パケットキャプチャに基づくIPトレースバックシステムの試作

M2010MM031 小田嶋 晃
指導教員：河野 浩之

1 はじめに

DoS 攻撃 (Denial of Service Attack) や DDoS 攻撃 (Distributed Denial of Service Attack) が増加している．DDoS 攻撃には，コネクション型の攻撃とコネクションレス型の攻撃がある．攻撃者は攻撃などに IP スプーフイング (アドレス詐称) を利用して攻撃元をわかりにくくしていることが多い．IP スプーフィングは性質上コネクションレス型の攻撃に利用されやすい．代表的なコネクションレス型の攻撃にはフラッディング攻撃，スマーフィング，ランドアタックなどがある．とりわけフラッディング攻撃はさまざまなプロトコルで利用可能な攻撃手法である．このような攻撃に対応するために侵入検知システム (Intrusion Detection System) や侵入防止システム (Intrusion Prevention System) などが開発されてきた．

本研究では，攻撃への対策として，IPS として本研究室で開発中のゲートキーパー [3] のために迅速なトレースバック機能を試作する．

昨今のトラフィックには電子商取引などの個人情報と密接に関連したデータのやりとりもなされるようになってきた．その分，トレースバックによって発信元ホストを特定することは非常に難しい．よって，現在 IP トレースバックの実証実験をするためにさまざまな規格の策定や商用 ISP と協力した実ネットワークを利用した IP トレースバック実証実験をしようとしている段階である [2]．本研究では，debug 機能を搭載していないルータのことも考えて本研究ではルータのロギングモジュールも作成する．性能評価基準はトレースバックに要する時間とする．

2 関連研究

本節では，先行研究のトレースバックシステムについて述べる．

2.1 既存の IP トレースバック手法

図 1，図 2 はそれぞれ先行研究 [1] で利用されていたトレースバックの通信シーケンス，本研究でのトレースバック通信シーケンスを示したものである．表 1 に図 1 と図 2 を基に先行研究と本研究のトレースバックシステムの主な違いを示す．

表 1 先行研究と本研究のシステム比較

	先行研究	本研究
要求発信元	被害者ホスト	サイトのルータ
応答経路	tracer を中継	発信元へ直接返答

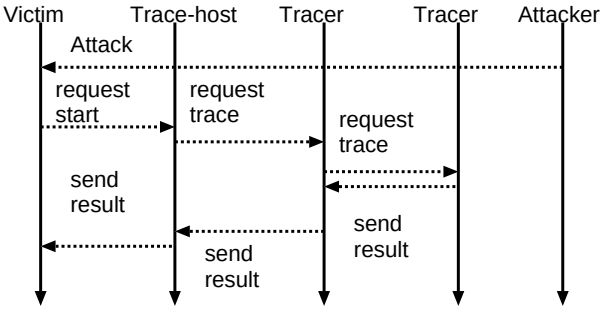

図 1 先行研究でのトレース通信シーケンス

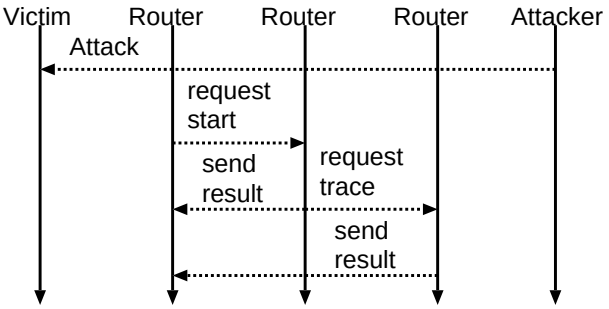

図 2 本研究でのトレース通信シーケンス

3 試作したトレースバックシステムの概要

本節では，試作したトレースバックシステムの設計について述べる．

3.1 トレースバック要求・応答プロトコル

本研究で実装するトレースバックスレッドの要求・応答時に用いる通信プロトコルを以下に示す．

- フラグ (QUERY, RELAY, FINAL)
- 状態 (INITIAL, NOTFOUND, FOUND)
- 問い合わせ元ホストの IP アドレス
- 問い合わせ元ホストのポート
- タイムスタンプ
- ホップカウント
- キー

