一种面向网络安全检测的高性能正则表达式匹配算法

张树壮[1)] 罗 浩[2)] 方滨兴[1),2)] 云晓春[2)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](中国科学院计算技术研究所信息安全研究中心 北京 100097)

摘 要 目前进行正则表达式匹配的典型工具 DFA 和 NFA 都存在匹配效率和内存需求之间不可调和的矛盾,无法胜任网络安全检测中大规模正则表达式的匹配. 为了解决这个问题,文中从网络安全检测的行为特点出发,结合 DFA、NFA 模型各自的特性,提出了一种基于猜测-验证的匹配方法. 首先使用 DFA 对正则表达式中的部分子特征进行搜索,完成特征存在性的猜测;当猜测到有可能匹配某个特征后,再使用 NFA 进行验证. 文中方法既充分利用了 DFA 的高效性,减少了对相对较慢的验证过程的调用,又借助 NFA 避免了内存消耗过于巨大. 结果表明,该方法可以在大大减少内存需求的情况下,实现正则表达式的高效匹配.

关键词 特征匹配;正则表达式;有穷自动机;子特征;猜测-验证

中图法分类号 TP393 **DOI 号**: 10.3724/SP.J.1016.2010.01976

An Efficient Regular Expression Matching Algorithm for Network Security Inspection

ZHANG Shu-Zhuang[1)] LUO Hao[2)] FANG Bin-Xing[1),2)] YUN Xiao-Chun[2)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100097)

Abstract Signature matching has been one of the key techniques in network security because of its high efficiency and exactness. As the attacks and malwares on network are becoming more complex, regular expressions which are more expressive than exact strings are widely used in various security systems. However, the strong description capability also makes the matching of regular expressions faces serious challenges. Traditionally, regular expression matching is based on either DFA or NFA. However, both of them have an irreconcilable contradiction between memory requirement and processing speed, which makes them impractical on large-scale rule set. In order to address this issue, this paper proposes a matching algorithm based on guessing and verification. It first searches special sub patterns of each rule by DFA, and checks the result by NFA once the previous guessing is successful. This algorithm takes advantage of the high processing efficiency of DFA and compact representation of NFA. It can improve the matching efficiency by reducing the frequency of calling the slower verification behaviors. The result shows that this proposal can provide a high throughout as well as a moderate memory requirement.

Keywords signature matching; regular expression; finite automaton; sub pattern; guessing and verification

---

收稿日期:2010-08-22. 本课题得到国家“八六三”高技术研究发展计划项目基金(2007AA01Z406,2007AA01Z467,2007AA01Z442,2007AA01Z474)、国家“九七三”重点基础研究发展规划项目基金(2007CB311101)、国家自然科学基金(60903209)资助.**张树壮**,男,1982 年生,博士研究生,主要研究方向为网络安全、信息内容安全. E-mail: zhangshuzhuang@software.ict.ac.cn. **罗 浩**,男,1979 年生,博士,副研究员,主要研究方向为网络安全. **方滨兴**,男,1960 年生,教授,博士生导师,中国工程院院士,主要研究领域为计算机网络与信息安全. **云晓春**,男,1971 年生,研究员,博士生导师,主要研究领域为网络安全.

1 引　言

在网络上的信息随时间呈指数增长的今天，网络安全的一个重要任务就是阻止有害信息在网络上蔓延以及机密信息通过网络泄露. 使用特征匹配技术对数据进行搜索是发现这种有害信息流的基本手段之一. 在网络安全检测中，特征匹配的特点是规则数量多且对实时性要求高. 作为重要的支撑技术，匹配算法性能的好坏是影响系统性能的决定性因素之一. 对于精确字符串匹配，过去的几十年里从理论到技术都已经进行了深入的研究，并取得了重大的突破，出现了多个接近理论性能上限的算法，如 AC、WU-MANBER、SBOM 等等. 这些经典算法在网络安全检测中发挥了巨大的作用. 然而，随着网络的不断发展，攻击者也在不断地针对各种安全检测技术进行躲避和隐藏，这使得网络中的攻击和恶意代码变得越来越复杂，简单的精确字符串模式已经难以准确地描述其特征. 因此，正则表达式以其强大、灵活的表达能力，迅速成为描述新一代规则的主要工具，开源社区和商业界都开始广泛使用正则表达式来增强协议特征和安全特征的描述. 例如 Linux 应用层协议分类器(Linux Application Protocol Classifier，L7-filter①)就全部采用正则表达式来识别 100 多种应用层协议. 入侵检测系统 Snort② 的过滤规则在 2003 年以前全部为精确字符串，但最新发布的规则集中，正则表达式的比例已经超过了 40%. 入侵检测系统 Bro③ 也直接使用正则表达式来描述它的过滤规则. 在商业市场上，3Com 的 TippingPoint X505④ 以及 Cisco 的各种网络安全系统⑤，都开始使用正则表达式. 目前 Cisco 已经将基于正则表达式的内容检测集成到了其网络操作系统中.

使用正则表达式来描述攻击的特征，比传统的提取精确字符串方法更准确、方便和有效，然而其强大的能力也令它在实际应用中面临诸多的挑战. 有穷自动机(FA)和正则表达式所表示的语言都是正则语言，因此有穷自动机通常用来实现对正则表达式的匹配. 有两种典型的有穷自动机可以完成这个任务：非确定型有穷自动机(NFA)和确定型有穷自动机(DFA). NFA 的优点是空间复杂度比较低，其状态数目与正则表达式的长度呈线性关系. 然而在处理每一个字符时，NFA 必须逐个处理活动状态集合中的所有状态，匹配效率非常低. 相比之下，DFA 处理一个字符则只需要访问一个状态. 但若将每条正则表达式编译成单独的 DFA，其时间复杂度将随着规则数目的增多而增大，而将所有的正则表达式编译成一个混合 DFA 时，又会导致其空间需求大大增加，当前的硬件条件将无法满足如此大的内存需求. 例如在 L7-filter 中，当规则数目为 40 时，其 DFA 的状态数目将超过 136000，内存需求为 1.5GB 以上[1]. 由于 DFA 和 NFA 在内存需求和处理效率上的矛盾，它们都不能同时满足网络安全检测中处理大规模规则集和实时性的要求. 针对这种情况，近些年来出现了一系列的研究. UC Berkeley、Bell-Labs 以及 Google 的 Yu 等人首先系统地论述了正则表达式在网络安全检测和协议识别中的优势，对不同的匹配方法进行了分析和评价，并提出了规则改写和分组匹配等方法. 随后由于 Cisco 公司的推动，Washington University 的 Kumar 和 Becchi 等人对这个问题进行了更为深入和细致的研究，提出了 D^2FA[2]、状态合并(State Merging)[3]、H-FA[4]、混合自动机(Hybrid-FA)[5] 以及 XFA[6] 等方法来实现大规模正则表达式的实用化. 这些工作大大提高了某些特殊类型正则表达式的实用化和匹配效率. 国内对高性能正则表达式匹配技术的研究也在不断加强，哈尔滨工业大学网络与信息安全技术研究中心、清华大学、国防科技大学、中国科学院计算技术研究所以及国家互联网应急响应中心等等，都已投入到这一关键技术的研究中.

