ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.8, August 2009, pp.2214−2226
doi: 10.3724/SP.J.1001.2009.03311

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

深度包检测中一种高效的正则表达式压缩算法*

徐 乾[1,2+], 鄂跃鹏[1,2], 葛敬国[1], 钱华林[1]

[1](中国科学院 计算机网络信息中心,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Efficient Regular Expression Compression Algorithm for Deep Packet Inspection

XU Qian[1,2+], E Yue-Peng[1,2], GE Jing-Guo[1], QIAN Hua-Lin[1]

[1](Computer Network Information Center, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: xuqian@cnic.cn, http://www.cnic.cn

Xu Q, E YP, Ge JG, Qian HL. Efficient regular expression compression algorithm for deep packet inspection. *Journal of Software*, 2009,20(8):2214−2226. http://www.jos.org.cn/1000-9825/3311.htm

Abstract: A memory efficient regular expression compression algorithm is devised for deep packet inspection. First, a parameter DR (distending rate) is defined to quantify the explosive quality of regular expressions. Then based on DR, a regular expression cutting algorithm is proposed to downsize the storage needs of individual regular expression, by detecting and confining the parts which cause DFA (deterministic finite automaton) states' exponential growth. Then according to the relation of different regular expressions, a selective grouping algorithm is introduced for regular expression sets, which could cut down the number of finite automata, and reduce the runtime memory consumption.

Key words: regular expression; DFA (deterministic finite automaton); deep packet inspection; multi-pattern matching algorithm; intrusion detection

摘 要: 提出一种基于确定的有穷状态自动机(deterministic finite automaton,简称 DFA)的正则表达式压缩算法.首先,定义了膨胀率 DR(distending rate)来描述正则表达式的膨胀特性.然后基于 DR 提出一种分片的算法 RECCADR (regular expressions cut and combine algorithm based on DR),有效地选择出导致 DFA 状态膨胀的片段并隔离,降低了单个正则表达式存储需求.同时,基于正则表达式的组合关系提出一种选择性分群算法 REGADR(regular expressions group algorithm based on DR),在可以接受的存储需求总量下,通过选择性分群大幅度减少了状态机的个数,有效地降低了匹配算法的复杂性.

关键词: 正则表达式;确定的有穷状态自动机(deterministic finite automaton,简称 DFA);深度包检测(deep packet inspection,简称 DPI);多模式匹配算法;入侵检测

中图法分类号: TP393 **文献标识码**: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z214 (国家高技术研究发展计划(863)); the Youth Foundation of the Computer Network Information Center (CNIC), the Chinese Academy of Sciences under Grant No.0714071101 (CNIC 青年基金)

Received 2007-10-17; Accepted 2008-03-14

现代的深度包检测系统往往需要对数据包的内容进行识别、分析和分类.以往的系统采用基于字符串的多模式匹配算法[1],如 Aho-Corasick 算法[2];随着被检测的内容日渐复杂,基于正则表达式的多模式匹配算法逐渐代替了基于字符串的匹配算法.正则表达式的实现方式有两种,一种是非确定的有穷状态自动机(nondeterministic finite automaton,简称 NFA),另一种是确定的有穷状态自动机(deterministic finite automaton,简称 DFA).为了尽可能地减少匹配过程的访存次数,多数基于 DFA 的方法都使用了二维表的存储结构.利用二维表的直接寻址特性,通过一次访存就可以找到当前状态的下一跳.DFA 的存储代价取决于 DFA 的边数和状态数,任意状态有至多 2^8(ASCII 的字符集)条出边,因此,本文重点研究如何减少 DFA 的状态数.

本文基于 Snort 正则表达式集合,分析造成 DFA 状态数膨胀的正则表达式结构特点,并提出了针对这些问题的优化算法.主要贡献有:

- 分析了造成 DFA 状态数膨胀的正则表达式的类型,提出一种量化 DFA 状态数膨胀的参数——正则表达式的膨胀率 DR(distending rate),并将这个参数应用到之后的算法中.
- 提出了一种正则表达式分片的算法(regular expressions cut and combine algorithm based on DR,简称 RECCADR),将正则表达式分成了头部、中部、尾部 3 个部分,其中,头部和尾部是造成正则表达式的 DFA 状态数膨胀较轻的部分,中部是膨胀严重的部分.这样分割正则表达式可以显著降低 DFA 的状态数,从而降低 DFA 的存储空间.
- 提出了一种正则表达式集合的分群算法(regular expressions group algorithm based on DR,简称 REGADR),基于正则表达式的膨胀率 DR 将正则表达式集合有选择地分成 DFA 状态数相近的群,降低了正则表达式匹配算法的空间复杂性.

针对上面所述的 3 个方面,本文进行了大量实验,结果表明,在不同情况下,RECCADR 算法都有效地隔离了正则表达式中容易引起 DFA 状态数膨胀的部分,存储空间的消耗降低了 80%以上;REGADR 算法可以有效地将集合中的正则表达式分成几个群,分群的个数远远小于正则表达式的个数.与其他分群算法相比,本文提出的分群算法可使存储空间进一步降低 30%左右,分群的个数也有很大的改善.

本文第 1 节描述基于正则表达式的多模式匹配算法的主要背景.第 2 节描述所研究正则表达式集合的特点,分析造成 DFA 状态数膨胀的正则表达式的主要类型,描述膨胀率 DR 的定义形式.第 3 节、第 4 节描述 RECCADR 算法和 REGADR 算法.第 5 节给出实验结果,说明算法的优越性.第 6 节是相关总结和下一步的工作.

1 相关研究背景

Sommer 和 Paxson[3]认为,用正则表达式描述网络中的攻击行为比用字符串描述更为高效、灵活.开源的入侵检测系统 Snort[4]和 Bro[5]均采用正则表达式来描述攻击类型,很多商用系统如 3Com 公司的 Tippingpoint X505 和 Cisco 公司[6]的很多安全产品也使用了正则表达式描述集.除此之外,正则表达式作为模式描述语言也有其他用途,开源项目 Linux Layer-7 filter[7]使用了大约 70 条正则表达式用来描述服务类型.尽管这些深度包检测系统大多声称可以达到线速处理,但由于受到正则表达式当前的匹配算法的限制,处理的速度远没有达到 Gbit 级.Yu[8]提到,当 Linux Layer-7 filter 完全启用时,链路的速度低于 10Mbps,且超过 90%的 CPU 时间被正则表达式匹配算法占用.Johnson[9]提出了在数据包乱序的情况下,基于正则表达式的匹配算法来改善系统的整体特性.

