段階的通信制限を実現するゲートキーパーの提案と試作

2003MT003 青山 正樹　　2003MT040 小島 正和
指導教員　後藤 邦夫

1　はじめに

インターネットが普及している近年，その安全性が大きな問題となっている．特に DoS 攻撃（Denial of Service attack）や不正アクセスなどについては，掲示板や検索サイトなどがターゲットとして狙われ，その被害について報道される場合が多い．最近では 1 秒間に 5 億回ものアクセスが有名掲示板に仕掛けられた事例が公表されており，システム管理者にとっては深刻な問題となっている [5][6].

このような攻撃に対応するために侵入検知システム（以下，IDS；Intrusion Detection System とする）[3] や攻撃検知防御システム（IDP；Intrusion Detection and Prevention），さらに侵入防止システム（IPS；Intrusion Prevention System）[4] などが開発され，対策手段として有名になった．しかし侵入を検知することが主な目的であるため，攻撃を防御するという点から考えると完全ではないのが現状である [7].

そこで本研究ではパーソナルコンピュータ（以下，PC とする）をブリッジとして用い，リアルタイムに通信をフィルタリングする．そして攻撃の量と時間に応じてパケットの到着時刻を遅らしたり，スループット制限を設定したり，送信パケット数を減らしたり，最終的には受信したパケットをすべて落とすという段階的に通信を制限する安価で拡張性の高い防御を重視したゲートキーパー（以下，GK とする）を提案し，試作する．なお待ち行列における処理には GINE 論文の手法 [1] を用いる．また，本研究では通信制限処理に重点をおいているため，パケットの監視は既存の IDS に任せて IDS 出力を利用した通信制限を行う．

このような段階的通信制限を設けることにより，すべての通信を完全に遮断させることなくある程度異常な通信のみを監視した上，攻撃ごとに的確なフィルタリングを適用させることを可能としている．

GK 本体の作成は共同で行い，青山は主に通信制限の実装，小島は主に IDS と GK とを連携させる処理の実装を担当した．

2　システムの概要と期待できる効果

この節では，本研究で考える GK と IDS の配置と処理の流れ，GK で行う通信制限と期待出来る効果について述べる．

GK は外部ネットワークと内部ネットワーク間で IP アドレスをつけずにブリッジとして動作させ，フレーム通過時に通信を制限する．IP アドレスをつけないので設置が容易であり，また故障時には GK に接続した

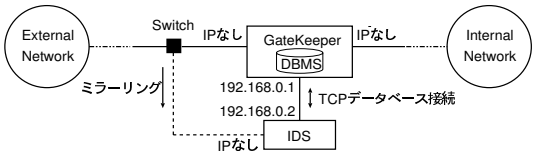

図 1　ネットワーク構成

LAN ケーブルを直結してバイパス出来る．同時に攻撃対象にならないという利点も持つ (図 1 参照)．IDS はスイッチでミラーリングされたネットワーク上を流れるパケットを監視し，検知した攻撃の種類，攻撃元の IP アドレスなどの情報を直結した GK に渡す．GK はこの情報に基づいてリアルタイムに通信制限を行う．通信制限は外部ネットワーク，内部ネットワーク間の両方向でできる．

GK で行う通信制限は以下の 5 段階に分ける．

- THROUGH（素通し）：通常の通信処理と同じ役割を果たす．正常なパケットのみをこの状態で通す．
- DELAY（遅延）：任意の秒数の遅延を起こす．
- THROTTLE（スループット制限）：帯域幅を絞り，通信速度を制限する．
- LOSS（パケット損失）：任意の確率でパケットを破棄し，パケット損失を起こす．
- DROP（パケット破棄）：全てのパケットを破棄する．

GK では，抑止効果が見込まれるものに対し DELAY，THROTTLE，LOSS の状態を与え様子を見る．抑止効果が見込まれないものに対しては遮断である DROP の状態を与える．また，DELAY，THROTTLE，LOSS の状態を与えられた通信に対しても，断続的に行われるようであれば DROP の状態に移行させる．

