入侵检测规则的冲突发现和解决[1]

孙美凤[①,②]　　龚俭[①]

①东南大学计算机系网络中心　南京 210096

②扬州大学信息工程学院计算机系　扬州

摘要： IDS 中当前输入事件同时匹配入侵规则库中多条规则的现象称为检测冲突，它导致漏报和误报，降低检测的精度。随着规则库规模的增大，冲突的可能性不断增加，也使得人工避免冲突的工作变得越来越困难，因此冲突的自动检测是入侵规则库管理和维护的重要问题。本文首次提出入侵检测规则的冲突问题，形式定义了冲突的检测标准，并给出冲突检测和解决的算法。对 Snort 规则库分析的结果表明了本文方法的必要性和有效性。

关键字： 冲突检测；入侵规则；入侵检测系统；报文分类

中国图分类号： TP393

Detecting Conflict for Network Intrusion Rule

Sun Meifeng　　Gong Jian

Department of Computer Science and Technology

Southeast University

Nanjing 210096, P.R. China

{msun, jgong}@njnet.edu.cn

Abstract： The fact that input event matches more than one rule of intrusion rule base in IDS is considered as conflict, conflict potentially creates false positive and false negative, and moreover reduces detection efficiency. As the number of intrusion rules increases, the possibility of conflict will increase, and the manual work to avoid conflict will be difficult. So automatically detecting and resolving conflict is an important aspect in rule management and maintenance. This paper studies the conflict detection problem of intrusion detection rules, defines formally a set of principles to determine the relationship of two intrusion rules, then proposes algorithms to detect and resolve conflict. The analysis of snort rule base implies the necessity and the effectivity of the method presented in this paper.

Keyword： intrusion detection system; intrusion rule; conflict detection; conflict resolution

1　引言

基于攻击特征进行入侵行为识别的滥用入侵检测系统（IDS）包括两个关键部分：规则库和检测算法。入侵规则编码了入侵过程的知识以及响应行为，检测算法依据入侵规则在输入事件流中寻找恶意行为的证据并做出结论。显然,正确的入侵规则是做出正确检测结论的前提。

目前入侵规则库的构造是渐进的和经验主义的，通常是入侵规则的简单罗列，语义矛盾和重复不可避免的存在，从而导致了同一报文同时匹配两条或多条入侵规则的冲突现象的出现。由于实时性的要求，IDS 通常在发现一个匹配时就形成检测结论，因此冲突可能导致检测结论的不确定性，并进一步导致漏报和误报。随着规模的扩大，入侵规则库存在矛盾和重复的可能性将增大，同时人工发现这些不足的工作变得越来越困难。因此在实际系统开发当中，如何根据规则描述形式的具体特点，有针对性的发现和消除一些潜在的冲突规则，则是一种非常现实的考虑。

[1]本文的研究得到国家 973 计划课题（2003CB314804）和江苏省网络与信息安全重点实验室（BM2003201）的资助. 孙美凤，女，1970 年生，博士研究生，主要研究方向为网络安全监测. E-mail: msun@njnet.edu.cn. 龚俭，男，1957 年生，博士生导师，主要研究方向为网络安全、网络管理和网络体系结构.

在滥用 IDS 发展的二十年中入侵检测规则的表示技术一直被广泛的关注，出现了大量的检测语言如 P-BEST[1,2]、状态语言[3，4]、有色 Petri 网[5]、简单规则语言[6~8]、陈述性的语言[9~11]等类型，它们对表示的完备性、计算有效性和使用方便性有不同的侧重。近年对检测语言的研究比较少，我们没有检索到 2002 年以后的相关文献。上述所有文献都没有涉及到入侵检测规则的正确性问题。

少量相关的研究可见于近年的报文分类[12，13]和防火墙安全政策[14]的管理领域。报文分类规则通常包括协议类型、源 IP 地址、源端口、目的 IP 地址、目的端口的描述，形式上表示为一个 K 元组。例如：传统路由表的规则只包含目的 IP，组播路由是一个二元组（源 IP，组），而防火墙的访问规则是包含上述全部内容的五元组。IDS 规则包括过滤器规则和多阶段特征规则。过滤器规则如 Snort 将 WEB-IIS mkilog.exe 攻击定义为“alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS mkilog.exe access"; flow:to_server,established; uricontent:"/mkilog.exe"; nocase; classtype:web-application-activity; sid:1485; rev:3;)”[15]。过滤器规则与报文分类规则比较，存在两个方面的差异：首先，报文分类规则简单，每个域或为一个具体的值（如协议类型），或为前缀值（如源宿 IP 地址），范围通常被拆分成多个前缀，IDS 规则语法支持任意范围的直接表示，同样以 Snort 为例，端口号有如下形式：any,27374,12345:12346,1024: ,!80 , !53:80 ,:1023；其次，IDS 领域没有被统一接受的规则语法。IDS 规则的冲突问题应从一般意义上讨论，而不应该局限于某种具体形式。多阶段特征包含多个过滤器以及过滤器之间的约束关系，由于多阶段特征的复杂性以及表示形式的多样性，多阶段特征的冲突更为隐蔽。

