高技术通讯 2010 年 第 20 卷 第 12 期:1217 ~ 1223

doi:10.3772/j.issn.1002-0470.2010.12.002

大规模复杂规则匹配技术研究①

张树壮②* 罗 浩** 方滨兴*,**

(*哈尔滨工业大学计算机网络与信息安全技术研究中心 哈尔滨 150001)

(**中国科学院计算技术研究所 北京 100190)

摘 要 针对当前安全系统对复杂规则的需求和复杂规则匹配技术的状况,提出了一种新的规则表示方式——字符串表达式,并给出了对应的匹配方法——基于扩展的有限状态自动机(XFA)实现大规模复杂规则匹配的算法。字符串表达式可以描述多个精确字符串之间的逻辑关系与空间位置关系,从而满足安全系统对复杂特征的描述需求。匹配使用二维结构来完成,首先用经典串匹配算法进行字符串的存在性验证,然后将其结果作为输入,驱动以表达式中字符串为"字符"的 XFA 完成逻辑关系的验证。基于 XFA 的匹配方法的空间效率和时间效率都接近多模精确串匹配算法。实验结果表明,文中提出的方法既能满足安全系统对关联特征的描述需求,又能提供高效的匹配性能,较好地解决了大规模(万条)的复杂规则匹配问题。

关键词 串匹配,正则表达式,字符串表达式,扩展字符串匹配,扩展有限自动机

0 引 言

随着网络应用的不断增多,安全系统(如病毒检测、入侵检测、垃圾邮件过滤、垃圾短信息过滤等)中过滤规则的表示方法也越来越复杂。例如 Snort[1] 规则,Alert tcp $HOME _ NET any → $EXTERNAL _ NET 1863 (msg:"CHAT MSN login attempt"; flow:to _ server, established; content:"USR"; depth:4; nocase; content:"TWN"; distance:1; nocase;)表示要同时匹配"USR"和"TWN"这两个关键词,且这两个词之间要有 1 个字符的距离。同样,对于原本为简单字符串的关键词,在实际应用中也可能以各种不同的形式出现(例如"模式匹配"可能会写成"模#式#匹#配"),这就需要逐一识别各个部分后再进行一定限定条件(位置顺序、有限间距等)的判断来进行识别。可见,为了表达更精确的语义和上下文信息,需要将多个关键词按照一定的顺序和限定条件进行组合,形成一个复合的匹配规则,也称为复杂规则。在 Snort 中有几千条规则,当启用的规则数目为几百条时,特征匹配的时间就占整个处理流程的 31%,而在处理 Web 数据时,最高可以达到 81%,是最耗时的操作。在 L7-filter[2] 中,启用的规则有 70 条时,匹配所占用时间超过整个流程的 90%。因此改进匹配效率,对提高整个系统的效率具有非常重要的意义。

上述这些问题所需要的复杂规则无法用精确串来描述,因此经典的多模串匹配算法如 AC[3]、SBOM[4]、WU-MANBER[5] 等无法直接使用。近些年来表达能力强大的正则表达式逐渐应用在各类安全系统中。由于确定型有限自动机(DFA)对每个输入字符具有 $O(1)$ 的处理速度,适合在线的实时匹配,因此目前对正则表达式的研究主要集中在 DFA 的匹配方法上。然而 DFA 具有指数的空间复杂度,且当多条正则表达式编译在同一个 DFA 中时,也会引起内存需求的"爆炸性"膨胀[6],所以对正则表达式匹配的研究又都集中于如何降低 DFA 的存储空间上,如规则改写和规则分组[6]、合并转换函数(D^2FA)[7]、合并状态[8]、对输入进行预编码[9] 及改变 DFA 结构[10] 等。但由于 DFA 的内存为指数级增长,而目前 DFA 内存压缩方法效果都为线性,因此对 DFA 进行内存压缩无法从根本上解决内存膨胀问题。为了实现大规模复杂规则的实用化,曹京等人先后提出了布尔表达式[11] 和定序窗口布尔表达

