基于 Snort 的入侵检测系统安全性研究

袁 晖
(深圳信息职业技术学院软件工程系 深圳 518029)

摘 要 Snort是一个成熟的开放源代码的网络入侵检测系统,所以这使得Snort的规模越来越庞大,但是随之也产生了安全缺陷。本文主要讨论了提高Snort的安全性的两种方法:规则优化和匹配模式优化。

关键词 Snort,规则优化,匹配模式

Safety Research of Instruction Detection System Based on Snort

YUAN Hui
(Department of Software Engineering, Shenzhen Institute of Information Technology, Shenzhen 518029)

Abstract Snort is a mature open source code network invasion detection system, this causes the scale of Snort more and huger, at the same time causes the shortage of security along with it. The paper mainly discusses two methods of improving the safety of Snort, they are rule optimization and match pattern optimization.

Keywords Snort, Rule optimization, Match pattern optimization

1 引言

Snort是一个成熟的开放源代码网络入侵检测系统[1]。Snort能设置实时发送告警信息。提供数据包嗅探和记录功能只是Snort的部分功能,Snort的重点是其入侵检测功能——根据入侵规则匹配数据包中的内容。通常称其为轻量级的网络入侵检测系统。这里"轻量级"的意思是占用的资源非常少[2],能运行在多种不同的操作系统上。另外,Snort还提供一些以前只有商业入侵检测系统才能提供的功能。Snort的流行也得益于日渐流行的Linux和其他免费操作系统[3],例如BSD系列的NetBSD、OpenBSD和FreeBSD等。虽然Snort是免费的基于开放源代码的软件,但这并不妨碍它在其它的免费的操作系统上应用,Snort能够在Solaris、HP-UX、IRIX甚至Windows操作系统上运行。当越来越来多的安全机制加入到Snort中时,Snort就变得越复杂。错误或缺陷就不可避免地引入到其中,在系统设计阶段及实现、维护和升级都会遇到这些问题。彻底检查Snort中所有的错误或缺陷已变得愈加困难,入侵者就有可能利用这些缺陷来进行攻击目标系统或Snort本身。本文主要研究的是针对Snort的安全隐患如何提高Snort的安全检测能。

2 规则优化

规则优化[4]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。为了有效减少对每个数据包进行匹配的平均的规则条数,规则优化必须满足下面两个条件:

1. 尽可能创建最小且最有效的规则集;

2. 创建离散的规则集,从而使对于每个到达的数据包只需检测一个规则集。

规则优化的关键是选择合适的特征参数作为划分集合的依据。协议解析的过程实际上就是沿着协议树,从根结点寻找叶子节点的过程,并用相关的数据填充Packet数据结构。所以我们可以选择一些协议特征作为划分集合的依据(如端口,协议选项等)。协议树示意图如图1所示。

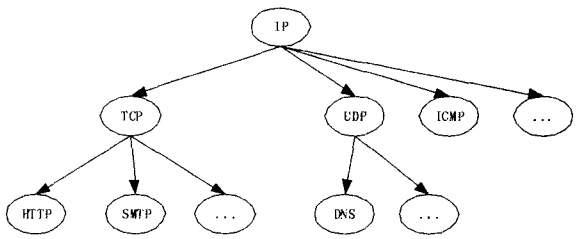

图1 协议树示意图

下面我们按Snort检测使用的协议分类详细讨论优化方法。

1. 特征参数的选取

· TCP/UDP协议

TCP/UDP协议最具有唯一性的属性就是源端口和目的端口。端口号通常被分成两种:保留端口号(1到1024)和非保留端口号(1024以上)。主机之间的大部分TCP/UDP通讯都是在一个保留端口(通常是服务器端)和一个非保留端口(通常是客户端)之间进行的。因此,可以使用保留端口作为一个表征参数来进行规则优化,譬如将规则集分为端口为80的规则子集、端口为23的规则子集等。这样就可以在对数据包进行规则检测的时候,根据源端口或者目的端口中是否有保留端口来选择相应的规则集。

但由于TCP/UDP是一个双向的通讯,即存在客户端和服务器端之分,因此在按保留端口进行规则优化的基础上,需要进一步根据源端口是保留端口,还是目的端口是保留端口,将规则集进行分组,即规则集分为两个子集:源端口是保留端口的规则集和目的端口是保留端口的规则集。

譬如,如果一个HTTP流的源端口是80,就能把它定位

袁 晖 讲师,硕士,研究方向:信息安全。

到源端口为 80 的规则集上进行检测，如果目的端口是 80，就把它定位到目的端口为 80 的规则集上进行检测。

· ICMP 协议

对于 ICMP 协议，我们使用 ICMP 类型字段作为 ICMP 规则集的特征参数。有些针对 ICMP 的入侵规则对所有类型的 ICMP 包都适用，我们将这些规则称为通用 ICMP 规则。在按照 ICMP 类型字段划分规则子集的时候，需要将通用 ICMP 规则加入到每个规则子集当中，同时还要建立一个通用 ICMP 规则子集，它仅包含所有的通用规则。

