通信制限システムにおけるTCPセッションの途中切替と安全なリモートアクセス機能の実装

M2008MM007 福井麻美
指導教員：河野浩之

1 はじめに

近年，インターネットの安全性が重要視されている [4]．トラフィックを増大させ，相手先のコンピュータに多大な負荷をかける DoS 攻撃などの不正アクセスが多発し，業務やサービスに深刻な被害をもたらしているが，現状ではその正確な検知と防御が困難である．

この対策手段として，南山大学 後藤研究室では,「通信制限システム」[1][6](以下，既存システムまたは GK とする) を開発している．このシステムは，正常な通信か攻撃かの判断が難しい通信に対して，遅延やパケット損失を発生させ，攻撃と断定したら通信を遮断する．しかし，TCP 通信の場合に応用層の通信を途中でハニーポットに切り替える機能やフィルタリングルール設定要求時のリモートアクセスにおける認証機能が未完成である．

本研究では，既存システムにおいて実装が不十分な点を改良する．ネットワークエミュレータ GINE[7] のライブラリクラスを用いてシステムを再設計し，より安定的にシステムが稼働することを目標とする．主な改良点は，攻撃の手法を監視，分析するためのハニーポットの利用を想定した TCP 通信の途中切替機能，リモートホストから安全にルールを設定するためのホスト認証機能，途中でシステムが停止した時に設定されているルールを保存するためのルールオブジェクトの保存機能の実装である．

これらの機能を実装したシステムに対して，動作確認と処理性能の評価，操作方法の利便性を評価する．

2 システムの概要

既存システムと本研究で提案するシステム (以下，新システムまたは NewGK とする) を比較し，既存システムの問題点と改善方法について説明する．

2.1 既存システムの概要

既存システムの概要を説明する．システムのネットワーク構成は，図 1 のようになる．

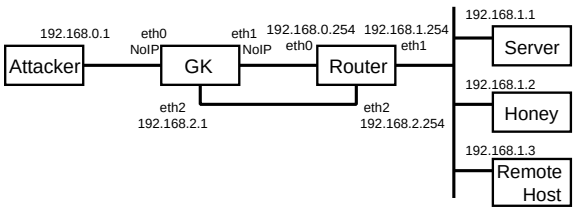

図 1 システムのネットワーク構成

システムをブリッジとして動作させるため，GK の eth0 と eth1 には IP アドレスを付与しない．そのため，内部ネットワークからルール設定を要求できない．内部ネットワークと GK が通信出来るようにするため，GK の eth2 と Router の eth2 のそれぞれに IP アドレスを付与した．通信制限は，内部ネットワーク側のホストが設定したフィルタリングルールに基づいて施される．

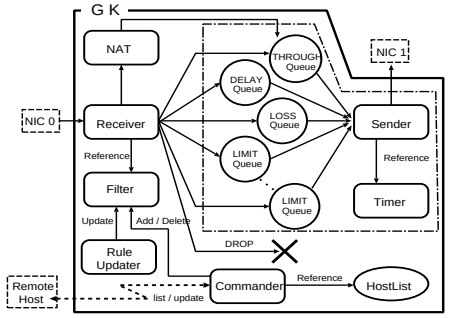

図 2 既存システム GK の構成

図 2 は，片方向の中継処理のみの構成である．点線で囲われた部分の待ち行列の処理には，GINE 論文の Queue 処理の手法 [7] を用いた．

GK は主に，Receiver，Sender，Commander，Filter，NAT のクラスによって構成される．

2.2 既存システムと新システムの比較

システムを再設計するにあたって，旧システム (GK) と新システム (NewGK) の違いを表 1 にまとめる．改良点は，GINE ライブラリクラスの使用，通信途中切替，リモートアクセス機能，Queue 処理，IPv6 への対応の 5 つである．

