社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－OS－109 (6)
2008／8／7

カーネルモジュールを用いた通信端点における侵入検知システム

松井 卓[†] 前田敦司[†] 西 孝王[†]
渡辺祐介[†] 山口喜教[†]

ネットワーク経由での攻撃に対処するためネットワーク侵入検知システム(NIDS)の導入が進んでいるが，内部からの攻撃を検知できない/スケーラビリティに欠けるなどの問題が指摘されている．これらの問題点を解決するため筆者らが提案している，通信端点における軽量ネットワーク侵入検知システムにおいて，カーネルモジュールを用いる手法を提案し，その実装と評価について述べる．提案システムは，一般的なNIDSで用いられている検知ルールをメモリ効率の良いDFA状態遷移表としてLinuxのカーネル空間にロードし，プロトコルスタック内のフックを用いてパケットのペイロードを検査する．提案システムを実装し評価を行った結果，実用的な範囲内での速度低下は25%未満であった．メモリ消費量も非常に小さく，十分実用的であると考えられる．

Intrusion Detection System at Network Endpoint using Kernel modules

MATSUI Taku,[†] MAEDA Atusi,[†] NISHI Takao,[†]
WATANABE Yusuke[†] and YAMAGUCHI Yoshinori[†]

To detect intrusion attempts through network, Network Intrusion Detection Systems (NIDSs) is being common as a tool for network administrators. Current NIDSs have inherent problems such as inability to detect attacks from internal network or lack of scalability. To resolve these problems, authors have proposed lightweight network intrusion detection framework at network endpoint. As a part of the framework, we describe in this paper a method using kernel module. Implementation overview and evaluation results are presented. Proposed system converts detection rules used in popular NIDS software into memory-efficient DFA state transition table and loads the table into Linux kernel space. The table is used to inspect packet payload at hook points inside protocol stack. Evaluation results show that overhead in throughput is less than 25% in practical settings, and memory overhead was very small, which supports practical usefulness of the approach.

1. はじめに

インターネットをはじめとするネットワーク社会の発展により著しい利便性の向上を享受する反面，ネットワークに接続することで不正アクセスやサービス拒否攻撃等の脅威に直面することになる．現実に，Webページの改竄や情報漏洩は既に珍しいことではなく，ニュースにも取り上げられ一般に認知されるまでになっている．ファイアーウォールの導入が一般的になってはいるものの，不正アクセスの被害は後を絶たない．これらネットワーク経由での攻撃に対処するため，近年，ネットワーク型侵入検知システム(ネットワーク型IDS・NIDS)の導入が進んでいる．一般的にNIDSは，外部ネットワークとの接続点付近に専用のマシンを設置して，外部から来る全パケットを検査する．しかしネットワークは急速に高速化しており，従来の接続点付近でネットワークを監視する手法では高速なネットワークに対応できないといった問題が指摘されている．また設置場所によっては内部から内部への攻撃も検知できない．

† 筑波大学
University of Tsukuba

そこでこれらの問題を解決するために，従来のように外部ネットワークとの接続点付近に専用のマシンを設置するのではなく，通信端点となるサーバやクライアントに軽量な検知モジュールを配置することで負荷を分散させる手法を筆者らは提案してきた．我々の最終目標は軽量検知フレームワークの構築である．本稿では分散検知モジュールの概要と実現のための諸技術を紹介したうえで，実装にカーネルモジュールを用いることでメモリアクセスのオーバーヘッドを抑えスループットの向上を図るとともに，UDP や ICMP といった従来のユーザプロセスレベルでは効率のよい検知が困難であった通信の検知が可能になることを示す．

2. 侵入検知システムとその問題点

2.1. NIDS とは

ネットワーク型 IDS(NIDS)は侵入や攻撃のパケットが対象のネットワークに流れていないか監視するシステムで，侵入やその兆候を検出した場合，管理者への通知やファイアーウォールと連携してパケットの遮断等を行うものである．

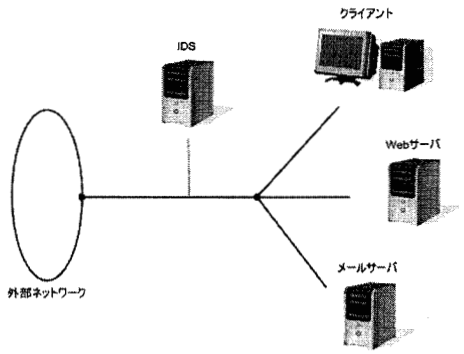

図 1 一般的な NIDS の構成

一般的に NIDS は図 1 のような外部ネットワークとの接続点付近に専用のマシンを設置して，スイッチングハブのミラーリングポートなどを利用して経路上を流れるすべてのパケットを取得し検査を行う．

