ポートバイオレーションに着目した侵入検知システムの実装

小林　孝史

要　旨

近年，コンピュータやネットワークセキュリティの事件（例えば，コンピュータウィルスや侵入事件など）が数多く報告され，毎日のように新聞記事として報道されるようになってきている．対策が不十分であることも原因の一つではあるが，大きな問題として捉えなければならないのは，正常な事象を異常として検出してしまうことや異常な事象を見過ごしてしまうことである．不確定性原理によると，これらの誤検出は検出システム自身に問題があるわけではなく，問題となっている事象の周辺知識や属性情報を完全に知ることができないためであるということになる．

本論文では，ファイアウォールの運用をより少ないコストで効率的に行うための情報量による分類基準と，それに対応したIDSの提案を行い，その検出実験を通じて実用可能性を検証する．

The Implementation of Intrusion Detection System focused on Port Violation

Takashi KOBAYASHI

Abstract

In recent years, the computer and network security incidents, such as viruses and intruders, were frequently reported and written on in newspaper articles. There are some problems, such as false positives or false negatives, in the computer and network security. A "false positive" is when the system detects an event as an anomaly, when it is actually a normal event. A "false negative" is when the system detects a normal event, but it is really an anomalous event. The reason why those problems have occurred is not due to the detection systems themselves, but we cannot understand all the attribute information of the event at hand.

In this paper, I propose a classifying standard based on the amount of information, and the IDS (Intrusion Detection System) corresponding to the standard. Then I verify the abilities of practical use through the experiment of detection.

1．はじめに

インターネットの急速な普及とともにネットワークのトラヒックが増大し，それらのパケットの量は人の手によって管理のできる範囲を越えているため，システムによる自動化を行うことが必要となっている．パケットの管理を行うシステムとしての侵入検知システム（IDS: Intrusion Detection System）では，パケットをキャプチャしそのパケットの中に既知の異常なコードが含まれていれば，それを異常な事象であると検出する．しかし，この検出方法では過去に発生した事象しか検出することができないため，未知の事象を検出することが不可能である．これがパターンマッチング型の検出手法の欠点である．

また，実は異常な事象ではないが，異常なものであると検出してしまうfalse positiveの問題，その逆である異常な事象を見過ごしてしまうfalse negativeの問題が発生する可能性を考えないわけにはいかない．不確定性原理によると，ある事象の背景あるいは原因を完全に知ることはできないことから，false positiveやfalse negativeの原因はシステムそのものに問題があるのではなく，原理的に不可能であることによる．しかし，少なくともその割合を減少する何らかの努力が必要である．

例えば，あるサービスポートに関する通信をファイアウォールで許可している場合，一般的にはそのポートを使う通信はすべて正しいと信頼している．しかし，そのポートを使って全く別のサービスを立ち上げることも可能であるから，そのポートを使うすべての通信が正しいと判断することは危険である．このように正しいサービスと別のサービスが混在している状況において，異常な事象を判断するためには，別の要素を持って判断しなければならないことは明らかである．例えば，TCP（Transmission Control Protocol）のスリーウェイ・ハンドシェイク（3-way handshake）[1]が終了した後の最初のパケットをキャプチャし，その内容によって正しくそのポートを使用しているかどうかを判断する手法である．この方法では，通信の内容部分についてキャプチャしているのではなく，通信内容の傍受とは一線を画していることから，組織的にも説明を行いやすい利点はある．

TCP/IPのモデルにおいては，送信元および宛先IPアドレスとポート番号を基にしたソケットによって送信元と宛先間の通信を行っており，使用するポート番号は基本的に自由に決めることができる．その中でもウェルノウン（Well-Known）ポートと呼ばれる1023番以下のポート番号についてはIANA（Internet Assigned Number Authorities）によって割り当てが決められており[2]，主としてサーバ用途に利用されている．Unixなどの一部のOS環境においてはWell-Knownポートを利用するためには管理者権限を要する仕様となっているため，利用者のアプリケーションプログラムによるポート番号の不正使用はほとんど発生することはない．しかし，一般的に利用されているOS環境においては，Well-Knownポートを利用するための権限が必要ないために，1023番以下のポート番号についても自由に使用することができるようになっている．また管理者（多くの場合，侵入者）によって，Well-Knownポートに全く別の機能を割り当てるといったこ

とが行われている可能性もある．このような状況では，ポート番号とその機能割り当ての違反が発生し，侵入や不正使用の検知をより難しくしている．1024 番以上のポート番号（エフェメラルポートと呼ばれる）は 65535 まで利用でき，Gnutella，WinMX などに代表される P2P 型ファイル共有／交換アプリケーションのほとんどが自由にポート番号を設定することができるようにする等，管理者以外のプログラムまたは権限によって自由に利用することが可能であるため，エフェメラルポートに関連したトラヒックの監視が事実上不可能になる状況が発生しつつある．こうした事情から，利用者の善意に頼ったポートの使用制限を求めていることが多く，ポート番号の正しい使用をシステム化できていないことも事実である．