① 863 计划(2009AA01Z437,2007AA01Z442,2007AA010501,2007AA01Z474),973 计划(2007CB311101)和国家自然科学基金(60903209)资助项目。

② 男,1982 年生,博士生;研究方向:网络安全,信息内容安全;联系人,E-mail: zhangshuzhuang@pact518.hit.edu.cn

(收稿日期:2009-09-01)

式[12]匹配技术。在定序窗口布尔表达式中所有“与关系”关键词必须按照次序全部出现在一个指定大小的文本窗口中。布尔表达式匹配分为模式串匹配层和布尔表达式匹配层两个层次,模式串匹配层首先找出所有在布尔表达式中出现的模式串,然后布尔表达式匹配层根据匹配的模式串信息来计算布尔表达式的值。但是这种表示方法以布尔代数为基础,虽然加上了定序和整体窗口限定,仍无法表示“元素”之间的相对位置关系。本文结合实际应用的需要提出了一种新的规则表示方法——字符串表达式,并给出了对应的匹配方法。该方法可满足描述复杂特征的需要而且匹配效率高,算法的内存占用量在当前硬件资源可接收的范围内。对方法和一些代表性串匹配方法进行的对比实验证明,该方法具有更高的实际应用价值。

1 字符串表达式匹配

为了能够满足当前安全系统对复杂语义和上下文关联信息的描述需求,本文提出了字符串表达式的概念。它通过特定的操作符将多个精确字符串特征连接起来,形成表达能力更强的复合规则。它可以准确地刻画在同一段待匹配文本中相互关联的多个特征之间的逻辑关系。

1.1 字符串表达式定义

已知有限非空字符集合$\sum$,称为字母表。下文中用 p 表示字母表中的字符, P 表示字母表中符号的有穷序列,称为字符串,即: $P = p_1p_2\cdots p_m$, $p_i \in \sum$, $1 \leqslant i \leqslant m$。

下面首先定义两个操作符:

定义 1:“$!P\{m,n\}$”称为否定限定符。否定限定符由否定符号!、限定字符串 P 以及距离限定界限 $\{m,n\}$(m,n 为整数,且 $0 \leqslant m \leqslant n$) 组成。$!P_i\{m,n\}P_j$ 表示在字符串 P_j 之前 m 至 n 个字符的区间内,不能出现字符串 P_i。

定义 2:“$.\{m,n\}$”称为连接符。连接符由通配符‘.’和距离限定界限 $\{m,n\}$(m,n 为整数,且 $0 \leqslant m \leqslant n$) 组成,用来连接两个字符串。$P_i.\{m,n\}P_j$ 表示字符串 P_j 位于 P_i 之后,且 P_i 与 P_j 之间有 m 至 n 个任意字符的间隔。

易知,使用上面两个操作符,可以表示出 Snort 规则中 content 的所有关于距离和逻辑修饰关系的关键字(!,within,distance,depth,offset)。

现在给出字符串表达式的定义:

定义 3:

(1)字符串是原子表达式。

(2)如果 P_i 是原子表达式,那么 $!P_i\{m,n\}P_j$ 也为原子表达式。

(3)原子表达式是字符串表达式。含有否定限定符的原子表达式称为限定原子表达式,否则称为自由原子表达式。

(4)将有限个原子表达式用连接符连接起来形成的复合表达式是字符串表达式。

定义 4:给定包含 m 个字符串表达式的集合 $SE = \{se_1, se_2, \cdots, se_m\}$ 和文本 $T = t_1t_2\cdots t_n$($t_i \in \sum$, $1 \leqslant j \leqslant n$),在文本 T 中找到所有位置 pos,使得在其之后的 k 个连续字符组成的序列 $t_{pos}t_{pos+1}\cdots t_{pos+k}$ 满足 se_i,其中 $se_i \in SE(1 \leqslant i \leqslant m)$,称为字符串表达式的匹配。