DELAY では小さな遅延を発生させれば TCP の通信を遅くする効果が，大きな遅延を発生させれば応用層の応答を遅くする効果が期待される．THROTTLE では全ての通信に対し制限効果が見込まれ，特に ICMP や UDP などのフローコントロールがない通信に対しての制限効果が期待される．LOSS は再送要求を増やすことで，DELAY と同じく TCP の通信を遅くする効果が見込まれる．

また，GK ではパラメータの異なる複数の制限を設定できる．

3 GK の実現

この節では，GK 内で作成した送受信処理の仕組みや，通信制限のレベル，フィルタリング方法について説明する．

3.1 構成と処理の流れ

本研究では，全ての PC の OS に VineLinux を使用する．また IDS にはオープンソースである Snort を使用する．

GK は大きく分けてパケットの受信処理，パケットの送信処理，Real Time Clock タイマ（以下，RTCTimer とする），IDS からの警告受信処理，そしてルールの作成・更新・削除の 5 つの処理から成り立っている．GK が RTC タイマに従いパケットを送受信している間に IDS より警告を受け取ったら，その警告情報をもとにしてパケットフィルタのルールを作成し，直ちに適用させる．なおこの 5 つの処理はプログラム内でそれぞれスレッドとして動作する仕組みになっており，その実現にはオブジェクト指向の C++ を用いた．また，逆方向も同様の構成となっている．(図 2 参照).

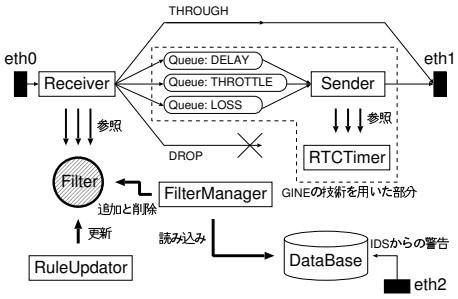

図 2 GK 全体の処理の流れ（片方向のみ）

3.2 Queue

本研究では，受信された各フレームは先着順に Queue に格納され，それぞれに出発予定時刻が与えられる．この出発予定時刻を操作することで，遅延やスループット制限を起こすことができる．

- DELAY：フレームの受信時刻に発生させる遅延の秒数を足した値を出発予定時刻とする．遅延の秒数をランダムにした場合，フレーム順序の入れ替わりが起きる．
- THROTTLE：前のフレームが出発した時刻に，設定スループット時における自身の通過時間を足したものを出発時刻とする．これにより，任意のスループットを実現できる．
- LOSS：受信したフレームを Queue に入れる直前に任意の確率で破棄することで実現できる．破棄されなかったフレームは，出発予定時刻を受信時刻の値として Queue に入れる．

本研究では，これらの障害を単独で起こす Queue を 3 つ用意したが，GINE 論文と同様に Queue は複数作ることができ，また複数の障害を組み合わせることが可能である．したがって様々な障害を起こす Queue をいくつも用意することができる．

3.3 RTCTimer

Sender は，各 Queue の先頭フレームの出発予定時刻を定期的にチェックし，現在時刻と比較して該当フレームを Queue から取り出し送信する．この処理には RTCTimer を利用した [2].