このようなファイアウォールの設定に際して，IP アドレスを基に行うのか，ポート番号を基に行うのかなどのように，どの情報を基にパケットを効率的に分類するための手法については管理者の経験則によるところが大きい．

本論文では，「インターネット上のホスト」から発生する情報量に着目し，情報量を基準とした分類方法をファイアウォールの設定に活用することにより，より小さなコストでファイアウォールを運用することが可能となることを示す．

また，IDS の中には TCP の状態遷移に着目したものも存在するが，一定期間状態遷移の状況を保存し状態遷移の違反を発見するためのオーバーヘッドがかなりのコストを必要とする．IANA で割り当てされているポート番号とその機能の対応の違反（以下，ポートバイオレーションと呼ぶ）を検出するために，TCP コネクションが確立した直後のパケットの内容を利用することができることを示す．また，侵入事件の発生によりバックドアが仕掛けられるといった場合において，バックドアが許可されたポートで設置されているような状況であってもそのポートに対応した機能が正しく使用されていない場合は，ポートの使用違反が発生していると検出することができることも示す．

次章では，エントロピー関数によって求めることができる情報量によって，ファイアウォールの構成方法を決定できることを示し，続いてその方針に基づいて運用を行った場合に発生することが予想されるポート番号の使用違反の発生メカニズムについて示す．さらに，ポート番号の使用違反を検出するための IDS の実装について説明し，最後に検出実験とその結果の検証を行う．

2．エントロピー関数による情報量の計算

ある IP アドレス，ポートに関係するトラヒックに関して，正常なトラヒックなのか異常なトラヒックなのかを分類する基準として，エントロピー関数から得られる情報量（amount of Information）を考える[3]．この情報量の大きさからその属性が分類に関してどの程度の影響力を与えているのかということの基準にする．ある情報の正負を決定するために属性 F_i（ここで $i = 1, 2, \cdots$）を利用するものとし，その生起確率を P_iとする．確率 P_iのうち，その情報が正であるときの属性 F_iが出現する確率を P_i^+，負であるときの出現確率を P_i^-とすると，F_iに関するエントロ

ピー関数は

$$H_{F_i} = -P_i^+ \log_2 P_i^+ - P_i^- \log_2 P_i^- \qquad (1)$$

で表すことができる．それぞれの属性 F_i についてエントロピー関数の値を求め，より小さな値を持つ属性の順に利用することで，より効率的に情報を分類することができる[4]．

この考え方をファイアウォールの構成方法に適用すると，ファイアウォールに入ってくるパケットという情報に関する属性 F_i，つまりIPアドレスという属性，ポート番号という属性のそれぞれに対して式(1)のエントロピー関数から計算できるエントロピーにより，パケットをどの属性で分類することがより効率的であるかという評価を行うことができることになる．

【エントロピー関数による情報量の計算例】

一つの組織に対して，IPv4アドレス空間のうちのBクラスIPアドレスが割り当てられている場合，収容できるホスト数は最大65536台である．これらのホストのうち300台がインターネット上にある別のホストに直接接続できるようになっており，そのうち30台が異常なトラヒックを発生しているとする．使用するポートについてはそれぞれのホストに対して65536個使用できるものとする．このようなホストを分類するための根拠として，これらのホストのIPアドレスに関する情報量を式(1)のエントロピー関数によって計算する．属性はそのIPアドレスがフィルタされているかどうかによって正負を判断するものとし，ここでは65536ホストのうちの300ホストが正，つまりインターネット上の別のホストと直接通信を行うことができるものとしている．その他は接続することができないものとして，そのエントロピーは0，つまり分類する必要がないホストとして取り扱うことになる．そして，直接通信を行うことのできるホストのうち30ホストが異常なトラヒックを発生している．つまり正であるとして扱う．このような条件に基づいてIPアドレスに関する情報量を計算すると次のようになる．

$$\begin{aligned} H_{IP} &= \frac{300}{65536} H_{IP=registered} + \frac{65236}{65536} H_{IP=unregistered} \\ &= -\frac{1}{65536}\left\{ 30\log_2 \frac{30}{300} + 270\log_2 \frac{270}{300} \right\} \\ &= -\frac{1}{65536} \times (-140.699) = 0.00215 \ . \end{aligned} \qquad (2)$$

また，65536個の利用可能なポートのうち20個を許可している場合，残りの65516個のポートについては発生する情報がないため情報量は0である．そして20個のポートについても同様に常に情報が発生するため情報量は0となる．

$$\begin{aligned} H_{port} &= \frac{20}{65536} H_{port=registered} + \frac{65516}{65536} H_{port=unregistered} \\ &= -\frac{1}{65536} 20\log_2 \frac{20}{20} = 0 \ . \end{aligned} \qquad (3)$$

