第 32 卷 第 18 期 计 算 机 工 程 2006 年 9 月
Vol.32 № 18 Computer Engineering September 2006
·安全技术· 文章编号：1000—3428(2006)18—0155—02 文献标识码：A 中图分类号：TP393

Snort 的高效规则匹配算法

谷晓钢，江荣安，赵铭伟
(大连理工大学电信学院计算机系，大连 116023)

摘 要：对入侵检测系统 Snort 的规则匹配算法进行了系统的分析，为了进一步提高 Snort 的规则匹配效率，提出了在匹配过程中，对于条件匹配处理函数应用参数链表驱动的方法。从而避免重复调用处理函数，充分利用参数之间的关系，并能动态地减少无效规则的匹配。通过两个实验来评估此方法的效率，结果表明改进方案较明显地提高了 Snort 的检测性能。

关键词：基于网络的入侵检测系统；规则匹配；参数驱动

Efficient Rule-matching Algorithms on Snort

GU Xiaogang, JIANG Rongan, ZHAO Mingwei
(Department of Computer, School of Electronic Information Engineering, Dalian University of Technology, Dalian 116023)

【Abstract】This paper systematically analyzes the rule matching algorithm of Snort, an open source-code NIDS. In order to increase effectively the rule matching speed, an approach of parameter-list-driven is proposed for the conditional checking subroutine during rule matching. The means can avoid repeatedly invoking the checking subroutines, can utilize relationship between parameters, and can significantly reduce invalid rules in the running time. Finally, two experiments are done for evaluating the efficiency of it. The result shows the approach can greatly improve the detecting performance of Snort.

【Key words】 NIDS; Rule matching; Parameter-driven

1 Snort 入侵检测系统

Snort[1]作为一种网络入侵检测系统，在共享网络上提取原始的传输数据，分析捕获的数据报，匹配入侵行为的特征或检测异常行为，进而完成入侵的预警或记录。从检测模式而言Snort属于误用检测，针对入侵行为提炼出其特征并编写检测规则，从而形成一个规则库。然后将数据报文按照规则库逐一匹配，若匹配成功则认为是入侵行为。

在 Snort 的实现中，相对耗时的操作主要有 4 个：(1)从网络传输介质上捕获数据包；(2)清除数据结构；(3)规则匹配；(4)对每一个数据包进行校验。其中，规则匹配是 Snort 核心处理模块，若能提高其处理速度，将有效提高 Snort 整体性能。

2 Snort 规则匹配算法分析

Snort[2]将所有已知的攻击以规则的形式存放在规则库中，每一条规则由规则头和规则选项两部分组成。规则头对应于规则树结点RTN(Rule Tree Node)，包含动作、协议、源(目的)地址和源(目的)端口及数据流向这样一些公共信息，Snort把这些具有相同条件的规则链接到一个集合中，用RTN结构来描述；规则选项对应于规则选项结点OTN(Optional Tree Node)，包含一些特定的检测标志、报警信息、匹配内容等条件，每个选项的匹配子函数(插件)放到FUNC链表中。只有当规则的各个条件都为真时才触发相应的操作。综合起来考虑，组成规则的各元素是“逻辑与”的关系；同时，规则库的各条规则为一个大的“逻辑或”关系。Snort解析规则时，分别生成TCP、UDP、ICMP和IP这 4 个不同的规则树，每一个规则树包含独立的三维链表：RTN(规则头)，OTN(规则选项)和FUNC(指向匹配子函数的指针)。

当 Snort 捕获一个数据报时，首先对其解码，然后进行预处理，再利用规则树对数据报进行匹配。如图 1 所示，在规则树匹配过程中：根据该数据报的 IP 协议决定与哪个规则树进行匹配；然后与 RTN 结点依次进行匹配，当与某个规则头相匹配时，接着向下与 OTN 结点进行匹配。每个 OTN 结点都包含了一条规则的全部选项，它包含的一组函数指针就是用来实现对这些条件的匹配操作。当检测得知数据报与某个 OTN 结点的所有条件相符合时，即判断此数据报为攻击报文。