本文从网络安全检测的行为特点出发，结合 DFA、NFA 模型各自的特性，提出了一种基于猜测-验证的匹配方法. 这种方法考虑到了网络中的数据只有很少一部分是带有攻击的恶意数据，因此使用猜测过程分检出不值得怀疑的正常数据，而只对少数具有部分恶意特征的数据进行验证. 由于使用了 DFA 和 NFA 两种不同的模型来分别完成猜测和验证，本文方法可以在大大减小内存需求的情况下，实现正则表达式的高速匹配. 本文第 2 节对当前的主要研究成果进行分类介绍；第 3 节对基于猜测-验证的匹配方法的原理和实现进行详细地描述和分析；第 4 节对第 3 节提出的方法进行扩展；第 5 节给出

① Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/
② Snort 2.x.x[EB/OL]. http://www.snort.org
③ Bro Intrusion Detection System. http://bro-ids.org/Overview.htm
④ TippingPoint X505. http://www.tippingpoint.com/products_ips.html
⑤ Cisco IOS IPS Deployment Guide. http://www.cisco.com

实验结果,并与当前代表性方法的结果进行对比分析;第 6 节是最后的结论,同时对未来的工作进行展望.

2 相关工作介绍

要想对正则表达式进行实际应用,必须降低 NFA 处理一个字符所需要的时间或者减少 DFA 所需要的内存空间. NFA 的时间复杂度是由其理论模型所决定的,所以在不改变处理器体系结构的情况下,很难对其进行改进,当前的大多数研究都集中于对 DFA 的内存进行缩减,这些研究可以大致分为两类,下面就分别对其中的代表性方法进行一下简要介绍.

DFA 的主要存储消耗为一个 $n\times|\Sigma|$ 的二维转换函数表,其中 n 为状态数目,$|\Sigma|$ 为字母表大小. 它占用 DFA 空间的 95% 以上,因此缩减 DFA 的第 1 类方法是缩减 DFA 的转换表. 要缩减二维表,第 1 种方法是缩减其列数. 陈曙晖等人[7] 提出一种基于集合交割的方法来对输入字符进行预编码,文献[3]和文献[8]中则提出了等价类分割的方法. 这两种方法都是通过将整个字母表分成多个不相交子集,并使用子集来代替字母表中的元素,从而减少表的列数. 文献[9]则进一步将状态和输入字符进行联合编码来提高对字母表的压缩效果. 第 2 种缩减二维表的方法是减少表项,使其成为稀疏表. 2006 年 Kumar 等人在文献[2]中首先提出了 D^{2}FA 的方法,其基本思想是:如果两个状态 s 和 t 对于相同的字符具有相同的跳转目标,则在其中一个状态 s 中去掉所有和 t 中等价的表项,然后再引入一条从 s 到 t 的转换,称为缺省转换(default transition),它不需要输入字符也能进行跳转,这种方法可以消除大量的转换表项. Kumar 还在文献[10]中对这种方法进行了改进,提出了 CD^{2}FA,以便在引入缺省转换的情况下也能通过访问一个状态完成一个输入字符的处理. Becchi 在文献[11]中提出了另一种简单且有效的改进办法,即使用状态的深度属性生成 D^{2}FA. 在处理一个长度为 n 的字符串时,这种方法访问的状态数目不大于 $2n$. 2008 年 Ficara 等人在文献[9]中提出了 δFA 结构,这种结构中每个状态仅仅存储与其相邻状态不同的表项,而其余表项直接从其父节点继承而来. 为了进一步消除转换项冗余,文献[3]中提出了状态合并(State Merging)的方法. 其基本思想是:如果两个状态具有目标相同的转换表项(无论是否对应于相同的字符),则将两个状态合并起来,形成一个混合状态,这样就大大减少了转换表的行数. 文献[12]中提出了转换表共享方法来改进状态合并方法. 上述冗余消除的方法虽然可以消减掉 DFA 中 90% 以上的转换边,但却无法彻底解决 DFA 的空间占用问题. 因为 DFA 的内存是随状态数目呈平方级甚至指数级的增长,但转换的消除只能是线性的缩减. 因为无论怎样缩减,要想保持与原始 DFA 的等价性,都至少要保持其状态之间的连通性,即保持树形结构.

使用 DFA 进行匹配的第 2 类方法是在匹配中不再使用原始的 DFA 结构,而是使用 NFA 和 DFA 的混合结构或者改进的 DFA 结构,从而避免 DFA 状态的膨胀. Yu 等人在文献[1]中提出了将正则表达式规则集进行简单分组的思想. 而文献[13]中则提出一种更加合理的分组策略:它首先定义膨胀率 DR 来描述正则表达式的空间膨胀特性,然后基于 DR 提出一种分片算法,这种方法有效地选择出导致 DFA 状态膨胀的片段并进行隔离,从而降低了单个正则表达式的存储需求. 同时文中还基于正则表达式的组合关系提出了一种选择性分组算法. 对规则进行分组匹配实际上是在广义上使用了 NFA 的组织形式,只是每次转换的活动状态数目是一定的(等于分组的数目). 文献[5]中将这种方法进一步推广和细化,提出了 Hybrid-FA. 其特点就是首先将整个规则集编译成一个 NFA,然后只将一部分 NFA 转化成 DFA,形成 DFA 状态和 NFA 状态同时存在的情形,匹配时对两种不同的状态使用不同过程. H-FA[4] 和 XFA[6,14] 则引入了额外的信息来对 DFA 的匹配过程进行记录,从而大大减少了 DFA 为了记录各种不同情况所需要的状态数目. 文献[4]还提出一种方法,依据字符出现的概率将正则表达式划分为前缀和后缀两部分,并使用不同的自动机(fast path automaton and slow path automaton)分别进行处理,通过减少对速度较慢的后缀匹配的调用来提高整个系统的匹配效率.

