数分别用来判断规则是否含有 Content 或 UriContent 关键字，如果有，则加入相应的内容集合中，否则加入通用集合中。此后代码转入 pcrn.c（包分类管理器），由它完成具体的规则分类工作，prmAddRule 函数把规则加入对应的索引集合中。

```
int fpCreateFastPacketDetection()
{
    PORT_RULE_MAP prmTcpRTNX = prmNewMap();
    PORT_RULE_MAP prmUdpRTNX = prmNewMap();
    for (rule=RuleLists; rule; rule=rule->next)
    {
        //处理 TCP 规则
        for(rtn = rule->RuleList->TcpList; rtn != NULL; rtn = rtn->right)
        {
            if( OtnHasContent( otn ) ){ /*内容集合*/
                sport = CheckPorts(rtn->hsp, rtn->lsp);
                dport = CheckPorts(rtn->hdp, rtn->ldp);
                for( otn = rtn->down; otn; otn=otn->next )
                { //如果是双向规则，交换源和目的端口并加入集合
                    prmAddRule(prmTcpRTNX, dport, sport, otnx);
                    prmAddRule(prmTcpRTNX, sport, dport, otnx);
                }
            }
            if( OtnHasUriContent( otn ) ){...} /*URI 集合，略*/
        }
    }
    ...//使用相同方式处理 UDP, ICMP, IP, 代码略
}
```

5.2.3 规则的编译

当快速引擎完成规则索引的创建后,调用 BuildMultiPatternGroups 函数编译每个内容集合下的模式。这个函数仅仅完成端口的遍历,而是将具体的编译工作交给 BuildMultiPatGroup 和 BuildMultiPatGroupUri 函数。

```
void BuildMultiPatternGroups( PORT_RULE_MAP * prm )
{
    PORT_GROUP * pg;
    for(i=0;i<MAX_PORTS;i++)    /*遍历端口*/
    {
        pg = prmFindSrcRuleGroup( prm, i ); /*查找源端口*/
        BuildMultiPatGroup( pg );
        BuildMultiPatGroupsUri( pg );
        .../*查找目标端口的代码略*/
    }
}
```

BuildMultiPatGroup 和 BuildMultiPatGroupUri 十分相似,它们均在 fpdetect.c 中定义。该函数首先通过 method 的值确定系统使用的匹配算法。Snort 支持 Modified WuMan (MWM)、BM、Aho-Corasick 等著名的模式匹配算法,在默认情况下使用 MWM 算法。之后,模式的内容传入 mpseAddPattern 函数,该函数负责调用对应的算法。这些算法都位于 SFUtil 目录中,大多算法较为复杂,不再细述。

```
void BuildMultiPatGroup( PORT_GROUP * pg )
{
    OptTreeNode *otn;RuleTreeNode *rtn;OTNX *otnx;
    PatternMatchData *pmd, *pmdmax;RULE_NODE *rnWalk;
    void *mpse_obj;int method;
    method = fpDetect.search_method;    /*匹配方式,默认为mwm*/
    mpse_obj = mpseNew( method ); pg->pgPatData = mpse_obj;
    for(rnWalk=pg->pgHead; rnWalk; rnWalk=rnWalk->rnNext){
        pmd = otn->ds_list[PLUGIN_PATTERN_MATCH];/*得到“与”模式内容*/
        if( pmd && IsPureNotRule( pmd ) ){...}
        else{
            pmdmax = FindLongestPattern( pmd );    /*得到端口中最长模式*/
            if( pmdmax ) mpseAddPattern(mpse_obj,pmdmax->pattern_buf,
                pmdmax->pattern_size,pmdmax->nocase,pmdmax->offset,
                pmdmax->depth,pmx,rnWalk->iRuleNodeID); /*将模式加入集合*/
            /*得到所有“或”模式,代码类似,略*/
            pmd = otn->ds_list[PLUGIN_PATTERN_MATCH_OR];
            while( pmd ){...}
        }
    }
    mpsePrepPatterns( mpse_obj );    /*编译模式*/
}
```

5.2.4 其他相关函数的分析

1. mpseSearch 函数:

Snort 提供三种完全不同的模式匹配算法：Aho-Corasick/Wu-Manber/Boyer-Moore，默认使用第二种。注意 mpseSearch 函数中有两个参数分别是：所要匹配的数据包指针 (Packet*) 和 先前为了快速匹配特征串而建立的对应的结构指针 (PORT_RULE_MAP->PORT_GROUP*)，后面的所调用的函数参数都是从这两个参数中得到。mpseSearch 函数调用 mwmSearch，继续调用 hbm_match 函数。这个函数即是用 Boyer-Moore-Horspool 模式匹配算法进行特征匹配。具体算法实现请参照 mstring.c 文件。

2. fpLogEvent 函数：

若匹配成功，则调用本函数记录或报警。该函数的参数分别是：匹配的 RTN/ONT 指针和数据

包指针(Packet*)，具体实现如下：

```
switch(rtn->type)
{
    case RULE_PASS: PassAction();
    case RULE_ACTIVATE: ActivateAction(p, otn, &otn->event_data);
    case RULE_ALERT: AlertAction(p, otn, &otn->event_data);
    case RULE_DYNAMIC: DynamicAction(p, otn, &otn->event_data);
    case RULE_LOG: LogAction(p, otn, &otn->event_data);
}
```