つまり，ポートを基にした分類を行うと，これ以上の分類は必要ないことになる．従って，IP

アドレスを基にしたアクセス制御やファイアウォールの構成を行うよりも，ポート番号を元にして行う方がより効率的であることが分かる．

今後，アドレス長 32 ビットの IPv4 から 128 ビットの IPv6 への移行により，広大なアドレス空間（IPv4 空間の 2 の 96 乗倍）が利用できるようになると，IP アドレスによるアクセス制御では設定するアドレス数の増大が予想されるため，ファイアウォールや設定の検証を行うことが次第に困難になっていくと考えられる．それと比較して，ポート番号によるアクセス制御を行うモデルの場合，IPv6 へのアドレス体系の変化が起きたとしても，現在のポート割り当てが劇的に増大しない限りこのモデルでの運用を継続することができる．

3．ポートバイオレーション

ポート番号を基にしたファイアウォールの構成を行っている場合，ファイアウォールを通過できるポート番号とその機能の対応を確実にすることが要求される．TELNET プロトコルに割り当てられているポート 23 番で SMTP（Simple Mail Transfer Protocol）が実行されていてはいけないし，SSH（Secure SHell）に割り当てられているポート 22 番で HTTP が利用可能であってはいけない．このような状況をポートバイオレーション（Port Violation）と呼ぶことにする．ポートバイオレーションが発生していると，信頼できると仮定したポート番号に関する通信を設定しているファイアウォールの機能は無意味なものとなるため，ポート番号とそれに対応しているプロトコルの関係を確実にすることが必要となる．

以下では，BSD rlogin，SMTP，POP3，SSH，HTTP の各プロトコルにおいて発生する可能性のあるポートバイオレーションについて説明する．

3.1 BSD rlogin

rlogin（remote login）[5]は，BSD Unix に実装されているリモートのシステムへログインするための TCP アプリケーションで，リモートシステムへのログイン時の認証を省略することができるプロトコルである．省略しているものの，プロトコルの手順内でクライアントとサーバそれぞれのユーザーの妥当性の確認を行っている．

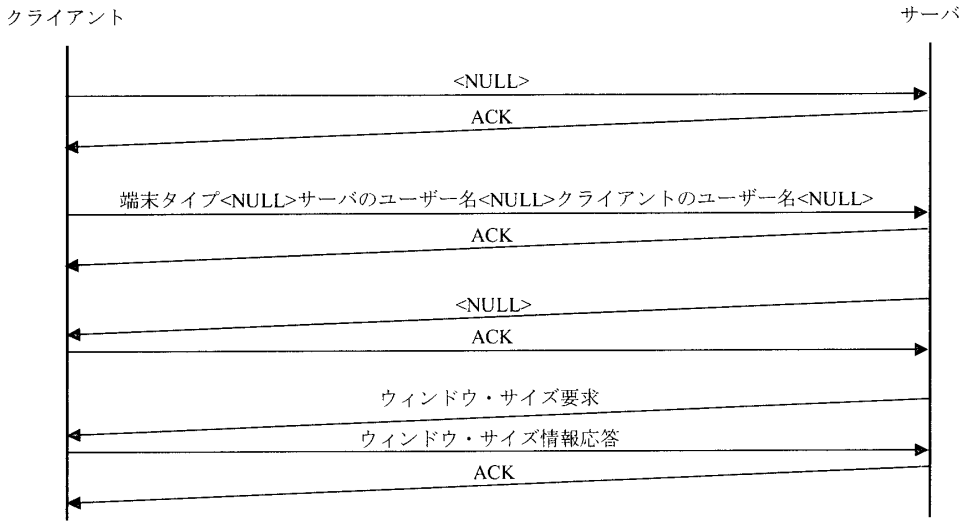

図1 rloginコネクション確立の流れ

図1にrloginにおけるコネクション確立の流れを示す．rloginプロトコルではTCPコネクション（ポート番号513）を確立した後，rloginコネクションを確立する．rloginの場合，プロトコルのイニシエーションの役目はクライアント側にあり，最初の<NULL>とそれに対するACK（ACKnowledgement）の対，クライアントのユーザー名，サーバのユーザー名，端末タイプとそれに対するACKの対，およびウィンドウ・サイズ要求とそれに対する応答の3種のデータのやり取りを必要とする．この最初の手順が存在しない場合，rloginプロトコルでの通信が行われていないので，ポートバイオレーションが発生していると考えることができる．

3.2 SMTP

SMTPプロトコル[6]はトランスポート・システムを選ばないプロトコルであるが，ここではTCPコネクション上での利用を考える．TCPコネクション（ポート番号25）の確立後，SMTPプロトコルによるサーバとクライアントの通信が行われる．