例如，对于类型代码为 8(请求回显)的 ICMP 数据包，会以类型代码 8 作为参数进行检测。然而，对于一个 ICMP 数据包，如果没有与它的类型代码相对应的规则子集，就需要在通用 ICMP 规则子集中对其进行检测。

· IP 协议

IP 包头的传输协议字段指明了该数据包所采用的传输协议类型。如果传输协议是 TCP，UCP 或 ICMP 协议，可以按照上面的讨论的特征化参数来进一步划分子集。

对于其他情况可以根据 IP 地址来进行划分，具体如何划分要通过分析规则文件中对应的规则的数目来决定，如果相应的规则数目较多，则进行划分，一般情况下不必要进行划分。

2. 匹配的唯一性

在按照数据包的特征参数选择需要进行匹配规则子集的时候，有可能出现有两个或者两个以上的规则子集同时被选中的情况，我们称之为唯一性冲突。这种情况只会发生在 TCP 和 UDP 协议中，譬如 TCPIUDP 规则子集 A 的特征参数是源端口号等于 80，TCP/UDP 规则子集 B 的特征参数是目的端口号等于 23，当一个 TCPIUDP 数据包的源端口为 80，目的端口为 23 时就发生唯一性冲突了，因为规则子集 A 和规则子集 B 都会被选中。

产生唯一性冲突时，最简单的方法就是对选中的规则子集都进行检测。但在有些情况下，这两个规则子集可能有很大一部分是重合的，所以如果将这两个规则子集合并成一个新的规则子集，就能够节约很多时间。合并后的新规则子集也加入到规则子集中，并不是取代原有的两个规则子集，只是它的特征参数有两个，即源端口和目的端口都满足条件时该规则子集才会被选中。

但将所有可能产生唯一性冲突的规则子集都事先进行合并是不可能的，因为，这样做所需要的地址空间太大了。例如，假定有 1000 个以源端口为特征参数的规则子集和 1000 个以目的端口为特征参数的规则集，那么理论上就会产生 1000000 个唯一性冲突。考虑到一个规则集所需要的内存空间，就可以知道对这些唯一性冲突都进行规则合并是不切实际的。所以我们只对合法的唯一性冲突合并规则子集。所谓合法的唯一性冲突，就是根据协议的定义，本身存在产生冲突的可能。譬如服务器间的 DNS 询问就是一个合法的唯一性冲突，因为在通讯的过程中，双方的端口号都是 530。

3 改进的模式匹配算法

BM 算法是 Boyer 和 Moore 在 1977 年提出来的，其特点是考虑到在匹配比较的过程中，不少情形是前面的许多字符都匹配而最后的若干个字符不匹配，因此，采用自右至左的方式扫描模式和正文。这样，一旦发现正文中出现模式中没有的字符时，就可以将模式、正文大幅度地滑过一段距离。假定字典表用 E 表示，模式和正文中出现的任意字符一定属于字典表 Σ。对于给定的模式 $P=p_1,p_2,\cdots,p_m$，定义一个从字母到正整数的映射(函数)

$dist:c\to\{1,2,\cdots,m\}$，这里 $c\in\Sigma$，函数 $dist$ 称为滑动距离函数，它给出正文中可能出现的任意字符 c 在模式中的位置。dist 函数的具体定义如下：

$$dist[c]=\begin{cases} m & \text{若任意字符 } c \text{ 不在 } P \text{ 中或者 } c=p_m \\ & \text{但 } c\neq p_j\ (1\leqslant j\leqslant m-1) \\ m-j & \text{否则，这里的 } j=\max\{j\mid p_j=c, \\ & 1\leqslant j\leqslant m-1\} \end{cases}$$

BM 算法的主要思想[4]是：假设将正文自位置 i 起“往左”的一个字串与模式进行自右至左的匹配。比较过程，若发现不匹配(不管是在何位置)，则下次应从正文的 $i+dist[t_i]$ 位置重新开始进行 BM 算法的匹配比较工作。其效果相当于把模式、正文向右滑过一段距离 $dist$ 间，即跳过 $dist[t_j]$ 个字符而无需进行比较。显然，若是字符 t_i 不在模式中出现，或者是仅仅在模式末端出现，则向右滑过最大。