フラグは要求と応答を区別するものである．状態は見付けたのかそうでないのかを示すものである．一番最初にトレースバック要求を出した IP アドレスとポートはトレースバック結果を中継するのではなく，直接ホストに返すために利用するものである．タイムスタンプは問い合わせ対象パケットが発生した時刻で，各トレーサがサーチ対象の限定に利用する．ホップカウントは通過してきたルータが何番目の経路に値するのかを示すものであり，経路逆

算に利用する．キーはトレースバックしたい通信のシグネチャである．

本研究では，ABR と Site 接続に利用されるルータに Tracer を設置してトレースバックを実現することを前提とする．

エミュレーション時には，全ての ABR に Tracer を付け，map を利用したハッシュテーブルに通信のログを入れる．それぞれの ABR で攻撃者までの経路により近い上位ルータを探して，最終的な攻撃ホストが所属する AS までの経路を特定する．そのために，各ルータでログを取得することになる．そのさい，全てのパケットの全ての情報を記録していたのでは，とても無駄であるため，通信の特定に必要であり変わらない情報だけを unordered_map を利用して格納し，ハッシュテーブルを作成する．格納するネットワークレイヤーのデータは

- 送信元・送信先 IP アドレス
- 送信元 MAC アドレス
- プロトコル番号
- データの 20Byte

の文字列をキーにする．またプロトコル別に以下のデータをキーとして追加する．

表 2　プロトコル毎のシグネチャ

プロトコル名	値
TCP	送信元ポート, 送信先ポート
UDP	送信元ポート, 送信先ポート
ICMP	タイプ, コード

実際に LogEntry に格納されているデータの例を図 3 に示す．

LogEntry に格納されたデータのサンプル

```
key: 10.64.6.216|10.64.6.127|144|6|51086|
5632|
ded96ceedae3814ab0402100da08733c793a0a25
srcMAC: 0:a:79:88:4b:b6
timestamp: 1327984136
```

図 3　データサンプル

key は前から順番に発信元 IP アドレス, 送信先 IP アドレス, ペイロード長, 次レイヤーのプロトコル情報, データ先頭の 20 バイトを 16 進表記で記録する．

3.2　トレースバックスレッドの起動

要求発生ルータでは，PacketLog, ClientThread を動かし，要求を受けるルータでは PacketLog と要求に備えた ServerThread を動かす．要求発生ルータとそれ以外のルータでは起動するスレッドが違うので，それぞれ示す．図 4 にはトレースバック要求を発するルータのスレッド起動を示す．

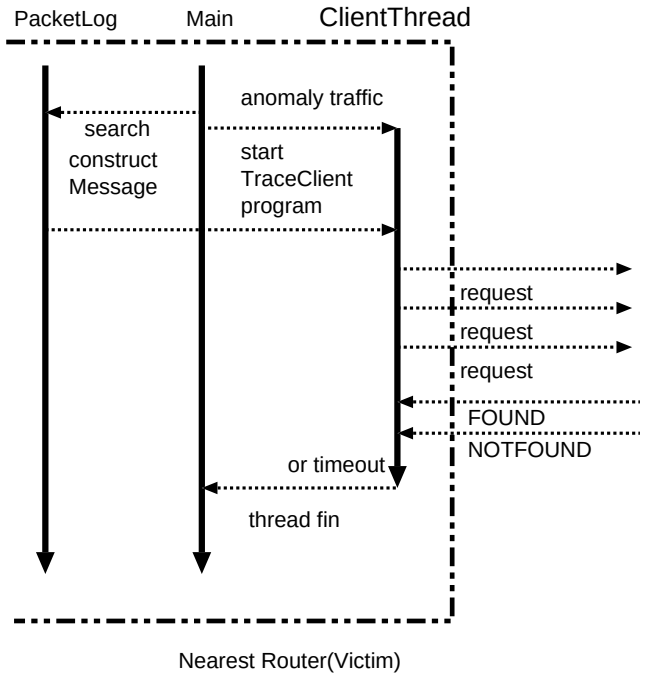

図 4　iptrace(トレースバック要求発信プログラム)