当前的多数正则表达式匹配引擎[10],如 PCRE[11],采用 NFA 实现正则表达式.Cox[12]比较了两种 NFA 匹配算法,证明了要么存在指数增长的匹配路径,导致匹配算法的计算复杂度很高;要么存在多个同时匹配状态,最坏情况下,NFA 的所有状态都会激活,匹配算法的空间复杂度很高.Sidhu[13]和 Clark[14]提出用硬件实现 NFA 来加速匹配算法,弥补 NFA 计算复杂度高的缺点.但模式集合往往十分庞大而且在不断增加,这种方法将会消耗大量的硬件逻辑结构,导致系统难以控制.

另一方面,使用 DFA 的正则表达式匹配算法每消耗一个字符只有一个确定的状态迁移,匹配算法的时间复杂度与目标字符串长度成正比,空间复杂度是常量.但是,复杂的正则表达式结构可能引起 DFA 的状态数膨胀,

导致存储代价巨大,在 Cisco 的安全产品中已达到 GB 级别的存储代价,且在不断增加[6,15],传统的二维表压缩算法对 DFA 状态转移表几乎没有效果.Kumar 提出了带有默认前移的 DFA 表示法(D^2FA)[6,15],通过合并状态之间重复的转移边来降低存储消耗.文献[6,15]证明了可以合并 DFA 中的大量转移边(超过 90%),但未能提出一种合理的表存储方法.原有的状态转移表是整齐的二维表,可以直接寻址找到对应字符的下一跳.如果状态转移表是不整齐的二维表,将会大量增加访存次数.Yu 等人[8]针对 Snort 规则集提出两条改写规则,把 DFA 状态数存在指数膨胀的正则表达式改写成 DFA 状态数是线性增长的形式,但适用性有限;而且他们忽略了集合中其他 DFA 也膨胀的正则表达式,未能提出一种普遍适用的解决方法.Lunteren[16]提出将 DFA 的状态转移分成不同的优先级,大幅度合并 DFA 的状态转移边.

最后,由于正则表达式集合数量巨大,当前的很多系统将多个正则表达式组合成一个 FA,实现一次匹配多个正则表达式.Hopcroft 等人[17]证明了组合 NFA 的状态数与原来每个 NFA 状态数的加和水平相当,而组合 DFA 则会产生状态数的膨胀.Yu[8]提出了一种选择性分群算法并给出实验结果,但算法忽略了正则表达式本身的 DFA 性质,算法仍然存在可改进之处.

2 正则表达式和DFA状态膨胀率DR

Hopcroft 等人[17]证明,任意一个正则表达式都可以通过形式化的方法构造出 DFA 和 NFA.NFA 的状态数与正则表达式的长度线性相关.在构造 NFA 的过程中,正则表达式中的每一个字符(属于字符集)、元字符和操作符,都对应着一定的 NFA 状态数.但随着正则表达式自身结构的复杂度不同,DFA 状态数会发生不同的变化,最坏的情况下,DFA 的状态数发生指数级膨胀.图 1 所示 4 个正则表达式 *RE*1,*RE*2,*RE*3 和 *RE*4 都具有相同的 NFA 表示形式,但产生了形态完全不同的 DFA,状态数分别是 4,5,7,8 个.其中,*RE*2,*RE*3 和 *RE*4 都有计数结构,它们的 DFA 状态数和正则表达式的计数结构计数值存在着不同的量级关系.

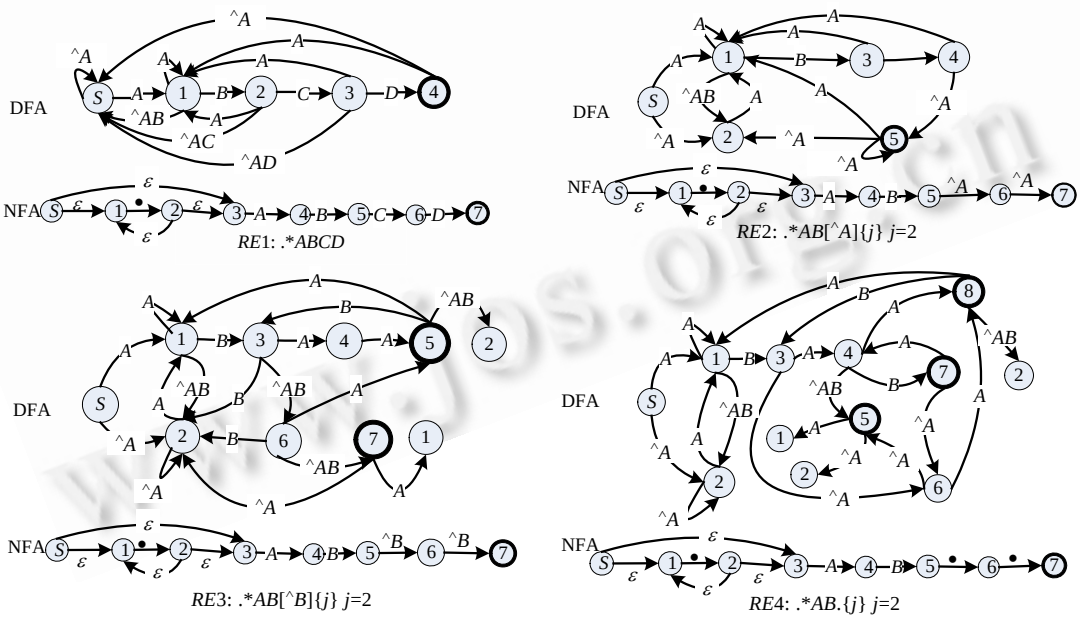

Fig.1 DFA and NFA of four regular expressions *RE*1, *RE*2, *RE*3, *RE*4

图 1 4 个正则表达式 *RE*1,*RE*2,*RE*3,*RE*4 的 DFA 和 NFA

上述分析说明,正则表达式的 DFA 状态数和正则表达式自身的结构紧密联系,具体可以归纳为:正则表达式可识别字符串集合越大,DFA 的状态数有可能越多;目标字符串可以对应于正则表达式中的不同部分,产生歧义

匹配, DFA 利用不同的状态来区分歧义,从而状态数增多.图 1 中,*RE*2,*RE*3,*RE*4 的识别能力依次变强且都存在歧义匹配,其 DFA 状态数依次增多(图中的 FA 采用参考文献[5]中的形式化方法生成,手工绘制的 FA 可能不同).