2.2. Snort

フリーの NIDS の代表例として Snort がある．オープンソースの IDS で，ネットワークを流れるパケットを検査し攻撃特有の文字列（シグネチャ）がパケットに含まれていないか検査するものである．導入が比較的容易で，オープンソースでありながら市販の IDS と同程度の機能を持っていることから，現在広く利用されている．検知に用いられる検知ルールは 2007/07 時点で約 8300 個ある．なお本研究においてもこの検知ルールを使用している．

2.3. NIDS の問題点

NIDS は外部ネットワークとの接続点付近に設置されるという性質上，いくつかの問題を持っている．

パケットはまとまって流れてくるのではなく他の通信のパケットと混ざって任意の順序で到達するため，サーバの中では順序等を判断して TCP reassembly や IP defragmentation の処理を行っている．そのため 1 つの通信のパケットがすべて到着するまで一時的にパケットを保持しておく必要がある．

このような通信を検査するためには，通信を構成する任意の順番で届くパケットに対して，IDS 側でも TCP reassembly や IP defragmentation の処理をしなければならない．処理すべきセッションの数が少なければ問題はないが，高速なネット環境が普及し外部ネットワークとの接続点を通るセッション数が膨大になると，これらの処理に必要なメモリ量も膨大になりメモリ不足が問題となりうる．また IDS の処理性能を超える通信量が発生した場合，全てのパケットを取得し検査することができず，不正アクセスを許してしまう可能性がある．

また検査を行うのはネットワークの接続点付近など局所的であり，IDS の設置場所によっては，ネットワーク内部のコンピュータからそのネットワーク内のサーバに対して攻撃や侵入が行われた場合その通信のパケットを取得することが原理的に不可能であるため，攻撃を許してしまうことになる．

3. 分散検知モジュール

3.1. 分散検知モジュールの概要

我々は IDS の持つ諸問題に対応するため，従来のように集中的に通信の検査を行うのではなく，通信端点となる個々のマシンに軽量な分散検知モジュールを分散的に配置する手法を提案している．これにより次のような利点が得られる．

まず IDS によるパケットの TCP reassembly や IP defragmentation の処理が必要なくなる．これらの処理は通信端点ではすでに行われており，処理が行われた後にパケットを取得すれば IDS による独自の処理は必要ないので IDS を軽量に実装できる．また実際にマシンが受け取るデータと同じデータを IDS が取得できるため，検知漏れが発生することはない．内部から内部への攻撃など，IDS の設置場所によっては検知できない経路での攻撃にも対応できる．

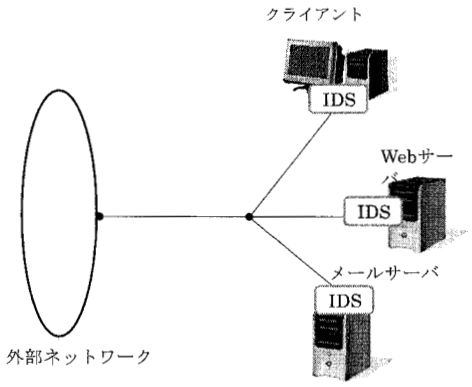

図 2 提案する通信端点における IDS

3.2. 実現に必要な技術的課題

通信端点における侵入検知システムを考える場合，以下の二つのフェーズを考えることができる．

まず，通信データを取得するフェーズである．パケットの TCP reassembly や IP defragmentation の処理後のデータを取得しなければならないため，Snort 等で使われている libpcap（TCPDUMP 等，パケットをキャプチャする時に広く使用されるライブラリ）ではこれらの処理後のパケットは取得できず使用できない．これは xinetd を使用する方法と Linux のカーネルモジュールを使用する方法を後述する． なお xinetd を用いる手法は従来提案してきたものであり，本稿ではカーネルモジュールを使用する手法を提案する．