本文的方法借鉴了上述的研究成果,但又不同于上述研究. 本文方法分为猜测和验证两个步骤,分别由单独一个 DFA 和一组 NFA 来完成,但是并不将它们编译在一个统一的结构中,而是在 DFA 的接受状态和 NFA 之间建立联系,以便完成猜测后的验证. 本文方法中也使用额外的信息来辅助匹配,但并不试图使得到的 DFA 和经典方法构建的 DFA

完全等价，而是为了针对网络安全检测的特点，将同一个正则表达式的两个邻接子特征的匹配结果关联起来，使猜测更准确并保证匹配结果的正确性.

3　基于猜测-验证的正则表达式匹配

3.1　网络安全检测中的正则表达式匹配

网络安全中使用正则表达式作为匹配规则，是为了能够更准确地捕获各种攻击和恶意代码的特征，这与用它来表示形式化语言的初衷有着很大的不同，这些不同也正是本文的基本出发点. 首先，检测规则更加注重描述攻击行为的多个阶段或恶意代码多个部分之间的逻辑关系与位置关系，所以网络检测类的应用中所使用的规则一般都可以明显地分成若干个部分，如图 1 所示. 图中每个部分表示一个完整规则的子特征，下文中称为 sub_pattern，子特征之间用正则表达式的运算符连接起来；第二，网络安全检测所关心的结果并不是两种不同形式语言的等价性，而是更注重验证正则表达式所描述特征的存在性. 因此这里的匹配更确切的说法应该是搜索（为了保持一致，下文仍称之为匹配）. 第三，在网络检测中，通常对要检验的数据没有任何先验知识，因此不知道在输入数据中是否存在一个子串能够命中规则集中某个正则表达式，也不知道这个子串从哪里开始. 通常的做法是假定匹配子串可能在任意位置出现，从而在每个不是以“^”开始的正则表达式前面加上“.*”操作符，但这种做法也导致了自动机所需要内存的增多. 第四，网络检测中，匹配是一个短暂但频率很高的动作，通常是以包或者流为输入单位，并且只有极少数输入才能完全命中某个完整特征，绝大多数被检测的数据都是正常数据，不含有任何目标特征.

图 1　安全规则的示意图（⊕表示正则运算符）

3.2　基于猜测-验证的匹配算法

DFA 和 NFA 的结构都可以看成是一个有向图，其中状态构成图的顶点，状态之间的转换构成有向边. 上面的分析中提到，在网络安全检测中只有极少数输入才会命中特征. 假设初始状态的深度为 0，那么这个特点反映在具体的匹配过程中则表现为大部分遍历行为都只是访问图的浅层顶点，这部分顶点只占整个图的一小部分. 深度比较大的顶点则必须当某个输入和特征吻合程度较大乃至完整匹配到某个特征时才会被访问. 直接使用原始 DFA 来进行大规模正则表达式匹配之所以不可行，就是因为它需要将所有的正则表达式完整地表示在一个结构中，而安全规则中将多个 sub_pattern 连接成一个整体的时候使用了特殊的操作符，这些操作符导致了 DFA 状态数目的剧烈增长，从而带来在现有资源下无法接受的内存消耗. 这种将规则全部编译到一个 DFA 中的行为，不仅造成了 DFA 的内存需求不可接受，同时也不符合当前计算机结构的分级内存体系特点，从而降低处理效率. 因为匹配类算法的过程主要是通过查表来进行，它的时间消耗取决于访存的次数和 cache 的使用效率[15].

为此本文提出了基于猜测-验证的匹配方法：它首先从整条规则中还原出各个 sub_pattern，然后以 sub_pattern 的出现为依据来猜测一个输入是否有可能匹配某个特征. 只有某个输入通过猜测后，才进行最后的完整性验证. 验证过程主要是检验各个 sub_pattern 之间是否符合一定的逻辑和位置关系，即处理图 1 中所示的连接符号. 本文主要针对“.*”、“.{n}”、“[^$c_1c_2\cdots c_k$]*”和“[^$c_1c_2\cdots c_k$]{n}”4 类连接符. 因为以往的大量研究表明，正是这些操作符导致了在使用子集构造法从 NFA 生成 DFA 时状态数目的剧烈增长. 这种方法有两个好处：(1) 在提取的时候选择那些简单形式的 sub_pattern，从而可以将这些 sub_pattern 放在一起形成一个混合的 DFA 结构而不会引起巨大的内存需求，使猜测过程可以依靠 DFA 快速完成. (2) 猜测过程可以将大部分输入过滤掉，只剩下少数能命中某条规则中所有 sub_pattern 的数据才需要进行完整的验证过程，因此可以使用速度相对较慢但内存需求对操作符不敏感的 NFA 来完成.

为了描述方便，规定：R 表示正则表达式集合，r 表示 R 中的某一条正则表达式，即$\{r_j \in R \mid 1 \le j \le |R|\}$. M 表示每个正则表达式中 sub_pattern 的数目，$s_{j,k}$表示 r_j的第 k 个 sub_pattern（$1 \le k \le M$）. S 表示 sub_pattern 组成的集合，而由只属于 r_j 中的 sub_pattern 组成的集合则表示为 S_j. 由 r_j 形成的 NFA 表示为 s_NFA_j，所有 s_NFA 形成的集合称为 S_NFA. 字符串用 T 表示，T 的子串用 t 表示，不同的子串用下标区分. 基于猜测-验证的正则表达式匹配算法其预处理和匹配过程如算法 1 和算法 2 所示.