DFA 的状态数还与正则表达式的长度相关,如果用 NFA 的状态数表示正则表达式的实际长度(正则表达式由元字符组成,元字符或是普通元素的集合,或是语法修饰结构,修饰其他普通字符.所以,正则表达式的长度不能用其本身的长度表示),那么也可以认为正则表达式的实际长度增大,DFA 的状态数也会增多.

图 2 中显示了 6 个同类型正则表达式计数结构的计数值与 DFA 状态数的关系.虽然它们的形式几乎一致,但 DFA 状态数的变化却不相同,说明正则表达式自身的结构也影响 DFA 的状态数.因此,本文定义正则表达式的膨胀率 DR 来描述正则表达式的膨胀特性,其定义形式如下:

$$DR = \frac{\#(DFA) - \#(NFA)}{\#(NFA)} \tag{1}$$

#()表示 FA 的状态数.公式(1)的含义是:NFA 的状态数表示了正则表达式本身的实际长度,DFA 的状态数减去 NFA 的状态数表示了 DFA 膨胀的部分,DR 表示了 DFA 膨胀的相对量.

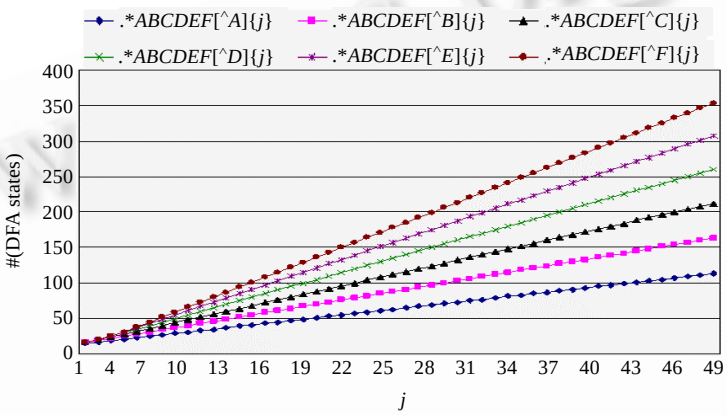

Fig.2 Relations of DFA and length restrictions for the same type regular expressions

图 2 相同类型正则表达式和 DFA 状态数的关系

3 基于正则表达式膨胀率的分片重组算法——RECCADR

3.1 Snort规则集的特点

利用膨胀率 DR 可将 Snort 规则集中 $DR \geq 1$ 的正则表达式近似地分成几类,见表 1.第 1 类和第 5 类的 DFA 状态数在实际计算时已经溢出,没有确定的数值.基于有限状态自动机的匹配算法无法记忆已经匹配的字符数,所以正则表达式的计数结构会导致 FA 的状态增长,其中不存在歧义匹配的正则表达式 DFA 状态数和计数结构的计数值成线性增长关系,而存在歧义匹配的正则表达式 DFA 的状态数和计数结构的计数值则成更高阶的增长关系.

表 1 中前 4 类正则表达式都包含计数结构,随着计数值的变化,DFA 状态数都有不同程度的增长.其中:第 1 类呈指数增长;第 3 类和第 5 类呈平方数增长;第 2 类和第 4 类呈线性增长,但这两类正则表达式的实例 DR 分别为 12.475 和 8.349,也存在较严重的 DFA 状态数膨胀.因而,即使 DFA 状态数和计数结构的计数值成线性关系,但由于正则表达式其他部分导致增长的系数很大,最终也会产生严重的 DFA 状态数膨胀.闭包运算一般不会导致 DFA 状态数膨胀,但当正则表达式包含多个闭包运算存在歧义匹配时,也会存在 DFA 状态数的膨胀.第 6 类正则表达式包含大字符集的闭包,存在较严重的 DFA 状态数膨胀.因而可知:

第一,正则表达式如果包含字符集的闭包和计数,尤其是大字符集的这类运算常会导致 DFA 状态数的膨胀;

第二,仅用 DFA 状态数与正则表达式计数结构计数值的量级关系来衡量膨胀程度是不完善的,低量级的正

则表达式(表 1 中第 2 类、第 4 类)也可能存在严重的 DFA 状态数膨胀.

Table 1 Snort rules set regular expressions

表 1 Snort规则集正则表达式**

	Description	Formulization	Instance	DR	DFA and length restriction
1	Big class ([]) covers prefix and length restriction	.*AB.{j}	.*contenttype=[^\r\n\x3b\x38]{100}	>9 999	$O(2^j)$
2	Big class overlaps prefix and length restriction	.*AB[^B]{j}	.*\x28\s*name\s*\x22[^\x22]{260,}	12.452	$O(j)$ prefix decides step
3	With ^, big class with length restriction covers small class closure	^AB+.{j}	^TEST\s+[^\n]{100,}	43.585	$O(j^2)$
4	With ^, big class with length restriction overlaps small class closure	^AB+[^A]{j}	.*cache_lastpostdate\[[^\]]+\]=[^\x00\x3B\x3D]{30}	8.349	$O(j)$ class decides step
5	Multiple big classes overlaps prefix and length restriction with link OR(|)	.*A(B[^B]{j}|A[^A]{j})	.*\(\s*(\x27[^\x27]{1024,}|\x22[^\x22]{1024,})	>9 999	$O(j^2)$
6	Multiple class closure	.*A.*B.*C	.*\x2Fcgi\x2Flogurl\.cgi.*User-Agent\x3A[^\r\n]*MyPost.*form-data\x3B\x20name=\x22pid\x22.*internal	2.393	N/A

3.2 RECCADR算法

若把闭包(*,+)、或(|)计数结构({})、集合([])及其修饰的结构看成整体,则正则表达式可以被看成由若干个不同类型的片依次连接而成.片的类型分为以下两种:一种是闭包(*,+)、或(|)计数结构({})、集合([])及其修饰的结构组成的片段,定义为 ET1(element type 1)片;另一种是正则表达式中非 ET1 类型的片,定义为 ET2 片(element type 2).