次に，取得したデータの中に攻撃のシグネチャが含まれていないかどうか検査する，パターンマッチングを行うフェーズである．NIDS はシグネチャとペイロードのパターンマッチングを行わなければならず，ここに多くの処理時間がかかってしまっている．よって使用されるパターンマッチングエンジンは高速であることが強く求められてきた．しかし本手法ではすでに他のサービスが起動しているサーバ等での使用が想定されているため，高速さだけでなく省メモリであるという軽量さも強く求められる．

4. NIDS 向け省メモリ検知ルール圧縮法

パターンマッチングエンジンの省メモリ化手法として提案している圧縮法[5]は，Aho-Corasick アルゴリズムの圧縮法である Johnson 法を拡張し，メモリ消費削減，スループット向上を図ったものである（詳細は[5]を参照）．

4.1. アルゴリズムの概要

Aho-Corasick アルゴリズムとは，複数のパターンを受理する DFA に，マッチに失敗したときに先頭からたどり直さなくても良いように遷移（failure link）を付け加えて，効率よくパターンマッチングを行う手法である．ここで用いる状態遷移表は，素朴には二次元配列で保持するという手法をとる．この二次元配列は初期状態へ戻る failure link がほとんどであり，これを除くと二次元配列はかなり疎になる．このことを利用して圧縮が可能である．

Johnson 法は，疎になった二次元配列を重ね合わせて base, next, check の 3 つの一次元配列で状態遷移表を表現するものである．

我々はこの Johnson 法にさらに拡張を加えた手法を提案してきた．Johnson 法で 3 つの一次元配列で表現していたものを，2 つの配列で表現するように拡張し，初期状態の直接の子への link（レベル 1 エッジ）も除去するという，2 つの改善を加えた．

4.2. 省メモリ検知ルール圧縮法の評価

[5]にて行われた評価を引用する．Snort のエンジン部分にレベル 1 エッジの除去を行った Johnson 法と我々の手法を組み込み，Snort の ftp 等のルールを読み込みランダム文字列を与えた．結果を表 1 と表 2 に示す．

表 1 メモリ消費量の評価（[5]より引用）

	メモリ消費量(バイト)		
ルール	FULL	Johnson	我々の手法
ftp	1,824,160	41,752	26,094
http	1,670,240	36,840	22,824
smtp	1,251,120	24,753	14,910

表 2 スループットの評価（[5]より引用）

	スループット(Mbyte/sec)		
ルール	FULL	Johnson	我々の手法
ftp	121.97	100.02	161.32
http	123.99	98.05	161.32
smtp	125.01	95.56	161.32

なお，表中の FULL はレベル 1 エッジ非除去である．我々の手法の方がスループットが向上しているが，これは保持するメモリ領域の極端な減少によるメモリアクセスの局所性の向上が考えられる．上記の通り圧縮率もスループットも良好な結果が得られたので，後述する xinetd を用いた TCP レベル侵入検知モジュールとカーネルモジュールを用いた侵入

検知モジュールは，どちらもこの圧縮化手法を用いたパターンマッチングエンジンを使用している．

5. xinetd を用いた TCP レベル侵入検知

TCP reassembly や IP defragmentation の処理が済んだデータを受け取りユーザ空間で検知を行うモジュールを，"idsmon"として提案してきた[6]．idsmon は TCP/IP 通信が来たときに xinetd 経由で検知プロセスを起動し，サーバプログラムに対する攻撃を検知する．

5.1. idsmon の特徴

idsmon は，1つの通信セッションに対して，1つの検知プロセスで動作し，その検知プロセスは対象のセッションのみを検査する．したがって，同時刻に多数の通信セッションが存在する場合は，idsmon プロセスも複数立ち上がることになる．よって，idsmon は可能な限り軽量であることが強く求められる．そのため，idsmon は，侵入検知ルール集合のうち，特定の通信の検査に必要なルールのみを抜き出し，それをメモリ上にコンパクトな形で保持して検査を行う．

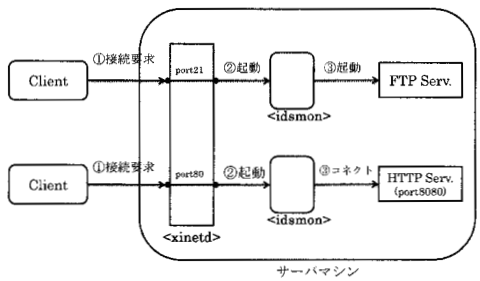

図 3 idsmon の概要

起動された idsmon は検査のためのセットアップを実行した後，子プロセスとしてサーバプログラムを起動し，クライアントとサーバ間の通信を中継する．この際に通信内容をルールに従って検査し，ルール中のパターンにマッチした場合に，指定されたアクションを実行する．