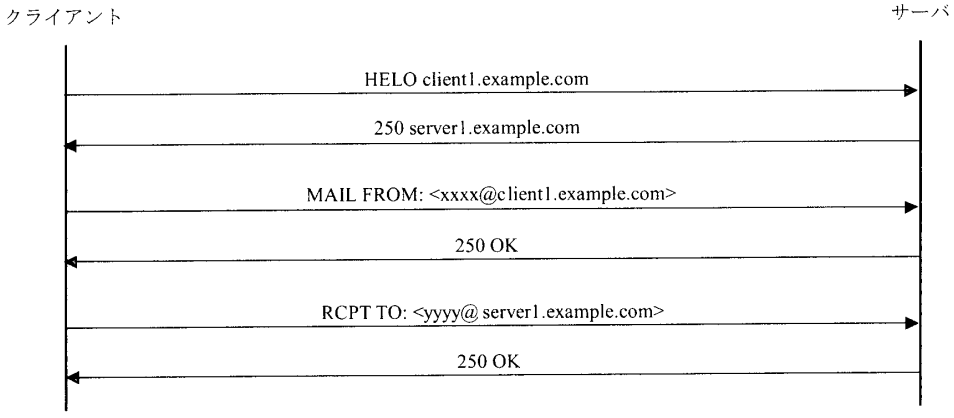

図2 SMTPプロトコルの流れ（先頭部分）

図2にSMTPプロトコルによる電子メールの送信の先頭部分を示す．SMTPでは，TCPコネクションの確立後にサーバから「220 サーバ名 SMTP・・・」メッセージが戻ってくるため，プロトコルのイニシエーションの役目はサーバ側にある．その後，クライアントがSMTPコマンドをサーバに送信し，サーバはそれに対する応答をクライアントに返す．まず，クライアントのホスト名（ドメイン名）を識別するためのHELOコマンドの送信から始まる．このコマンドは必須で，これにより通信を開始することになるため，このコマンドが存在していないとポートバイオレーションが発生していると考えることができる．

3.3 POP3

POP3（Post Office Protocol Version 3）[7]は，集中管理されているSMTPサーバに届いている電子メールをクライアント側に転送する際に用いられるプロトコルである．TCPコネクション上での利用を考えており，送信プロトコルであるSMTPと対で利用されるプロトコルである．

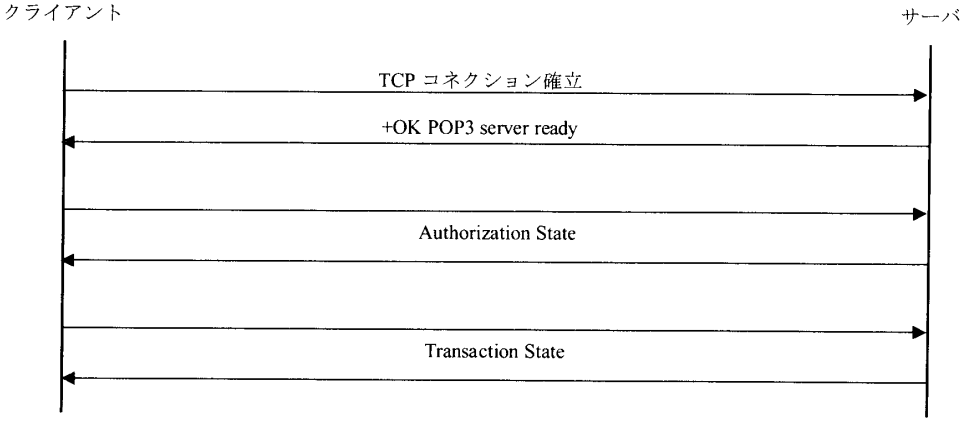

図3　POP3プロトコルの流れ

図3に示すように，TCPコネクション（ポート番号110）が確立した後，サーバからクライアントへ向けて「+OK POP3 server ready」という Greeting Message が送信されて，Authentication State と呼ばれる認証フェーズへ移行する．認証フェーズでは，「USER と PASS」，「APOP」サブコマンドや RFC1734[8]に規定されている「AUTH」コマンドによる認証を行うことができる．サーバ側に着目すると，Greeting Message の送信が認証フェーズへ移行する印となるため，サーバからこの Greeting Message が戻ってこない場合はポートバイオレーションが発生していると考えることができる．

3.4 SSH

SSH[9]は，Unix系のOSに実装されているr-系のコマンド（rlogin，rsh，rexec，rcp）のセキュリティ上の問題を解決し，安全にリモートシステムを利用できるように設計されたものである．「SSH」という用語はアプリケーションの名称だけではなく，プロトコルそのものを指す用語としても用いられているため，ここではプロトコルを指す用語として使用する．SSHプロトコルには互換性のないSSH-1.5（SSH Version1）とSSH-2.0（SSH Version2）が存在するが，

TCP コネクションの確立後の最初の送受信されるデータフォーマットには同じものが使われているため，同じ処理で検出することができる．SSH-1.5は単一のプロトコル上に複数の機能が搭載されているが，SSH-2.0ではモジュールに分割（トランスポート，認証，コネクション）され，3つのプロトコルレイヤーによりSSHのプロトコルアーキテクチャを構成している．