表 1 既存システムと新システムの比較

変更箇所	GK	NewGK	変更理由/目的
GINE ライブラリ	GINE の Queue 処理手法のみ使用	GINE クラスを利用	システムの安定性を高めるため
TCP 通信の途中切替機能	無し	実装	未実装のため
パスワード認証	平文パスワード	SSL クライアント	安定性向上のため
ルール保存機能	無し	実装	システム停止時にリセットされるため
Queue	ルールごと	往復各 3 種	メモリ節約 処理効率向上のため
IPv6	未対応	IPv6 処理を導入	IPv6 ネットワーク対応のため

表 1 からわかるように，既存システムでは Queue 処理に GINE の手法を利用していただけで，プログラムのほとんどを GK 用に作成したクラスによって構成していた．そこで，新システムでは，システムの安定性を図るために GINE のライブラリを使用するように変更する．また，未実装であった通信の途中切替機能を実装した．不完全であったリモートアクセス機能は，安全性を高めるために SSLv3(TLS) を用いたクライアント認証を採用する．

図 3 は，NewGK の構成を表した図である．既存システムではルール毎に Queue を作成していたが，効率化を図るために，THROUGH，DELAY，LIMIT の 3 種類の Queue を Inbound(NIC0 から NIC1 への通信) と Outbound(NIC1 から NIC0 への通信) の各方向に用意した．

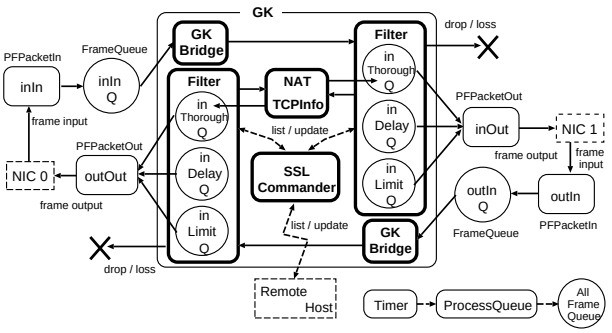

図 3 新システム NewGK の構成

GKBridge，Filter，NAT，TCPInfo，SSLCommander クラスは，GK 用のクラスである．これらのクラスのほかに，Filter の内部クラスであり，フィルタリングルールである Rule クラスも GK 用のクラスである．

3 システムの実現

本節では，提案するシステムの実現方法について述べる．

3.1 TCP 通信の途中切り替え機能

図 4 は実際は攻撃ホストと保護ホスト間で通信しているが，ハニーポットにもフレームのコピーを転送する場合の TCP 通信の流れを示したものである．

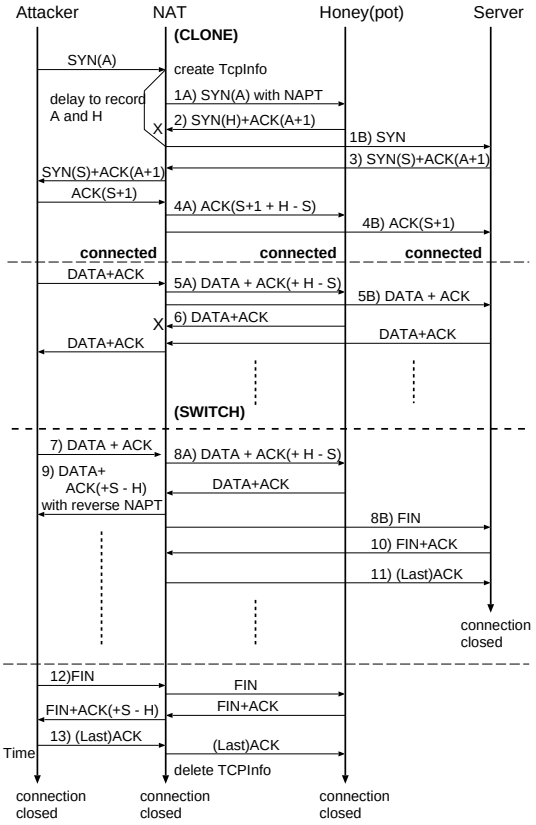

図 4 TCP 通信の途中切替の流れ

Attacker から Server へ送信されるフレームをコピーして Honey に送信する **CLONE** の状態からスタートする．

NAT 変換時のシーケンス番号 (seq)，確認応答番号 (ACK seq) の補正には，Server と Honey の seq の差を利用する．そのため，Server と Honey の seq の差を取得するために，初回の通信時は Honey に SYN を転送した後，10msec 程度の遅延を加えて Server に SYN を転送する．