算法 1. pre_process(R).

//预处理过滤规则,构建猜测用的 DFA 和验证用的 NFA

1. $S=\varnothing$, $S\_NFA=\varnothing$
2. for (each $r_j \in R$) do
3. $S_j = extract\_sub(r_j)$
4. $S = S \cup S_j$
5. for(k=1 to M) do
6. if($s_{j,k} \in S_j$) then
7. $set\_map(map_j, 1, k)$
8. end if
9. end for
10. $s\_NFA_j = build\_NFA(S_j)$
11. $S\_NFA = S\_NFA \cup s\_NFA_j$
12. end for
13. $DFA = build\_DFA(S)$

算法 2. Match(DFA, S_NFA, T).

//对输入数据 T 进行匹配

1. for(each $p \in T$) do
2. $state = dfa\_match(DFA, p)$
3. if($state \in DFA.accept\_states$) then
4. for(each s_NFA_j in $state.rule\_list$) do
5. $set\_bit(map_j, 0, state.index\_list[j])$
6. if($map_j == 0$) then
7. $nfa\_match(s\_NFA_j, T)$
8. end if
9. end for
10. end if
11. end for

算法 1 给出了预处理过程.对于每个正则表达式 r_j,首先从中抽取出若干个 sub_pattern 作为猜测条件.为了对同一正则表达式中的多个 sub_pattern 进行记录,算法中使用了一个 bitmap 型变量 map 来追踪各个 sub_pattern 的出现情况.如果一个 r_j 的第 k 个 sub_pattern 被抽取作为猜测条件,那么 map 的第 k 个 bit 则会被 set_map 函数设置为 1;预处理过程为每个 r_j 构建一个 s_NFA_j,并在最后使用选取出来的全部 sub_pattern 构建一个混合 DFA.需要注意的是,不同的正则表达式中可能存在相同的 sub_pattern,所以一个 sub_pattern 可能与多个正则表达式相关联.为了记录这种关联,在 DFA 的接受状态中增加了两个链表:rule_list 和 index_list. rule_list 用来记录包含这个接受状态所代表的 sub_pattern 的所有正则表达式,index_list 用来记录这个 sub_pattern 在相应 map 中的索引位置.

算法 2 对匹配过程进行了描述.对于一个输入字符串 T,匹配过程首先使用算法 1 中构建的 DFA 逐个处理每个字符,如果访问到了一个接受状态,则更新其对应 rule_list 中所有规则的 map 结构,但只有在同一个输入 T 中找到某个 map 所对应的全部 sub_pattern 后才会将这个数据交由相应的 s_NFA 进行完整的验证.需要注意的是:同一个表达式中的 sub_pattern 不需要按顺序出现,且预处理中生成的 map 结构并不参与真正的匹配过程,而是在每次匹配前作为模板来生成对应的实例,每次匹配过程的操作都是在其实例上进行.

3.3 算法分析

3.3.1 正确性证明

在网络安全应用中,匹配就是在给定规则集 R 和文本 T 的情况下,找出 T 中是否存在一个或者多个恰好是正则表达式 r_j($r_j \in R$)所表示的语言的子串 t.如果 T 的某个子串 t_i 是 r_j 表示的语言,则称 t_i 匹配 r_j.如果能够在 T 中找到 t_i 匹配 r_j,则称 T 被命中.下面证明本文提出的基于猜测-验证的匹配结果与使用 NFA 匹配的结果相同.

定理 1. T 在基于猜测-验证的匹配中被命中当且仅当它在 NFA 的匹配中被命中.

证明. 由于猜测-验证法在最后一步使用了 NFA 进行结果验证,因此能够被猜测-验证法命中的 T 必然也会被 NFA 命中.

假设一个输入 T 能够被 NFA 命中.不妨设子串 t_i 匹配了 r_j,那么 t_i 必然可以匹配所有的 $s_{j,k} \in S_j$. 而预处理过程中从 r_j 中抽取的 sub_pattern 集合是 S_j 的子集,因此 t_i 必然可以匹配 r_j 的所有猜测条件,因此 T 会被输入到 s_NFA_j 中进行验证.所以 T 必然也会被猜测-验证法命中.当 T 中出现多个匹配子串时,情况与此类似.得证. 证毕.

3.3.2 算法性能分析

下面对猜测-验证法进行一下讨论和分析,给出影响其性能的因素和优化策略.从算法 2 中可知,本文方法对一个输入的处理由两部分组成:使用 DFA 来完成的猜测过程和使用 NFA 来完成的验证过程.假设对于输入字符串 T,使用 DFA 处理它所需要的时间为 $Dt(T)$,使用 NFA 处理它所需要的时间为 $Nt(T)$,处理过程中验证过程被调用的概率为 P,则处理 T 所需要的总时间为

$$Dt(T)+PNt(T) \tag{1}$$

从式(1)中可以看出，在给定规则集和输入 T 的情况下，基于猜测-验证匹配算法的时间复杂度的决定性因素在于验证过程的调用概率 P，当 P 很小时，猜测-验证过程的效率将接近 DFA.

进行猜测-验证时，当且仅当预处理时从同一个正则表达式中提取出来的所有 sub_pattern 都被命中时，才会将数据送入相应 NFA 中进行一次完整的验证过程. 假设从某一条规则 r_j 中选择了 m 个 sub_pattern，且 $s_{j,k}(1\leq k\leq m)$ 被命中的概率为 $p_k(0<p_k<1)$. 则数据会被规则 r_j 生成的 NFA 进行验证的概率 P_{r_j} 可以用式(2)来表示，而在一次处理过程中验证过程被调用的概率 P 可以用式(3)来表示.

$$P_{r_j}=\prod_{k=1}^{m}p_k \tag{2}$$

$$P=\sum_{i=1}^{|R|}P_{r_i} \tag{3}$$

可见，预处理过程中选取出来的 sub_pattern 数目 m 越大或者每个 sub_pattern 的命中率 p_k 越小，那么调用 NFA 进行验证的概率 P 越小. 因此在进行预处理时，要遵循以下两条原则：

(1) 优先选取那些较长的纯字符串作为 sub_pattern. 因为一个字符串被命中的概率通常取决于它的长度以及它所包含字母的出现概率和分布. 通常认为每个字符的出现概率是相同的(每个字符出现的概率是 1/256)，因此一个串的长度越长，其命中的概率越小. 同时，纯字符串不会引起猜测用的 DFA 的内存需求剧烈增长.