5.2. idsmon の課題

xinetd を用いた TCP レベル侵入検知モジュールは，当然のことながら TCP/IP 以外のプロトコルの検知ができない．そのため TCP/IP 以外のプロトコルの検知を行う xinetd を用いる以外の別の手法が必要となる．またカーネル空間から idsmon に通信データのバッファをコピーし検査を行ったあと，該当するアプリケーションに通信データを渡すためもう一度バッファのコピーを実施しており，オーバーヘッドとなっている．

6. LKM を用いた侵入検知モジュール

本稿では新たに、通信端点における侵入検知モジュールを Linux のカーネルモジュールを用いて実装する手法を提案する．これによりプロトコルに寄らない検知が可能になり，またオーバーヘッドを削減してスループットの向上が見込まれるということを示す．

この手法では，実装にカーネルモジュールを用いることで，バッファに直接アクセスしメモリアクセスのオーバーヘッドを抑えることができる．これによりパターンマッチングの速度の向上を図る．またカーネル内で動作するため，UDP や ICMP といったユーザプロセスレベルでは効率のよい検知が困難であったプロトコルでの検知が可能になると考えられる．

6.1. カーネル空間で行うことによる利点

カーネル内からネットワークのデータを取得する方法として，カーネルモジュールを用いて検知プロセスをカーネル内に移動するという本手法以外に，libpcap 等を用いてカーネル内の通信データをユーザ空間に送り検知を行うという方法も存在する．なお Snort はこの libpcap を利用している。

そこで予備的な実験として，libpcap を用いてデータを取得するプログラムを実装し，単に取得するだけで検知を行わない場合のスループットを測定した．動作環境は 6.4 の Pentium4 マシンである．その結果 25%の速度低下が発生した．同様にカーネルモジュールで行った場合の速度低下は 1%程度であった．

これは libpcap を使う場合はカーネル空間からユーザ空間へのバッファのコピーが発生するのに対し，カーネルモジュールであれば構造体が渡されるだけでバッファのコピーが発生しないためであると考えられる．またそもそも libpcap の場合はこの他に IP defragmentation 等の処理が必要となる．以上のことからカーネル空間で行うことの利点は十分あると考える．

6.2. 基本構成

提案手法ではあらかじめ検知ルールからパターンマッチングエンジンを生成しておく．まず Snort のルールセットから検知に用いる文字列を抽出し C のソース形式で出力する．このソースは検知モジュールのコンパイル時にリンクされ，検知を行うカーネ

ルモジュールが生成される．生成されたカーネルモジュールをロードするとプロトコルスタック上のフックへ登録され，以後到着した通信の検知を行う．攻撃を検知した場合，記録用のバッファにログを記録し，出力を行うデバイスドライバを用いてユーザに通知を行う．

Snort のルールが更新された場合は，同様にモジュールを生成しなおし，カーネルモジュールのアンロード・再ロードを行う．

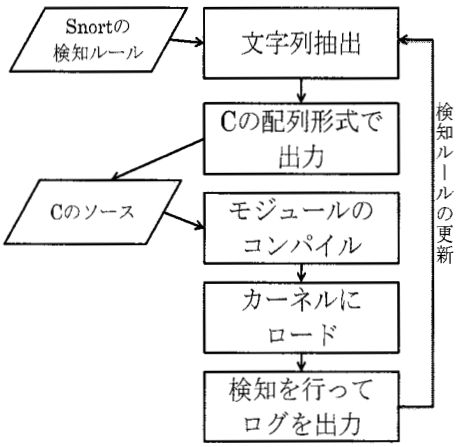

図 4 基本的な処理の流れ

6.3. 実装

検知モジュールは，パケットのパターンマッチングを行うマッチャーと検知ルールのセットで構成されている．そしてこれらはカーネルモジュールとしてカーネルにロードされる．

検知ルールから生成された DFA 状態遷移表をあらかじめ C 言語の配列形式で保存しておき，コンパイル時に検知モジュールにリンクする．検知ルールは，UDP/IP の場合・ICMP の場合等に分割してカスタマイズすることが可能で，対象となるサーバに必要なルールのみを抽出して読み込むことで，省メモリ化を果たしている．