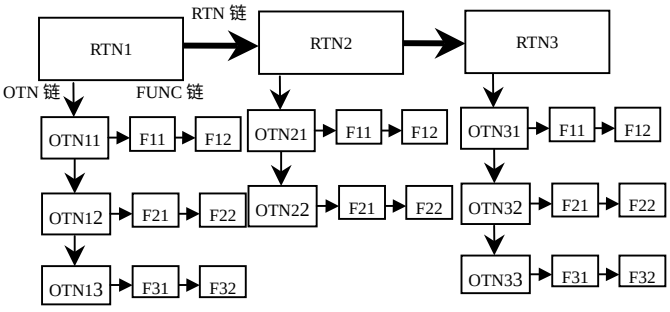

图 1 Snort 的规则树

为提高规则匹配的速度，Snort 采用了 Boyer- Moore 字符串匹配算法、二维列表递归检索(RTN 和 OTN)以及函数指针列表(称为“三维列表”)等方法。在保持这些方法的基础上，本文试图将匹配策略并行化，对于各条件(规则头中的和规则选项中的)既按照规则本身进行分类，也按照条件分类；同时引入横向淘汰方法，如果一个规则中的一个条件为假，

作者简介：谷晓钢(1975—)，男，硕士生，主研方向：网络安全；江荣安，副教授；赵铭伟，博士生
收稿日期：2005-12-06 **E-mail：**mwzhao@dl.cn

那么就不匹配此规则中的其他条件，从而能有效地减少数据包的匹配时间。

3 改善 Snort 规则匹配效率的方法

3.1 思想

下面是规则文件 icmp.rules 中的 2 条规则。

（1）alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP L3retriever Ping";icode:0;itype:8;content:"ABCDEFGHIJKLMNOPQRSTUVWABCDEFGHI";depth:32; reference: arachnids,311;classtype:attempted-recon;sid:466; rev:4;)

（2）alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP redirect net";icode:0;itype:5;reference:arachnids,199; reference:cve,1999-0265; classtype:bad-unknown;sid:473; rev:4;)

以这 2 条规则为例，分析一下 Snort 规则的一些特点：

(1)规则由规则头和若干选项条件构成，规则头和选项条件都对应着各自的匹配子函数，这虽然有利于程序的扩展，但从执行效率的角度看却是较差的。因为整个规则匹配过程中需要大量调用重复匹配子函数，而调用它们需要消耗额外的系统时间，但实际上只有 26 种规则选项匹配子函数和 2 种规则头匹配子函数。

(2)对于规则头和规则选项的处理，Snort 都要执行相应的判断函数，来决定二者是否匹配报文中的条件。但是在有些情况下，比如这里的规则(1)、规则(2)都有“ icode ”、“ itype ”条件，如果按照 Snort 现在的处理方法，先处理规则(1)中“ icode ”、“ itype ”条件，然后处理规则(2)中“ icode ”、“ itype ”条件，显然这 2 条规则中条件处理结果不能重用。

由此，根据以上列出的 2 个问题引发出相应的改进措施：

(1)对于规则头和规则选项中的条件参数，打破只按照规则归类的方法，而是既按照规则归类也按照匹配子函数归类的方法。这样在每个匹配子函数中都附有一个不同规则相同条件对应的参数集合。在进行规则匹配时先按照匹配子函数的顺序，然后在每个函数内只是针对不同规则中的条件参数进行处理，而不是一遍又一遍地调用相同的函数，这样可以节省重复调用函数的系统时间。

(2)每个条件除跟匹配子函数关联外，也同规则有联系，这里有 2 种情况要考虑：

1) 第 1 次遇到某个条件且值为真，在以后遇到其他规则的相同条件时，可以直接引用以前的处理结果。在对规则头的匹配中，比如规则(1)的源地址是$EXTERNAL_NET(外部地址)，目的地址是$HOME_NET(内部地址)，可以把地址分成几类，比如$EXTERNAL_NET，$HOME_NET 及 any(任意地址)等。如果检测得知报头的源地址是$EXTERNAL_NET，那么同为$EXTERNAL_NET 的其他 RTN，对于源地址就不需判断了，甚至$HOME_NET 和 any 类里的 RTN 也不用判断，因为源地址肯定不是$HOME_NET 并且肯定是 any，这样就可以减少不必要的比较了。