(2) 要选取尽量多的 sub_pattern 参与猜测. 但需要注意的是，检测规则中，一些词组的出现概率可能要更高一些，特别是一些协议控制字段. 例如在 snort 系统的 backdoor 规则集中“server”出现了 24 次. 如果选取它作为 sub_pattern，那么在每次命中这个 sub_pattern 时，都要对 24 条规则的 map 进行逐个更新. 这会严重影响处理速度，因此不能入选作为 sub_pattern.

4　定序的猜测-验证

算法 2 是一种理想的匹配方式，因为需要经 NFA 验证的数据将会极少，绝大部分正常数据都会在经过 DFA 处理之后直接过滤掉，其平均的处理效率与 DFA 相当. 然而，使用这种方法有一个前提：整个特征必须出现在同一次输入数据中，例如短信息内容过滤或者病毒检测等应用. 若是特征分散在两次或者多次输入中，就必须对数据进行缓存，否则将无法完成验证过程. 在实际的网络中，安全检测通常是要以流为单位的，而各个流的数据包到来顺序是不确定的. 为了能对分散到达的数据进行正确的匹配，有两种解决办法：(1) 将流进行缓存；(2) 改进匹配算法，使得验证部分以只需要猜测成功后的“剩余”数据，而不需要进行回溯. 在高速网络环境下，以当前的硬件条件对数据包进行缓存显然是不可能的，因此，本文将简单的 bitmap 进行扩展，为上文中提到的 4 种连接符分别设计了一种 e-map(extended map)结构，来实现无回溯的定序猜测-验证.

4.1　e-map 结构

对 Thompson 构造法进行分析可以知道，如果不考虑“失效”转换的话，则 NFA 的结构类似于一棵树，且其接受状态总是出现在“叶子”节点处. 在 NFA 中命中一个特征的过程是不需要回溯的，因为通向一个接受状态的路径只包括向深度更大的状态的转换以及向当前状态自身的转换. 因此要想在猜测成功之后只需要对“剩余”的数据进行验证，一个简便的办法就是只从每个表达式中取一个简短的前缀作为对本条规则的猜测条件，但是对于某些正则表达式来讲，这会大大提高其猜测成功的可能性，从而频繁地调用验证过程，降低系统的效率. 为此本文针对上文中提出的 4 类不同连接操作符，将原来的 bitmap 中的 bit 扩展成一个 e-map 结构，下面分别对其进行描述.

(1)“.*”在规则中的实际意义是要求被连接的两个 sub_pattern 按顺序出现，而不管它们之间的距离和字符. 因此它不需要额外的字段来记录其它信息，其本身的出现就表明了条件已满足.

(2)“[^$c_1c_2\cdots c_k$]*”要求被连接的两个 sub_pattern 按顺序出现，并且它们之间不能出现集合 $\{c_1,c_2,\cdots,c_k\}$ 中的字符. 例如有些攻击必须使特殊数据出现在同一行，而不能被/n 和/r 分隔开. 不过这个连接符只是关注这些字符是否出现过，而不管它们出现的次数和位置. 因此为它增加一个表示字母表的位图(alphabet_bitmap)，位图中每个 bit 表示字母表中的一个字符. 在匹配过程中，通过对这些 bit 进行置位来表示其对应的字符是否出现过.

(3)“.{n}”要被连接的两个 sub_pattern 按顺序出现并且它们之间要有 n 个字符的距离. 但并不对字符的种类进行要求. 因此为它增加一个距离计数器 distance，初始值为 n，在匹配过程中对其进行更

新操作来表示是否满足距离要求.

(4)"$[\hat{}c_1c_2\cdots c_k]\{n\}$"不但要求被连接的两个sub_pattern按顺序出现以及二者之间要有n个字符的距离,还要求间隔的n个字符中不能出现集合$\{c_1,c_2,\cdots,c_k\}$内的字符.因此它需要在第3类e-map结构的基础上再增加一个表示字母表的位图(alphabet_bitmap).在匹配过程中,不但要对distance做记录,还需要对位图中的bit进行置位以记录字符的出现情况.

4.2 定序的猜测-验证匹配

有了针对每一种连接操作符的e-map结构,预处理时就可以用和算法1相似的过程来构建猜测用的DFA和验证用的NFA,在此不再详述.下面主要介绍一下猜测和验证的过程.首先给出如下定义.

定义1. 在预处理所形成的DFA中,每个正则表达式中被抽取的最后一个sub_pattern所形成的接受状态,称为终结接受状态,其余的接受状态称为中间接受状态.

定义2. 在预处理所形成的NFA中,最后一个sub_pattern的下一个字符所形成的状态称为开始状态.

因为算法要处理多个流并发的情况,因此每个流都有一个属于自己的e-map实例列表.当这个流中的数据有效命中了某一个sub_pattern的时候,就会将这个sun_pattern所关联的全部e-map模板拷贝一份放在这个流的e-map列表中形成实例,并赋予每个实例一个唯一的id.对于同一个流来讲,如果重复命中一个sub_pattern,则有可能导致多次添加同一e-map模板的实例,但这些e-map实例的id是不同的.一旦某个e-map实例在处理某个流的过程中被创建并添加到这个流的e-map列表中,那么在处理这个流随后的数据时,也会同时对这个e-map实例的状态进行更新,直到它被从列表中移除.每一种e-map实例的移除条件如下:第1类e-map实例只有在对这个的流处理结束时候才会被移除.第2类e-map实例被创建后,会在遇见第一个属于其alphabet_bitmap中的字符时被移除.第3类e-map实例被创建之后,会在连续处理distance个字符之后被移除.第4类e-map实例被创建之后,会在第一次出现属于其alphabet_bitmap中的字符时或者连续处理distance个字符之后被移除.位于每个正则表达式开始处的sub_pattern的每一次命中都直接算作有效命中,而其它sub_pattern被命中时,首先要查看这个流的e-map列表中是否还有由其前驱所创建的e-map实例.如果存在,说明其前驱已经被命中过,并且如果这两次命中之间的字符满足它们之间连接操作符的限定关系,就将其设置为有效命中,并创建它所对应的e-map模板的实例.如果不存在,说明其前驱没有被命中过或者中间间隔的字符已经不满足它们之间的限定关系,因此为无效命中,不做其它处理.