RTCTimer はコンピュータ内部のマザーボード上に実装されている計時専用のチップであり，Linux では/dev/rtc として利用できる．なお，定期的にチェックする処理は RTC の割り込み機能を利用している．割り込みは 1 秒間に最大 8192 回できるので，本研究では は $\frac{1}{8192}$ 秒間隔で他のスレッドとの同期に用いている．なお，RTCTimer は 1 つのプロセスからしか利用できない．

3.4 ソケットの作成とパケットキャプチャライブラリとの比較

通常ネットワークを介してデータを送受信するためにはソケットの作成が必要となる．またここでは GK がブリッジとして動作しているため，生のフレーム入出力が必要となる．そこでソケットを作成する際にオプションとして PF_PACKET の SOCK_RAW を利用した．これにより，データリンク層フレーム中の IP パケットヘッダでプロトコル番号，SrcIP アドレス，DstIP アドレスを用いて通信制限を可能にした．また，本研究では TCP や UDP など上位のプロトコルに含まれる情報も加味して通信制限処理を実装している．

また，この他にも VineLinux にあらかじめインストールされている libpcap パケットキャプチャライブラリには PF_PACKET とフィルタによるパケット送受信処理が用意されている．特に最近の libpcap では，受信したパケットを pcap_sendpacket 関数を用いてインタフェースを指定して送信することが可能で，本研究におけるパケットの送受信とほぼ同等の処理を実現できる．

本研究においても当初 libpcap を用いて作成する予定であったが，ネットワークインタフェース 1 つに対してライブラリ関数を用いて複数キャプチャオープンしておき，そこに pcap_sendpacket 関数を用いた送信処理を記述すると，送信すべきパケットを同一インタフェースが再びキャプチャしてしまい，結果としてパケットがループしてしまう問題が発生した．

3.5 IDS からの警告受信とアラート管理

GK 側で特定のホストからのみ通信制限を行うためには，IDS で感知した情報を GK 側に伝える必要がある．ここでは GK 側にデータベースを構築し，IDS 側ではそのデータベースに向けて感知した情報を送信する環境を構築した．GK 側では格納された情報をもとにフィルタリングルールを作成し，直ちにパケット受信時のルールとして適用させた．データベース内のアラートは一定時

間ごとに自動的に消すようにして，検索に負荷がかからないようにした．なお，データベースは PostgreSQL を使用した．

3.6 フィルタリングルールの作成と適用・更新・削除

正常なパケットなのか異常なパケットなのかを判別するには何らかの情報が必要となる．そこで我々は通信制限の処理に先立ち，パケットフィルタリングのルールを設定した．

フィルタリングに用いる情報は以下の項目である．

- パケットの発信元・宛先 IP アドレス
- パケットの発信元・宛先ネットマスク
- 発信元・宛先ポート番号
- 発信先・宛先マスク
- プロトコル番号・マスク
- TCP フラグ (SYN/FIN/RESET/PUSH/ACK/URGENT)・ビットマスク
- アクション(THROUGH/DELAY/THROTTLE/LOSS/DROP)

上記項目のうち，TCP フラグはフィルタリングルールが TCP の場合のみ使用する．また，マスクについては攻撃元 IP アドレスなどが類似している場合に使用してルール数を減らすためのものである．

パケットを受信するとパケットが解析される．そして解析した情報と IDS からの警告情報が格納されているデータベースに含まれる属性（パケット情報）とを比較する．上記の情報全てに該当したパケットに対して制限を行い，監視を続ける（図 3 参照）．

なお IDS からの警告が増えるにつれてデータベースに含まれる情報も多くなり，検索に負担がかかるため，毎回全てのデータベース情報を検索するのではなく，前回検索時の時刻以降に新たに追加された情報のみを検索する．

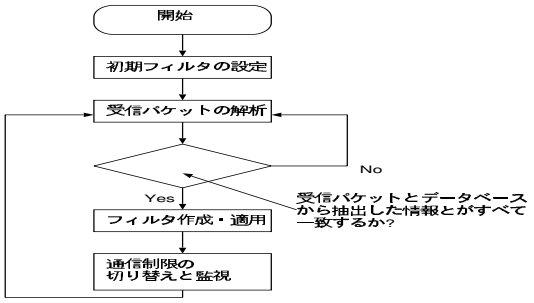

図 3 フィルタリング処理の流れ

また同一ホストからの攻撃が続く場合，制限を移行させるため該当するルールのアクションを更新する．なお一定時間使用されないルールは削除する．

4 実験

各制限の性能評価とフィルタの性能評価を行った．GK の両端に host_a，host_b を用意してそれぞれ外部ネットワークにある攻撃ホスト，内部ネットワークにある保護ホストと仮定した．また，IDS の監視対象は host_b とし，host_b 宛の攻撃フレームを検知するように設定した．

