軽量ネットワーク IDS 向け検知ルール圧縮法の提案

西　孝王[†]　前田　敦司[†]　山口　喜教[†]

[†]筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻

1. はじめに

ネットワーク侵入検知システム(NIDS)は、ネットワーク上のトラフィックを監視し，侵入や攻撃を検知するシステムであり，外部ネットワークとの接点付近に設置するような形態が多い．

しかし外部ネットワークとの接点付近に NIDS を設置すると，通信情報量が増えた場合に任意の順序で来るパケットの並びを直すためにメモリが大量に必要になるため設置形態として不適当であると考え，各 PC に NIDS を組み込むような環境を想定し，NIDS の軽量化，省メモリ化を図るための，NIDS における検知ルールの圧縮を行なった．今回評価にはオープンソースの NIDS である Snort[1]を使用した．

2. Aho-Corasick オートマトン

パターンを DFA 等のツリーとして作成しパターンマッチングを行なうが．単純な DFA による力任せのマッチングでは効率が悪い．一方，Aho-Corasick(以下 AC)オートマトン[2]はマッチした可能性のある最も長い接尾辞を表すノードまで戻るエッジ(failure link)をツリーに埋め込み，テキストを一度読むだけで全てのパターンを検知できる．しかしルートノードへのエッジ（ルートエッジ）については数が多すぎるため NIDS の状態遷移表からは省かれることが一般的である．

3. 検知ルールの表現及び圧縮法

3.1. 二次元配列による表現(FULL)

AC オートマトンを状態遷移表として表現する簡単な方法は二次元配列を用いる方法である(以後 FULL)．この表現では現在の状態を i，入力を j とすると，次の状態は FULL[i][j]を参照することで求めることができる．

3.2. FULL の圧縮法

FULL の圧縮法として本研究の圧縮法の元となった Johnson 法[3]と Snort でデフォルト使用されている圧縮法 AC_BANDED 法[4]について述べる．

3.2.1. Johnson 法(Johnson)

Johnson は 3 つの一次元配列(next，base，check)を用いて圧縮を行なう(図 1)．3 つの配列の用途と圧縮の流れを説明する．FULL の各行をスライドさせ重ならないように next に詰めるというのが目的である．

① FULL の各行における全ての要素について，空き場所までの距離を base の行番号の位置に格納する．
② base に基づき next を構築する．
③ check には next の情報が何行目から来たものかという行情報を格納する．

このように構築を行なうことで現在の状態を i，入力を j とすると，次の状態は check[base[i]+j]が i であった場合に next[base[i]+j]を参照することで求められる．

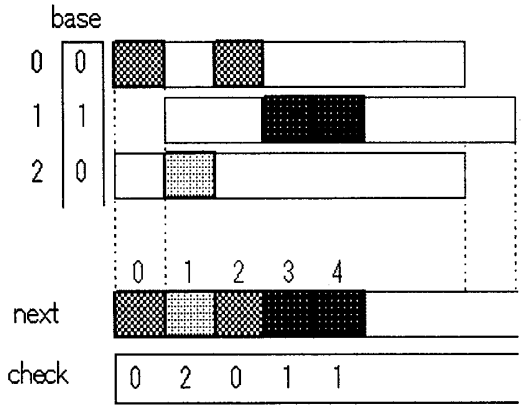

図 1　Johnson による圧縮

3.2.2. AC_BANDED 法(BANDED)

AC_BANDED 法は FULL の各行において最初の要素(start)から最後の要素(end)までを抜き出したもので構成される．FULL のような検知が可能であり，検知速度は比較的速いと考えられる．

この手法では現在の状態を i，入力を j とすると j が i 行の start 以上かつ end 以内である場合に，BANDED[i][j-start]とすることで次の状態を求められる．

4. 提案圧縮法(MySparse)

提案手法である MySparse は Johnson における base を排除し，次の状態を next[i+j]で求められるようにし(図 2)，さらにルール数の上昇と共に状態数よりも入力アルファベット数の方がはるかに少なくなるため，check には入力アルファベットを格納することでメモリ削減と同時に配列アクセス数減少によるスループットの向上を図る．このとき base を排除することで MySparse における状態番号は i から base[i]に変化することになる．その他の操作については Johnson と同じである．それでは圧縮手順について説明する．Johnson 同様，FULL の各行をスライドさせ重ならないように next に詰めるのが目的となる．

① FULL の各行における全ての要素についての空き場所までの距離を一時的な配列 base の行番号の位置に格納する．
② base に基づき next の構築を行い，next 内の古い状態を新しい状態に書き換える．
③ check には next の情報がどの入力アルファベットによって現れるかというアルファベットを格納する．

このように格納することで MySparse においては，現在の状態を i，入力アルファベットを j とすると，次の状態は check[i+j]が j であった場合に，next[i+j]とすることで求めることができる．