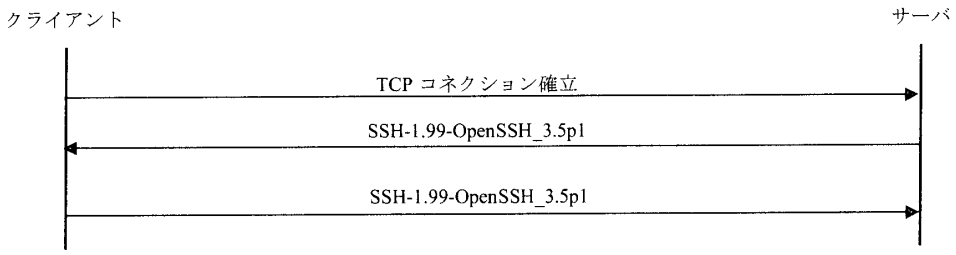

図4　SSH コネクション確立の流れ

図4にSSH コネクション確立の手順の先頭部分を示す．TCP コネクションの確立の後，バージョン文字列（Version Strings）がサーバからクライアントへ送られる．これによりサポートしている SSH プロトコルバージョンをクライアントに通知することができる．クライアントが送られてきたバージョン文字列に対応しているプロトコルを持っていれば，クライアントのバージョン文字列をサーバへ送る．その後，パケットベースのプロトコルに切り替わり，最終的にSSH コネクションが確立される．このようなバージョン文字列のやり取りが存在していない場合は，ポートバイオレーションが発生していると考えることができる．

ただし，SSH2アプリケーションにはコネクションレイヤー内（SSH Version1ではSSH コネクション内）に SSH トンネルによるポート転送機能を含んでいるため，ポートバイオレーションではない場合でも，第3者が設置した SSH トンネルを使った新たなポートバイオレーションがモニターしているホスト以外のホストで発生する可能性もある．しかし，モニターしているホストでSSHの利用が有効になっている場合にのみ，この危険性が顕在化するため，SSHの利用できるホストを限定することにより，新たなポートバイオレーションを防ぐことは可能である．

3.5　Webアプリケーション

現在の情報システム・アプリケーションの構築において，Webサーバ上でのアプリケーションとし，ブラウザを使ったユーザーインターフェイスとするような事例が多くなってきている．このようなアプリケーションの場合，使用するプロトコルはアプリケーションに依存しないHTTPプロトコルであり，ユーザーインターフェイスはクライアント環境に依存しないブラウザである．HTTPプロトコルの場合，クライアントからのリクエストの発行によって駆動するので，TCP コネクション確立直後のパケットはクライアントからのパケットとなる．

しかし，Web アプリケーションの場合は，「～ over HTTP」というプロトコルとなるため，HTTP プロトコルに加えて Web アプリケーションのプロトコルにも対応する必要がある．Web アプリケーションでは，URI（Uniform Resource Identifier）に格納されて渡されるデータ（GET

メソッドで渡される）や通信路を通じて渡されるデータ（POST メソッドで渡される）の他に，クッキーのようなデータ（サーバおよびクライアントに格納されているクライアント識別用情報）も利用するが，それぞれのアプリケーションによって情報の受け渡しをする手法も異なり，渡される属性もかなり異なる．従って，Web アプリケーションに対する侵入・不正アクセスの対策はアプリケーション自身にその機構を組み込む方法を取らざるを得ず，システム側での対策を行うことが非常に難しい．最近ではクロスサイトスクリプティング（XSS : Cross Site Scripting）と呼ばれる Web アプリケーションの脆弱性も報告されるようになってきており，XSS が発生すれば様々な形でシステムの持っている情報が危険にさらされることとなるため，影響範囲は計り知れない．このような脆弱性を自動検出し，データベースに登録する Proxy 型アプリケーションも存在し，XSS 脆弱性を自動的に検出してその情報を共有するクライアントに対して脆弱性を持った Web アプリケーションの利用を警告しようとする動きもある．

以上のように，Web アプリケーションではプロトコルの土台となっている HTTP だけでなく，アプリケーション自身の情報の流れにも注目する必要があるため，本研究において実装する IDS では対応できないことも考えられる．この問題は今後の課題としておきたい．

3.6 その他

その他のプロトコルについても，TCP コネクションの確立の後，同様のプロトコルの解析によって，ポートバイオレーションを検出することが可能である．使用するすべてのプロトコルに対して最初のいくつかの情報の送受信をモニターする必要はあるが，そのポートを正しく使用しているかどうかの判定を行うのに必要最低限の機能を実現することが可能であると考えられる．

4．ポートバイオレーションに着目した IDS の実装

IDS の実装は，bpf（Berkeley Packet Filter）インターフェイスを利用したパケットキャプチャ機能とキャプチャしたものから各プロトコルのコマンド部分を取り出す機能から構成される．bpf の持つプリミティブな機能を用いて IDS を実装することも可能であるが，本研究において実装した IDS は bpf のラッパー機能を持っている Libnet[10] および Libnids[11] アプリケーション・プログラミング・インターフェイスを用いてパケットストリームの再構成を行うことで，IDS 本体ではポートバイオレーションの検出に重点を置くことができるようになっている．