GK にはデュアルコア CPU 搭載のマシン，IDS には負荷に十分耐えることができるマシン，そして端末 A と端末 B には同じ動作環境のデスクトップ PC2 台を利用した．

4.1 RTT・フレーム損失率・スループット測定

host_a から host_b に向かって ICMP パケットを含むフレームを 0.2 秒間隔で 1000 個送り，RTT の平均値とその間のフレーム損失率を測定した．これを THROTTLE 以外の各制限でそれぞれ 5 回ずつ行い平均値を求めた (表 1 参照)．なお復路の host_b から host_a 向きの制限は全て THROUGH とした．また，THROUGH がどの程度の遅延を起こしているか測定するため，host_a と host_b を直接接続した場合（DIRECT）も行った．

表 1 各制限における設定値と測定値

	設定値		測定値	
経路	delay(ms)	loss(%)	RTT(ms)	loss(%)
DIR	—	—	0.20	0
THR	—	—	0.27	0
DEL	1000	—	1000.29	0
LOS	—	50	0.28	49
DRO	—	100	—	100

この結果より，それぞれの制限がほぼ設定通り正確に動作していることがわかる．THROUGH の RTT は DIRECT と比較すると差は 0.1ms 以下で, 実際の運用に影響を及ぼす値ではないと思われる．DELAY の RTT と LOSS と損失率を見ると，設定値がそれぞれ反映されていることがわかる．通信の遮断である DROP も正常に動作している．またフレーム損失が起きる LOSS と DROP 以外では損失は起きておらず，ブリッジとして問題なく動作していると言える．

また，オープンソースのスループット測定ツールである iperf を用いて host_a をクライアント，host_b をサーバとして各制限について TCP・UDP 各スループットを測定した．THROUGH については TCP・UDP ともに DIRECT に近い結果が得らた．DELAY については TCP では遅延を大きくするにつれてスループットが急激に低下し，UDP では遅延を大きくしても THROUGH に近い値が得られた．THROTTLE でも TCP・UDP ともに正確な帯域幅制限ができた．LOSS については DELAY と同様の結果となった．

4.2 フィルタの性能評価

IDS からの情報をリアルタイムにフィルタに追加しフレームを振り分けることが出来るか，また制限を段階的に移行させることが出来るか ping を用いて実験した．

IDS からの情報を読み込み，まず THROUGH のルールを与え，DELAY(1000ms)，THROTTLE(392bps)，DROP と移行させる設定とし，host_a から host_b に向かって ICMP パケットを含むフレームを 1 秒間隔で 50 個送信して，時間に沿った RTT の変化を測定した（図 4 参照）．THROTTLE の設定帯域幅が 392bps では，ICMP パケットを含むフレーム (98*8=784bit) が通過するのに 2 秒かかり，RTT は 1000ms ずつ増える．フィルタ管理の FilterManager と RuleUpdator は ping 開始 10 秒後にスタートさせた．またルールの参照間隔は 1 秒とし，10 回参照ごとに制限を移行させた．

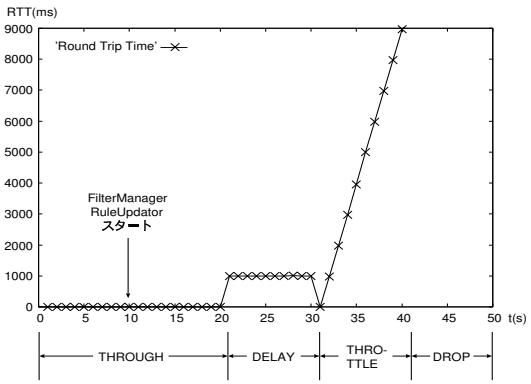

図 4　制限切替えが RTT におよぼす効果

図 4 からおよそ 10 秒間隔で制限が移行していることがわかり，また，host_b が受信したフレーム数が 41 個で損失率が 18% であり，DROP の状態での損失分のみとなっている．