その後の通信でも，常に Server より先に Honey にフレームを転送する．これは，Honey が ACK seq を補正に初回時の Server と Honey の seq の差を利用しており，Honey からの返信前に Attacker からの応答があると Honey への通信が確立できなくなるためである．

CLONE

1A) Attacker から届いた SYN パケットの DstIP，DstPort を Honey のものに変更し，TCP チェックサム，IP ヘッダチェックサム補正後，Honey に送信する．
(この接続の状態を記録する TcpInfo を生成する．)

1B) 4A) 以前に Server と Honey の seq の差を取得できるように，10msec 程度の遅延を加え Server に SYN を転送する．

2) Honey からの SYN+ACK 応答の seq H を保存し，フレームを廃棄する．

3) Server からの SYN+ACK 応答の seq S を保存し，Attacker に転送する．

4A) Attacker からの TCP 接続完了の ACK seq を $S+1+H-S$ に補正し，Honey に転送する．

—— **Honey の TCP 接続完了．** ——

4B) Attacker からの TCP 接続完了 ACK を Server に転送する．

—— **Server の TCP 接続完了．** ——

5A) Attacker からのフレームの NAT 変換，ACK seq 補正，TCP，IP ヘッダチェックサム補正後，Honey に転送する．

5B) Attacker からのフレームは，Server にそのまま転送する．

6) CLONE での Honey からの応答は廃棄する．

Attacker から Server へ送信されるフレームを NAT 変換処理し，Honey へ転送する **SWITCH** に切り替える．初期状態が SWITCH の場合は，2) のフレームは廃棄せず，逆方向に NAT 変換処理して，Attacker に転送する．

SWITCH

7) Attacker は，Server にフレームを送信する．

8A) 5A) と同様に Attacker からのフレームを NAT 変換，ACK seq 補正，TCP，IP ヘッダチェックサム補正後，Honey に転送する．

8B) Server の TCP 接続終了のために、TCP FIN flag を 1(ON) に変更して転送する

9) Honey からの応答は，逆方向の NAT 変換，ACK seq 補正，チェックサム補正後，Attacker に転送する．

10) Server は，8B) に対し FIN+ACK を応答する．

11) NAT 内で 10) の MAC，IP アドレス，ポートの Src/Dst を逆にして LastACK を生成し，Server に送る．

—— **Server の TCP 接続終了．** ——

12) Attacker からの FIN を Honey に転送する．

13) Attacker からの LastACK を Honey に転送後，この接続の状態を記録する TcpInfo を削除して，処理を完了する．

—— **Honey の TCP 接続終了．** ——

3.2 SSL(TLS) を用いたホスト認証機能

既存システムにも，平文パスワード認証機能は実装されている．しかし，このパスワード認証機能はホスト毎にパスワードを設定することが出来ず，パスワードを知っていれば，誰でもルールの設定をすることが出来てしまう．そこで，SSL クライアント認証を利用してホスト認証機能を実現する．

既存システムの Commander に SSL クライアント認証機能を追加し，新たに SSL Commander クラスを作成した．SSL クライアント認証の実装には，C++プログラミング言語で使用でき，SSL のすべての機能を実装しているオープンソースである openssl[8] を使用した．openssl でクライアント/サーバ方式の通信を実装するにあたり，SSLCommander をサーバとし，新たにクライアントプログラムを作成し，SSL クライアント認証を実装した．

SSLCommander は，SSL クライアント認証を用いた通信と外部ホストからの要求でフィルタ操作をするスレッドのクラスである．SSLCommander に，SSL クライアント認証でサーバとして通信をするスレッド実行内容の run メソッドと要求されたフィルタ操作をする accessFilter メソッドを準備した．作成した SSLCommander オブジェクトはスレッドとして動作する．