実装の方針としては，(1)指定したポートの監視を行う実装とする，(2)できるだけ軽量な IDS として実装する，などを採用している．

4.1 実装する IDS のスケルトンコード

実装する IDS のスケルトンコードは図 5 に示すようなものである．nids_init() 関数により Libnids の初期化を行い，nids_run() 関数によって IDS を駆動する．IDS のコードは tcp_callback() というコールバック関数に記述し，nids_register_tcp() 関数で登録して利用する．

```
#include "nids.h"
int main(int argc, char *argv)
{
    nids_init();
    nids_register_tcp(tcp_callback);
    nids_run();

    return 0;
}
```

図5　実装IDSのスケルトンコード

ポートに対応するプロトコルの正当性を検証するため，すべてのポートを1つのコールバック関数で処理するのではなく，ポートごとの関数によって処理する．このように，nids_register_tcp()によって登録する関数はリンクリストに保持されることになるので，後で登録されたものほど後で処理されることになる．

4.2　ポートバイオレーション検出機構

Libnidsで検出することができるTCPステータスは「established」，「data stream」，「close」，「reset」，「timed out」の5種類である．実装するIDSは「established」直後の「data stream」を検査することでポートバイオレーションを検出するので，IDSをできるだけ軽量に実装するためにもそれ以外のTCPステータスについてのコードは含めていない．

```
void tcp_callback(struct tcp_stream a_tcp, void **none)
{
    if (a_tcp->nids_state==NIDS_JUST_EST){
        a_tcp->client.collect++;
        ...
        return;
    }
    if (a_tcp->nids_state==NIDS_DATA){
        if (a_tcp->client.count_new){
            /* data stream from server to client */
            ...
        }
        return;
    }
    ...
}
```

図6　ポートバイオレーションの検出コード

NIDS_JUST_ESTで示されるTCPコネクションの確立の際に，Libnidsではtcp_stream構

造体にサーバおよびクライアントの IP アドレスとポートを格納する．サーバからのデータを収集したい場合は図6に示すように a_tcp->client.collect をインクリメントし，クライアントからのデータを収集したい場合は a_tcp->server.collect をインクリメントする．これにより，次の NIDS_DATA で示される TCP ステータス時に送受信されるデータをキャプチャすることができ，このときパケットに含まれているデータとプロトコルの成立に必要なデータとのパターンマッチングにより，ポートバイオレーションを検出することができる．

検出コードにはプロトコルごとにその検出するパターンを記述することになるが，その記述方法はパターンそのものを表す plain text 形式，正規表現形式，独自表記等を考えることができる．ここで実装する IDS は可読性を上げるために正規表現形式によって記述している．この IDS では，TCP コネクションが確立した直後のサーバもしくはクライアントから発生する1，2個のパケットに含まれる特定のメッセージを検査することを目的としており，複雑なデータ表現は必要がないことから plaint text 形式もしくは非常に簡単な正規表現での記述が適当と考えたためである．

ポートバイオレーションが発生していなければそれ以上のデータのキャプチャは必要ないので，a_tcp->client.collect および a_tcp->server.collect 変数をデクリメントしてそれ以降のキャプチャを停止し，アプリケーションの通信に影響を与えないようにする．ポートバイオレーションが発生した場合は，その tcp_stream 構造体の内容に基づいてその TCP コネクションを切断するというアクションを発生する．これによりポートバイオレーションが発生した瞬間に TCP コネクションを切断し，それ以降の通信の継続を妨げることができる．

4.3 IDS の設計方針とシステムの連携

本 IDS の設計方針として，できるだけ軽量なものにし，稼働時の負担を減少させることを目指すため，システム側で使用するポート番号を制限することが必要となる．

システム側で使用するポート番号以外へのアクセスを制限する対策を行う必要があるが，これは TCP セッションのうちの SYN パケットを拒否することで対応できる．オペレーティングシステムでポートに対するアクセス制御は，IP Filter，IPFIREWALL，iptables などによって設定することができるので，この設計方針は様々な OS 上で受け入れることが可能である．