一旦终结接受状态被有效匹配,说明已经完成了一次成功的猜测,需要进行下一步的验证过程.需要注意的是,每次验证过程不是从NFA的初始状态开始,而是从NFA的开始状态开始进行.由于只需要验证剩余的字节流是不是能够匹配正则表达式被提取的最后一个sub_pattern之后的部分,因此在NFA的每一步中,会排除活动状态集合中位于开始状态之前的状态,因为位于开始状态之前的匹配会在猜测中完成.如果某一步所得到的活动状态集合全部位于开始状态之前,那么就可以终止验证过程.在验证过程中,如果访问到NFA的接受状态,则表示验证成功,当前数据流成功地命中了一个特征.

5 实验结果与分析

高性能正则表达式匹配算法的主要评价指标有两个:处理速度和内存需求.本节将在实际规则集上给出本文方法在这两方面的性能评测结果,并与Hybrid-FA方法的结果进行比较和分析.实验中使用RegEx[①]工具集将正则表达式转化成DFA、NFA以及生成对应规则集的trace数据.

5.1 正则表达式规则集分析

实验采用的正则表达式来自Snort入侵检测系统和Linux Layer-7 filter(L7).虽然Snort系统中有数千条正则表达式规则,但是它们并不会同时被启用,因为Snort是在分析完数据包的头部之后才进行内容部分的检查.本文中选取了其中的两大类:web-misc和backdoor.本文中将"x+"类型的符号都改写成了等价的"xx*"的形式,并在统计时将".{m,n}"与".{n}"视为同类.下面在表1中给出3个规则集各自的特性.

① Regex http://regex.wustl.edu/index.php/Main_Page

表 1　不同规则集的特性

规则集名称	规则数目	各种操作符的数目	含有 4 种操作符之一的规则数目	猜测使用的 sub_pattern 数目	定序猜测 sub_pattern 数目	原始 NFA/DFA 状态数目
web-misc	55	3/75/0/31	53	79	73	7615/＞3000000
backdoor	153	9/388/0/10	130	356	261	3364/＞3000000
L7	89	23/33/23/3	57	142	133	1450/＞3000000

表中第 2 列给出了每个规则集的规则数目，规模从 50 条递增到 150 条左右. 第 3 列给出了规则集中 4 种连接操作符的出现次数. 4 个用“/”号分开的数目顺次对应“.*”、“[^$c_1 c_2 \cdots c_k$]*”、“.{n}”、“[^$c_1 c_2 \cdots c_k$]{n}”. 可以看出，在 Snort 规则集中，出现最多的是[^$c_1 c_2 \cdots c_k$]* 类型的操作符. 特别是 backdoor 规则集中，这种操作符在每条规则中出现将近 3 次. 而 L7 规则集中前 3 种操作符出现的次数都比较多. 第 4 列中给出了每个规则集中至少带有一个连接操作符的规则数目. 可见，连接操作符在实际的过滤规则中是十分普遍的，特别是在 Snort 的两个规则集中，占了 90%左右. 第 5、6 列给出了本文实验中所选取的 sub_pattern 的数目，选取的原则如第 3.3.2 节所述. 最后一列则给出了用这些规则直接构建有穷自动机时，所得到的 NFA 和 DFA 状态数目. 表中 DFA 的状态数目为一个范围，因为三组规则在构建过程中都因为状态超过 3000000（内存需求超过 3GB）而无法构建成功，可见当表达式中包含有特殊连接符时 DFA 所需要的内存是十分巨大的. 需要注意的是，虽然 web-misc 规则集比较小，但是它所形成的 NFA 自动机数目是最多的，这是因为在 web-misc 的规则中，出现了很多第 4 种类型([^$c_1 c_2 \cdots c_k$]{n})的连接符，并且 n 的值都比较大（最大的为 1024），因而出现了大量的重复状态.

5.2　性能评价和分析

下面给出本文方法的内存需求以及处理速度，并与 Hybrid-FA 方法在相同条件下的结果进行对比.

5.2.1　内存需求

表 2～4 中分别给出了本文方法和 Hybrid-FA 方法在完成预处理后的内存占用情况. 对于本文方法，表中列出了猜测用的混合 DFA 的状态数目，每条规则形成的原始 NFA 的状态数目之和以及 *map* 模板的数目. 对于 Hybrid-FA，则给出了 Head-DFA 和 Tail-FA 的状态数目.

表 2　web-misc 规则上的内存需求对比

方法	NFA 状态数	DFA 状态数	总的状态数目	*map* 数量
猜测-验证	7952	763	8715	55
定序猜测-验证	7952	715	8667	18
Hybrid-FA	6714	29343	36057	—

表 3　backdoor 规则上的内存需求对比

方法	NFA 状态数	DFA 状态数	总的状态数目	*map* 数量
猜测-验证	4725	2993	7718	153
定序猜测-验证	4725	2160	6885	108
Hybrid-FA	2539	30936	33475	—

表 4　L7 规则上的内存需求对比

方法	NFA 状态数	DFA 状态数	总的状态数目	*map* 数量
猜测-验证	1842	2603	4445	89
定序猜测-验证	1842	6244	8086	43
Hybrid-FA	1091	8249	9340	—

从表中数据可以得到如下结论：

(1) 自动机所占用的空间更多地取决于规则集的特性，即连接符的种类和出现频率对整个内存需求的影响远远大于规则集中规则数目的影响. 例如 web-misc 的规则数目最少，但是它需要的自动机状态数目最多. 通过与表 1 中的数据相比较还可以发现，当每条规则单独构建 NFA 时，总的状态数目比构建混合 NFA 略大，因为此时各个 NFA 不能共享公共前缀.