~~因此~~本文探讨了~~工作在~~入侵检测领域中一个被忽略的问题~~具有显著的先进性~~，~~它~~给出了入侵检测规则冲突发现和解决的基本框架。本文结构如下：第 2 节定义冲突概念；第 3 节描述过滤器冲突的判定准则以及一种发现并解决冲突的算法；第 4 节描述多阶段特征冲突的表现形式以及一种发现并解决冲突的算法；第 5 节对 Snort 规则库进行分析，以表明冲突发现和解决的必要性和有效性。

2 冲突分类

定义 1

Field：域，记为 f。每个域都有名字和值域两个属性，分别记为 N(f)和 D(f)，所有域的集合记为 FS。如果建立了域名到自然数的影射则可以用下标标识域。

Filter：过滤器，记为 F，是一组作用于单事件的约束 F=[F[1] F[2]…F[n]]。F[i]是对 i 域的约束，定义与 F 匹配的事件的 i 域的取值范围。某个域在 F 中可以不出现，表示该域取任意值。例如：F[1]=(端口号,大于,80)；F[2]=(源地址,属于,某个子网范围)；F[3]=(传输层协议类型,等于,TCP)；则 F=[F[1]F[2]F[3]]定义了一种报文类型。

Signature：特征，记为 S。令∑是 F 的集合，$S = \{\omega \in \Sigma^{+} : \omega$具有性质 P$\}$，$\Sigma^{+}$是∑上所有不为空的 F 的序列集合，P 是定义在∑上的一组约束 IC。例如：某入侵类型“包含 A、B、C 三个行为，要求 A 优先于 B，B 优先于 C”，则该入侵类型的 S 中的∑={A,B,C}，P={A 优先于 B,B 优先于 C}，S={ABC}。如果某入侵特征 S 中存在ω且|ω|>1，则 S 称做多阶段特征。过滤器是入侵特征的最简形式。

首先定义两个集合间的关系：部分相交$\cap_{PD}$和完全不相交$\cap_{CD}$。

定义 2 令 A，B 是任意的集合，

$A \cap_{CD} B$：A 和 B 完全不相交，如果 $A \cap B = \varnothing$。

$A \cap_{PD} B$：A 和 B 部分相交，如果 $A \cap B \neq \varnothing$ 且 $A \cap B \neq A$ 且 $A \cap B \neq B$。

定理 1 $\delta = \{=, \supset, \subset, \cap_{PD}, \cap_{CD}\}$，δ是任意集合 A、B 间关系的全集且δ中关系互斥。

证明：考虑 A 中所有元素，与集合 B 之间存在三种可能性：

（1） 所有 A 中元素都在 B 中，$\forall\alpha(\alpha\in A\rightarrow\alpha\in B)$。

（2） 部分 A 中元素在 B 中，$\exists\alpha\beta(\alpha\in A\rightarrow\alpha\in B\wedge\beta\in A\rightarrow\beta\notin B)$。

（3） 所有 A 中元素都不在 B 中，$\forall\alpha(\alpha\in A\rightarrow\alpha\notin B)$。

同理，考虑 B 中所有元素，与集合 A 之间也存在三种可能性：

（4） 所有 B 中元素都在 A 中，$\forall\alpha(\alpha\in B\rightarrow\alpha\in A)$。

（5） 部分 B 中元素在 A 中，$\exists\alpha\beta(\alpha\in B\rightarrow\alpha\in A\wedge\beta\in B\rightarrow\beta\notin A)$。

（6） 所有 B 中元素都不在 A 中，$\forall\alpha(\alpha\in B\rightarrow\alpha\notin A)$。

则 A，B 间关系的 9 种组合中，（1）（6）、（2）（6）、（3）（4）和（3）（5）组合不可能出现，可能的 5 种组合分别是=，⊃，⊂，$\cap_{PD}$，$\cap_{CD}$。

得证。

不失一般性，过滤器和特征都可被抽象为一个集合，过滤器是符合特定模式的事件集合，特征是入侵行为（事件）序列的集合。理想情况下，任意入侵特征集合完全不相交。=、⊂、⊃和$\cap_{PD}$ 关系都会导致同一入侵行为序列匹配多个特征，冲突出现。

定义 3 令 S1 和 S2 是任意规则，

冗余冲突：S1 = S2，表现为两条特征定义的入侵行为集合相同。由于大多数算法的速度都与规则个数有关，规则冗余通常降低 IDS 的检测效率同时增加了空间要求，是规则库精练和维护中必须解决的问题，应消除冗余规则。