3.3 設定済みルールの保存機能

本研究では，ルールの追加や削除，変更時に設定したルールを自動的に保存する機能を追加する．

Rule オブジェクトの保存は GINE のオブジェクト保存機能 [2] を参考に，GNU common C++の Persistence を用いて実装する [3]．本研究では，Persistence ライブラリクラスの直列化を可能にする Engine クラスと Persistence オブジェクトを生成するためにタイプを定義する BaseObject クラスを使用する．

Filter が Rule リストを持つため，Filter の保存時に Rule リストを保存すれば Rule がすべて保存できる．

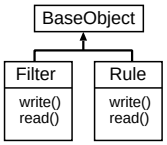

図 5 Rule オブジェクト保存の継承関係

そのため，図 5 のように，Filter，Rule クラスにオブジェクトの保存を指示する write メソッド，読み出しを指示する read メソッドを実装した．

4 性能評価

旧システムと本研究で試作した新システムの実験結果を比較し，その性能を評価する．また，改良した機能について，その動作を確認する．

4.1 実験ネットワークの構成

図 6 で示すように，実験環境に実機を用いると，最低 5 台の PC が必要となる．そこで，実験環境は，後藤研究室で開発中のネットワークエミュレータ GINE[7] を用いて 1 台だけで構築した．

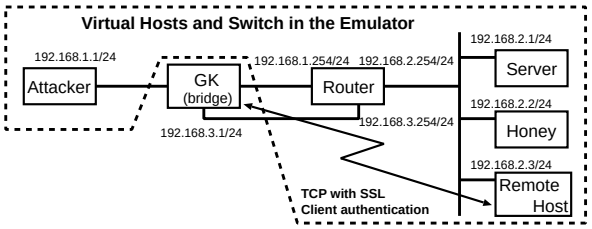

図 6 実験ネットワークの構成

仮想ネットワーク上のホストには，Network Namespace(NETNS) を使用した．GK(NewGK) は OS で直接実行する．処理速度以外は現実と同じ構成である．GK(NewGK) の Timer は NANOSLEEP を使用した．

4.2 NewGK の性能評価

4.1 節の図 6 に示した実験環境の Attacker から Server に対して，ICMP 通信の ping と TCP，UDP 通信の iperf を用いて，往復遅延時間 (RTT)，スループットを測定する．

○ NewGK がブリッジとして高速に動作するか

Attacker と Router の間を Kernel Bridge で接続した状態と NewGK，GK を配置した状態の RTT を比較して，NewGK がブリッジとして高速に動作するかを判断する．

表 2 ブリッジと NewGK の RTT 測定結果

	Kernel Bridge	NewGK	GK
RTT(msec)	0.006	0.037	0.501

表 2 の RTT の測定値を見ると，Kernel Bridge よりも NewGK の方が遅延時間が小さくなっているが，その誤差は 0.031msec と非常に小さなものである．また，GK の RTT と比較すると NewGK の方が 0.464msec 小さくなっていることから，GK よりも NewGK の方が高速に動作していることがわかる．

○ NewGK が通信自体を妨害しないか

システムが通信を妨害しないかどうか，TCP スループットを測定して評価する．測定結果は表 3 の通りである．TCP スループット測定のウインドウサイズは，iperf のデフォルト値を使用した．デフォルト値は，クライアント側は 16.0KBytes，サーバ側は 85.3KBytes である．それぞれ 10 回ずつ測定し，その最大値と最小値，平均値をまとめた．

表 3 TCP スループット測定

	Kernel Bridge			NewGK			GK		
	min	max	ave	min	max	ave	min	max	ave
帯域幅 (Mbps)	942	1126	973	637	697	677	701	1000	873