若是应用 pass 规则动作，则仅设置通过包的数目加一；

若是 dynamic/log 规则动作，则调用 detect.c 文件中的 CallLogFuncs 函数进行记录；

若是 activate/alert 规则动作，则调用 detect.c 文件中的 CallAlertFuncs 函数进行报警。

5.2.5 要修改的函数分析

经过查看资料，发现 CheckANDPatternMatch 为对包的匹配函数，在 CheckANDPatternMatch 函数处设立断点，运行可以得出 Snort 对相关函数的调用过程，具体过程见下图(5-2)。

otnx_match 以前的函数在上文已经介绍。当 CheckANDPatternMatch 对包匹配成功后会返回 1。最后 1 返回至 otnx_match 中。otnx_match 调用 DEBUG_WRAP 增加一个事件。也就是找到了一个成功匹配的包。

结合看 otnx_match 的代码：

```
if( fpEvalRTNSW(otnx->rtn, otnx->otn, omd->p, omd->check_ports) )
{
    omd->pg->pgQEvents++;
    UpdateQEvents();
}
```

```

DEBUG_WRAP(DebugMessage(DEBUG_DETECT,
    "[**] SID %d added to event queue [**]\n",
    otnx->otn->sigInfo.id));

fpAddMatch(omd, otnx, pmd->pattern_size );
}
else
{
    omd->pg->pgNQEvents++;
    UpdateNQEvents();
}

```

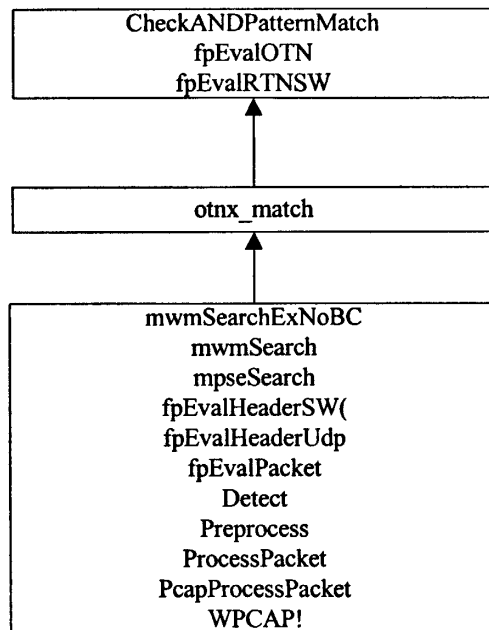


图 5-2 Snort 处理包时对相关函数的调用过程

Fig.5-2 Correlation function calling process of Snort processing packet

可以发现 `otnx_match` 函数是包匹配后要运行的函数。

5.3 算法改进实现

经过以上基础知识的准备，现在就要开始具体的代码修改的实现了。首先，介绍如何搭建编译及调试环境，然后是代码修改的思想，最后是代码修改的实现。

5.3.1 搭建编译环境及调试

可以在 Snort 的官方网站 www.snort.org 上得到源码包。本文使用的 Snort 版本为 2.4。Snort2.4 是直接支持 Win32 平台的。

打开 snort.c, 可以找到系统入口函数 main, 如下图 5-3 所示。

Snort 在 Windows 平台下, 支持 Mysql、MSSQL、Oracle 三种类型的数据库输出插件, 它们是分开编译的。选择 Build/Set Active Configuration 菜单。

选择要编译的平台。如下图 5-4。

调试的时候选择 Debug 版本, 采用的是 MySQL 数据库, 这个时候, 我们就可以进行编译了。经过一阵耐心的等待会生成 Win32\WIN32-Prj\snort__Win32__Debug\snort.exe。