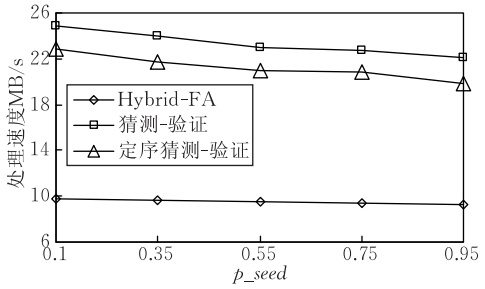
(2) 猜测-验证法和 Hybrid-FA 所需要的内存要远远低于原始 DFA，前两者仅仅相当于后者的 1%. 同时，在 L7 规则上，本文方法的内存需求为 Hybrid-FA 的 50%左右，对于 Snort 规则，前者则仅仅相当于后者的 25%左右. 这是因为 Hybrid-FA 对规则类型的敏感度要大于本文的方法. 它只在特定的连接符处才会中断子集构造过程，若连接符出现的位置比较靠后，它就不得不扩大确定化的范围，从而造成 Head-DFA 比较巨大. 而本文则通过灵活的 sub_pattern 提取策略避免了对连接符的处理.

(3) 如果都是用简单形式的 sub_pattern，那么本文的两种方法所形成的 subs_DFA 的大小相当. 但由于定序使用的 sub_pattern 数目较少（只能按序选择），所以其对应的 subs_DFA 状态数目也较少. 表 2 和表 3 中两个 Snort 规则集就属于这种情况. 但是在某些情况下，由于规则的开头不是简单串，因此需要在进行定序猜测中不得不选取部分带有连接符的 sub_pattern，此时它将比简单猜测需要更多的空间. L7 中的规则主要描述协议的格式，因此出现了较多此类情况.

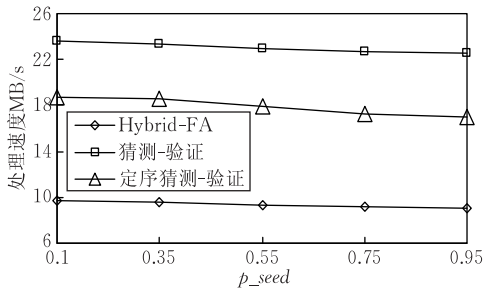
5.2.2　处理速度

本节中将对 Hybrid-FA 与本文方法的处理速

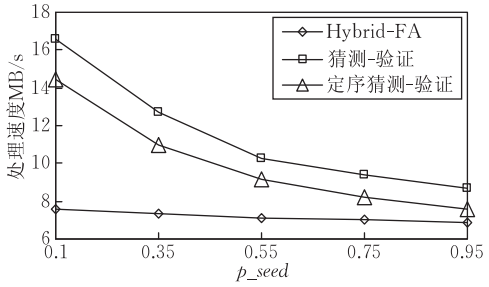
度做一下对比. 实验中所使用的测试数据由 regex 的 trace 生成工具产生. 这个工具可以为指定的正则表达式规则集生成具有一定特性的 trace 数据. 为了做一个全面的对比,本文选取了 5 个不同的参数($p\_seed=0.1,0.35,0.55,0.75,0.95$)为 3 个规则集分别生成了一组 trace 数据集. 其中 p_seed 表示在处理过程中对自动机中深层节点的访问概率,p_seed 越大,访问深层节点的概率越大. 两种方法在相同规则集和 trace 数据集上所得到的实验结果如图 2 所示.



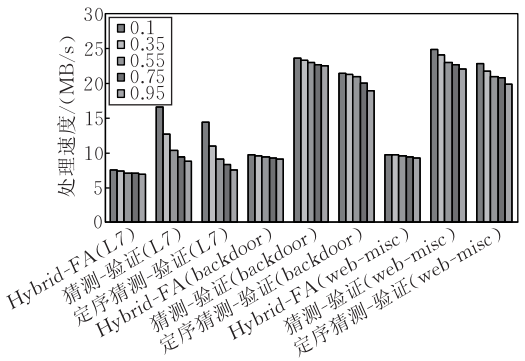
(a) web-misc规则上的处理性能对比



(b) backdoorc规则上的处理性能对比



(c) L7规则上的处理性能对比



(d) 两种方法的整体性能对比

图 2 两种方法的处理效率对比

图 2 中(a)~(c)分别给出了两种方法在相同规则集上处理不同特性 trace 数据时所得到的对比结果. 而图(d)则给出了整体的性能对比结果. 从图中可以看出,在所有的规则集上本文方法的处理速度都高于 Hybrid-FA,特别是在 web-misc 和 backdoor 规则集上,性能都提高了一倍左右. 这可以归结为两个原因:(1)这两个规则集中的连接符更复杂一些,因而 Hybrid-FA 只能选取很短的前缀构建 head-DFA,这就使得它的 tail-NFA 部分被激活得更频繁一些. (2)本文方法构建的 subs_DFA 的状态数目要远远小于 Hybrid-FA 中 head-DFA 的状态数目,因此其 cache miss 也更少一些,减少了由于访存而带来的时间延迟.

通过对本文的两种方法结果的对比可以看出,在 L7 和 web-misc 规则集上,猜测-验证与定序猜测-验证的处理速度相近,而在 backdoor 规则集上,猜测-验证方法的效率则要明显高于定序猜测-验证. 这是因为在 L7 和 web-misc 两个规则集中,两种方法所选取的 sub_pattern 数目几乎相同,且大部分规则中只取一条. 此时二者都会在命中一个 sub_pattern 后直接进行验证过程,因此具有相近的处理速度. 而在 backdoor 规则集中,猜测-验证方法则选取了比定序-猜测多将近一倍的 sub_pattern 用于猜测,因而其调用验证过程的频率相对较少,处理速度也更高

本文的两种方法在 Snort 规则集上的处理速度都随着 p_seed 的增加而略有降低,但在 L7 规则集上本文方法的性能退化相比 Hybrid_FA 则严重得多,同时本文方法在 L7 规则集上的处理速度也低于在 Snort 规则集上的处理速度. 这是因为 Snort 规则集是针对攻击的特征集,而 L7 规则集是针对协议类型的,因此 L7 规则集中的特征命中频率要远远高于 Snort 规则集,其验证过程被调用的频率也相应地高一些,从而导致处理的速度较慢. 这也表明了本文方法在面向网络安全检测的应用上更有优势.