メインプログラムが異常通信を検知すると，PacketLog スレッドに問い合わせ，ClientThread のスレッドを生成する．この ClientThread は設定ファイルに静的に設定された同一 AS 内ルータや隣接ルータでトレースバックスレッドが動いているルータへそれぞれ問い合わせ要求を送信する．隣接ルータにのみ request を送ればよい場合は request は 1 つであるが，それ以外の場合は複数ルータにそれぞれ request を送信する．

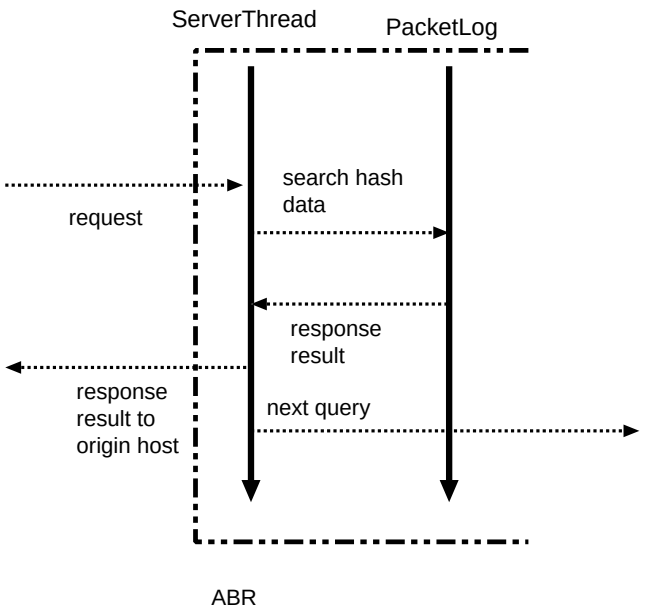

図 5　iptraceServer(トレースバック応答受信プログラム)

図 5 にトレースバック要求を受信し，返答するルータ内で動くプログラムのスレッドに関して示す．他のトレースバックルータから要求を受け取ると，トレースバックプ

ログラムサーバは PacketLog スレッドに問い合わせてそのパケットが通過したかどうかを問い合わせる. そして, 送られてきたメッセージを基に, 問い合わせ元のルータに返答し, 他のルータに問い合わせる必要があればさらに next request として次のルータへトレースバック要求を送信する. このさい, メッセージ内のオリジナル IP アドレスやポートについては値を変更しない. そのことにより, 次のルータも要求発信ルータへと応答を返すことが可能となる.

3 節で述べた要素をハッシュのキーを基として, パケットの情報と時間と上位ルータの MAC アドレスを LogEntry に格納する. 時間を格納するのは, 無尽蔵に通信のログを収集出来るわけではなく, 効率よくログを利用するために, 一定時間ごとにログをローテーションさせてログを収集するためである.

4 IP トレースバックの実装

本節では, 実験ネットワークや作成したクラスについて述べる. 実験環境の構築には後藤研究室で開発中のネットワークエミュレータ GINE(Goto's IP Network Emulator) を利用し, 1 台の PC で仮想ネットワークを構築し実験する.

4.1 IP トレースバックメインスレッド

図 6 にメインスレッドの処理の流れを示す.