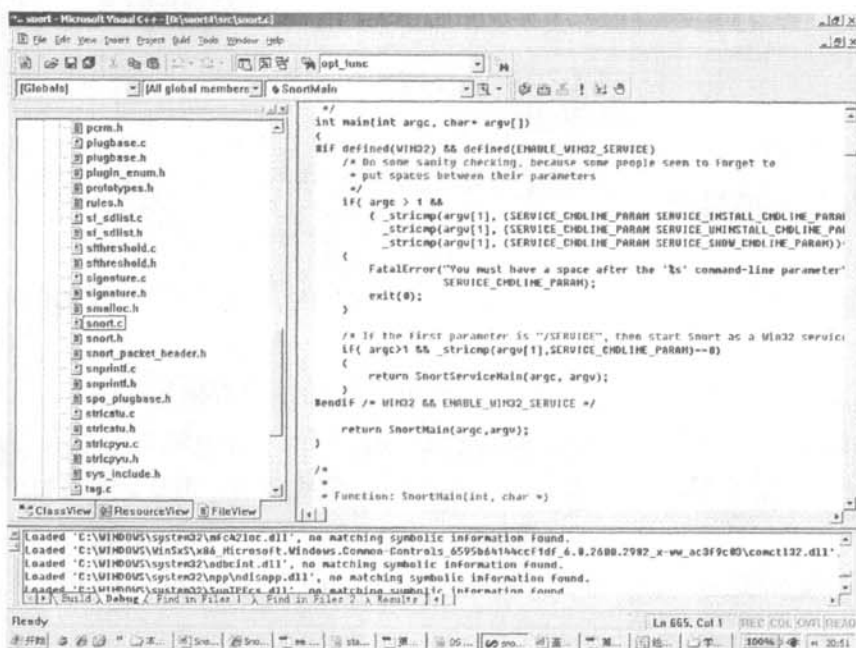


图 5-3 Snort 的 main 函数
Fig.5-3 Main function of Snort

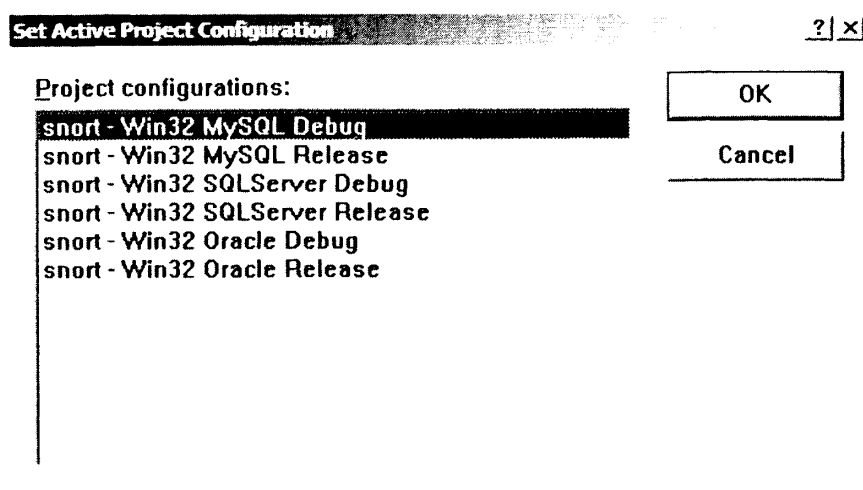


图 5-4 选择 Snort 的编译平台
Fig.5-4 Select the compiler of Snort

5.3.2 代码修改的思想

结合提出的动态集的思想以及 Snort2.4 版本的特点，可以得出以下的程序修改流程图。如图 5-5。

当匹配成功时，有一个 `OptTreeNode *otnstr[999999]`；保存的是所有 `otn` 前面的 `otn` 的指针。当某个 `otn` 没索引的时候，就需要通过遍历 `rtn->down->next` 来进行索引。而且这种索引只用遍历一次，所以建立索引，虽然对内存的消耗会提升，但是会明显加快 `otn` 寻找上面的 `otn` 的速度。可以快速的修改 `otn` 的顺序。

修改完成 `otn` 的顺序后，需要更新索引，这样程序才不会出错。这样就可以动态修改 `otn` 顺序，实现活跃规则集的算法。

5.3.3 具体修改的代码

```
otnTemp=otnstr[otnx->otn->chain_node_number];
if(otnTemp!=NULL&&(otnx->rtn->down!=otnx->otn))//交换位置
{
    otnTemp->next=otnx->otn->next;
    if(otnx->otn->next!=NULL){
otnstr[otnx->otn->next->chain_node_number]=otnTemp;
    }//判断下一个是否为空
```