包含冲突：$S1 \supset S2 \vee S1 \subset S2$，表现为一条特征包含另一条特征。依赖 IDS 采取何种冲突解决方案，这种情形对检测结论有不同影响。如系统采取前规则优先冲突解决方案，通用规则在前专用规则在后，专用规则将永远得不到激活的机会，原该匹配专用规则的事件引起通用规则报警，既产生误报又产生漏报。如果系统不加选择的发出报警(如 snort 2.1.3 版本以上)则又提高误报率。

交叉冲突：$S1 \cap_{PD} S2$，表现为两条特征定义的入侵行为集合部分相交。同样是两条规则重叠，交叉冲突与包含冲突不同，包含冲突对重叠部分有明确的定义，而交叉冲突的两条规则没有明确重叠部分的行为。

3 过滤器冲突

3.1 冲突判定准则

我们从最基础的问题入手：如何判定任意两个过滤器的关系？下面的定理将这种关系的判定转化为判定过滤器对应域的关系。

定理 2

对任意的过滤器 F 和 G，F[i]，G[i]表示 F 和 G 的第 i 域，

（1）$F = G \quad iff \quad \forall i\ (F[i] = G[i])$；

（2）$F \supset G \quad iff$

$\forall i\ (F[i] \supseteq G[i]) \wedge \exists j(F[j] \neq G[j])$；

（3）$F \subset G \quad iff$

$\forall i\ (F[i] \subseteq G[i]) \wedge \exists j(F[j] \neq G[j])$；

（4）$F \cap_{CD} G \quad iff \quad \exists i\ (F[i]) \cap_{CD} G[i])$；

（5）$F\cap_{PD} G$ iff

$(\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j]\supset G[j] \wedge\forall k(k\neq i\wedge K\neq j \rightarrow F[k]\cap G[k]\neq\varnothing) \vee$

$(\exists i\ F[i] \cap_{PD} G[i] \wedge\forall k(k\neq i \rightarrow F[k]\cap G[k]\neq\varnothing)$

证明：

（1）~（3）是直观的。

对于（4），$F\cap_{CD}G$ 等价于 div(F)和 div(G)的交集为空。假设 F 与 G 在所有域存在交点，分别是 d1, d2, d3, …，则多元组 k = （d1,d2,d3,…）,$k\in$div(F)且 $k\in$div(G)，与 div(F1)和 div(F2)的交集为空矛盾。所以$\exists i$ (F[i]) $\cap_{CD}$ G[i])，必要条件成立。

假设 div(F)和 div(G)存在交点 p=(d1,d2,d3,…)，在域 fi 上，P 的对应值 di。则 di$\in$ F[i], di$\in$ G[i], F 与 G 在所有域存在交点，与存在某域 F 和 G 的交集为空矛盾。所以 div(F)和 div(G)的交集为空，$F\cap_{CD}G$，充分条件成立。

对于（5），定理 1 证明了δ作为集合的关系空间，既互斥又完备。则：

$F\cap_{PD}G$

$= \neg F\cap_{CD}G \wedge \neg F=G \wedge \neg F\supset G \wedge \neg F\subset G$

$= \neg F\cap_{CD}G \wedge \neg F\supseteq G \wedge \neg F\subseteq G$

$= \neg F\cap_{CD}G \wedge \neg(\forall i\ F[i] \supseteq G[i]) \wedge \neg F\subseteq G$

$= \neg F\cap_{CD}G \wedge \exists i\ (\neg F[i] \supseteq G[i])\wedge \neg F\subseteq G$

$= \neg F\cap_{CD}G \wedge \exists i\ (F[i]\subset G[i] \vee F[i] \cap_{CD} G[i] \vee F[i] \cap_{PD} G[i])\wedge \neg F\subseteq G$

$= \neg F\cap_{CD}G \wedge (\exists i\ F[i]\subset G[i] \vee \exists i\ F[i] \cap_{CD} G[i] \vee \exists i\ F[i] \cap_{PD} G[i])\wedge \neg F\subseteq G$

$= \neg F\cap_{CD}G \wedge (\exists i\ F[i]\subset G[i] \vee F\cap_{CD}G\vee \exists i\ F[i] \cap_{PD} G[i]) \wedge \neg F\subseteq G$

$=\neg F\cap_{CD}G \wedge (\exists i\ F[i]\subset G[i] \vee \exists i\ F[i] \cap_{PD} G[i])\wedge (\exists j\ F[j]\supset G[j] \vee \exists j\ F[j] \cap_{PD} G[j])$

$=(\neg F\cap_{CD}G \wedge\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j]\supset G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j] \cap_{PD} G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i] \cap_{PD} G[i] \wedge\exists j\ F[j]\supset G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i] \cap_{PD} G[i] \wedge\exists j\ F[j] \cap_{PD} G[j])$