5．検出実験および結果

検出実験は次のようなサーバとクライアントがスイッチングハブを介して接続されている環境で行った．サーバでは通常どおりのサービスを運用している状態としておき，実装した IDS を同じサーバ上で稼働させて必要なポートを監視し，クライアントからのアクセスの際にポートバイオレーションを検出する．ポートバイオレーションが発生したことを検出したかどうかを確かめるためにサーバのネットワークインターフェイスでのパケットログを採取し，パケットログによって実験結果を検証する．

```
192.168.1.2.32797 > 192.168.1.1.22: S 2328151036:2328151036(0)
                                win 49640 <mss 1460,nop,nop,sackOK> (DF)   ・・・(1)
192.168.1.1.22 > 192.168.1.2.32797: S 3589575220:3589575220(0)
                                ack 2328151037 win 65535 <mss 1460> (DF)   ・・・(2)
192.168.1.2.32797 > 192.168.1.1.22: . ack 1 win 49640 (DF)                 ・・・(3)
192.168.1.1.22 > 192.168.1.2.32797: P 1:41(40) ack 1 win 65535 (DF)        ・・・(4)
0x0000   4500 0050 f1df 4000 4006 aece c0a8 0101        E..P..@.@.......
0x0010   c0a8 0102 0016 801d d5f4 9235 8ac4 c3fd        ..........5....
0x0020   5018 ffff 37b6 0000 5353 482d 312e 3939        P...7...SSH-1.99
0x0030   2d4f 7065 6e53 5348 5f33 2e35 7031 2046        -OpenSSH_3.5p1.F
0x0040   7265 6542 5344 2d32 3030 3231 3032 390a        reeBSD-20021029.
192.168.1.2.32797 > 192.168.1.1.22: . ack 41 win 49640 (DF)
0x0000   4500 0028 e647 4000 4006 ba8e c0a8 0102        E..(.G@.@.......
0x0010   c0a8 0101 801d 0016 8ac4 c3fd d5f4 925d        ...............]
0x0020   5010 c1e8 1caa 0000 0000 0000 0000             P.............
192.168.1.2.32797 > 192.168.1.1.22: P 1:21(20) ack 41 win 49640 (DF)       ・・・(5)
0x0000   4500 003c e648 4000 4006 ba79 c0a8 0102        E..<.H@.@..y....
0x0010   c0a8 0101 801d 0016 8ac4 c3fd d5f4 925d        ...............]
0x0020   5018 c1e8 5ff2 0000 5353 482d 322e 302d        P..._...SSH-2.0-
0x0030   5375 6e5f 5353 485f 312e 300a                  Sun_SSH_1.0.
192.168.1.1.22 > 192.168.1.2.32797: P 41:577(536) ack 21 win 65535 (DF)
```

図7　正常なSSHコネクション確立（バージョン文字列交換段階）

図7に正常にSSHコネクションが確立されていく状況を示す．図7－(1)～(3)はTCPの3-way handshakeによるTCPコネクションの確立である．TCPコネクションが確立された後，図7－(4)のパケットに示すようにサーバからSSHのバージョン文字列が送信されていることが分かる．それに続く図7－(5)のパケットではクライアントのバージョン文字列が送信される．この場合ではバージョン文字列が必要であり，検出コードでは正規表現で記述される「SSH-?¥. *」のような形式で表現されるパターンとのマッチングをとる必要がある．正常なSSHの場合にはこのようなバージョン文字列が存在するが，SSH用のWell-Knownポートを別のサーバに割り当てていると，このようなバージョン文字列が出現しない．すなわちポートバイオレーションが発生していることになる．

図8にSSHポートにポートバイオレーションが発生した場合の通信記録を示す．図8－(1)から(3)はTCPコネクションの確立である．その直後の図8－(4)がサーバからクライアントへのデータ送信となっているが，この記録からも図7に示したような文字列は見当たらない．そこで，このポートではSSHがサービスされていないものとして図8－(6)でIDSからクライアントへ向けたRSTフラグの付いたパケットが送信される．このときにクライアント側では図9に示すように，一旦は実際のサービスであるTELNETによるlogin:プロンプトが表示されているが，IDS

からRSTパケットが送信され，TCPコネクションが切断されていることが分かる．

```
192.168.1.2.32804 > 192.168.1.1.22: S 540245806:540245806(0)
                                win 49640 <mss 1460,nop,nop,sackOK> (DF)   ・・・(1)
192.168.1.1.22 > 192.168.1.2.32804: S 634391624:634391624(0)
                                ack 540245807 win 65535 <mss 1460> (DF)    ・・・(2)
192.168.1.2.32804 > 192.168.1.1.22: . ack 1 win 49640 (DF)                 ・・・(3)
192.168.1.1.22 > 192.168.1.2.32804: P 1:4(3) ack 1 win 65535 (DF) [tos 0x10]・・(4)
0x0000   4510 002b 0933 4000 4006 9790 c0a8 0101        E..+.3@.@.......
0x0010   c0a8 0102 0016 8024 25d0 0c49 2033 7f2f        .......$%..I.3./
0x0020   5018 ffff 9f1b 0000 fffd 25                    P.........%
192.168.1.2.32804 > 192.168.1.1.22: . ack 4 win 49640 (DF)                 ・・・(5)
0x0000   4500 0028 2802 4000 4006 78d4 c0a8 0102        E..((.@.@.x.....
0x0010   c0a8 0101 8024 0016 2033 7f2f 25d0 0c4c        .....$...3./%..L
0x0020   5010 c1e8 0239 0000 0000 0000 0000             P....9........
192.168.1.1.22 > 192.168.1.2.32804: R 634416445:634416445(0) win 32000     ・・・(6)
0x0000   4500 0028 3039 0000 4006 b09d c0a8 0101        E..(09..@.......
0x0010   c0a8 0102 0016 8024 25d0 6d3d 0000 0000        ........$%.m=....
0x0020   5004 7d00 859e 0000                            P.}.....
```