1.2 匹配算法框架

从字符串表达式的定义可以看到,表达式由连接符将各个原子表达式组合起来而形成。如果将字符串表达式中的每个字符串看成是一个广义的“字符”的话,那么每一个具体的字符串表达式在匹配时除了要满足“字符”相同外还要符合“字符”间的逻辑关系和空间限定条件。本文提出的字符串表达式匹配方法分为两个层次:首先对每一个原子表达式和否定限定符中的字符串进行存在性验证,然后对各个字符串之间的限定关系和位置关系进行满足性判定。

第一个层次属于多模串匹配问题,可以使用现有的经典算法来完成。本文选用基于 DFA 的 AC 自动机来实现字符串的存在性验证。第二层对原子表达式内部以及相邻两个原子表达式之间的限定关系进行判定。这类问题可以使用自动机来完成。但由于经典自动机(NFA,DFA)只能够接收字母表中的字符,如果要使用它对字符串间的否定限定关系以及位置关系进行判定,需要对其转换函数作出必要改进,使其能够处理本文所定义的两类操作符。为此本文提出了一种扩展的有限状态自动机 XFA。这里先给出匹配方法的框架,XFA 将在下一节中详细介绍。

对于一个大的规则集,处理方法有两种:(1)将所有的表达式编译在一起,形成一个混合 XFA。(2)将每一个表达式单独编译成一个 XFA。基于自动机的算法所需要的内存与字母表的大小相关,因为自动机中每个状态对字母表中每个字符都应该有一个转换函数。对于一个大规模字符串表达式规则集,

其包含的所有字符串形成的“字母表”通常很大(远远大于通常选用的 ASCII 字母表(256)),若将所有表达式构建成一个混合 XFA,则每个状态需要巨大的存储空间来形成转换表。因此本文选用第二种方法。虽然将每个表达式单独编译成一个 XFA 时,由于其“字母表”很小而降低了内存需求,却同时增大了时间复杂度。因为大规模规则集会形成等数量的 XFA,而同时更新数量巨大的 XFA 显然在效率上是无法接受的。为了改进这种情况,在使用表达式中字符串所构建的 AC 自动机中,每一个接受状态处都会被附加一个列表,用来存放这个接受状态所表示的字符串和哪些 XFA 相关联,从而在第二层的验证过程中,只对相关的 XFA 进行状态更新,这可以大大提高匹配效率。从第 3 节中对实际规则的分析中可以看到,每个字符串所关联的表达式数量很少,因此可以很快地完成第二层验证。

从上文可知,本文提出的匹配方法形成了一个二维结构,如图 1 所示。给定一个字符串表达式规则集,首先经过预处理过程,抽取出所有的字符串,构建成一个基于 DFA 的 AC 自动机,同时以自由原子表达式和否定限定操作符中的字符串作为“字符”,将每个表达式构建成一个 XFA。在匹配时,首先将待匹配文本送入 AC 自动机进行字符串的存在性验证,当匹配过程到达 AC 自动机的接受状态时,就根据这个状态的关联列表去更新相应的 XFA。如果有某个 XFA 到达接受状态,表示一个字符串表达式规则匹配成功。

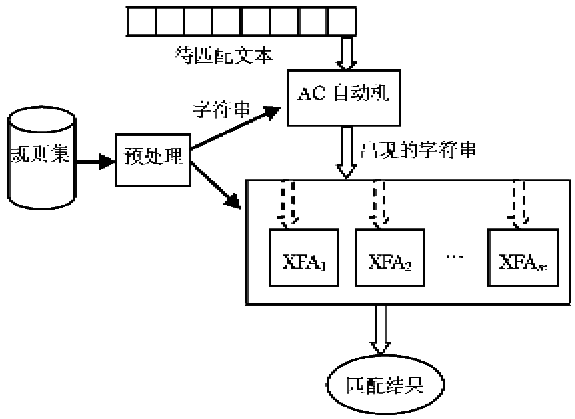

图 1　字符串表达式匹配算法框架

2　扩展有限自动机

由前面的叙述可知,为了能够匹配表达式中字符串之间的逻辑次序关系和空间位置关系,必须对有限状态自动机做出必要的改进。

2.1　XFA 的定义

定义 5:扩展有限状态自动机 XFA = $(Q, P_Q, \sum, \delta, (q_0, P_0), F)$ 其中:

Q:有限的扩展状态集合,它记录状态所代表的字符串以及与其它字符串之间的限定条件;

P_Q:每个状态的匹配位置的集合;

$\sum$:字母表,其每一个字符为自由原子表达式或否定限定符中的字符串;

$\delta: Q \times \sum \rightarrow Q$,扩展的跳转函数;

q_0:初始状态;

P_0:P_Q 的初始值;

F:接受状态的集合,且 $F \subseteq Q$。

从定义可以看出 XFA 与 DFA 相似,但与 DFA 相比有两个不同点:(1)XFA 多了一个状态匹配位置的集合 P_Q,它存储了在匹配过程中每个“字符”的匹配位置,并将信息提供给跳转函数;(2)跳转函数 δ 在进行跳转时,不仅需要考虑“字符”的相等性,还需要考虑位置关系的匹配性。

2.2　XFA 的构建

字符串表达式是由原子表达式顺序连接而成,而一个原子表达式的基本形式为一个自由原子表达式(P_i)前面有 w 个(,$w \geqslant 0$)个否定限制符,可以表示为

$$.\{m_i, n_i\}!P_j\{x_j, y_j\}!P_{j+1}\{x_{j+1}, y_{j+1}\}\cdots!P_{j+w}\{x_{j+w}, y_{j+w}\}P_i$$

下面详细说明一下这段原子表达式的状态结构。在 XFA 中,第一个状态都是初始状态 s_0,对应于定义 5 中的 q_0,是进行匹配的初始状态。每个自由原子表达式和否定限定符中的字符串对应一个状态。为了描述方便,给出如下定义:

定义 6:由表达式中自由原子表达式中字符串生成的 XFA 状态称为自由状态,由否定限定符中字符串生成的 XFA 状态称为限定状态。整个 XFA 中的状态可以分为 3 类:初始状态,自由状态和限定状态。

定义 7:修饰同一个自由原子表达式的所有否定限定符所生成的状态形成的集合称为一个限定状态组。

假设 s_k 为初始状态 s_0 或者第 $i-1$ 个原子表达式中自由原子表达式中字符串生成的状态,那么第 i 段对应的 XFA 结构如图 2 所示。图中,每个自由状态由实线圆圈表示,每个限定状态由虚线圆圈表示,矩形框中的所有限定状态同属于同一个限定状

态组。

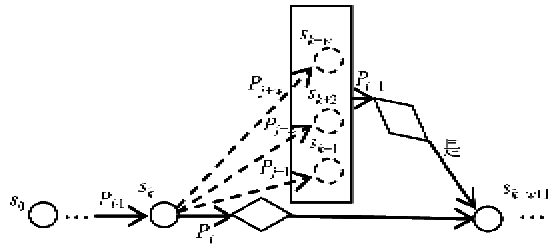

图 2 $P_i \cdot \{m,n\} ! P_j\{x_j,y_j\} ! P_{j+1}\{x_{j+1},y_{j+1}\} \cdots ! P_{j+w}\{x_{j+w},y_{j+w}\} P_{i+1}$ **对应的 XFA 片段(省略了所有目标为初始状态的跳转和限定状态之间的跳转)**

这些状态之间的跳转函数定义为:相邻的两个自由状态之间可以由字符串 P_i 触发一次跳转判断。s_k 与修饰 P_i 的每一个限定状态 s_{k+w} 之间可以通过字符串 P_{j+w} 进行跳转。同一个限定状态组内限定状态之间也可以通过相应的字符串 P_{j+w} 进行跳转,并且每个限定状态都可以通过 P_i 触发一次向状态 s_{k+w+1} 的跳转判断。除此之外的所有跳转目标均为初始状态 s_0。可以看到,如果一个跳转的目标状态是由自由原子表达式生成的,那么就需要进行一次判断来决定跳转的目标状态。

构建完成第 i 个原子表达式时,将以 s_{k+w+1} 为起始状态构建下一个原子表达式的状态集合,重复这个过程,直到处理完整个字符串表达式。最后一个自由状态为 XFA 的接受状态。