$=(\neg F\cap_{CD}G \wedge\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j]\supset G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j] \cap_{PD} G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i] \cap_{PD} G[i] \wedge\exists j\ F[j]\supset G[j]) \vee$

$(\neg F\cap_{CD}G \wedge\exists i\ F[i] \cap_{PD} G[i])$

$=(\neg F\cap_{CD}G \wedge\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j]\supset G[j]) \vee (\neg F\cap_{CD}G \wedge\exists i\ F[i] \cap_{PD} G[i])$

$=(\exists i\ F[i]\subset G[i] \wedge \exists j\ F[j]\supset G[j] \wedge\forall k(k\neq i \wedge K\neq j \rightarrow F[k]\cap G[k]\neq\varnothing) \vee$

$(\exists i\ F[i] \cap_{PD} G[i] \wedge\forall k(k\neq i \rightarrow F[k]\cap G[k]\neq\varnothing)$

得证。

图 1 给出任意过滤器按照定理 2 进行冲突判定的算法。

```
Algorithm ConflictDetect(F,G)
    /* 判定 F 和 G 是否冲突以及冲突的类型  */
    1.  for I = 1 to d do
    2.      if F[I] ∩CD G[I]  return "F ∩CD G"
            else if F[I] = G[I]  count=++  /*对相等域计数*/
            else if F[I] ⊃ G[I]  count⊃++  /*对包含域计数*/
            else if F[I] ⊂ G[I]  count⊂++  /*对被包含域计数*/
            else  F[I]∩PDG[I]  count∩PD++ ; /*对部分相交域计数*/
    3.  if count∩PD>0  return "F ∩PD G"
```

```
            else if count= = d  return  "F = G"
            else if count⊃= d  return  "F ⊃ G"
            else if count⊂= d  return  "F ⊂ G"
            else  return  "F ∩PD G";
end  Algorithm
```

图 1　过滤器冲突判定算法

3.2　冲突发现和解决算法

算法的思想是分裂冲突规则为多个不相交的子规则，彻底地解决冲突问题。根据定理 2，只要∃i (F[i]) ∩CD G[i]) 则 F∩CDG，通过分裂规则保证任意规则至少存在某个域完全不相交，可实现冲突解决。在讨论本文的冲突解决算法之前，首先给出限定的冲突无关∩DCD的定义。令 F=[F[1]F[2]…]，G=[G[1]G[2]…]；

定义 4　F ∩DCD G：F 和 G 限定冲突无关，如果∃i(F[i]) ∩CD G[i])) ∧∀j(j<i →F[j]=G[j])。

∩DCD是∩CD的子集，显然如果一个规则库是限定冲突无关的，则它一定是冲突无关的。使用∩DCD而不是∩CD可能导致不冲突的规则分裂，但其对算法的时空复杂性不会有根本性影响。