また図では表していないが，制限が切り替わる時の遅れはなく次の制限が有効になったことが確認できた．

これらのことからフィルタのリアルタイム制限機能と段階的移行機能が正しく動作していると言える．

以上の結果を踏まえて，攻撃に対して抑止効果が得られるかについても実験した．実際に行った攻撃は，PING flood，nmap ポートスキャン，同報メールソフトによるスパムメール大量送信の 3 種類であり，すべての攻撃を検知とほぼ同時に制限することができた．

PING flood とポートスキャンについては抑止効果が確認できたが，スパムメール送信は確認することができなかった．

5　おわりに

本研究を通して，ヘッダ解析によるパケットの振り分け方法を理解することができた．また，ソケットの利用と独自のフィルタ作成による 5 種類の通信制限経路を確立し，IDS からの連絡に従って攻撃別かつ動的にフィルタリングルールを追加してリアルタイムに通信を制限することに成功した．さらに，RTT 計測値と簡単な攻撃実験により，各通信制限経路において設定した通信制限が正確に反映されていることが確認できた．今後の課題は以下のとおりである．

- 運用レベルとしての GK の性能評価：作成したフィルタリングルールが正しく反映されているかどうかは本研究において確認できた．しかし運用レベルとしての性能評価は行っていないため，実際に様々なパケットが流れる実ネットワーク下に設置してあらゆる攻撃を想定した性能評価を行う必要がある．
- ルールの統合と検索効率：現状では 1 回の攻撃で 1 つのルールが生成され，同一ホストから攻撃が続いた場合，同一ルールが多数生成されてしまう．この場合一番最初にできたルールのみが使用され，残りが無駄となる．そのため，類似したルールは 1 つのルールとして統合する必要がある．
  また頻繁に使用されるルールは検索に時間がかからないようにする必要もある．

上記の研究課題はいずれも重要課題である．またハニーポットへの切り替えなどさらに攻撃を追跡できるようにすることも行う必要がある．

参考文献

[1] Ihara, A., Murase, S. and Goto, K.: IPv4/v6 Network Emulator using Divert Socket, *Proc. of 18th International Conference on Systems Engineering (ICSE2006)*, Coventry, UK, pp. 159–166 (Sep. 2006).

[2] Paul, G. and Kawasaki, T.: Linux 用リアルタイムクロックドライバ，JF（Japanese FAQ） Project (2002). (`http://www.linux.or.jp/JF/JFdocs/kernel-docs-2.6/rtc.txt`).

[3] 牛窪裕一：Tcpdump 簡易表現形式によるネットワーク侵入検知，修士論文，群馬大学大学院 (2006). (`http://www.ail.cs.gunma-u.ac.jp/asaka-lab/data2005/[Ushikubo]master.pdf`).

[4] 桝本　圭，原田季栄：OSS による IPS の実現，株式会社 NTT データ (2005). (`http://lc.linux.or.jp/paper/lc2005/CP-11.pdf`).

[5] 警察庁技術対策課：DoS/DDoS 対策について（検証），警察庁 (2004). (`http://www.cyberpolice.go.jp/server/rd_env/pdf/DDoS_Inspection_2.pdf`).

[6] 国家公安委員会：不正アクセス行為の発生状況，総務省 (2006). (`http://www.soumu.go.jp/s-news/2006/pdf/060223_1_2.pdf`).

[7] 武田圭史：侵入検知システムに関する研究の現状，情報処理， Vol. 42, No. 12, pp. 1169–1174 (2001).