对于每一个 XFA, P_0 中元素的初始值 P_{s0} 为 0,表示初始状态 s_0 已经在 0 处被匹配,其余元素的初始值均为 -1,表示未匹配。

2.3 XFA 的匹配

从上一节的介绍中可以看出,在一个 XFA 中,跳转函数分为以下三种情况:

(a) 从自由状态跳转向限定状态,以及从限定状态跳转向限定状态;

(b) 从自由状态跳转向自由状态;

(c) 从限定状态跳转向自由状态。

下面就以图 2 为例分别给出这三种跳转函数。

假设表达式已经匹配完第 $i-1$ 个原子表达式,需要匹配第 i 个原子表达式,目前的输入为 P。s 为当前的活动状态:

(1)如果当前输入 $P \in \{P_{j+r} \mid 1(r(w\}$,那么属于(a)类跳转,不需要进行判断就直接跳转到对应的目标状态 s_{k+r},并记录下匹配位置。

(2)如果 s 为自由状态,且当前输入 $P = P_i$,那么属于(b)类跳转。(b)类跳转是需要进行距离判断的跳转。由于 s 为自由状态,由第 2.2 节可知其为原子表达式 P_{i-1} 形成的状态(图 2 中 s_k),假设 $distance$ 为 P_i 的匹配位置和自由状态 s_k 的匹配位置之间的距离,那么有:如果 $distance \leqslant m_i$,则本次不符合跳转条件,但是以后文本中还可能出现符合距离的 P_i,即可能出现成功的匹配,因此不进行跳转;如果 $distance \geqslant n_i$,则不仅这一次匹配失败,且无论后续文本是什么内容,在当前状态下都无法再使它跳转到状态 s_{k+w+1},因此不会有成功的匹配,因此跳转到初始状态 s_0,重新开始匹配;如果 $m_i \leqslant distance \leqslant n_i$,说明满足匹配条件,跳转到状态 s_{k+w+1},将 s_{k+w+1} 的匹配位置设置为 P_i 的匹配位置。

(3)如果 s 为限定状态,且当输入 $P = P_i$,那么属于(c)类跳转。(c)类跳转是从限定状态向自由状态进行跳转,需要进行两次判断。首先判断 P_i 和 P_{i-1} 形成的自由状态 s_k 的匹配位置之间的距离是否满足要求,即进行一次(b)类跳转判断。唯一不同的是,当满足 $m_i \leqslant distance \leqslant n_i$,不能直接跳转到 s_{k+w+1},还需要判断 P_i 与它所有的否定限定符之间的限定关系是否满足要求:对于和当前状态 s 在同一个限定状态组内的限定状态 $s_l(1 \leqslant l \leqslant w)$,如果其中一个状态所表示的字符串曾经出现过且匹配位置和 P_i 之间的距离不满足限定关系,则说明在 P_i 之前某个区域的文本中,存在不该出现的字符串,因此,无论后续文本是什么内容,都不同通过当前状态到达一个接受状态,因此跳转到 XFA 的初始状态 s_0 重新开始匹配。只有所有的限定状态与目标状态之间距离都满足限定关系,才能跳转到状态 s_{k+w+1},并设置其匹配位置。

3 结果评测

3.1 匹配效率

通常串匹配算法的评测指标有两个:时间效率(匹配速度)和空间效率(最终的匹配结构所占用的内存总量),且这两个指标是被分开考虑的。这种评测方法忽略了两点:一是算法空间的占用对串匹配算法本身的匹配速度有很大影响[13],二是由于系统的硬件环境(资源)是有限的,算法空间的占用对整个系统的效率也有很大的影响。因此,在对一个串匹配算法进行评测时,尤其是在针对某项应用来选择算法时,应该将两者结合起来考虑。