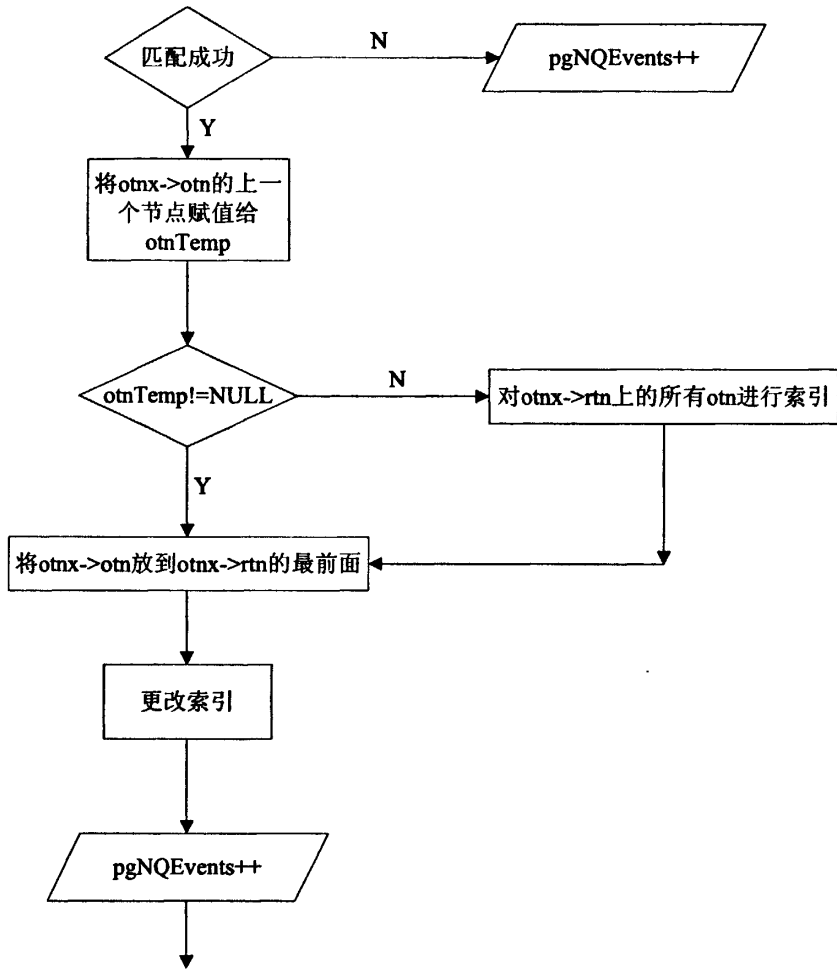


图 5-5 修改 Snort 程序的流程

Fig.5-5 The process of modifying Snort program

```

otnTemp=otnx->rtn->down;
otnx->rtn->down=otnx->otn;
otnx->otn->next=otnTemp;
otnstr[otnx->otn->chain_node_number]=NULL;
    otnstr[otnTemp->next->chain_node_number]=otnx->otn;
}
else
{
    otnTemp=otnx->rtn->down;//第一个 otn 节点
    while(otnTemp->next!=NULL)
    {otnstr[otnTemp->next->chain_node_number]=otnTemp;//将每一个 otn 的上一个保存进
otnstr 中

```

```

otnTemp=otnTemp->next;
}
otnTemp=otnstr[otnx->otn->chain_node_number];
if(otnTemp!=NULL&&otnx->rtn->down!=otnx->otn)//交换位置
{
otnTemp->next=otnx->otn->next;
f(otnx->otn->next!=NULL){
otnstr[otnx->otn->next->chain_node_number]=otnTemp;
} //判断下一个是否为空
otnTemp=otnx->rtn->down;
otnx->rtn->down=otnx->otn;
otnx->otn->next=otnTemp;
otnstr[otnx->otn->chain_node_number]=NULL;
otnstr[otnTemp->next->chain_node_number]=otnx->otn;
}
}

```

在 otnx_match 函数外增加函数声明

```

OptTreeNode * otnTemp=NULL;//临时保存 otn 的变量
OptTreeNode * otnWalk=NULL;//遍历用的
OptTreeNode * otnstr[999999];//索引使用

```

现在就完成了对 Snort2.4 的修改，接下来的任务就是测试了。

5.4 算法改进性能测试

我们的测试环境是 windows 2003 server，测试工具是 vc 的 Profile，测试数据选自 MIT Lincon Lab 提供的 1999 DARPA intrusion Detection Evaluation Data Sets，我们采用其中一天的包含攻击的网络流量作为评估数据集。

5.4.1 测试工具 profile 的使用

Profile 工具可以帮助程序员发现程序运行的瓶颈，找到耗时所在，同时也能帮助程序员发现不会被执行的代码。从而最终实现程序的优化。

Profile 包括 3 个命令行工具：PREP, PROFILE, PLIST。可以以命令行方式运行 Profile，其过程是：PREP 读取应用程序的可执行文件并生成一个.PBI 文件和一个.PBT 文件；PROFILE 根据.PBI 文件，实际运行并分析程序，生成.PBO 输出文件；PREP 再根据.PBO

文件和.PBT 文件, 生成新的.PBT 文件; PLIST 根据.PBT 文件生成可阅读的输出。

选择 Project->Settings->Link, 选择 Enable profiling 复选框, 重建项目在命令行中进入 Snort 的 bin 目录。

1.prep snort, 生成与 snort 对应的.PBI.PBT 文件。

2.profile -i snort.pbi snort -c "d:\snort4\etc\snort.conf"

"d:\snort\bao\outside.tcpdump" -l "D:\snort4\logs" -d -e -X 运行分析程序。

```

15.075 0.1 34.576 0.1 108379 _sixnash_inu_node_row (sixnash.obj)
14.933 0.1 14.933 0.1 110789 _OverlapCompareFunc (spp_stream4.obj)
14.519 0.1 179.384 0.8 43143 _DecodeIP (decode.obj)
14.493 0.1 14.493 0.1 7002 _make_shift (mstring.obj)
14.082 0.1 14.082 0.1 1_pcap_lookupnet (wpcap.dll)
13.506 0.1 13.506 0.1 262144 _prmFindDstRuleGroup (pcrm.obj)
12.870 0.1 30.514 0.1 35768 _ps_detect (portscan.obj)
12.789 0.1 12.789 0.1 640503 _hi_util_in_bounds (hi_client.obj)
12.784 0.1 18.706 0.1 2648 _SegmentCleanTraverse (spp_stream4.obj)
12.720 0.1 77.649 0.3 49292 _FlowPreprocessor (spp_flow.obj)
12.143 0.1 18.498 0.1 3941 _SetFinSent (spp_stream4.obj)
11.990 0.1 13613.254 58.7 43358 _Detect (detect.obj)
11.838 0.1 11.838 0.1 346 _prmGetFirstRule (pcrm.obj)
10.664 0.0 22.959 0.1 13637 _AddOptFuncToList (plugbase.obj)
10.318 0.0 10.318 0.0 1_ProcessPorts (xlink2state.obj)
10.096 0.0 2875.570 12.4 1228 _PrintXrefs (log.obj)
9.936 0.0 11451.881 49.4 32790 _CheckANDPatternMatch (sp_pattern_match.obj)
9.508 0.0 9.508 0.0 524288 _BoRand (spp_bo.obj)