表 2 是表 1 中的正则表达式实例经过分片的结果(【】是分隔符,包含的是 ET1 片,其余的是 ET2 片).可以看出,ET2 片与普通字符串相似,通常不会引起 DFA 的状态数的膨胀;而 ET1 片是大字符集的闭包或者计数结构,可能导致 DFA 状态膨胀.如果可以隔离导致 DFA 的状态数膨胀的部分,消除正则表达式中造成歧义匹配的因素,则可以显著降低 DFA 的状态数.

Table 2 Cut result of regular expression

表 2 正则表达式的分片结果

	Regular expression
1	【.*】contenttype=【[^\r\n\x3b\x38]{100}】
2	【.*】\x28\s*name\s*\x22【[^\x22]{260,}】
3	^TEST\s+【[^\n]{100,}】
4	【.*】cache_lastpostdate\[【[^\]]+】\]=【[^\x00\x3B\x3D]{30}】
5	【.*】\(\s*【(\x27[^\x27]{1024,}|\x22[^\x22]{1024,})】
6	【.*】\x2Fcgi\x2Flogurl\.cgi【.*】User-Agent\x3A【[^\r\n]*】MyPost【.*】form-data\x3B\x20name=\x22pid\x22【.*】internal

理论上可以将正则表达式中ET2类型的片全部分割出来,但这样做导致匹配算法复杂,性能严重下降.因此,本文采用一个折衷的方案.选择造成 DFA 状态数膨胀最严重的 ET2 片作为分隔点,定义为中部(middle);位于该片之前的所有片连接起来作为一部分,定义为头部(head);位于该片之后的所有片连接起来定义为尾部(tail).图 3 描述了这一过程.其中,中部的选择十分重要,但选择最优的 ET1 片时间复杂性高.所以,下文设计了一个寻找较优解的算法.

** 在 Snort 规则集中的正则表达式存在这样一个问题,按照惯例,转义字符\s 代表的含义[17]是[\x20\t\r\n\f],但是在 Snort 规则集中却存在模糊现象:有的地方认为\s 包括\r,有的地方却认为不包括.这里的这个例子是\s 不包括\r,在此说明.

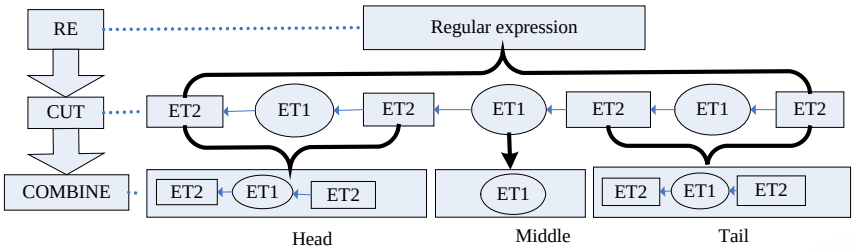

Fig.3 Process of regular expressions cut and combine

图 3 正则表达式分片和组合过程图

(1) 算法的形式化描述

for (集合中的每一个正则表达式) {

计算正则表达式的 NFA 状态数#(*NFA*),DFA 状态数#(*DFA*),膨胀率 DR_i;

if ($DR_i>\sigma$) {

按照特征将正则表达式切成小片并存放在数组 *REArray*;

for (*REArray* 中最后一个元素 to *REArray* 第 1 个元素) {

if (是 ET1 类型的片) {

Head=*REArray* 中位于当前片之前的元素依次连接,*Tail*=*REArray* 中位于当前片之后的元素依次连接组成尾部,*Middle*=当前片;

计算出三者的膨胀率 *DR*(*Head*),*DR*(*Middle*),*DR*(*Tail*);

if (*DR*(*Head*)<δ,*DR*(*Middle*)<δ,*DR*(*Tail*)<δ) {停止计算,记录下切片结果;}

判断当前膨胀率是否比记录的头部、中部和尾部最小值都小,记录最小值和切片位置;

}

}

if (没有切片成功) {在记录的最小值处切片;}

}

}

(2) 算法的分析

算法中有两个参数σ和δ:对 $DR\geq\sigma$的正则表达式进行分片;在分片的过程中,当正则表达式的头部、中部、尾部都小于δ时,取当前的组合作为分片结果返回,否则继续试探.实际操作时,δ可为 3 个参数,即头部、中部和尾部分别取不同上限值.

RECCADR 算法是启发式的算法,依次试探正则表达式中不同的 ET2 类型片作为中部时 DR 的变化情况,当头部和尾部的 DR 出现大幅降低的时候,说明已经找到了正则表达式引起歧义匹配的部分.一个 DR 很大的正则表达式通常都包含引起歧义匹配的 ET2 类型片,从这样的 ET2 类型片处切片后,正则表达式其余部分的 DR 一般会出现大幅度变化.RECCADR 算法通过门限值σ有效地选择出了正则表达式中这样的 ET2 片,所以,对于 DR 异常的正则表达式来说,RECCADR 算法的切片方案是有效的,后文的实验数据也说明了这一点.算法的复杂度是 *O*(*L*),*L* 是正则表达式的分片个数.

经过分片,正则表达式匹配算法也要有相应的变化,以保证不影响原有语义.若匹配算法依次进行头部、中部和尾部的匹配,则时间复杂性高;若同时进行头部、中部和尾部的匹配,则空间复杂性高.正则表达式的中部往往是大字符集的闭包或者计数(.[100],.*),其特点是可接受的目标字符串集合大,如果直接用中部匹配目标字符串,会产生大量的成功匹配,算法性能严重恶化;而头部和尾部两个部分占据了正则表达式的绝大部分,相对于中部语义更确定,产生的成功匹配比中部少得多.因此,匹配算法可以在第 1 阶段同时对目标字符串进行头部和尾部的匹配,在头部和尾部匹配成功的时候进入第 2 阶段,匹配中部,充分利用头部和尾部匹配更确定的特点跳

过大多数目标字符串,达到加速匹配的目的.本文限于篇幅,不深入论述.

4 基于正则表达式膨胀率的分群算法——REGADR

现代深度包检测系统的规则集往往很大,Snort 有 4 000 多条,一种攻击类型的正则表达式有数百个.若正则表达式 DFA 一一对应,则匹配算法需要同时持有多个当前状态或反复扫描目标串,造成严重的性能下降.Aho[2]将多个字符串组合成一个 DFA,同时完成多个字符串的匹配.将多个正则表达式组合在一起生成组合 DFA(composite DFA)也有同样的效果,但正则表达式比字符串复杂,若对组合不加区分,则会造成严重的 DFA 状态膨胀.下文分析了单个正则表达式的膨胀率与组合 DFA 之间的关系,设计了一种正则表达式分群算法——REGADR.