对于一个内存有限且确定的应用环境来讲,当

算法的内存占用比较少时,其对整个系统造成的影响很小,而当内存占用到达一定程度时,开始对系统造成较大影响,当内存占用继续增大并超过一定的量(物理内存容量、进程可持有的内存量等)时,将会使得算法在当前环境下不可用。这个过程类似于在化学反应中的量-效关系[14,15]。因此本文提出了一个算法效率函数 f 来反映这种规律,f 为一个无量纲实数,定义为

$$f = \frac{s}{1 + ce^{-k(1-\frac{m}{M})}}$$

式中:s 表示算法的匹配速度(单位时间内处理的数据量),m 表示算法所占用的内存。M 表示系统可供算法使用的内存的上限,c 和 k 为常数,且 $\frac{s}{1+c}$ 为算法内存占用达到极限时的函数值,因此 c 通常取值较大。k 决定了 f 曲线的形状(内存占用到什么程度时算法接近不可用),其取值通常根据系统实际情况而定。f 的值反应了在一个确定环境中,匹配算法的适用程度。

3.2 评测结果

在第 2 节中曾经提到,若第一层(AC)处理中得到一个出现在文本中的字符串,则需要对与其关联的所有 XFA 进行更新。因此每次需要更新的 XFA 数目等于相应字符串在整个规则集中出现的次数。表 1 所示为对 Snort 2.8 中不同类型的规则进行分析统计后得出的结果。表中平均关联规则数目表示每一个字符串特征平均在整个规则集中的多少条规则中出现过,可见,在实际应用的规则集中,每个字符串所关联的规则数目一般都很少(小于 2),这就说明由第一层匹配验证出一个字符串存在于文本中之后,需要更新的 XFA 数目很少,因此其整体效率接近于第一层所用的多模匹配算法的效率。

表 1 Snort 规则中的关键词关联统计

规则名称	web-misc	spyware-put	oracle	exploit
规则数目	369	972	307	201
平均关联规则数目	1.14	1.64	1.028	1.30

为了测试本文提出的方法(strexp)的性能,对本文方法进行了实现,并与文献[11]中提出的双映射表法(boolexp)以及基于 DFA 的匹配在完全相同的环境下进行了测试,对它们的时间、空间性能以及效率进行了对比。其中 strexp 和 boolexp 的第一层匹配使用了相同的 AC 自动机。

实验环境:(1)硬件:曙光 I610、CPU 为 Intel(R) Xeon(R)、主频 2.4GHz、内存 4G;(2)软件:Cent os5.2 Linux 操作系统、gcc 4.1.2 编译器。为了能够产生大规模规则集,以便测试本文方法的适应性,规则集和待匹配的文本生成方法如下:将全宋词中的每个句子根据其长度拆成最少 2 段最多 4 段,通过在段间加入连接符和否定限定符将其扩展成字符串表达式。使用同一句子并在加入连接符的地方插入段首字的拼音作为填充字符生成待匹配文本。这样,插入了拼音的文本是否匹配由同一句形成的表达式完全取决于其拼音的长度,具有随机性。如:"万家掩.{0,4}映翠.{0,3}微间"可以匹配"万家掩 ying 映翠 wei 微间",而"野人只.{0,3}合其.{0,4}中老"却由于距离不符合而无法匹配"野人只 he 合其 zhong 中老"。表达式个数的变化范围为 100~50000,带有否定限定符的规则占全部规则总数的 50%。步长随着规模的增大而增大。为了保证模式串能在文本中命中,首先从新浪新闻中获取 100MB 的文本数据。然后在每 500 个字节的区域内随机位置处插入一条待匹配的文本,这样,整个测试数据文本就呈现出整体均匀而局部随机的特性,接近真实情况。由于 strexp 算法和 boolexp 算法都使用了基于 DFA 的 AC 自动机进行字符串的存在性验证,因此本文中也同时给出了使用规则集中所有字符串构建成的 AC 自动机的匹配速度和内存需求。结果如图 3 和图 4 所示。