```

图 5-6 Profile 输出的结果

Fig.5-6 Profile output results

3.prep -io snort.pbo -it snort.pbt -ot snort1.pbt 生成 snort1.pbt 文件。

4.plist snort1.pbt >result.txt 将结果以记事本的方式输出。

测试结果如图 5-6。从测试结果上可以看出_Detect 函数在测试中一共被调用 43358 次, 也就是说一共分析了 43358 个数据包, 用时为 13613.254millisecond 则每一个数据包用的时间为 13613.254 除以 43358 等于 0.314ms。

5.4.2 DARPA 与 IDS 测试理论

DARPA (美国国防高级研究计划局) 于 1998 年启动了一项入侵检测系统评估项目。它主要涉及背景流的生成和恶意攻击行为的研究。通过使用在背景流中混杂恶意数据的方式, 可以测试 IDS 的算法和系统设计是否合理。虽然该项目于 1999 年终止, 但它留下了大量可用的测试数据, 成为评估入侵检测系统的重要数据来源, 是迄今为止最权威的 IDS 评估。

DARPA 是第一个系统地测试和评估 IDS 的项目, 其工作是开创性的。DARPA 所关注的不是测试一个完整的 IDS, 而是评估 IDS 时所采用的技术手段。由于从未有人研究过

类似的问题，在项目的开始阶段，没有测试标准可依，也没有用于测试的标准攻击、背景流和已知的理论。

DARPA 采用如下的方式对 IDS 进行评估。它包含两种测试数据：离线训练数据（Training Data）和在线测试数据（Testing Data）。离线数据是时长 7 星期，其中标明了哪些数据包是正常的，哪些数据包是恶意的，它们可以用于训练入侵检测系统，使入侵检测系统完成对网络情况的学习。在线数据时长 2 星期，用于模拟真实网络。在使用这些数据时，使用 tcpreplay 将其重放。

有几种方式可以得到背景流数据。最简单的方式是在真实网络中安装侦听器，收集网络中所有数据。然而由于这些数据中常常包含用户名、密码等隐私信息，公布它们是不妥当的。另一种方式是仔细分析这些数据，把隐私信息从中剔除。这项工作是艰巨的，研究者不但要面对海量的数据，还需要有扎实的专业基础，因为在这些数据中可能混杂着攻击数据。最后一种方式是将都提取并重新合成所有会话。这将避免隐私信息的泄露，还可以自动生成实验所需要的数据。

DARPA 使用最后一种方式获取网络数据，其网络环境主要是 Unix 服务器群，主要的协议有 HTTP、XWindows、SQL、SMTP、POP3、FTP、Telnet 和 DNS 等。大约 120 种攻击工具被分为 38 类加入这些数据中，后来，又在实验网络中加入了一些受害 WindowsNT 服务器。

虽然业界对 DARPA 存有很多批评，但它对 IDS 测试有着极其重要的贡献，也为今后的工作奠定了基础，其测试数据在今天仍然有一定价值。近年来，网络数据的成分已发生重大变化。过去，网络中大约 70% 左右的流量都是 HTTP 协议。现在，P2P 软件（如 BT、Emule、PPLive）的盛行使网络中有大约一半流量流向不知名的端口，高峰时可能达到整个网络流量的 20%-30%。而且更多的攻击工具和攻击手段也层出不穷，使用 1999 年的数据测试现在的 IDS，已是不合时宜的了。但 DARPA 所指出的测试 IDS 的方法，仍然是适用的。

5.4.3 性能改进对比测试结果

为验证修改后的源程序对数据包匹配时间的影响，采用 MIT Lincon Lab 提供的 1999 DARPA intrusion Detection Evaluation Data Sets 中一天的包含攻击的网络流量作为评估数据集，测试环境为 Windows Server 2003，cpu 2.40G Hz，内存 256M。通过读取 tcpdump 格式的入侵检测数据集，以入侵检测模式运行 Snort。对于改进系统的测试和对于原系统的对比实验在同一台机子上进行。利用 windows 和 vc++ 下的 profile 得出实验结果。计算每个数据包的匹配时间，给出对比结果。

进入 bin 目录后按照上面 profile 的使用过程输入。每隔一段时间测试修改过的 Snort，并且和原来的 Snort2.4 进行速度对比。如图 5-7 是 Snort 运行测试时的截图。

对 Snort 的性能对比测试共进行了五组，分别计算出每一个包的检测时间，实验数据如图 5-8 所示。

实验结果表明，在检测到的攻击数据包与原程序相同的情况下，程序修改后的数据包平均匹配时间却比原 Snort 平均匹配时间减少了 0.06195 毫秒。Snort 对每个数据包的平均匹配速度提高了 6%~21%。

```
命令提示符 - snort - c "d:\snort\etc\snort.conf" -r "d:\snort\ba0\outside.tcpdump" -i "D:\snort\ba0"
! Alphabet Size      : 256 Chars
! Size of State      : 2 bytes
! Storage Format      : Full
! Num States         : 141466
! Num Transitions    : 5839883
! State Density      : 16.1%
! Finite Automaton   : DFA
! Memory             : 204.81Mbytes

----- Initialization Complete -----

--> Snort! <*-
o" >~ Version 2.6.1.3-ODBC-MySQL-FlexRESP-WIN32 [DEBUG] <Build 36>
'" ~ By Martin Roesch & The Snort Team: http://www.snort.org/team.html
    ~ (C) Copyright 1998-2006 Sourcefire Inc., et al.

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 1.6 <Build 11>
Preprocessor Object: SF_SSH Version 1.0 <Build 1>
Preprocessor Object: SF_SMIP Version 1.0 <Build 6>
Preprocessor Object: SF_FTPTELNET Version 1.0 <Build 8>
Preprocessor Object: SF_DNS Version 1.0 <Build 1>
Preprocessor Object: SF_DCERPC Version 1.0 <Build 3>

Not Using PCAP_FRAMES
```