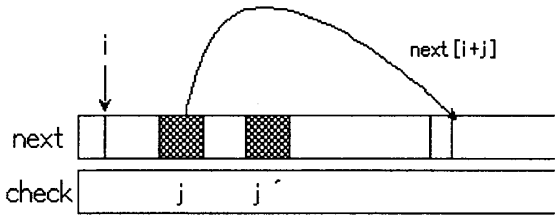

図 2　提案圧縮法による検知

また Aho-Corasick オートマトンの failure link においてはルートエッジの割合が一番高いが，その次にルートノードの直接の子供へのエッジ(レベル 1 エッジ)の割合が高いことが調査から判明した．従ってレベル 1 エッジも状態遷移表から省くことでさらなる省メモリ化を図る．レベル 1 エッジ除去時での状態遷移プログラムは以下の図 3 のようになる．

```
state = start_state;
while(input(j)){
   if(check[i+j] == j) state = next[i+j];
   else state = next[start_state+j];
}
```

図 3　レベル 1 エッジ除去時の状態遷移プログラム

5.　評価

Johnson，BANDED，MySparse の 3 つの圧縮法について評価を行なった．Snort に BANDED と MySparse を組み込み，検知文字列にはランダム文字列 10Mbyte を与えた．検知ルールは Snort の ftp，http，smtp の 3 つのルールを採用し，レベル 1 エッジ非除去時，除去時においてのメモリ消費量とスループット評価を行なった．

評価環境は Opteron2220SE 2.80GHz×2，12GB RAM，RedHatEnterpriseLinuxVersion4，Linux2.6.955.0.9.ELsmp，GCC3.4.6 で，コンパイラの最適化オプション-O2 をコンパイル時に用いた．

5.1.　レベル 1 エッジ非除去時評価

まずメモリ消費量についての評価結果を述べる(表 1)

表 1　レベル 1 エッジ非除去時メモリ消費量評価

ルール	メモリ消費量(キロバイト)			
	FULL	BANDED	Johnson	MySparse
ftp	1,824	1,429	1,516	1,132
http	1,670	1,352	1,365	1,019
smtp	1,251	1,007	1,017	759

MySparse でのメモリ消費量が最も少なく，FULL の約 60%，BANDED の約 75%となっている．

次にスループット評価の結果について述べる(表 2).

表 2　レベル 1 エッジ非除去時スループット評価結果

ルール				
	FULL	BANDED	Johnson	MySparse
ftp	275.27(1.0)	0.534	0.472	0.422
http	286.17(1.0)	0.519	0.479	0.485
smtp	312.55(1.0)	0.500	0.457	0.463

FULL には Mbyte/sec，その他の方法は FULL を 1.0 とした相対速度で示す．

FULL の速度が最も速く，Johnson が最も遅くなった．MySparse においては FULL の半分，BANDED の約 8 割となった．FULL よりも全体的に速度が下がっているのは配列アクセス数増加によるものと思われる．またそれぞれの圧縮法についてそれぞれのルール見ると速度に大きな差はなく．実行速度がパターン数に依存しない AC 法の特徴が表れている．

5.2.　レベル 1 エッジ除去時評価

まずはメモリ消費量について評価結果を述べる(表 3).

表 3　レベル 1 エッジ除去時メモリ消費量評価

ルール	メモリ消費量(キロバイト)		
	BANDED	Johnson	MySparse
ftp	192.9	41.7	26.1
http	145.1	36.8	22.8
smtp	96.0	24.7	14.9

MySparse のメモリ消費量が一番少なく，FULL(表 1)の約 1/83，BANDED の約 1/7 となった．

次にスループット評価の結果について述べる(表 4)

表 4　レベル 1 エッジ除去時スループット評価結果

ルール			
	BANDED	Johnson	MySparse
ftp	0.293	0.490	0.924
http	0.289	0.466	0.881
smtp	0.267	0.427	0.800

MySparse の速度が極端に上昇し，8 割～FULL に匹敵する速度になっている．これは極端にメモリサイズが小さいことによるメモリアクセスの局所性の向上によるものであると考えられる．他の手法では配列アクセス数の増加により速度が落ちている．

6.　おわりに

軽量なネットワーク IDS 向けの検知ルール圧縮法を提案した．最終的な圧縮によるメモリ消費は FULL の約 1/83，BANDED の約 1/7 となり，スループットは FULL の約 0.9 倍，BANDED の約 3 倍となった．今後は検知アルゴリズムを見直し，さらに新たな圧縮法の提案やスループット向上を目指していく予定である．

謝辞

本研究の一部は，科学研究費補助金特定領域研究「情報爆発 IT 基盤」(領域番号 456)「通信端点における分散検知モジュールによる侵入防止機構」(課題番号 19024008)をうけて行われた．

参考文献

[1]B.Caswell, J.Beale, J.C.Foster and J.Faiecloth, "*Snort 2.0 Intrusion Detection*", Syngress Publishing (2003)

[2]A.V.Aho and M.J.Corasick, "*Efficient string matching: an aid to bibliographic search*", CACM, 18(6),pp.333-340 (1975)

[3]S.C.Johnson, "*Yacc: Yet another compiler compiler*",UNIX Programmer's Manual, Vol.2, Holt, Rinehart, and Winston, UA, pp.353-387 (1979)

[4]M.Norton, "*Optimizing pattern matching for intrusion detection*", Sourcefire White Paper (2004)