【学术研究】

Snort 的高效规则匹配算法的研究

顾 明
(阜新高等专科学校，辽宁 阜新 123000)

摘 要：通过对 Snort 的工作流程和 Snort 匹配方法的分析，改进匹配过程、改进其搜索并行性，并对其改进性能进行分析.

关键词：Snort；匹配过程；规则树结构
中图分类号：TP301.6　　**文献标识码**：A　　**文章编号**：1008 - 5688(2009)01 - 0022 - 03

1 Snort 的工作流程分析

Snort 的基本工作流程：首先为每一类攻击事件定义一条或多条“规则”(如网络数据包的某些头信息等)，形成规则库；然后分析规则库，生成树型规则结构(本文中称为“规则树”)；接着实时捕获网络数据包，并将数据包进行解码；最后检索规则树，将解码后的数据包与规则进行匹配；如果匹配成功，判定其为入侵行为，按照规则的设定方式进行报警. Snort 启动后必须先执行初始化(INT)工作[1]，包括命令参数的解析、处理模块初始化、检测规则的解析和网络包抓取过程. Snort 入侵检测初始化流程图见图 1.

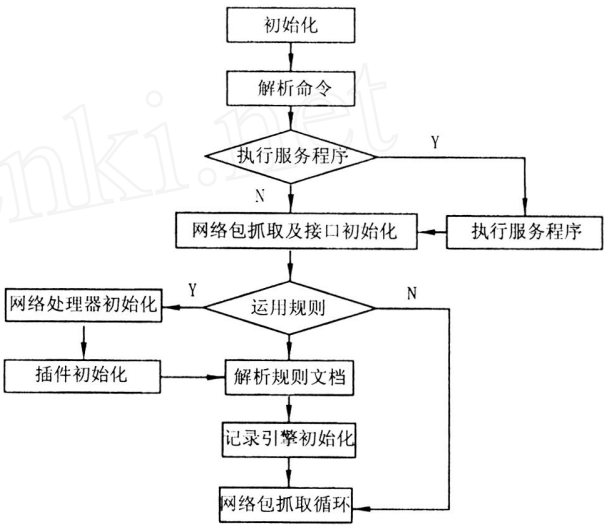

图1 snort入侵检测初始化流程图

在主程序 Main.c 内进行的主要是 ParseCmdLine()解析命令行相关参数设定及设置相关规则变量值，InitializeInterface()接口初始条件的设定，以及可加入自己编写检测条件的 InitPlugins()函数进行设定动作，然后进入监听循环 Pcaploop()中. 初始化的过程中，Rules.c 内的 ParseRulesFile()分析用户定义的规则来进行网络包检测、过滤规则的设定，Spo-database.c 及 Sp-session.c 内的各函数也会被调用，以进行数据库和日志的相关记录设定.

网络包过滤时，ProcessPacket()首先由函数 DecodeEthPkt()、DecodeIP()、DecodeTCP()进行一连串的解码动作，然后进行 Preprocessor()、Detect()等检测动作. 按照使用者的设定规定，可疑网络包会再经由 Log.c 或数据库 Spo-database.c 记录下来，或是启动其他的规则来进行进一步地分析，或是执行外部程序进行防护动作.

在匹配规则过程中，必须通过 ParseRuleFile()读取所有规则，并运用规则解析器 ParseRule()解析每一单独的子规则条，并将其加入至二维树状结构中.

2 Snort 规则匹配方法分析

Snort 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)地址和端口以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的时候才触发相应的操作. 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.

收稿日期：2008—09—15
作者简介：顾明 (1974 -)，男，辽宁阜新市人，讲师，主要从事计算机方面教学研究.

Snort 初始化并解析规则时，根据规则的第一个关键字 Alert、Log、Activate 和 Dynamic 分别建立规则链表，每条链表是一个 ListHead 结构，结构中有 IpList、TcpList、UdpList 和 IcmpList 链；对应规则的第二个关键字 IP、TCP、UDP 和 ICMP 分别归类到相应链中. ProcessHeadNode 的第一个参数是当前规则的 RuleTreeNode 结点，ProcessHeadNode 函数的功能就是把 RuleTreeNode 结点放入规则链中. 所有的规则根据适用的协议分为 IP、TCP、UDP、ICMP 共 4 棵树，当 Snort 捕获一个数据包时，首先根据协议匹配 IP、TCP、UDP 或 ICMP 的一个，如果匹配成功则告警，否则匹配 IP 规则树. 每棵树又根据源 IP、源端口、目的 IP 和目的端口以及通信方向分为若干子树，待匹配的包如果地址信息的方向与子树相符，则进入相应的规则链，否则匹配下一棵子树直至结束. 此时进入的规则链里的每个结点对应一条规则，在每个结点上挂着一系列的检测插件，如果每个检测插件都匹配成功，则这个结点对应的规则被匹配，进入告警. 为提高规则的匹配速度，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. 它的规则树结构如图 2 所示.