Wu 氏算法是对 BM 算法的改进。Wu 氏算法采用了 BM 算法的思路，并使用了 3 个表来加速匹配过程移位表、哈希表和前缀表。移位表和 BM 算法中的类似，但和 BM 算法的处理有所不同。Wu 氏算法根据最后 2 个(甚至 3 个)字符来计算移位值，当移位值大于 0 的时候，可以移动位置继续扫描；但是当匹配发生时，我们需要知道到底是哪一个模式发生了匹配。为了避免逐个地和每个模式串进行比较，需要使用哈希技术来进行加速。

Wu 氏算法主要分为两个阶段：预处理阶段和扫描阶段。预处理阶段的主要任务是构造移位表、哈希表和前缀表，为扫描阶段做准备；扫描阶段的主要任务是进行多模式匹配。

(1)预处理阶段

预处理阶段要做的第一件事是计算模式的最小长度，记作 m，在进行匹配的过程中只考虑每个模式的前 m 个字符。也就是说，我们假定所有的模式的长度都相同。模式的最小长度对算法的效率有很大的影响。

在 Wu 氏算法中，不是对文本中的字符逐个地检查，而是将它们看作是长度为 B 的块。假定所有模式的总长度是 M，$M=k*m$，其中 k 为模式的个数，并且假定 c 是字母表的大小。B 的最优值是 $\log_c^{2M}$ 的倍数，但实际上，通常取 $B=2$ 或者 $B=3$。移位表的作用与 BM 算法中一样，只是它是根据最后 B 个字符决定移位的位数而不是根据最后一个字符。在移位表中，每个长度为 B 的字符串都被映射成一个整数，作为移位表的索引。移位表中的值给出了它在扫描文本的过程中可以向前移动的距离。移位表可以根据需要进行压缩。

当匹配发生时，我们需要知道到底是哪一个模式发生了匹配。为了避免逐个地与每个模式串进行比较，需要使用哈希表进行加速。

首先，计算出 B 个字符到一个整数的映射，用作移位表的索引。然后将同样的数据也用于哈希表中。哈希表的第 i 个入口记作 HASH[i]，它包含了一个指向最后 B 位字符的哈希值为 i 的模式列表的指针。当发生多模式匹配时，可以根据该指针快速定位到匹配的模式串列表中，提高算法的效率。

自然语言文本不是随机的。有些后缀不只是在文本中经常出现，而且还非常有可能在几个模式中同时出现。这会带来哈希表冲突的问题，即具有相同后缀的所有模式被映射到哈希表的相同的入口。在文本中遇到这样的后缀时，就会发

(下转第 138 页)

```
        FOR 每个 r∈ Child DO
          IF q. ext ⊆ r. ext THEN
            tag=FALSE;
            break;
          IF tag THEN
            IF q. con 和 C 间存在边 THEN 删除 q. con 和 C 之
              间的边;
            C_new 和 q. con 间增加一条边;
            Child=Child ∪{q};
      i=i-1;
    IF {m*}'⊆ext THEN
      exit;
END
```

5 实验结果及其分析

在 windows XP 下用 Java 编程实现了 RRCL_A 算法，在 P4 2.8G 的计算机上对随机生成的数据集进行了实验。对象数目为 200，属性数目为 100，关系概率为 30%和 40%，生成了 2 个形式背景。实验将属性分成 10 组，每组 10 个属性，分别采用 Godin 算法[3]、CLIF_A 算法[4]、RRCL_A 和文[7]算法生成概念格，2 个形式背景的实验结果分别如图 3 和图 4 所示。

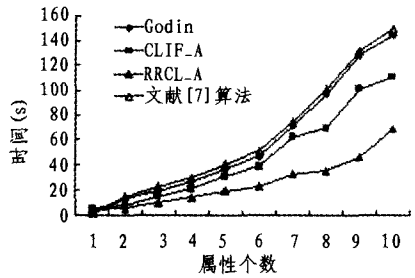

图 3 测试结果(|G|=200，|M|=100，关系概率为 30%)

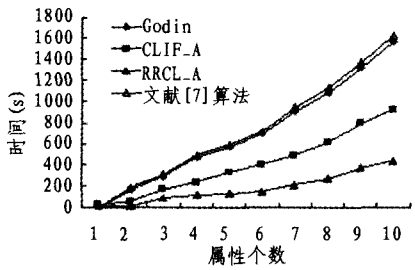

图 4 测试结果(|G|=200，|M|=100，关系概率为 40%)