ロードされると Linux のプロトコルスタックにフック処理を登録する機能を用いて，IP 層のフックポイントで検知処理を起動するように登録する．このフックは NF_IP_LOCAL_IN(自分宛のパケットと判断し IP デフラグメンテーションを実行後で，IP 層から上位プロトコルに渡される直前)の位置に登録される．そのため IP defragmentation の処理の後である．

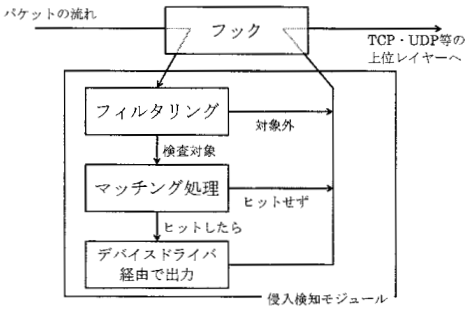

図 5 検知動作の概要

システムの概略を図 5 に示す．マシンに何らかのパケットが到着するとフックに登録した関数が起動され，検知モジュールにパケットのバッファが渡される．パケットのバッファは構造体の形式で渡され，その際にデータ自体のコピーは行わない．

検知モジュールは受け取ったパケットのヘッダを検査し，検査対象であればパケットのペイロードをマッチャーに渡し検査を行い，攻撃を検知した場合はデバイスドライバを通して警告を行う．

6.4. 評価

スループットの計測には Netperf[7]を用いた．検知ルールは Snort のルールのうち，UDP と IP に関するルール(670 個)で現れる文字列を使用している．なお，検知モジュールのサイズは 180396byte であり，現在のサーバであれば十分に軽量と言える．

まず Netperf を用いて同一マシン内にサーバとクライアントを動作させて UDP/IP の送受信を行い，スループットを算出した．評価環境は CPU: Pentium 4 (2.20GHz), RAM:512MB, kernel: 2.6.18, GCC 4.1.2 で，コンパイラオプションは-O2 で行った．ここで，一般的な MTU である 1500 の場合と，ローカルループバックのデフォルトである 16436 の場合，2 種類を計測した．これは，MTU が小さい場合パケットのメッセージサイズを大きくするとフラグメント化も比例して多く発生するため，IP defragmentation も多く発生し負荷となる．MTU のサイズを変えることでフラグメント化の回数を変化させ，スループットに影響があるかどうかを見る．

結果を以下の図と表に示す．図は縦軸が速度(Mbps)，横軸が 1 パケットあたりのパケットのサイズを表している．

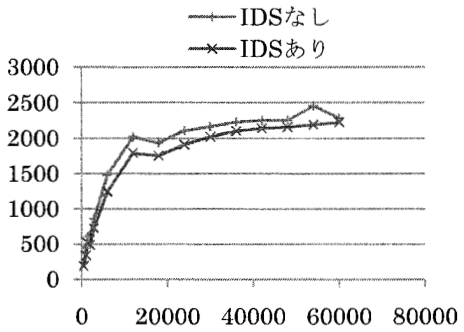

図 6 MTU=1500 でのスループット

表 3 MTU=1500 でのスループットの抜粋

	Packet size(byte)		
	500	1000	60000
IDS なし	227.8(1)	440.0(1)	2280.9(1)
IDS あり	190.9(0.84)	337.7(0.77)	2230.0(0.98)

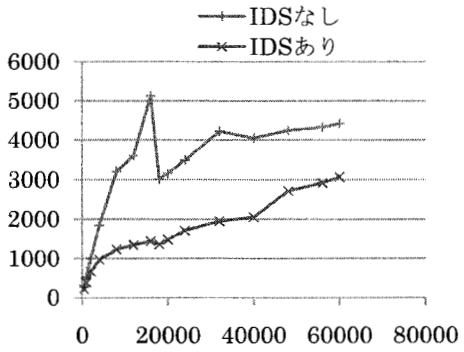

図 7 MTU=16436 でのスループット

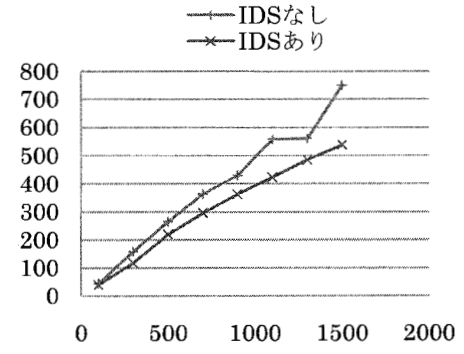

図 8 図 7 のパケットサイズ 2000 までを拡大

表 4 MTU=16436 でのスループットの抜粋

	Packet size(byte)		
	500	1000	60000
IDS なし	270.7(1)	523.6(1)	4423.4(1)
IDS あり	225.7(0.83)	399.9(0.76)	3040.7(0.69)