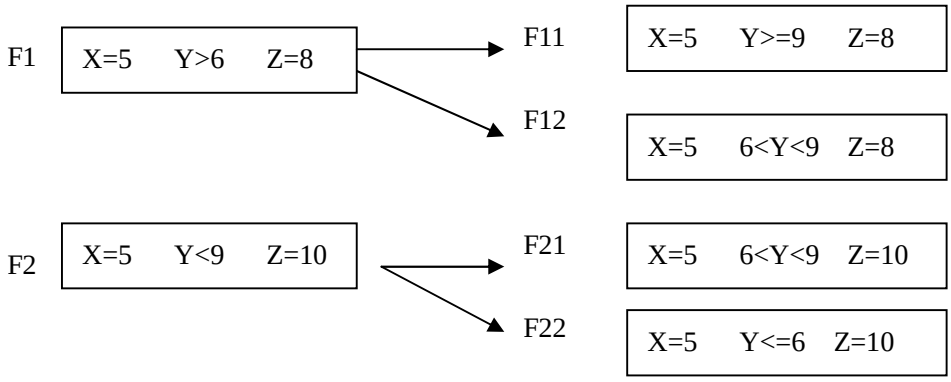

图 2 规则分裂的示例

图 3 是将任意规则库 B 改造成冲突无关的算法，算法由 d 重循环构成，d 是规则的维数。当第 1 重循环结束，以第 1 域不相交为原则，B 被划分成若干规则子集，子集内所有规则的第 1 域相等，不同子集的第 1 域完全不相交，因此属于不同子集的任意规则限定冲突无关。对第 i 重循环，以第 i 域不相交为原则，继续分割规则子集，直到 d 重循环结束，通过发现并合并相等规则，剩余规则必定不冲突。以图 2 中规则为例，当第 1 重循环结束时，因为 X 域相同规则 F1 和 F2 属于同一子集；第 2 重循环根据 Y 域取值，将由 F1 和 F2 组成的规则集进一步划分为三个子集，分别是：X=5 且 Y>=9 的规则集，由 F11 组成；X=5 且 6<Y<9 的规则集，由 F12 和 F21 组成；X=5 且 Y<=6 的规则集，由 F22 组成。

```
Algorithm FilterDetect_Resolve(B)
    /*  B 是规则集 */
1.   令 S，T 是规则集 B 的划分，初始化 S= {B}；
2.   For  i = 1 to d
```

```
3.      {
4.       For   j = 1 to |S|
5.         If   |Sj|> 1    then /*Sj 是 S 中规则子集*/
6.          {按照 i 域将 Sj 分割成 Sj1,Sj2…，每个子集具有相同的 i 域值，
7.           且不同子集的 i 域值完全不相交；
8.           从 S 中删除 Sj，将 Sj1,Sj2…加入 T；
9.          }
10.        Else 将 Sj 从 S 移到 T 集合；
11.      S <= T，T <= {}；
12.    }
13.  For j = 1 to |S|
14.     If   |Sj|> 1    then 发现冲突；
15.  End  Algorithm
```

图 3　过滤器冲突发现和解决算法

3.3　算法分析

不考虑单个域上的逻辑操作，n 个过滤器的 i 域在数轴上最多确定 2n 个点（考虑域上的逻辑操作，n 个过滤器最多确定 kn 个点，对复杂性没有本质影响），因此以 i 域不相交为原则由 n 个过滤器组成的过滤器集最多被划分为 2n+1 个不相交的子集，每个子集最多包含 n 个新的过滤器。同样以图 2 中过滤器集为例，F1 和 F2 在 Y 数轴上确定 2 个点：6 和 9，将 Y 域划分为 3 个不相交的区间，由 F1 和 F2 组成的过滤器集也相应地被划分为 3 个 Y 域不相交的子集。因为过滤器有 d 个域，所以算法的最坏时间复杂性和空间复杂性都为 $O(n^d)$，n 是过滤器数，d 是维数。

【1】?给出了冲突发现和解决的通用算法。*这句话与引言中最后的结论有矛盾，这个问题有人讨论过，而且有通解？*为了进行冲突检测，算法对规则两两比较，则对大小为 n 的规则集，冲突检测的时间复杂性是 O（n^2）。当冲突发现后，该算法为每一对交叉冲突规则 F 和 G 的重叠部分创建新规则，记为 ∩(F,G)，∩(F,G)=[F[1]∩G[1],F[2]∩G[2],F[3]∩G[3],…]。当报文匹配多条规则时，其中必有一条规则被所有其它规则包含（最长匹配），该规则的行为决定了报文的处理动作。这是我们已知的最好的通用解决方法，报文分类领域的其它研究成果由于利用了报文分类规则的特殊性，不能被直接应用。

为了将冲突规则库 B 改造为最长匹配规则优先条件下的冲突无关规则库 B’，该算法需要逐个处理 B 中元素，令 B 中规则数为 n，最坏情况下，B 中所有元素交叉冲突，因此每处理一个 B 中元素，都引起 B’中元素成倍增长，算法的最坏时间复杂性 $O(2^n)$，引起的规则数增长率最坏情况也是 $O(2^n)$，这种情况是因为对交叉区域重复定义专用规则引起的。另一方面，为了实现最长匹配规则优先，该方案下的报文分类算法必须对每个到达报文检查所有的规则，并在匹配规则集中进行选择，因此延长了报文分类时间。

本文算法性能为 $O(n^d)$，一方面 IDS 只在启动时才需要完全的运行该算法，另一方面 d 是个比较小的数值（通常是 5）且因为冲突需要分裂的规则数也比较小，所以该算法实际可行。本文给出的快速冲突解决算法将冲突规则库改造成真正意义上的冲突无关规则库，提高了入侵检测的精度，同时具有较好的规则预处理性能。

4 多阶段特征的冲突

4.1 冲突判定准则

多阶段特征形式上表现为多个过滤器的组合，可能的组合方式包括：then、and、or，定理 3 证明了任意入侵特征总可以表示为有穷的过滤器序列集合。

定义 5 令 S1,S2,S 为任意的场景，其中 S 由 S1 和 S2 组合而成；

顺序与（S = S1 then S2）:S = {ω1ω2: ω1∈S1,ω2∈S2}；

或(S = S1 or S2): S = S1∪S2；

与(S = S1 and S2): S = {ω1•ω2: ω1∈S1, ω2∈S2}。

•操作表示两个 F 序列任意位置连接，ω1•ω2 = {“ABCD”,“ACBD”,“ACDB”,“CABD”,“CADB”,“CDBA”}。•操作是可交换的：ω1•ω2 = ω2•ω1；•操作可结合：ω1•ω2•ω3 =（ω1•ω2）•ω3。•操作具有非常强的表示和生成序列的能力。