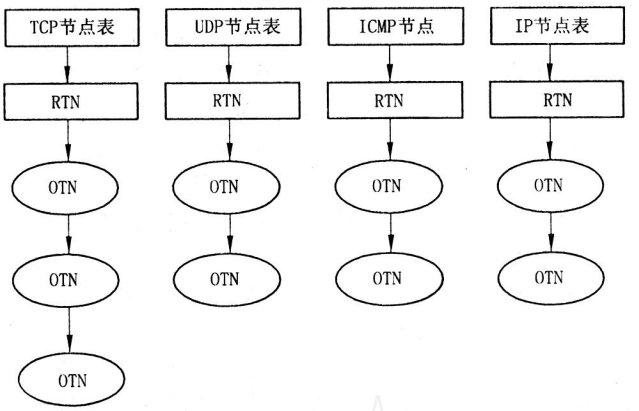

图2 snort规则树

目前在规则匹配中的通用方法[2]是：入侵检测系统将检测到的数据包生成相应的结构数据后，再通过匹配函数遍历规则树：首先对规则头链表进行依次匹配，当与其中一个规则头相匹配时，再与规则库中选项进行比较，当发现匹配的内容时，即报告发生了一次攻击. 可以看出，随着规则库中规则选项的增多，如果遍历所有的规则，然后匹配规则选项，当匹配相同时才报告发生了攻击，显然效率是比较低的. 首先，直到匹配到相同的选项才跳出系统循环，这样就无端地匹配了前面不同的选项规则，消耗了系统资源. 其次，随着操作系统漏洞或其它软件漏洞的升级，致使单纯依赖于这些漏洞的攻击方法将不起任何作用. 那么，如果这些规则不能很好地进行调整或删除也将影响规则匹配速度，因为遍历规则库时，那些无用的规则仍然将被进行匹配比较，除非这些无用的规则排在成功匹配项的后面.

3 改进搜索并行性

3.1 匹配过程改进分析

Snort 按照深度链式搜索算法对规则进行逐条匹配，规则中的每一个选项对应各自的匹配函数，以实现不同类型的匹配操作. 这种算法的优点是程序结构清晰，具有非常好的可扩展性. 在具体实现中，假如同类规则集中有多条规则具有某一相同值的选项值，系统仍需依次对每一选项进行匹配操作. 例如在 Web - iis 类规则集中，85 条规则的 Flags 选项值都为“A + ”(即在 TCP 包头中，标志位 ACK 为 1，其他标志位无关.

如果将这类具有共性的选项提取出来，在匹配头结点之后、选项结点之前，集中先对 Flags 选项进行一次匹配操作，将能够有效减少每一条规则中的重复匹配操作. 在其他一些规则集中，Flags 选项的值并不都是“A + ”，因此在图 3 中提取出的 Flags 选项的基础上，在宽度方向增加一个 Flags 选项链，使得每一个具有相同规则头内容的多条规则，根据 Flags 选项值的不同再一次被分类，从而在整体上提高搜索过程的并行性.

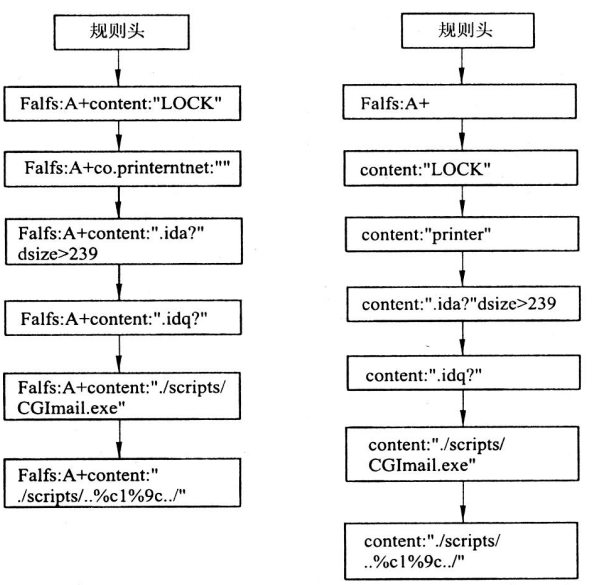

图3 对于Flags选项的两种匹配方式

3.2 改造规则树结构

为合并 Flags 选项，需对规则树的结构进行改造. 在生成规则树时，当解析完一条规则的头结点后，首先对 Flags 选项进行单独处理，根据其内容，在规则头之下生成一条 Flags 选项链. 图 24 页.