4.1 正则表达式膨胀率DR和组合DFA

正则表达式在匹配字符串的过程中存在歧义匹配的问题,若成功匹配的目标字符串可对应正则表达式的不同片段,则会导致单个正则表达式的 DFA 状态数膨胀;若成功匹配的字符串可对应多个正则表达式中的不同片段,则将这两个正则表达式组合在一起时会造成组合 DFA 的状态数膨胀.图 4 中两个正则表达式“.*AB[^A]{2}”和“.*DEF”合并成一个组合 FA,由于存在歧义匹配,目标字符串可同时匹配两个正则表达式中的任意一个,DFA 中的状态 7,8,9 是由于这种歧义匹配所膨胀出的状态.

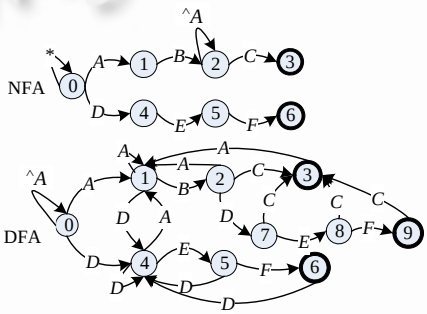

Fig.4 Composite DFA and NFA regular expressions: “.*AB[^A]{2}” and “.*DEF”

图 4 正则比表达式“.*AB[^A]{2}”和“.*DEF”的组合 NFA 和 DFA

不同的正则表达式膨胀率不同、复杂程度不同,对组合 DFA 的影响也不同.可以直观地认为:膨胀率越大,正则表达式本身的复杂程度就越高,生成的组合 DFA 也就越容易膨胀.

图 5 和图 6 所描述的是单个正则表达式的 DR 与所产生的组合 DFA 的 DR 的关系.图 5 是从 Snort 规则集中的两两组合产生的组合 DFA 记录中任意抽出的大约 50 000 条记录,图 6 是由 Linux L7 中的正则表达式产生的.图中符号的含义是:DR(com)含义是组合 DFA,DR(ind)是组合 DFA 中单个正则表达式 DR 的集合.曲线的含义是:当正则表达式集合的最大值处于不同的区间时,组合 DFA 的 DR 大于每一个正则表达式的 DR,组合 DFA 的 DR 在每一个正则表达式的 DR 之间,以及组合 DFA 的 DR 小于每一个正则表达式 DR 的个数占所在区间的总个数的百分比.

图 5 和图 6 说明:

第一,组成组合 DFA 的单个正则表达式的 DR 越大,组合 DFA 就越容易膨胀.随着正则表达式的 DR 的增大,组合 DFA 的 DR 大于单个正则表达式 DR 最大值的个数占该区间内组合 DFA 总个数的百分比呈上升趋势;组合 DFA 的 DR 小于单个正则表达式 DR 的最大值的个数占该区间内组合 DFA 总个数的百分比呈下降趋势;组合 DFA 的 DR 小于所有单个正则表达式 DR 的情况极少出现,且随着正则表达式 DR 的增加,这种情况接近 0;

第二,当单个正则表达式 DR 很大时(比如超过 1),几乎必然导致组合 DFA 状态膨胀.在图 4 中,当正则表达式的 DR 超过 1 后,在它参与的组合 DFA 中,超过 90%都存在状态膨胀.

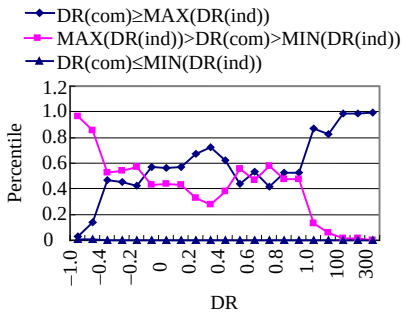

Fig.5 Snort rule set (about 50 000 pieces of record)

图 5 Snort 规则集(约 50 000 条记录)

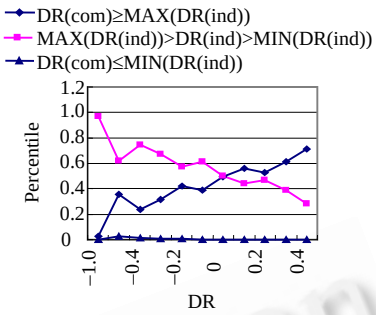

Fig.6 LinuxL7 rule set

图 6 LinuxL7 规则集

4.2 REGADR算法

(1) 算法的形式化描述

define *upbound*, *δ*, *RESet*; //定义组合 DFA 状态数上限,每加入一个正则表达式,组合 DFA 膨胀率的增长差值上限

while (#(*RESet*)!=0) { //要组合的正则表达式集合对 *RESet* 按照膨胀率升序排序

define *CompositeDFA_RESet*; //组合 DFA 的正则表达式集合

define *UnComposte_Reset*; //未进入组合 DFA 的正则表达式集合

for (*RESet* 中第 1 个元素 to *RESet* 最后一个元素) {

将当前的正则表达式加入组合 DFA;

if (组合 DFA 膨胀率的增长差值>*δ*||组合 DFA 的状态数>*upbound*) {

将当前正则表达式加入 *UnComposite_Reset*;

} else {将当前正则表达式加入 *CompositeDFA_RESet*;}

} 输出 *CompositeDFA_RESet*,*RESet*=*UnComposite_Reset*, *CompositeDFA_RESet*=NULL;

}

(2) 算法的分析

算法中,*upbound* 为组合 DFA 状态数上限值.当组合 DFA 的状态数大于 upbound 时,剩余的正则表达式组合成一个新的组合 DFA;*δ*限定每加入一个新的正则表达式组合 DFA 的 DR 增长值上限.在实验中,组合 DFA 的 DR 一般是上升的,这是由于组合 DFA 的复杂程度随着正则表达式的个数增加而增加,决定了组合 DFA 的 DR 总体趋势是上升的;但存在一类正则表达式加入组合 DFA 时会降低它的 DR,这是因为该正则表达式的匹配路径与组合 DFA 中已有正则表达式的匹配路径重合,从而降低了组合 DFA 的状态数,此时组合 DFA 的膨胀率会下降.