图 5-7 测试运行 Snort 时的截图

Fig.5-7 The truncation chart of Snort testing and running

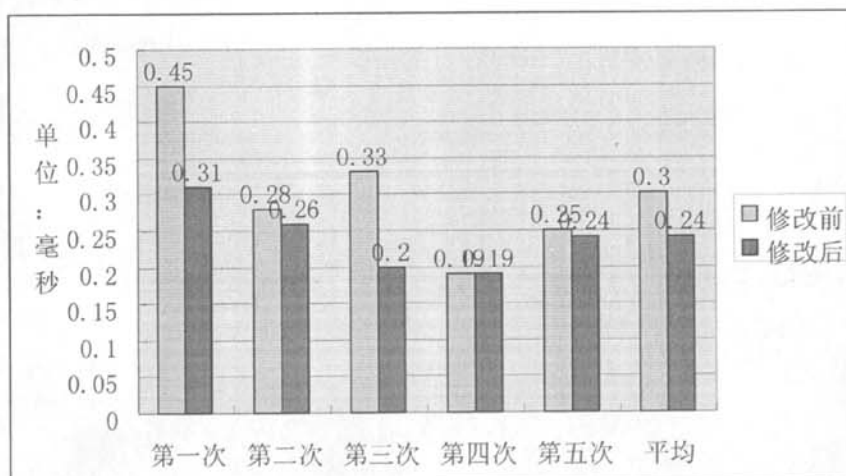


图 5-8 Snort 性能测试对比图

Fig.5-8 The contrast chart of Snort performance testing

5.5 本章小结

本章给出了基于活跃规则集的 Snort 高效规则匹配算法的实现，并采用 MIT Lincon Lab 提供的 1999 DARPA intrusion Detection Evaluation Data Sets 中一天的包含攻击的网络流量作为评估数据集，对改进后的 Snort 做了性能对比测试实验，用每个数据包的平均匹配时间作为 Snort 性能的度量标准，对比测试了五组数据，结果表明应用基于活跃规则集的 Snort 规则匹配方法对 Snort 性能有显著的提升。

6 总结与展望

6.1 总结

Snort作为免费的开源入侵检测工具，不仅仅是一种网络安全工具，作为开源项目，还能给更多的网络安全从业者提供一个学习、动手和研究的平台，这也是Snort能快速发展的重要原因。

Snort采用了可独立编写的插件机制，使程序具有很强的可扩展性，简化了编码工作。相对简单的体系结构、独立的插件机制和灵活的规则集使得开发者很容易对Snort进行修改以便应对新的入侵。但Snort和其他IDS一样，仍然面临着一系列问题，如误报、漏报、自身系统的安全以及处理速度等问题。

本课题首先完成了 Snort 的安装、配置和使用，Snort 运行良好，可以检测出异常的网络数据包，并在 ACID 界面显示报警，对 IDS 和 Snort 有了一个基本的理解。

本论文以 Snort为研究对象，来了解、研究一般的IDS。通过阅读源代码，对Snort的整体实现结构、各个组件及2.0后采用的全新的规则索引和多模式匹配引擎进行了分析。相对于以前的单模式匹配算法，在多模式匹配中，Snort对经过预处理的一组模式集并行地进行模式匹配，使字符匹配性能得到显著提高，从而提高了Snort的检测性能。

本文通过实验对较新的 Snort2.4 版本进行了性能瓶颈分析，讨论了改进 Snort 性能的一系列关键技术，然后在前人工作的基础上提出了基于活跃规则集的 Snort 规则匹配方法，最后通过算法改进实现和性能对比测试，说明了本文提出的改进方法行之有效，达到了预期设想的效果。

本课题工作的不足之处在于，基于活跃规则集的模式匹配算法的实现不够理想，对活跃规则的定义只是动态的调整了 OTN 的匹配顺序，虽然在理论上可以看作是两个集合，但是物理上仍然是一个集合。

下一步能改进的方向是严格按照基于活跃规则集的 Snort 规则匹配方法的思想，构造两个集合，并且修改 OTN 的数据结构，添加是否为活跃的选项并随着时间的推移动态的修改活跃规则集，这样可能更好的提高匹配速度。

6.2 展望

作为Snort核心和基础的规则匹配技术存在着性能瓶颈，高效的算法将是提高Snort总体性能的手段之一。对规则匹配算法，接下来研究的主要方向是提高算法的平均性能和减少资源耗费：

1) 可以根据实际的网络应用，减少需要匹配的规则数量，或减少n或m的值来减少实际匹配的次數；

2) 设法将部分匹配算法移植到硬件的实现中或在并行的体系结构下实现等,以减少匹配所耗费的时间。

Snort整体性能受多方面因素的影响。Snort可通过通讯机制与其他工具协同处理,也可借鉴其他IDS工具中成功的方法来实现新的检测方法,以提高检测速度,减少误报和漏报率,保护计算机系统和网络的整体安全。