定理 3 任意入侵特征可被表示为有穷的过滤器序列集合。

证明：

任意使用 and、or、then 组合的入侵场景**显式地**表达每个入侵行为以及行为之间的约束关系。因此通过 and 组合在一起的并发行为、通过 or 组合在一起的可选行为以及通过 then 组合在一起的顺序行为数一定有穷，否则入侵特征既不能被表示也不能被检测。

按照定义 5，任意入侵特征是过滤器序列集合。由有穷的顺序行为组成的序列长度有穷，由有穷的并发行为和可选行为生成的序列数也一定有穷。

得证。

与形式语言理论的研究对象不同，不存在预定义的有穷过滤器表。因此两个有穷过滤器序列集合的关系并不象两个有穷字符串集合的关系一样显而易见。同时既然无法枚举任意过滤器给出的事件集合，更无法枚举入侵特征定义的事件序列集合，因此无法根据原始定义判定两个入侵特征之间的关系。

再看两个特征：S1={abcd,ef},S2={efgh,ab},形式上这两个集合完全不相交。但如果考虑入侵检测的背景：“ab”包含“abcd”，“ef”包含“efgh”。

综上所述，对于多阶段特征，冲突只能在两个过滤器序列之间进行检查，因为两个特征集合的不同过滤器序列之间可能存在不同的冲突关系。定理 4 给出过滤器序列冲突的两种形式。

定理 4 对任意的过滤器序列ω_1，ω_2；

（1）$\omega_1 \supset \omega_2$，$\omega_1$ 和ω_2 包含冲突。如果$\omega_2 = \wp_1\omega_1\wp_2$，$\wp_1$、$\wp_2$ 是任意过滤器序列，至少一个长度不为 0，即ω_1 是ω_2 的子序列，那么所有触发ω2 行为的事件序列均将触发ω1。IDS 对过滤器的包含冲突通常采取最长规则优先冲突解决方案，由于 F1…属于未来事件，多事件情形的该方案在实现上不同。一种实现方法是当检测到模式ω1 时，检测引擎等待一个时间窗口，如果在该时间窗口内ω2 的剩余模式出现，则采取ω2 的行为，否则采取ω1 的行为。

（2）ω1=ω2，ω1 和ω2 交叉冲突。如果ω1=ω2，即触发ω1 和ω2 的行为序列完全相同。与过滤器的交叉冲突一样，检测引擎的任何选择都可能是不确定的，最好的方法是提供所有可能行为信息由高层进一步处理。

证明：略。

4.2 冲突发现和解决算法

即使是两个序列特征，其中是否包含冲突也不是显而易见的，原因就是过滤器是事件模式而不是基本事件。本文方法是按照首过滤器不相交的原则不断划分特征，直到能够断言两个特征不冲突或发现冲突区域。

下面以一个简单的例子说明算法的执行过程：

令 S1、S2 是两个简单的序列特征，且都由两个过滤器组成。为了表示的方便，过滤器利用其覆盖的区域集表示。例如 F=（a,b）表示 F 是覆盖区域 a 和 b 的过滤器。

S1=(a,b)(c,d)；S2=(a)(c,e)；由 S1 和 S2 组成的特征集 S={(a,b)(c,d)#1，(a)(c,e)#2}，给每个特征附上特殊的结束标志#和特征号，是为了方便冲突的识别。

算法第一步：

获取特征的首过滤器集{(a,b),(a)}，参照图 3 算法，对其进行划分，得到{(a),(b)}

算法第二步：

以首过滤器不相交为原则，分裂原始特征集 S，得到两个新的特征子集：

S1={(a)(c, d)#1，(a)(c, e)#2}，S2={(b)(c, d)#1}。每个集合的首过滤器相同，不同集合的首过滤器不同，显然 S1 和 S2 中的特征互相不冲突。

算法第三步：

S1 中元素个数>1，将其表示成(a)S ，S={(c,d)#1，(c,e)#2}，由于 S 元素超出 1，按照首过滤器不相交原则，S 被分裂为 3 个特征子集：S1={(c)#1，(c)#2}，S2={(d)#1}，S3={(e)#2}

算法第四步：

S1=(c){#1,#2}，表明分裂出一个新的过滤器序列(a) (c)同时属于特征 S1 和 S2，即发现交叉冲突，冲突序列是(a)(c)。

本文算法有一个预处理过程:

（1)为每个过滤器序列添加一个结束标志和特征编号，如#3 表示入侵特征 3 的结尾；

（2)为了发现包含冲突，为每个过滤器序列（令长度为 n）生成所有长度为 k 的子序列，k 是特征集中长度小于 n 的序列的长度，并添加一个特殊的结束标志和特征编号，如$3 表示这是入侵特征 3 的子序列；