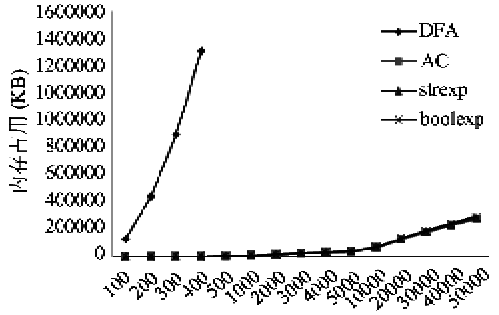

图 3 不同规模规则集下各种方法的内存占用

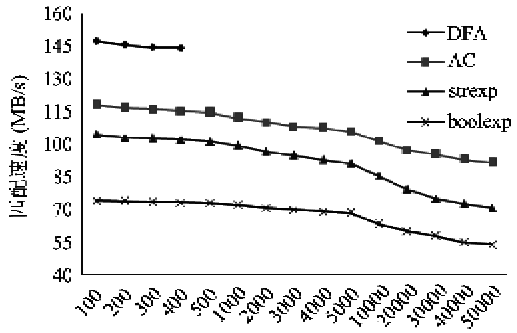

图 4 不同规模规则集下各种算法的匹配速度

图 3 给出了各个方法的内存需求随着规则规模的变化情况。可以看到，DFA 的内存需求随规则数目呈指数级增长，且在 400 条时，内存需求超过 1.3G，当规则数继续增加时，需要的内存太大而使得 DFA 无法构建成功。其余三种方法的内存需求要相对低很多，且整体呈线性增长，在规则规模达到 5 万条时，内存需求为 250MB 左右。strexp 和 boolexp 比 AC 自动机的内存需求稍大，这是因为它们都首先需要一个 AC 自动机来做字符串匹配，然后 strexp 还需要为每条规则构建一个 XFA，boolexp 也需要额外的内存来做映射表。值得注意的是，这两部分额外内存都在 AC 自动机所需的内存的 1%以下，这意味着 strexp 的空间复杂度取决于所用精确串匹配算法的空间复杂度。

图 4 给出了在规则规模逐渐增大的情况下，各个方法匹配速度的变化情况。可以看出，同等条件下各种方法中 DFA 的匹配速度最高，strexp 和 boolexp 方法速度低于 AC，且 strexp 比 boolexp 速度高 20%～30%。strexp、boolexp 以及 AC 自动机的匹配速度都是随着规则数目的增加而降低。在规则数据小于 1000 时，降低的幅度非常小，随着规模数目的继续增大，匹配速度降低的幅度也逐渐加大。这是由于 strexp 和 boolexp 都首先取决于 AC 自动机的匹配速度，因而呈相同规律下降。另外值得注意的是 strexp 匹配速度与 AC 的匹配速度并不是成固定比例，而是下降的幅度要稍大于后者，这是由于同一个关键字所关联的表达式的数目增多，使得每次 AC 成功匹配到一个字符串后需要更新的 XFA 数目增多，导致了性能的降低。

使用 3.1 节提出的效率评测方法对上述三种方法进行评测（$M = 1000000$(KB)，$c = 100$，$k = 10$），其结果如图 5 所示。可以看到，DFA 匹配方法的效率由于内存的增大而迅速降低，而 strexp 和 boolexp

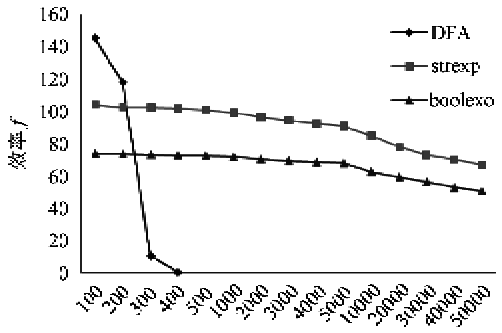

图 5 三种方法的效率评测

的效率降低趋势和匹配速度趋势相同，说明此时它们的效率主要取决于匹配速度，而受内存的影响较小，这和实际情形是一致的。从图中可知，在规则集规模在 300 条到 50000 条之间时，strexp 比其他两种方法具有更高的效率，即能够在实际应用中取得更好的整体性能。