Flags 选项链由 Flags 选项结点组成，每个结点包括三个内容：Flags 的值、一个向右的指针（指向下一个 Flags 选项结点）以及一个向下的指针（指向选项链）. 具体数据结构描述如下：

```
typedef struct FlagNode
{
char flag; //flags 选项的值
struct FlagNode *right; //向右指向下一 flags 选项结点
Opt TreeNode *down; //向下指向选项链
} FlagsNode;
```

Flags 选项链的生成方法是：如果当前解析的规则中 Flags 选项的内容与已生成的 Flags 选项结点都不相同，则在 Flags 选项链的最右边生成一个新的结点；否则找到对应 Flags 选项，在其下的纵向选项链中增加一个新的选项结点（包括除去 Flags 选项之外的其他选项，以及各自对应的匹配函数）.

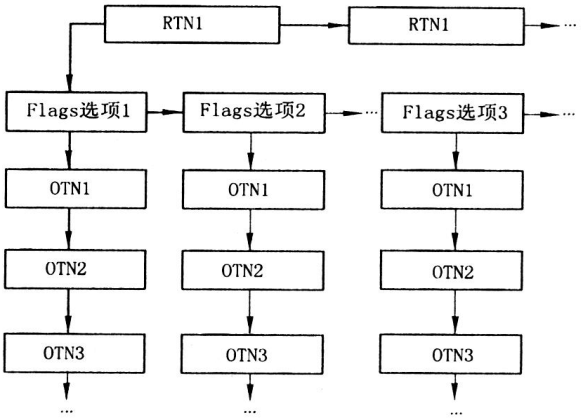

图4 增加Flags选项链后规则树示意图

在 Snort 原程序中，函数 ParseRuleOptions（char * rule，int rule-type，int protocol）完成对一条规则中全部选项的解析；为增加 Flags 选项链，程序需作以下改动：

(1) 从规则中分解出 Flags 选项名和其对应的内容，对其内容进行格式化；

(2) 在 Flags 选项链中，自左向右查找是否存在与该 Flags 选项内容相同的结点；如没有，在链表最后创建一个新的 FlagsNode 结点，其右指针为空，下指针指向选项链；

(3) 继续向下生成其他选项链.

3.3 性能比较

由于对 Flags 选项的特殊处理，系统生成规则树时，会相应增加处理时间. 表 1 所示的实验数据是函数 ParseRulesFile() 对 85 条 Web - iis 类规则进行处理的时间值对比.

表 1 规则树生成时间对比

	1	2	3	4	5	均值
原执行时间（s）	0.302	0.335	0.338	0.324	0.356	0.331
改后执行时间（s）	0.314	0.354	0.341	0.332	0.360	0.340

实验结果表明，修改后的程序增加了约 2.7 %的处理时间. 由于规则树的生成是在系统初始化时完成，不影响系统对数据包的匹配操作，因此规则树处理时间的增加对系统的整体性能无直接影响. 为验证增加 Flags 选项链对系统匹配时间的影响，在匹配函数 Detect（Packet *p）中增加时间计算函数. 根据系统对 Check Tcp Flags() 函数的调用次数，记录 10 组 10 000 个 TCP 数据包的总匹配时间，并与原程序进行对比，见表 2.

表 2 程序修改前后 10 000 个数据包总检测时间对比

组数	原检测总时间（s）	改进后检测总时间（s）
1	2.8120	2.1718
2	2.4233	2.2101
3	2.6890	2.4436
4	2.9850	2.6116
5	2.4671	2.0754
6	2.3690	2.0643
7	2.2538	2.1971
8	2.7936	2.6211
9	2.9439	2.8641
10	2.2745	2.1456
均值	2.0611	2.3410

实验数据表明，原程序的检测时间平均为 2.601 1 s，修改后平均为 2.341 0 s. 通过优化规则树构成和匹配方法，提高搜索过程的并行性，系统的匹配速度比原来提高了约 10.00 %.

参考文献：

[1] 许勇，向智勇. 入侵检测系统在分布式环境下的协作协议 [J]. 计算机与现代化. 2002，(8)，23 - 26.

[2] 李索科，刘宇虹. 分布式入侵检测系统 [J]. 信息网络安全. 2002 (3) 41 - 43.

（责任编辑 李树东，王 巍）

（上接 9 页）

和不断矫正方向的过程，因此上述分析只是便于学生理解而作的定性分析与探讨.

参考文献：

[1] 周衍柏. 理论力学教程 [M]. 北京：人民教育出版社，1999.

（责任编辑 王立俊，于 海）