（3)按长度将特征集划分为多个子集。

```
Algorithm SignatureDetect_Resolv(S)
    /*  S 是预处理后的特征集  */
1.  ActiveSet={S} /* 设置变量保存所有待分割的特征集，初始为 S */
2.  {
3.      从 ActiveSet 中取出一个元素，一定具有 prefix_T 的形式；
/*    prefix_是指该集合元素拥有的共同的前缀，T 是提取前缀后的特征集      */
4.      将 prefix_T 从 ActiveSet 中删除；
5.      If T 中元素仅由结束标志组成 then
6.         T 中以#结束的特征相互交叉冲突，
7.         T 中所有以#结束的特征包含以$结束的特征；/*冲突发现解决*/
8.      Else {
9.          取出 T 中所有特征的首过滤器，按照图 3 中算法，对其进行分割并
            相应地将特征集划分为若干特征子集 T1,T2…；
10.         将 T1,T2…加入 ActiveSet；
11.     }
12.  }
```

13. End Algorithm

图 4 入侵特征冲突发现和解决算法

4.3 算法分析

图 4 和图 3 的两个算法有相似之处，都是通过分裂规则，直到断言两个规则不冲突或发现冲突区域。图 3 中的算法按域分裂过滤器，图 4 中的算法按首过滤器划分特征。令特征集有 n 条特征序列，序列最大长度为 m，最小长度为 1，过滤器维数 d，每个序列生成的子序列最多为(m-1)!，则预处理后生成的新的特征集有（m-1）!n 条特征，则过滤器分割的时间复杂性为 $O(((m-1)!n)^d)$，分割成 $O(((m-1)!n)^d)$个新的特征子集，由于最大序列长度为 m，则图 4 中算法的时间和空间复杂性为 $O(((m-1)!n)^{md})$。该算法的时间复杂性是超指数次的，只能应用于少量特征的场合。

5 Snort 规则库的冲突检测

本文利用图 1 中算法对 Snort 规则库进行检查。Snort 是由 Sourcefire 公司开发的开放源码的 IDS，由于具有速度快、使用方便等优点，是世界上用户最多的同类产品之一。Snort 规则库由若干按类组织的规则文件组成，本文利用定理 2 对 103~2103 范围的规则进行实验。*选取的原则是什么？*需要注意的是规则号并不连续，部分规则号不存在，如：123、139、144、178、242、263 等 48 个规则号，另外组织在 deleted.rules 文件中的规则没有参与比较。由于比较既考虑了规则头域也考虑了规则选项，如果某个选项域 i 在规则 F 中存在而在规则 G 中不存在，则 G[i]⊃ F[i]。规则的两两比较次数为 1693720 *这个数字有意义吗？*，共有冲突 5485 个，其中冗余冲突占 0.04%、包含冲突（一对规则总是重复出现于包含冲突情形，因为 F⊃G 那么 G⊂F）1.44%、交叉冲突 98.52%。

限于篇幅，仅选择每类冲突的一个示例进行说明：

冗余冲突：规则 1485 和 1665 除报警信息不同外，所有项的定义完全相同。

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS mkilog.exe access"; flow:to_server,established; uricontent:"/mkilog.exe"; nocase; classtype:web-application-activity; sid:1485; rev:3;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC mkilog.exe access"; flow:to_server,established; uricontent:"/mkilog.exe"; nocase; classtype:web-application-activity; sid:1665; rev:4;)
```

包含冲突：规则 453 比 454 多了 icode 域的约束。

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Timestamp Request (Undefined Code!)"; itype: 13; sid:454; classtype:misc-activity; rev:4;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Timestamp Request"; itype: 13; icode: 0; sid:453; classtype:misc-activity; rev:4;)
```

交叉冲突：

按照定理 2，交叉冲突有两种情况，第一是存在某个域部分相交，第二是存在两个域相互包含，这两种情形在 snort 规则库同时存在。例如规则 2101 和规则 2102 因为关键字 content 及其相关的 offset，depth 和 nocase 综合起来检测的部分相交导致的交叉冲突，规则 108 和规则 731 因为不同域相互包含引起的交叉冲突。