目前IDS新的研究方向主要是使用智能化的方法与手段来进行入侵检测,常用的方法有神经网络、遗传算法、模糊技术、免疫原理等,这些方法常用于入侵特征的辨识与泛化。当前较为一致的解决方案是入侵检测系统与具有智能检测功能的检测软件或模块的结合使用。通过与其他先进技术的结合,IDS定能取得更大的发展,为网络安全做出应有的贡献。

网络安全是技术问题,也是管理的问题,只有将网络安全作为一个整体工程来处理,加强网络管理,提高使用者的安全意识,合理使用网络及安全工具,才能达到网络的真正安全。

致谢

日月如梭，转眼已经到了毕业的时节，回忆起两年半研究生学习生活中的点点滴滴，不由得生出许多感激之情来。

首先我要感谢我的导师张亚玲副教授，张老师渊博的知识，敬业的精神，严谨的作风和随和的态度都给我留下了很深的印象。感谢张老师对我一如既往的悉心教导和关怀，您对我的指导和帮助将使我终身受益。同时我也要感谢您不厌其烦地对我的论文加以审阅和指点，正是由于您的辛勤工作，我的论文工作才能更上层楼。最后我还要感谢王尚平老师，感谢王老师高屋建瓴的指点。

我要感谢我的家人给我的支持，他们的支持是我前进的动力，我将会用我的努力来回报家人以及所有关心我的人。

同时，还要感谢计算机学院 104 研究室的所有同学，在平时的学习和生活中，你们的帮助使我受益良多。

再次衷心地感谢母校，恩师和同学！

参考文献

- 【1】 2004 年全国网络安全状况调查报告[R]. 北京: CNCERT/CC, 2005: 4.
- 【2】 杨琼, 杨建华, 王习平, 马斌. 基于防火墙与入侵检测联动技术的系统设计[J]. 武汉理工大学学报, 2005, 27(7): 112-115.
- 【3】 卿斯汉, 蒋建春等. 入侵检测技术研究综述[J]. 通信学报, 2004, 25(7): 19-30.
- 【4】 唐正军, 网络入侵检测系统的设计与实现[M]. 北京: 电子工业出版社, 2002: 30-35.
- 【5】 Snort IDS[EB/OL]. <http://www.snort.org>, 2006-12-15.
- 【6】 Paxson P V. Bro: A System for Detecting Network Intruders in RealTime. In: Proceedings of the 7th USENIX Security Symposium. San Antonio[C], 1999, 2435-2463.
- 【7】 Firestorm IDS[EB/OL]. <http://www.scaramanga.co.uk/firestorm/index.html>, 2006-12-15.
- 【8】 Prelude-IDS[EB/OL]. <http://www.prelude-ids.org>, 2006-12-15.
- 【9】 Knuth D E, Morris J H, Pratt V R. Fast Pattern Matching in Strings[J]. SIAM Journal On Computing, 1977, 6(2): 323-350.
- 【10】 Boyer R S, Moore J S. A fast string searching algorithm[J]. Communications of the ACM, 1977, 20(10): 762-772.
- 【11】 Karp R M, Rabin M O. Efficient randomized pattern-matching algorithms[J]. IBM J. RES. DEVELOP, 1987, 31(2): 249-259.
- 【12】 Aho A V, Corasick M J. Efficient string matching. an aid to bibliographic search[J]. Communication of the ACM, 1975, 18(6): 333-340.
- 【13】 Wu S, Manber U. A fast algorithm for multi-pattern searching[C]. Department of Computer Science, University of Arizona, Report TR-94-17. 1994, 1-13.
- 【14】 Coit C J, Staniford S, McAlerney J. Towards faster pattern matching for intrusion detection or exceeding the speed of snort[C]. In: DARPA Information Survivability Conference and exposition. U.S.A, 2001, 367-373.
- 【15】 Walter B C. A string matching algorithm fast on the average[C]. In: Proceedings of the 6th International Colloquium on Automata, Language and Programming. U.S.A, 1979, 118-132.
- 【16】 Liu R T. A fast string-matching algorithm for network processor-based intrusion detection system[C]. ACM Transactions on Embedded Computing Systems, 2004, 3(3): 614-633.
- 【17】 Anagnostakis K G, Anotonatos S, Markatos E P, et al. E2XB: A domain-specific string matching algorithm for intrusion detection[C]. In: Proceedings of the 18th IFIP International Information Security Conference, Athens, 2003, 217-228.
- 【18】 贺龙涛, 方滨兴, 余翔湛. 一种时间复杂度最优的精确串匹配算法[J]. 软件学报, 2005, 16(5): 676-683.