REGADR 算法按照 DR 从低到高依次选择正则表达式加入组合 DFA,同时将导致组合 DFA 状态剧烈膨胀的正则表达式挑出,这样做是基于第 4.1 节中的结论,即 DR 越大的正则表达式越容易引起组合 DFA 的状态膨胀,该算法可以将任何一个正则表达式规则集合转换成组合 DFA.算法复杂度为 $O(mn)$,m 是最后组合 DFA 的个数,n 是正则表达式的个数.通常情况下,$m<<n$,因而算法的时间复杂度近似为 $O(n)$.

算法中 *δ*的选择很重要,有两种选择:一种为静态值;另一种为动态值,定义为 $\delta=F$(#States(Com DFA), #(Residual Regular Expression)).#States(Com DFA)是组合 DFA 的状态数,#(Residual Regular Expression)是未组合正则表达式的个数,*δ*是这两个参数的函数.这两种方法中,前者简单,本文的算法采用的就是这种方法;后者需要找出一个合适的函数,针对不同的正则表达式集合具体分析.直观上认为:组合 DFA 的状态数(#States(Com DFA))越大,允许的 DFA 膨胀率增长越小,这样才可加入更多的正则表达式;剩余未组合正则表达式个数(#(Residual Regular Expression))越少,允许的 DFA 膨胀率越大.一般的函数形式为一个阶梯形的衰减关系.当每

次开始一个新的组合 DFA 时,剩余的正则表达式个数下降,δ的初值比在上一个组合 DFA 时的初值大一些.在生成组合 DFA 的过程中,随着组合 DFA 的状态数上升,δ值下降.

5 实验结果与分析

实验采用的正则表达式集合来自 Snort 入侵检测系统中的 4 个规则集和 Linux Layer-7 filter,其中,Snort 规则集约有 1 600 条正则表达式,Linux Layer-7 filter 规则集有 70 条正则表达式.实验中使用 Flex[18]将正则表达式转化成 DFA,实验平台是 PC,Pentium4 2.80GHz CPU,512MB 内存.

5.1 RECCADR实验结果

实验中,参数σ设为 0.5,在 Snort 正则表达式规则集中有 28.2%的正则表达式的膨胀率大于 0.5.其中,27.7%的正则表达式的 DFA 状态数超过 10 000(计算时溢出),这些正则表达式在 Snort 规则集中的分布也是不平衡的.考虑到 Snort 规则集中的正则表达式不是同时激活的,只有在数据包头匹配重叠的情况下才会同时使用,所以我们选择描述相似攻击类型的模式集合作为实验数据.实验中选取了 Snort 规则集的 4 个规则子集:*imap.rules*, *Backdoor.rules*,*Spyware.rules*,*Ftp.rules.imap.rules* 规则集共有 38 条正则表达式,92%的正则表达式的膨胀率超过 0.5.其中:20 条正则表达式存在严重的状态膨胀,DFA 状态数超过 10 000;另外,15 条正则表达式的膨胀率超过了 0.5.下面的实验数据列举了结果,给出了正则表达式的分片结果和状态数总和的统计量.

RECCADR 把原有的正则表达式分割成以下 3 种情况:第一,分割的部位位于正则表达式的中部,正则表达式被分成头部、中部、尾部 3 段;第二,分割的部位位于正则表达式的尾部,即导致 DFA 状态数膨胀的部分位于正则表达式的尾部,正则表达式被分成头部和中部两部分;第三,分割的部位位于正则表达式的头部,正则表达式被分成中部和尾部两个部分.表 3 分别列出了这 3 种正则表达式的一个实例.

Table 3 Regular expression cut result (head, middle, tail separated by •)

表 3 正则表达式分片(头部、中部和尾部由•分隔)

Regular expression	Before		After					
	DFA	DR	Head		Middle		Tail	
			DFA	DR	DFA	DR	DFA	DR
.*\sRENAME\s•[^\n]*•\s\{	49	0.887	24	0.2	6	–0.455	11	–0.214
.*\sLOGIN\s•[^\n]{100}•	>10 000	>10 000	22	0.158	104	–0.046	N/A	
•.*\s{23}•DIMBUS\s+Server\s+v\d+\x2E\d+	938	15.456	N/A		27	–0.156	37	0.121

对实验结果分析如下:

表 3 中前两个正则表达式来自 Snort 规则集中的 imap.rules 规则,第 3 个来自 Backdoor.rules 规则集.经过切片,正则表达式 3 部分的膨胀率都降到了 0.5 以下,分片算法把正则表达式中最容易引起 DFA 状态膨胀的部分隔离起来.分片后,正则表达式 3 部分的 DFA 状态数和膨胀率都大为降低.

表 4 说明了分片后的正则表达式集合的 DFA 状态数总和显著减少,正则表达式的膨胀率均值明显降低.第 3 节详细分析了可能引起正则表达式 DFA 状态膨胀的特征字串是大字符集的计数和闭包.将中部隔离后,正则表达式的头部和尾部的膨胀率与原正则表达式相比会大为减小.

Table 4 Cut results of *imap.rules* regular expression set

表 4 *imap.rules* 正则表达式集合分片结果

#Regular expression	Before		After					
	SUM(DFA)	AVG(DR)	Head		Middle		Tail	
			SUM(DFA)	AVG(DR)	SUM(DFA)	AVG(DR)	SUM(DFA)	AVG(DR)
38	>200 863	>2 631.925	1 011	0.210	5 410	–0.208	175	–0.259

5.2 REGADR实验结果

实验提取了 Snort 规则集中的 3 个典型的规则集合,它们具有不同的正则表达式特征,基本上可以代表 Snort

规则集的普遍情况,同时也实验了 LinuxL7 中的正则表达式集合.Yu 提出了一种基于正则表达式两两关系的正则表达式的多处理器结构的分群算法(Yu algorithm)[8].实验比较了这两种算法的结果.实验中共设置了 9 组不同的 *upbound* 取值,δ设置为 0.7.4 个规则集合中的正则表达式都由分片算法将正则表达式分成头部、中部和尾部 3 个部分.分片的正则表达式只有头部参与组合,但是这 4 个正则表达式中的正则表达式达到分片要求的很少,都只有一两条,基本上不改变原来状态集的性质.