6 结束语

本文首先阐述了正则表达式在网络安全检测中的重要性及其匹配所面临的严峻挑战. 然后针对网络安全检测应用本身的特点以及 DFA、NFA 各自的特性,提出了基于猜测-验证的匹配方法. 这种方法首先使用 DFA 对正则表达式中的 sub_pattern

进行搜索，完成特征存在性的猜测；在猜测到有可能匹配某个特征后，再使用 NFA 来完成少量但复杂的逻辑关系和距离关系验证. 文中还对这种方法进行了深入的分析，给出了其性能的估算方法并指出了影响其性能的主要因素. 这种方法充分地利用了安全检测中成功命中的稀疏性，结果表明，本文提出的方法可以在大大减小内存需求的情况下，实现大规模正则表达式的高速匹配.

虽然正则表达式具有很强的描述能力，但它毕竟不是专门为网络安全设计的，在描述特征时可能会放大问题的范围，从而给匹配造成困难. 作者的下一步方向是设计出更适合表示安全特征的规则，并给出高效的匹配算法.

参 考 文 献

[1] Yu Fang, Chen Zhifeng, Diao Yanlei et al. Fast and memory-efficient regular expression matching for deep packet inspection//Proceedings of the IEEE/ACM ANCS. San Jose, California, 2006: 93-102

[2] Kumar S, Dharmapurikar S, Yu F et al. Algorithms to accelerate multiple regular expressions matching for deep packet inspection//Proceedings of the ACM SIGCOMM. Pisa, Italy, 2006: 339-350

[3] Becchi M, Cadambi S. Memory-efficient regular expression search using state merging//Proceedings of the IEEE Infocom. Anchorage, Alaska, 2007: 1064-1072

[4] Kumar S, Chandrasekaran G, Turner J et al. Curing regular expressions matching from insomnia, amnesia and acalculia//Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems. Orlando, Florida, USA, 2007: 155-164

[5] Becchi M, Crowley P. A hybrid finite automaton for practical deep packet inspection//Proceedings of the ACM CoNEXT. New York, 2007: 1-12

[6] Smith Randy, Estan Cristian, Jha Somesh et al. Fast signature matching using extended finite automaton (XFA)//Proceedings of the ICISS Hyderabad. India, 2008: 158-172

[7] Chen Shu-Hui, Su Jin-Shu, Fan Hui-Ping et al. An FSM state table compressing method based on deep packet inspection. Journal of Computer Research and Development, 2008, 42(8): 1299-1306(in Chinese)
(陈曙晖，苏金树，范慧萍等. 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)

[8] Kong Shijin, Smith Randy, Estan Cristian. Efficient signature matching with multiple alphabet compression tables//Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks. Istanbul, Turkey, 2008: 1-10

[9] Ficara D, Giordano S, Procissi G et al. An improved DFA for fast regular expression matching. ACM SIGCOMM Computer Communication Review, 2008, 38(5): 29-40

[10] Kumar S, Turner J, Williams J. Advanced algorithms for fast and scalable deep packet inspection//Proceedings of the IEEE/ACM ANCS. San Jose, California, 2006: 81-92

[11] Becchi M, Cadambi S. An improved algorithm to accelerate regular expression evaluation//Proceedings of the IEEE/ACM ANCS. Orlando, Florida, 2007: 145-154

[12] Zhang Shuzhuang, Luo Hao, Fang Binxing et al. Fast and memory-efficient regular expression matching using transition sharing. IEICE Transactions on Information and Systems, 2009, E92-D(10): 1953-1960

[13] Xu Qian, E Yue-Peng, Ge Yue-Guo et al. Efficient regular expression compression algorithm for deep packet inspection. Journal of Software, 2009, 20(8): 2214-2226(in Chinese)
(徐乾，鄂跃鹏，葛敬国等. 深度包检测中一种高效的正则表达式压缩算法. 软件学报，2009，20(8)：2214-2226)

[14] Smith Randy, Estan Cristian, Jha Somesh. Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. ACM SIGCOMM Computer Communication Review, 2008, 38(4): 207-218

[15] Tan Jianlong, Liu Yanbing, Liu Ping. Accelerating multiple string matching by using cache-efficient strategy//Proceedings of the 9th International Conference on Web-Age Information Management. Zhangjiajie, China, 2008: 539-545

ZHANG Shu-Zhuang, born in 1982, Ph. D. candidate. His research interests include network security and information content security.

LUO Hao, born in 1979, Ph. D., assistant professor. His research interests focus on network security.

FANG Bin-Xing, born in 1960, professor, Ph. D. supervisor, member of Chinese Academy of Engineering. His research interests include computer network and information security.

YUN Xiao-Chun born in 1971, professor, Ph. D. supervisor. His research interests focus on network security.

Background

This research mainly focus on defending the various evading and hiding behaviors in the content filtering and network intrusion detection. Regular expressions can describe filtering rules more exact, efficient and convenient than traditional simple strings. It can integrate different format rules into unified format and help implement a universal matching engine in UTM device. However, the strong description capability also makes the matching of regular expressions faces serious challenges. Traditionally, regular expression matching is based on either DFA or NFA, but both of them have an irreconcilable contradiction between memory requirement and processing speed, which make them impractical on large-scale rule set. Recent years, most researches focus on reducing the memory requirement of DFAs. However, DFA memory requirements increase exponentially but DFA memory reduction efficiency is linear. Furthermore, some these proposals have constraints which limit their applicability. Therefore, DFA memory reduction techniques cannot resolve memory explosion thoroughly.

In order to make large-scale regular expressions matching practical, this paper proposes a matching algorithm based on guessing and verification. it takes advantage of the high processing efficiency of DFA and compact representation of NFA. It first searches special sub patterns of each rule by efficient DFA, and checks the result by slower NFA once the previous guessing is successful. The result shows that this proposal can provide a high throughout as well as a moderate memory requirement.

This research is partly supported by the National High Technology Research and Development Program (863 Program) of China (grant Nos. 2007AA01Z406, 2007AA01Z467, 2007AA01Z442, 2007AA01Z474), National Basic Research Program (973 Program) of China (No. 2007CB311101), National Natural Science Foundation of China (grant No. 60903209).