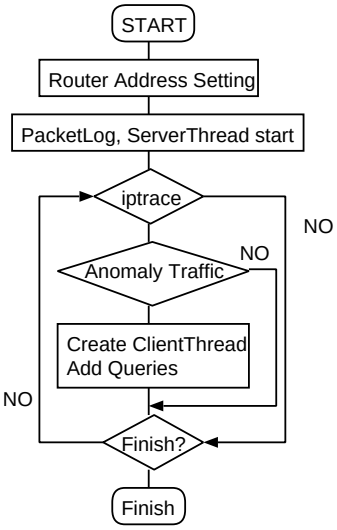

図 6 メインプログラム (iptrace) フローチャート

実験では, 通信が出るはずのないアドレスからの通信が発生した場合にトレースバック要求が発生するようにした. 例えば, (10.0.0.1) のアドレスが送信元アドレスとなる通信である. 本来は, 図 8 のような構成となり, アドレスのファーストオクテットが 192 から始まるネットワークであり, ファーストオクテットが 10 から始まる (10.0.0.1) のアドレスからの通信は発生しない. そこで Site 外から 10.0.0.1 の通信が来ていないか定期的に監視する. メインプログラムは起動すると, それぞれのルータにあわせた設定ファイルを読み込み, 自分が問い合わせるべきトレースバックルータのアドレスを設定し, PacketLog スレッドを起動する. ホップカウントが 1 のルータがオリジン IP, ポートを代入する. そして, 異常通信が発生すると, ClientThread を起動し, 設定ファイルに基づき上位ルータへ問い合わせ要求を送信し, 応答を一定時間待つ. 一定時間内に応答がない場合は, ClientThread の処理を終える.

4.2 PacketLog クラス

本節では, PacketLog クラスについて述べる. PacketLog クラスで設定するメソッドは以下である.

- `LogEntry* find(std::string key)`
  全文一致検索するメソッドである.
- `LogEntry* findByStr(std::string key)`
  先頭文字列部分一致検索するメソッドである.

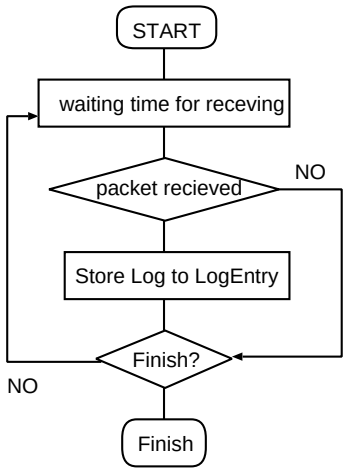

図 7 PacketLog のスレッド内処理

またログ件数の上限は指定せず, 一定時間毎 (例えば 1 分) でローテーションして 2 つのログテーブルを使用する.

4.3 ClientThread クラス

本節では, ClientThread クラスのメソッドについて述べる.

- void run(); 実際に ClientThread として動作するメインの動作内容がここに含まれる.

この ClientThread クラスは, iptrace で異常検知によりスレッドが生成され, 残り時間を select で監視することで, 一定時間が経つとスレッドが終了しオブジェクトが消えるよう設計した.

4.4 ServerThread クラス

本節では, ServerThread クラスの概要について述べる.

- `ServerThread(std::vector<std::string> neighbors, int listenPort, PacketLog &log);`
  このメソッドはコンストラクタである.
- `void run();`
  受信待機, ログサーチ, 返答・中継, Message 作成送信

この ServerThread クラスは, 全てのルータで PacketLog スレッドと同様常に動いており, 他のトレースバック機能

をもつルータからのトレースバック要求に常に備えるスレッドである.

5 実験

本節では実験環境と結果について述べる. 本研究での実験には Ubuntu Linux 10.04LTS(Linux kernel 2.6.32-38-generic), CPU は Intel(R) Xeon(R) CPU X3220 2.40GHz クアッドコアのものを利用した. メモリは 2GB である.

5.1 実験ネットワークトポロジ

図 8 に実験で利用するネットワークトポロジを示す.