由图 3 和图 4 可以看出：随着属性个数的增加，RRCL_A 算法和 CLIF_A 算法在时间上比 Godin 算法、文[7]算法更优。作为基于属性的算法，RRCL_A 算法优于 CLIF_A 算法，且随着关系概率的增大，RRCL_A 算法的优势体现得愈加明显。这种现象是由于较高的关系概率使得具备一个属性的对象数目增大，从而使得每个格节点的子节点数目增多，被 RRCL_A 算法忽略的节点个数也随之增多。从概念格存储角度，RRCL_A 算法最小，文[7]算法其次，CLIF_A 算法和 Godin 算法最差，这是由于 RRCL_A 算法只存储概念的初始内涵和初始外延，文[7]算法存储概念的初始内涵，而 CLIF_A 算法和 Godin 算法存储概念的真正内涵和真正外延。从算法的空间耗费角度，RRCL_A 算法需建立概念树，导致 RRCL_A 算法空间耗费最高。但相比概念格存储空间和构造时间上的节省，概念树的空间耗费可以忽略。

结论 本文提出了一种基于属性的相对约简格快速渐进式构造算法。该算法利用树结构组织格节点，通过概念树的按层遍历实现对格节点的访问。算法通过缩小产生子格节点判定时所需搜索的格节点的范围、缩小更新格节点和产生子格节点的搜索范围以及缩小新生格节点的直接子节点的搜索范围，达到了加速相对约简格构造的作用。实验结果表明，本文提出的算法优于 Godin 算法和现有的相对约简格构造算法。

参 考 文 献

1 Ganter B, Wille R. Formal Concept Analysis: Mathematical Foundations. Berlin Heidelberg: Springer-Verlag, 1999
2 胡可云，陆玉昌，石纯一. 概念格及其应用进展. 清华大学学报(自然科学版)，2000，40(9)：77～81
3 Godin R, M issaoui R, A laoui H. Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence, 1995, 11(2): 246～267
4 李云，刘宗田，陈峻. 基于属性的概念格渐进式生成算法. 小型微型计算机系统，2004，25(10)：1768～1771
5 张凯，胡运发，王瑜. 基于互关联后继树的概念格构造算法. 计算机研究与发展，2004，41(9)：1493～1499
6 谢志鹏，刘宗田. 概念格的快速渐进式构造算法. 计算机学报，2002，25(5)：490～496
7 张意德，宋简全，赵文兵，等. 相对约简格及其构造. 计算机工程与应用，2002，38(6)：196～197

(上接第 123 页)

现移位表的值为 0(如果它是某些模式的后缀)，因此，不得不分别检查具有这种后缀的所有模式，看它们是否与文本匹配。为了加速这个处理，引入了另外一张表格，叫做前缀表。当发现移位表哈希值为 0，并需要利用哈希表确定是否有匹配的时候，可以先检查前缀表中的值与文本中相应的前缀(通过向左移动 $m-B'$ 个字符，B' 是前缀表中字符块的大小)是否相符，过滤掉大量的模式。但是，只有当移位表的值经常为 0，也就是规则条数很多，且具有很高的冲突可能性的时候，使用前缀表才有意义。因此，在具体的应用中，是否需要前缀表要根据具体应用环境而定。

(2)扫描阶段

算法的主循环有以下几个步骤：

1)根据文本中当前的 B 个字符，计算哈希值；

2)检查 SHIFT[h]的值，如果该值大于 0，移动文本，转向 1；否则转向 3；

3)计算文本前缀的哈希值，(从第 m 个字符开始，到当前位置的左侧)，记为 text Prefix；

4)检查每个 p，其中 HASH[h]$\leqslant p\leqslant$HASH[$h+l$]，PREFIX[p]=test_prefix 是否成立。如果它们相等，使用真正的模式(由 PAT_PIONT[p]得到)对文本直接进行检查。

结论 本文主要讨论了基于 Snort 的网络入侵检测的问题，Snort 是一个开放源代码的网络入侵检测系统，这就使得我们可以对入侵检测的方法进行改进，本文主要从规则优化和匹配规则优化两个角度讨论了对 Snort 安全性能提高的方法。本文创新点在于将基于 Snort 的网络入侵检测系统进行了研究，并对入侵检测的算法进行了改进，使得检测的有效率更高。

参 考 文 献

1 INSKI A C~. A synchronization, Algorithm for processes with dynamic priorities in computer networks with node failures[J]. Information Processing Letters, 1989, 32(3): 129～136
2 INSKIA Two Algorithms for Mutual Exclusion in. Real-time Distributed Computer Systems[J]. Journal of Parallel and Distributed Computing, 1990, 9(1): 77～82
3 CASWELLBrian, BEALEJay, FOSTERJamesC, et al. Snort2.0 intrusion detection[M]. 北京：国防工业出版社，2004
4 唐正军. 黑客入侵防护系统源代码分析[M]. 北京：机械工业出版社，2002
5 孙振龙，等. 基于数据挖掘技术的 snort 入侵检测系统的研究[M]，微计算机信息，2006，22(11-3)