2)第 1 次遇到某个条件且值为假，在以后匹配时，根本不处理此规则的其他条件。Snort 的规则和规则之间是或关系，规则内条件与条件之间是与关系。当某一条件值为假时，就可以通知其他处理函数不用处理此规则的其它条件了。

本文正是按照以上思路提出改进规则匹配效率的方案，下面给出实现的具体描述。

3.2 具体描述

在匹配方式上可分为规则头匹配和规则选项匹配。在工作流程中可分为前期准备和匹配过程两个阶段。下面按以上的分类方法分别阐述。

3.2.1 前期准备

规则头链表如图 2 所示，在规则头链表基础上，把各自的条件参数组成新的自上而下的链表构建十字链表结构。取消原先在 RTN 中规则头函数链表，把它们上移到 ListHead 中。参数链包括：地址参数链和端口参数链，对于地址链就可以如上所述，分为几个类，处理函数只处理某类中第 1 个参数，其他参数都引用这个参数的处理结果。

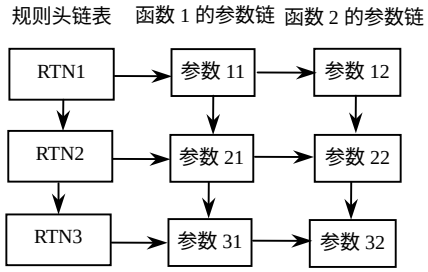

图 2　改进方案的规则头 RTN 链表结构

规则选项链表如图 3 所示，在选项条件按规则分类的基础上，再按照插件进行分类构建十字链表结构。首先在 OTN 结构中增加一个指针，用于指向本规则对应的选项参数链表，具体的参数结构除有参数内容外还要有横向指针(指向本规则中的下一项参数)和竖向指针(指向本插件的下一个参数)，当遇到一个选项关键字时，找到对应插件后，构建此条件的参数结构并同时放入到按规则分类的横向参数列表中和按照插件分类的竖向参数列表中。最后对于同一个匹配子函数下的参数链表可以按照某种方式进行分类或排序。

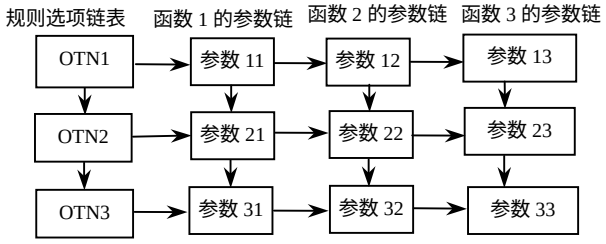

图 3 改进方案的规则选项 OTN 链表结构

3.2.2 规则匹配过程

规则头链表：在找到规则树后，首先按照规则头条件的顺序依次处理(先地址，后端口)，然后在具体的处理函数中对参数列表中的每个参数进行处理，对于具有相同分类的参数，只引用上次的处理结果就可以了。若结果为真，继续下一个参数，否则，则通知其他处理函数：不用处理此规则头的其他条件。执行 2 个匹配函数后，就找到了特定的 RTN。

规则选项链表：首先按照插件的顺序依次处理每个插件，然后在具体的插件中对参数列表中的每个参数进行处理。若结果为真，继续下一个参数，否则，则通知其他插件：不用处理此规则的其他条件了。对于规则中同时含有重型选项条件(需较多检测时间)和轻型选项条件(需较少的检测时间)，一般把轻型插件放到重型之前处理，若轻型插件检测出此条件为假，后面的重型条件就没有必要处理，显然这样会减少规则匹配的时间。

4 结果比较

实验是在 Linux 环境下进行的，通过构建规则树和规则匹配两个过程，分别考察 Snort 和改进方案所消耗的时间，以此来对比二者之间的效率。

首先是构建规则树的时间对比，这里选取了 icmp-info 规则文件，共有 93 条规则。

(下转第 213 页)