- 【19】 王永成, 沈洲, 许一震. 改进的多模式匹配算法[J]. 计算机研究与发展, 2002, 39(1): 55-60.
- 【20】 许一震, 王用成, 沈洲. 一种新的多模式字符串匹配算法[J]. 上海交通大学学报, 2002, 36(4): 516-520.
- 【21】 万国根, 秦志光. 改进的 AC-BM 字符串匹配算法[J]. 电子科技大学学报, 2006, 35(4): 531-541.
- 【22】 张邈, 徐辉, 潘爱民. 高效匹配型入侵检测系统[J]. 计算机工程, 2003, 29(19): 104-105.
- 【23】 陈一航, 薛质. 一种针对网络入侵检测系统的字符串匹配算法[J]. 计算机应用与软件, 2005, 22(4): 5-6.
- 【24】 李雪莹, 刘宝旭, 许榕生. 字符串匹配技术研究[J]. 计算机工程, 2004, 30(22): 24-26.
- 【25】 Martin Roesch. Snort-Lightweight Intrusion Detection for Networks[C], Proceedings of the 13th USENIX conference on System administration 1999, 12:7-12.
- 【26】 张晓芬, 牛少彰. 入侵检测系统的标准化[J]. 网络与应用, 2003, 7(7): 18-21.
- 【27】 Denning DE. An Intrusion Detection Model[J]. IEEE Transaction on Software Engineering, 1987, 13(2): 222-232.
- 【28】 Anderson J P. Computer Security Threat Monitoring and Surveillance[R]. Fort Washington, PA: Jame P Anderson Co, 1980.
- 【29】 Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search[J]. Communications of the ACM, 1975, 18(6): 333-343.
- 【30】 蒋建国, 冯登国. 网络入侵检测原理与技术[M]. 国防工业出版社, 2001.7.
- 【31】 Pedro A. Diaz-Gomez, Dean F. Hougen. Analysis of an Off-Line Intrusion Detection System: A Case Study in Multi-Objective Genetic Algorithms[C], FLAIRS Conference 2005, 822-823.
- 【32】 Martin Roesch, Chris Green. Snort Users Manual[M]. Sourcefire Inc, 2003.7.
- 【33】 范华春, 王颖, 杨彬等. 基于网络处理器及协处理器的高速网 IDS 的研究[J]. 计算机工程与应用, 2005(1): 124-126.
- 【34】 Fisk M, Varghese G. Fast Content-based Packet Handling for Intrusion Detection[R]. UCSD Technical Report CS2001-0670, 2001-05.
- 【35】 Norton M, Roelker D. Hi-performance Multi-rule Inspection Engine [EB/OL]. <http://www.snort.org>, 2004-04.
- 【36】 Snort 的高效匹配算法[J]. 计算机工程, 2006(9): 213-215.
- 【37】 史亮, 李斌, 庄镇泉. 基于多主体的分布式入侵检测系统研究与设计[J]. 计算机工程与科学, 2005, 27(2): 5-8.
- 【38】 薛华, 李雪莹, 陈宇等. 利用数据分流实现高速网络入侵检测的研究与实现[J]. 计算机应用研究, 2004(1): 112-114.
- 【39】 褚永刚, 魏战松, 杨义先等. 代理技术在大规模分布式入侵检测系统的应用研究[J]. 计算机工程与应用, 2004, (6): 150-151.
- 【40】 Snort 官方网站. www.snort.org.

- 【41】 Protocol, Anomaly, Detection for Network-based Intrusion Detection[EB/OL].
<http://www.sans.org/rr/intrusion/anomaly.php>,2003-04.
- 【42】 Sun Wu and Udi Manber. A Fast Algorithm for Multi-Pattern Searching[EB/OL].
<http://webglimpse.net/pubs/TR94-17.pdf>,Technical Reports,1994-17.
- 【43】 Ravi S., Sandhu, Pierangela Samarati. Authentication, Access Controls, and Intrusion Detection[J].
The Computer Science and Engineering Handbook 1997: 1929-1948.
- 【44】 Robert Graham. NIDS Pattern Search vs. Protocol Decode[J]. Computers & Security 20(1),2001:
37-41.
- 【45】 S. Cheung, U. Lindqvist, and M. W. Fong.. Modeling multistep cyber attacks for scenario
recognition[C],The 3rd DARPA Information Survivability Conference and Exposition, Apr 2003:
284-292.
- 【46】 S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed
firewall[C],In ACM Conf.on Computer and Communications Security, May 2000: 190-199.
- 【47】 Andres Felipe Arboleda. Snort Development Diagrams [EB/OL].
<http://afrodita.unicauca.edu.co/~cbedon/snort/snortdevdiagrams.pdf>,2005-04-14.
- 【48】 Sushant Sinha,Farnam Jahanian,Jinnesh M.patel.WIND:Workload-AwareIntrusion Detection[C],
Proceedings of Recent Advances in Intrusion Detection(RAID),Hamberg,Germany,2006.
- 【49】 Chistopher Kruegel,Thomas Toth. Using Decision Trees to Improve Signature-Based Intrusion
Detection[C],Proceedings of Recent Advances in Intrusion Detection(RAID),Hamberg, Germany,
2003.
- 【50】 唐谦, 张大方. 基于 Snort 的入侵检测引擎比较分析[J]. 计算机工程与设计, 2005, 26(11):
2884-2856.
- 【51】 任晓峰, 董占球. 提高 Snort 规则匹配速度方法的研究与实现[J]. 计算机应用, 2003, 23(4):
59-61.
- 【52】 陈欢响, 史浩山, 侯蓉晖. 基于反馈信息加速 Snort 规则匹配的研究与实现[J]. 计算机工程与
应用, 2005, 32(3): 133-135.

在校学习期间所发表的论文和参加的项目

1. 张亚玲, 谢少春. 基于活跃规则集的 Snort 高效规则匹配方法. 计算机工程与应用, 2008 年第 10 期.
2. 航天 210 所信息系统开发