表 3 の TCP スループットの測定値を見ると，NewGK は，Kernel Bridge，GK と比較するとスループットが小さいことが分かるが，最大値と最小値の差は，NewGK が一番小さく，安定している．また，NewGK は平均 670Mbps 程度のスループットを保つことができており，GINE クラスの再利用でシステムの安定性が向上したといえる．

○ 各通信制限が設定値通りの挙動を示すか

システムに DELAY，LOSS，LIMIT のそれぞれのルールを設定し，設定したルール通りに通信制限が出来ているかどうかを確認する．測定結果を表 4 に示す．

表 4 通信制限の測定

制限	設定値	測定値 (NewGK)			測定値 (GK)		
		RTT (msec)	帯域幅 (Mbps)	損失率 (%)	RTT (msec)	帯域幅 (Mbps)	損失率 (%)
THROUGH	-	0.037	-	0	0.501	-	0
DELAY	50	50.051	-	0	50.361	-	0
	100	100.057	-	0	100.335	-	0
	200	200.047	-	0	200.277	-	0
	300	300.041	-	0	300.285	-	0
LIMIT	50	-	(UDP)48.8 (TCP)47.9	0	-	(UDP)48.8 (TCP)47.9	0
	100	-	(UDP)98.0 (TCP)95.7	0	-	(UDP)98.0 (TCP)95.7	0
	500	-	(UDP)490 (TCP)478	0	-	(UDP)489 (TCP)482	0
LOSS	25	-	-	24	-	-	27
	50	-	-	49	-	-	52
	75	-	-	74	-	-	75
	100	-	-	100	-	-	100

NewGK の測定値と GK の測定値のどちらとも，設定値に近い値が測定できた．この結果から，設定したルール通りに通信制限ができていることがわかる．

4.3 改良した機能の動作確認

○ TCP 通信途中切替の評価

4.1 節の図 6 に示した実験環境の Attacker から Server に対して独自に作成した TCP 通信のサーバ，クライアントプログラムを用いて通信した．設定した NAT ルール通りに Honey に転送，切替ができているかどうか，3.1 節の図 4 と同様の流れで実験し，確認する．

Router で tcpdump を起動し，パケットを表示して期待通りに通信切替が動作しているかを確認した．tcpdump の一部を次に示す．

```
// 接続開始
18:33:52.166146 IP 192.168.1.1.42463 > 192.168.2.2.60000:S
18:33:52.170249 IP 192.168.2.2.60000 > 192.168.1.1.42463:S
  // Attacker から Server, Honey に SYN を送信
18:33:52.176121 IP 192.168.1.1.42463 > 192.168.2.1.60000:S
18:33:52.178733 IP 192.168.2.1.60000 > 192.168.1.1.42463:S
  // Server, Honey が応答
18:33:52.178781 IP 192.168.1.1.42463 > 192.168.2.2.60000: . ack 1 win 92
18:33:52.188780 IP 192.168.1.1.42463 > 192.168.2.1.60000: . ack 1 win 92
// Server, Honey ともに TCP による通信を確立

// CLONE から SWITCH に切替
18:34:22.186231 IP 192.168.1.1.42463 > 192.168.2.1.60000: FP 31:37(6) ack 31
18:34:22.186266 IP 192.168.2.1.60000 > 192.168.1.1.42463: P  31:37(6) ack 38
18:34:22.186297 IP 192.168.2.1.60000 > 192.168.1.1.42463: F  37:37(0) ack 38
// Attacker から Server に FIN 送信し，TCP 通信終了

// 接続終了
18:34:41.770859 IP 192.168.1.1.42463 > 192.168.2.2.60000: F 55:55(0) ack 55
18:34:41.770914 IP 192.168.2.2.60000 > 192.168.1.1.42463: F 55:55(0) ack 56
18:34:41.770983 IP 192.168.1.1.42463 > 192.168.2.2.60000: . ack 56
// Attacker から Honey に FIN 送信し，TCP 通信終了
```