表 5 列出了 Snort 的 *Spyware.rules* 正则表达式集合在上限 *upbound* 为 4 000,6 000,8 000 和 16 000 时的组合 DFA 生成的实验结果.可以看出,分组结果有以下两个特点:第一,当组合 DFA 中的正则表达式个数超过 20 时,组合 DFA 的状态数大小接近上限.也就是说,在给定上限的条件下,已接近最优地生成了组合 DFA.称这类组合 DFA 为第 1 类型组合 DFA.第二,当组合 DFA 的正则表达式个数小于 20 时,很多组合 DFA 所包含的正则表达式只有 1 个或者接近 1 个.同时,组合 DFA 的状态数与上限 *upbound* 相差很远.称这类组合 DFA 为第 2 类型组合 DFA.

Table 5 Composite DFA of *Spyware.rules*

表 5 *Spyware.rules*组合DFA***

upbound	Type1 composite DFA (#(Regular Expressions)>20)	Type 2 composite DFA (#(Regular Expressions)<20)
4 000	120/3999,70/3999,152/3993, 48/3982,36/3963,26/3417	3/592,11/1430,1/33,1/33,7/566,2/121,2/117,1/74,1/46,1/67,1/49, 3/156,1/295,1/189,1/233,1/390,4/533,1/51,1/25,1/283,1/64,1/48,1/76
6 000	207/5999,46/6000,131/5989, 44/5986,27/4041	3/529,1/33,3/195,2/66,5/344,2/121,1/74,1/46,1/67,1/49,3/156, 1/295,1/189,1/233,1/390,4/553,1/51,1/25,1/283,1/64,1/48,1/76
8 000	242/7983,56/7985, 119/7995,35/6126	3/529,12/1283,2/82,1/53,2/66,4/168,2/121,1/74,1/46,1/50,1/67,1/49, 3/156,1/295,1/189,1/233,1/390,4/553,1/51,1/25,1/283,1/64,1/48,1/76
16 000	240/10984,93/15950, 114/15865	3/529,8/1367,1/26,1/33,2/66,12/1430,2/121,2/117,1/74,1/67,2/222,3/156, 1/295,1/189,1/233,1/390,1/52,4/553,1/51,1/25,1/283,1/64,1/48,1/76

算法根据正则表达式膨胀率来生成组合 DFA.当组合 DFA 的膨胀率不产生大跳跃时,认为当前正则表达式对组合 DFA 存在良性影响,并将其加入组合 DFA.算法尽可能多地找到这样的良性正则表达式,生成组合 DFA,这种情况下生成的是第 1 类型组合 DFA.第 2 类型组合 DFA 包含的正则表达式往往使组合 DFA 的膨胀率发生较大的跳跃,算法把这样的正则表达式隔离出来.

由于第 2 类型组合 DFA 中的正则表达式的个数很少,如果这些正则表达式组合在一起,则即使生成的组合 DFA 膨胀率会发生大的跳跃,它的状态总数也不会超过上限.基于上面的分析,这里对分群算法进行优化,将第 2 类型组合 DFA 中的正则表达式再次组合,调整算法中的参数δ,实验中δ=10,放宽了限制.表 6 是重新组合的结果,发现在 4 个 upbound 限制下,第 2 类型组合 DFA 的个数可以进一步压缩.

Table 6 Composite DFA of *Spyware.rules*

表 6 *Spyware.rules* 组合 DFA

upbound	#(Type 2 composite DFA)	#(New composite DFA)	New composite DFA of composite DFA(<20)
4 000	23	9	9/3005,4/1176,4/2509,5/1873,6/3472, 2/382,3/2522,15/851,1/76
6 000	22	9	35/1678,16/3329,12/4624,5/4430, 5/2820,5/2178,5/3914,3/5255,3/1018
8 000	24	7	25/2469,10/4261,4/2763,5/3514, 7/7812,3/5255,3/1018
16 000	24	6	23/3062,10/6842,6/11129,7/10617, 4/13725,3/1018

表 7~表 10 中给出几个 Snort 规则子集的实验数据,同时比较了基于正则表达式膨胀率的分群算法和 Yu 分群算法.实验数据表明:基于正则表达式膨胀率的分群算法对几个正则表达式集合所生成的组合 DFA 的个数明

*** 表中数据的含义是组合 DFA 中正则表达式的个数和组合 DFA 的状态数.例如,第 2 行第 2 列的第 1 个数据 120/3999 的含义是:这个组合 DFA 的正则表达式个数是 120,组合 DFA 的状态数是 3 999.

显低于其他分群算法;组合 DFA 的状态数总和小于其他分群算法,平均 30%以上.

Table 7 *Backdoor.rules* (147 patterns)

表 7 *Backdoor.rules* (147 条规则)

upbound	REGADR		Yu algorithm		States reduction (%)
	Groups	Total states	Groups	Total states	
4 000	4	7 825	6	10 086	22
6 000	4	16 527	5	23 045	28
8 000	4	18 855	5	24 026	22
10 000	3	24 159	4	28 115	14
12 000	3	22 392	4	32 851	32
14 000	3	22 392	4	32 851	32
16 000	3	23 752	4	33 530	30
18 000	2	27 190	4	33 530	20
20 000	2	27 043	4	47 143	43

Table 8 *Spyware.rules* (500 patterns)

表 8 *Spyware.rules* (500 条规则)

upbound	REGADR		Yu algorithm		States reduction (%)
	Groups	Total states	Groups	Total states	
4 000	15	42 921	24	64 952	34
6 000	13	56 940	20	93 425	39
8 000	12	68 260	18	101 756	33
10 000	11	77 888	16	124 015	37
12 000	10	90 844	16	131 366	31
14 000	10	93 218	15	141 761	34
16 000	9	115 196	14	151 274	24
18 000	9	111 136	14	174 223	36
20 000	9	143 966	13	177 926	20

Table 9 *Ftp.rules* (61 patterns)

表 9 *Ftp.rules* (61 条规则)

upbound	REGADR		Yu algorithm		States reduction (%)
	Groups	Total states	Groups	Total states	
4 000	3	6 665	6	9 073	26
6 000	3	6 657	5	11 437	42
8 000	3	6 657	5	23 867	72
10 000	3	6 657	4	18 147	69
12 000	3	6 657	4	18 147	69
14 000	3	6 657	4	18 147	69
16 000	3	6 657	4	18 147	69
18 000	3	6 657	4	29 230	77
20 000	3	6 657	4	29 230	77

Table 10 LinuxL7 (70 patterns)

表 10 LinuxL7 (70 条规则)

upbound	REGADR		Yu algorithm		States reduction (%)
	Groups	Total states	Groups	Total states	
4 000	4	9 646	5	14 988	35
6 000	3	14 629	5	18 719	22
8 000	3	16 178	4	23 438	31
10 000	3	20 415	4	28 411	28
12 000	3	21 847	4	26 471	17
14 000	3	25 591	3	32 584	21
16 000	3	25 591	3	34 669	26
18 000	3	25 591	3	34 669	26
20 000	3	25 591	3	34 669	26

实验数据表明了 REGADR 算法的优良特性.实验中,δ为常数,初次分群结果并不理想,需要把算法分为两个阶段.如果在δ与组合 DFA 的状态数以及剩余的未组合正则表达式个数之间建立一定的函数关系,则生成的组合 DFA 的结果可以更优化,本文限于篇幅没有对这两者的关系作进一步讨论.