4 结 论

本文分析了当前安全系统对复杂规则的需求以及复杂规则匹配技术的现状，提出了字符串表达式的概念并给出了匹配方法。字符串表达式通过限定符和连接符来描述各种特征之间的逻辑关系和位置关系，这种规则表示方法的表达能力强于单纯的字符串，能够更好地满足当前安全系统中用户对复杂语义和上下文信息的描述需求。本文所提出的基于二维结构的匹配方法使得其匹配效率接近经典的多模匹配算法的同时内存需求随规则规模呈线性增长。实验证明，使用本文方法可以实现大规模（万级别）复杂规则匹配，具有非常大的实用价值。

在今后的工作中，我们将进一步完善这种语法，使其能够更灵活更全面地表示安全系统中的各种复杂特征，并进一步提高匹配效率。

参考文献

[1] Snort 2.8.x[EB/OL]. http://www.snort.org, 2009

[2] Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/,2009

[3] Aho A, Corasick M. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*. 1975, 18(6):333-340

[4] Wu S, Manber U. A fast algorithm for multi-pattern searching. Technical Report: TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994

[5] Allauzen C, Raffinot M. Factor Oracle of a Set of Words. Technical Report, Institute Gaspard-Monge, University, 1999.99-11

[6] Fang Y, Zhifeng C, Yanlei D, et al. Fast and memory-efficient regular expression matching for deep packet inspection. In: Proceedings of the IEEE/ACM Architecture for Networking and Communications Systems. San Jose, USA: ACM, 2006. 93-102

[7] Becchi M, Cadambi S. Memory-efficient regular expression search using state merging. In: Proceedings of the 26th IEEE International Conference on Computer Communications, Anchorage, Alaska, USA :IEEE, 2007.1064-1072

[8] Kumar S, Dharmapurikar S, Yu F, et al. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pisa, Italy: ACM, 2006. 339-350

[9] 陈曙晖,苏金树等. 一种基于深度报文检测的 FSM 状态表压缩技术. 计算机研究与发展, 2008, 42(8):1299-1306

[10] Smith R, Estan C, Jha S. XFA: Faster signature matching with extended automata. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 2008. 158-172

[11] 曹京,刘燕兵. 布尔表达式问题研究. 计算机应用研究, 2007,24(12):70-72

[12] 曹京,刘燕兵,刘萍等. 定序窗口布尔表达式匹配技术研究. 通信学报,2007,28(12):125-130

[13] Tan J L, Liu Y B, Liu P. Accelerating multiple string matching by using cache-efficient strategy. In: Proceedings of the 9th International Conference on Web-Age Information Management, Zhangjiajie, China, 2008. 539-545

[14] Smith P, Handelsman J, Goodman M. Modeling doseresponse relationships in biological control: Partitioning host responses to the pathogen and biocontrol agent. *Phytopathology*, 1997, 87(7):720-729

[15] Malkin R, Entcheva E. The mechanism of the ULV dose-response curve: a model. Study. In: Proceedings of the Computers in Cardiology, IEEE Computer Society Press, Los Alamitos, USA, 1996. 213-216

Research on complex rules matching on a large scale

Zhang shuzhuang*, Luo Hao**, Fang Binxing*,**

(* Research Centre of Computer Network and Information Security Technology, Harbin Institute of Technology, Harbin 150001)

(** Institute of Computing Technology, Chinese Academy of Science, Beijing 100190)

Abstract

In view of current security systems' needs for complex rules and the present state of the complex rules matching, this paper proposes the concept of the string expression, a new type of rule expression, and correspondingly, gives its matching method, an algorithm for complex rules matching on a large scale based on the extended finite automation. String expression can describe the logical relation and the position relation between multiple strings. The matching is achieved by using the two-level matching structure. First, it checks the existing of each string using the classical string matching algorithm, and then drives the extended finite automaton using the previous checking results. Its time complexity and space complexity are near to the classical string matching algorithm. The experimental result shows that the string expression and its matching method can provide a high matching efficiency as well as satisfy the complex request on semantic of security systems. It resolves the complex rules matching problem on the scale of 10000 better.

Key words: string matching, regular expression, string expression, extending string matching, extended finite automaton