规则 2101 和规则 2102 交叉：

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB
```

SMB_COM_TRANSACTION Max Parameter of 0 DOS Attempt"; flow:to_server,established; content:"|00|"; offset:0; depth:1; content:"|FF 53 4D 42 25|"; offset:4; depth:5; content:"|00 00|"; offset:43; depth:2; reference:cve,CAN-2002-0724; reference:url,www.microsoft.com/technet/security/bulletin/MS02-045.asp; reference:url,www.corest.com/common/showdoc.php?idx=262; classtype:denial-of-service; sid:2101; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB SMB_COM_TRANSACTION Max Data Count of 0 DOS Attempt"; flow:to_server,established; content:"|00|"; offset:0; depth:1; content:"|FF 53 4D 42 25|"; offset:4; depth:5; content:"|00 00|"; offset:45; depth:2; reference:cve,CAN-2002-0724; reference:url,www.microsoft.com/technet/security/bulletin/MS02-045.asp; reference:url,www.corest.com/common/showdoc.php?idx=262; classtype:denial-of-service; sid:2102; rev:1;)

规则 108 和规则 731 交叉：

alert tcp $EXTERNAL_NET any -> $HOME_NET 7597 (msg:"BACKDOOR QAZ Worm Client Login access"; flags: A+; content:"|71 61 7a 77 73 78 2e 68 73 71|"; reference:MCAFEE,98775; sid:108; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible QAZ Worm"; content: "|71 61 7a 77 73 78 2e 68 73 71|"; reference:MCAFEE,98775; sid:731; classtype:misc-activity; rev:3;)*要考虑冲突解决之后规则数增加了多少，这些新规则的增加是否给 IDS 的性能带来负面影响。另外在上述的例子中还要说明如果不消除这个冲突会给检测结果带来什么影响，即如何造成漏报和误报。例子的选取应当尽量能够展现冲突消除的好处。*

6 结论

检测冲突是检测规则实现的不合理性的一种表现形式，它直接导致了检测结果的不准确，对 Snort 规则库的分析结果表明了各类冲突的实际存在。本文研究了入侵检测规则，包括过滤器规则和多阶段特征规则的冲突发现和解决的问题。定义了冲突的检测标准，在此基础上给出了规则库的冲突发现和解决的算法。

本文给出的过滤器冲突发现和解决算法的性能为 $O(n^d)$，优于目前报文分类领域可被借鉴的同类算法性能。本文给出的多阶段特征的冲突发现和解决算法的性能是超指数次的，因此寻找更优算法将是今后研究的重点。

关于冲突解决，算法从形式上只能发现并分割出冲突区域，但为该区域定义行为需要管理员的参与。对此本文的设想是冲突检测和解决分两个步骤进行：第一步是模拟的冲突解决，输出需要明确行为的新规则以及引用的原规则，提供管理员参考；第二步按照管理员确定的方案分裂规则并为新规则定义行为，解决冲突。

参考文献

1. ebra Anderson, Thane Frivold, Alfonso Valdes . Next-generation Intrusion Detection Expert System (NIDES) A Summary. SRI-CSL-95-07, 1995. http://www.sdl.sri.com/nides/reports/4sri.pdf

2. A. Porras and P. G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In : Proceedings of the 20th National Information Systems Security Conference. Baltimore, Maryland,USA, 1997. 353—365. http://www.sdl.sri.com/emerald/emerald-niss97.html.
3. K. Ilgun. USTAT: A real-time intrusion detection system for UNIX[Master's thesis]. Computer Science Dept., University of California, Santa Barbara, USA, 1992.
4. Vigna, G. and R.A. Kemmerer. NetSTAT: a network-based intrusion detection system. Journal of Computer Security, 1999,7(1): 37-71.
5. S. Kumar. Classification and Detection of Computer Intrusions[PhD thesis], Dept. of Computer Science, Purdue University, USA, 1995.
6. .Habra, B.Le Charlier, A. Mounji, and I. Mathieu. ASAX: Software Architecture and Rule-based Language for Universal Audit Trail Analysis. In : Proc Of (ESORRICS)' 92. Springer-Verlag, 1992:435-450.
7. V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks, 1999,31(23--24):2435—2463.
8. M.Roesch. Snort-Lightweight Intrusion Detection for networks. In: Proceedings of USENIX LISA' 99 conference,1999: 229-238.
9. J.-L.Lin and X.Sean Wang amd S.Jajodia. Abstraction-Based Misuse Detection: High-level Specifications and Adaptable Strategies. In: Proc. Of the 11th Computer Security Foundations Workshop, Rockport, MA, 1998:190--201.
10. C.Michel and L.Me. Adele: an Attack Description Language for knowledge-based Intrusion Detection. In: Proc. Of the 16th International Conference on Information Security,2001. http://citeseer.nj.nec.com/michel01adele.html.
11. J-P.Pouzol and M.Ducasse. From Declarative Signatures to Misuse IDS. In : Proceedings of the RAID International Symposium, Davis, CA, 2001,2212:1-21.
12. A.Hari,S.Suri, and G.Parulkar.Detecting and Resolving Packet Filter Conflicts.In: proc of Infocom,Israel,2000(3):1203-1212.
13. Florin Baboescu and George Varghese. Fast and Scalable Conflict Detection for Packet Classifiers. The International Journal of Computer and Telecommunications Networking,2003,42(6):717－735.
14. Ehab Al-Shaer and Hazem Hamed. Modeling and Management of Firewall Policies. eTransaction on Network&Service Management,2004,1(1).
15. Snort.org. http://www.snort.org/cgi-bin/done.cgi