当前,使用正则表达式的深度包检测系统实现组合 DFA 的算法通常为试探组合,没有深入考虑正则表达式不同形式之间的关系对组合 DFA 的影响.本文提出的 REGADR 算法和 Yu 算法从两个不同的角度把正则表达式的自身特点和生成组合 DFA 的算法结合起来,都有效地压缩了空间.Yu 在文献[8]中证明了经过选择性组合,可以有效地生成组合 DFA.

6 结论和下一步工作

本文主要解决的问题是正则表达式的存储问题.首先提出了正则表达式膨胀率 DR,说明用 DR 来衡量正则表达式潜在的状态膨胀特性是有效的,实验结果说明了这个结论的正确性.然后基于正则表达式的膨胀率 DR 提出了两个正则表达式的压缩算法 RECCADR 和 REGADR,证明了它们可以大幅度降低正则表达式状态数,从而降低了正则表达式的存储空间.同时还比较了 REGADR 算法和 Yu 算法,说明了 REGADR 算法分群的个数和 DFA 的状态数总和都显著降低,表现了算法的优良特性.

下一步的工作是,针对组合 DFA 和δ的关系,设计合适的函数从而得到更好的分群效果.同时,基于本文提出的算法,将 RECCADR 算法与 REGADR 算法结合起来,设计相应的正则表达式匹配算法.

致谢 感谢中国科学院计算机网络信息中心的各位同事和同学,尤其要感谢网络体系结构研究组的所有成员,与诸位的讨论使我们受益匪浅.

References:

[1] Li WN, E YP, Ge JG, Qian HL. Multi-Pattern matching algorithms and hardware based implementation. Journal of Software, 2006, 17(12):2403−2415 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/2403.htm

[2] Aho AV, Corasick MJ. Efficient string matching: An aid to bibliographic search. Communications of the ACM, 1975,18(6): 333−340.

[3] Sommer R, Paxson V. Enhancing byte-level network intrusion detection signatures with context. In: Proc. of the 10th ACM Conf. on Computer and Communications Security. New York: ACM, 2003. 262−271.

[4] Roesch M. SNORT network intrusion detection system. 2007. http://www.snort.org

[5] Paxson V. Bro intrusion detection system. 2007. http://www.icir.org/vern/bro-info.html

[6] Kumar S, Turner J, Williams J. Advanced algorithms for fast and scalable deep packet inspection. In: Bhuyan LN, Dubois M, Eatherton W, eds. Proc. of the 2006 ACM/IEEE Symp. on Architecture for Networking and Communications Systems. New York: ACM, 2006. 81−92.

[7] Levandoski J, Sommer E, Strait M. Application layer packet classifier for Linux. 2007. http://l7-filter.sourceforge.net/

[8] Yu F, Chen ZF, Diao YL, Lakshman TV, Katz RH. Fast and memory-efficient regular expression matching for deep packet inspection. In: Bhuyan LN, Dubois M, Eatherton W, eds. Proc. of the 2006 ACM/IEEE Symp. on Architecture for Networking and Communications Systems. New York: ACM, 2006. 93−102.

[9] Johnson T, Muthukrishnan S, Rozenbaum I. Monitoring regular expressions on out-of-order streams. In: Proc. of 2007 IEEE the 23rd Int'l Conf. on Data Engineering. Piscataway: IEEE, 2007. 1315−1319.

[10] Allen J. Perl version 5.8.8 documentation—Perlre. 2007. http://perldoc.perl.org

[11] Hazel P. PCRE—Perl compatible regular expressions. 1997. http://www.pcre.org

[12] Cox R. Regular expression matching can be simple and fast (but is slow in Java, Perl, PHP, Python, Ruby, …). 2007. http://swtch.com/~rsc/regexp/regexp1.html

[13] Sidhu R, Prasanna VK. Fast regular expression matching using FPGAs. In: Proc. of FCCM 2001, the 9th Annual IEEE Symp. on Field-Programmable Custom Computing Machines. Piscataway: IEEE, 2001. 227−238.

[14] Clark CR, Schimmel DE. Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns. In: Field Programmable Logic and Application: The 13th Int'l Conf., FPL 2003. New York: Springer-Verlag, 2003.

[15] Kumar S, Dharmapurikar S, Yu F, Crowley P, Turner J. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: Rizzo L, Anderson TE, McKeown N, eds. Proc. of the ACM SIGCOMM 2006 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM, 2006. 339–350.

[16] van Lunteren J. High-Performance pattern-matching for intrusion detection. In: Proc. of the INFOCOM 2006, the 25th IEEE Int'l Conf. on Computer Communications. Piscataway: IEEE, 2006. 1–13.

[17] Hopcroft JE, Motwani R, Ullman JD. Introduction to Automata Theory, Languages, and Computation. 2nd ed., New York: Addison Wesley, 2001.

[18] Paxson V. Flex: A fast scanner generator. 2007. http://www.gnu.org/software/flex/

附中文参考文献:

[1] 李伟男,鄂跃鹏,葛敬国,钱华林.多模式匹配算法及硬件实现.软件学报,2006,17(12):2403–2415. http://www.jos.org.cn/1000-9825/17/2403.htm

徐乾(1983－),男,河南洛阳人,硕士,主要研究领域为网络协议,网络安全,入侵检测技术.

葛敬国(1973－),男,博士,副研究员,主要研究领域为互联网体系结构.

鄂跃鹏(1976－),男,博士生,主要研究领域为网络协议,嵌入式.

钱华林(1940－),男,研究员,博士生导师,主要研究领域为下一代网络体系结构.