この結果から，期待通りに TCP 通信の途中切替が実装できていることが確認できた．しかし，iperf を使用した場合，Server の端末では通信切替後にきちんと通信が終了しないという問題があることがわかった．その原因は NewGK プログラムの TCP 通信の終了処理にある．通信の終了処理は図 7 のように Server からの FIN と ACK を同時に送信するように実装したが，正式な処理は図 8 のようにの FIN と ACK を別々に送信するためである．

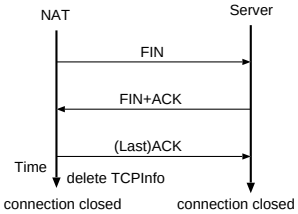

図 7 NewGK の終了処理

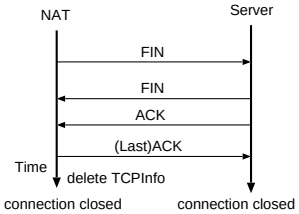

図 8 正式な終了処理

○ ホスト認証機能

登録されている CN からの通信であり，証明した認証局が同じときのみ認証を突破することがわかった．この結果から，期待通りにホスト認証機能が動作していることが確認できた．

○ 設定済みルール保存機能

設定したルールがきちんとファイルに保存できていることを確認した．また，保存したファイルを読み出し，ルールの設定ができることも確認できた．

しかし，LOSS や NAT のルールの時は，再起動後に正常に通信制限を施すことができないという問題があることもわかった．これは，LossGenerater や NAT オブジェクトを保存できていないことが原因であると考える．

5 おわりに

本研究によって，通信制限システムに TCP 通信の途中切替，SSL クライアント認証を用いた安全なリモートアクセス，システム停止時のルール保存機能を実装した．

実験結果から，GINE のライブラリクラスを再利用してシステムを再設計することで，システムの安定性を高めることができたと考える．通信の終了処理に問題は残るが，TCP 通信の途中切替機能も実装することができた．また，外部ホストのリモートアクセスに，SSL クライアント認証を用いたことで，システムのセキュリティの向上にもつながった．さらに，LOSS や NAT のルール保存はまだ不完全であるが，設定済みルール保存機能も実装することができた．

今後の課題として，TCP 通信の途中切替機能の終了処理，LOSS や NAT のルールの保存方法の改良がある．また，IPv6 環境での性能評価や，実ネットワークにおける運用実験も必要である．

さらに，後藤研究室，松永氏の研究 [5] である Web 経由での遠隔操作手法を組み込むとさらに使いやすいシステムになると考える．

参考文献

[1] Aoyama, M., Kojima, M., Goto, K.: Design and Implementation of a Traffic Limiter for Network Security, Proc. of 16th International Conference on Systems Science, Vol.II, pp.213-220 (2007).

[2] 浅野 洋介: 大規模ネットワーク構築のための GINE の管理機能の追加, 修士論文, 南山大学 大学院 数理情報研究科 数理情報専攻 (2010).

[3] Free Software Foundation: Gnu common C++, http://www.gnu.org/software/commoncpp/ (accessed Aug. 2009).

[4] 警察庁セキュリティポータルサイト@ police: 平成 21 年上半期におけるインターネット治安情勢, http://www.cyberpolice.go.jp/detect/pdf/H_kamihanki.pdf (accessed Aug. 2009).

[5] 松永 龍太朗: 通信制限システムの Web 経由での安全な遠隔操作, 卒業論文, 南山大学 数理情報学部 情報通信学科 (2010).

[6] 中西 忠夫, 坂口 由佳: 段階的通信制限システムの拡張, 卒業論文, 南山大学 数理情報学部 情報通信学科 (2009).

[7] Sugiyama, Y. and Goto, K.: Design and Implementation of a Network Emulator using Virtual Network Stack, Proc. of the Seventh International Symposium on Operations Research and Its Applications (ISORA2008), Lecture Notes in Operations Research, Vol.8, pp.351-358 (2008).

[8] Viega, J., Messier, M. and Chandra, P.: OpenSSL 暗号・PKI・SSL/TLS ライブラリの詳細, オーム社, (2004).