図８　SSHポートにポートバイオレーションが発生した場合

```
% telnet 192.168.1.1 22
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

FreeBSD/i386 (192.168.1.1) (ttyp0)

login: Connection to 192.168.1.1 closed by foreign host.
%
```

図９　クライアント側での通信記録

６．おわりに

本論文では，ポート番号によるアクセス制御を行うことの利点を示し，その方針に基づいた運用を行っている場合の問題点としてポートバイオレーションについて説明した．さらに，この問題に対する解決策として，ポートの使用に関するIDSの設計，実装を行い，その有効性を検証した．

検証作業に関する課題としては，実装したIDSの検証作業には同一セグメントに配置されたサーバとクライアントを用いていることであり，この場合サーバ対クライアントという１対１の関係の中での検証であり，ネットワーク全体を含めた検証となっていないことが課題となっている．

運用上の問題としては「通信の傍受」との関係がある．TCP/IPにおける「通信の傍受」とは，IP層におけるman-in-the-middle攻撃などのような，中間の第3者が通信内容を完全に（故意または偶然に）受信することであり，本論文で問題としているポートバイオレーションの検出では，TCPコネクションが確立した直後のアプリケーションプロトコルの最初の部分の検出を追跡することに限定し，通信している内容についての追跡はしていないため，一般に言われている「通信の傍受」とは異なる種類のものになる．

技術的な問題としては，物理的なネットワークインターフェイスのみに対応となっており，仮想インターフェイス（IPv4-IPv4トンネルなど）やVLANインターフェイスには対応していないため，IPv6トンネルでの検出ができないことである．また，実装したIDSはIPv4用のAPIであるLibnetおよびLibnidsを利用しているため，IPv6に対するモニターを行うことができていない．これらの技術的問題については，仮想インターフェイスを含めたデータリンク層におけるパケット処理用のコードおよびIPv6用のコードに書き直すことが必要である．

またIDS単独の問題でなく複合的なものとして考えていく必要のある問題としては，Webアプリケーションの箇所でも述べたように，「～ over HTTP」のようなプロトコル階層を持っている場合には，本研究において実装したIDSでは十分な対応ができないことである．この問題はIDS単独で解決するよりも，IDSとアプリケーションの連携によって解決していかなければならない問題であると認識しており，今後の課題としておく．

Well-Knownポートでは現在1000近くのサービスが提供されているがそれらの全てが稼働することは考えにくく，またインターネットを通じて安全に利用できるプロトコルは非常に限定されており，実際にはこれよりもかなり少ない数のポートを監視対象とすることになる．そのため，プロトコルが増加したとしても，すべてを監視しなければならない可能性は非常に低い．実際にインターネットから利用すると考えることのできる代表的なものを数え上げてみると，22（SSH），25（SMTP），79（finger），80（HTTP），110（POP3），143（IMAP4），443（HTTP over TLS/SSL），993（IMAP4 over TLS/SSL）などである．かなり限定的なものであり，パスワード，通信路の暗号化などセキュリティのことも考慮に入れると，さらに減少するものと予想できる．セキュリティポリシーの観点からも，すべてのポートがオープンであるような運用を組織が認めるということはほとんどあり得ないため，全面的なオープンという状況は今後改めるべきであると考えている．

謝辞

本研究の一部は，平成13年度関西大学研修員研究費によって行われたものである．

参考文献

［1］W. Richard Stevens：「TCP/IP Illustrated, Volume 1: The Protocols」，Addison Wesley（1994）．
［2］Assigned numbers: RFC 1700 is Replaced by an On-line Database，http://www.ietf.org/rfc/rfc3232.

txt?number=3232 (2002).
[3] Norman Abramson 著，宮川訳：「情報理論入門」，好学社 (1969).
[4] 溝口，石田共編：「人工知能」，オーム社 (2000).
[5] BSD rlogin, http://www.ietf.org/rfc/rfc1282.txt?number=1282 (1991).
[6] Simple Mail Transfer Protocol, http://www.ietf.org/rfc/rfc2821.txt?number=2821 (2001).
[7] Post Office Protocol - Version 3, http://www.ietf.org/rfc/rfc1939.txt?number=1939 (1996).
[8] POP3 AUTHentication command, http://www.ietf.org/rfc/rfc1734.txt?number=1734 (1994).
[9] D. J. Barrett and R. E. Silverman :「SSH, The Secure Shell: The Definitive Guide」, O'Reilly & Associates Inc. (2002).
[10] Libnids, http://www.packetfactory.net/projects/libnids/ (2002).
[11] Libnet Packet Assembly, http://www.packetfactory.net/projects/libnet/ (2001).