MTU が 1500 の場合，サイズを変化させても IDS による大幅な速度低下は見られなかった．ただし次の MTU が 16436 の場合と比べて全体的に遅いスループットとなっている．

それに対し MTU が 16436 の場合は，パケットサイズが大きくなるにつれて差が大きくなっている．これはパケット数が少なくなった分オーバーヘッドが減少し，パターンマッチングの負荷が顕著に表れたためと考えられる．16000byte や 32000byte 付近でのスループットの周期的な悪化は MTU との関連である．

次に実際のネットワークにてスループットの計測を行った．評価環境は侵入検知モジュールを動作させるマシンとして、CPU: Opteron 2220SE 2.8GHz*2, RAM:12GB, kernel:2.6.9, gcc 3.4.6 のオプション-O2, クライアントマシンとして、CPU: Xeon 5130 2.0GHz, RAM:2GB, kernel:2.6.9 を使用した．

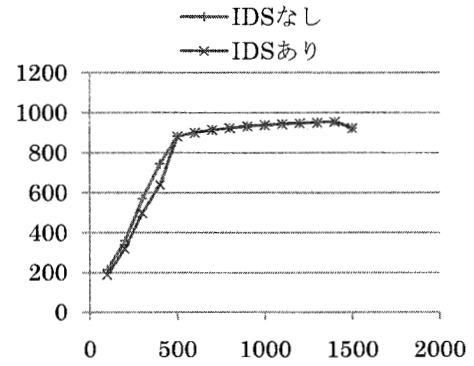

図 9 実マシン間でのスループットの測定

図 9 より，実マシン間では侵入検知モジュールによる速度低下が起こっても，ネットワークの性能の限界の影響の方が大きく，問題とならないことが示された．

7. おわりに

本研究では，通信端点において侵入検知を行う軽量な検知モジュールの，カーネルモジュールを用い

た手法の提案し実装を行った．本手法を用いることで TCP/IP 以外のプロトコルの検知が可能になり，比較的高速な検知を実現することができた．また TCP/IP 以外の検知をおこなう場合，通常の使用であれば速度低下は 25%未満で，組み込むモジュールのサイズも数百 KB 程度である．これらの結果より，十分実用的で許容し得るレベルのオーバーヘッドで個々のサーバに検知モジュールを導入できることが示せたと考える．

ただし本手法ではフックポイントの位置に制約があるため，IP defragmentation 後にフックすることはできるが，TCP reassembly 後にフックすることはできない．そのため TCP の検知を行うにはカーネルの書き換えを行うか idsmon とお互いを補い合う形で実装する必要がある．

また今後の課題として，さらに効率を向上させオーバーヘッドの削減を目指すとともに，Snort の検知ルールのオプションにも対応させていくことが挙げられる．侵入検知システムとして完成度を高めるべく，研究を進めていく予定である．

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] M. Roesch : "Snort - lightweight intrusion detection for networks", Proc. 13th USENIX conf. on System Administration, pp. 229-238 (1999). [2] A. V. Aho and M. J. Corasick: "Efficient string matching: an aid to bibliographic search", CACM 18(6), pp.333-340 (1975)

[3] 前田敦司，渡辺祐介，西孝王，山口喜教: "通信端点における軽量侵入検知モジュールの試作" : 信学技報, Vol.106, No.436, pp.13-18 (2006)

[4] 高橋浩和，小田逸郎，山幡為佐久:"Linux カーネル 2.6 解読室" :ソフトバンククリエイティブ(2006)

[5] 西孝王，前田敦司，山口喜教，" 軽量ネットワーク IDS 向け検知ルール圧縮法の提案" ，情報処理学会第 70 回全国大会(2008)

[6] 渡辺祐介，前田敦司，山口喜教，" 通信端点における TCP レベル侵入検知モジュールの実装" ，情報処理学会第 70 回全国大会(2008)

[7] http://www.netperf.org/