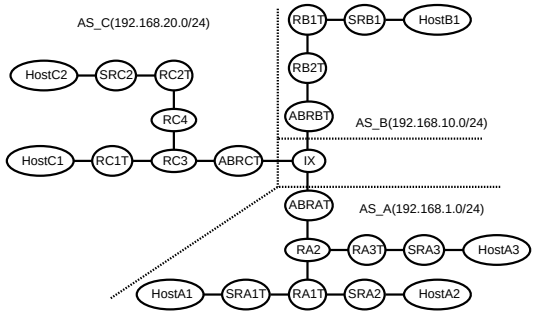

図 8 テストネットワークトポロジ

このトポロジでは, 保護対象ホストと攻撃者が同一 AS に居る場合や, 他の AS からの攻撃が発生している場合などさまざまなケースのエミュレートが可能である.

頭文字が S で始まるルータは顧客となるホストをそれぞれ持つ Site のルータである. 末尾に T と付くルータはトレースバックスレッドが動くトレースバック用のルータである.

5.2 実験結果

本節では, 本研究で作成したトレースバックプログラムの実験の一例について述べる. この実験は, HostB1 から発信元アドレスを 10.0.0.1 に偽装した通信が HostA1 に送られたときに, SRA1T でトレースバック要求が発生する想定である. 図 9 に SRA1T に iptrace プログラムを動かしたときの出力結果を示す. また, HostC1 から HostA1 にかけて ping flood を流し, トレースバックプログラムが高負荷環境で動いている実験のものを示す. 同じアドレスからホップ数が増えているのは, ルータの設定ファイルにアドレスが記載されていて送信されるからと思われる. しかし, 一度 Found になっていれば問題はない. 各トレースバックルータから帰ってきたメッセージは, IP アドレス・フラグ・状態・トレースバック要求をしたルータのオリジナル IP アドレス・トレースバック要求を出したルータのポート・ホップカウントと続く. 図 9 には IP アドレス (ルータの名前), Flag, stat, hops だけ示した. どのルータを通ってきたかは, stat が Found であり, hops が増えているものを辿ることでわかる. よって, 出力結果抽出末尾の hops と stat が FOUND であるものを辿ると RA1T, ABRA, ABRB, RA2T, RA1T を通ってきたことがわかる.

図 9 HostB1 からの攻撃 (高負荷時)

```
QUERY Xmit at 1328767841.858686|
QUERY|INITIAL|||0|1328767818.223364|
10.0.0.1|192.168.1.1|72|1|8|0|
From: srcIP timestamp|Flag|Stat|||hops|
From: 192.168.100.253(RA1T)
at 1328767841.859132|REPLY|FOUND|||0
From: 192.168.104.254
at 1328767841.859177|REPLY|NOTFOUND|||1
From: 192.168.105.253 (ABRAT)
at 1328767841.859518|REPLY|FOUND|||1
From: 192.168.104.254
at 1328767841.859546|REPLY|NOTFOUND|||2
From: 192.168.255.2 (ABRBT)
at 1328767841.868639|REPLY|FOUND|||2
From: 192.168.152.254 (RB2T)
at 1328767841.868679|REPLY|FOUND|||3
From: 192.168.151.254 (RB1T)
at 1328767841.868703|REPLY|FOUND|||4
Total time: 0.010017
```

トレースバックに要した時間はそれぞれ高負荷環境では, 0.01 秒, 負荷があまりない状態では, 0.001 秒程度であった. IX で繋がっている ABR で時間がかかっている. これは, ログの量も多くトラフィックも多いため通信速度も遅くなっているからと考えられる.

6 おわりに

本研究では, IP スプーフィングをするホストの攻撃により発生する不適切に偽装されたアドレスからの通信に対してトレースバックするシステムの試作に成功した. また, 実ネットワークのクラッカーからの攻撃を精度評価の指標には入れていない. 実験により, 同一 AS 内の別 Site や別の AS のさまざまな Site からの IP スプーフィングに対し, それぞれ経路を 出すことが出来た. Site Router でトレースバックスレッドが動いていることで Site まで特定でき, Site Router で動いていない場合は AS まで特定出来ることが実験から分かった. 発信元の AS 等の特定が高負荷でも 0.01 秒以内で可能である. したがって, このシステムは実用的である.

参考文献

[1] 伊藤健司, 川本高弘：既存手法を組み合わせた IP トレースバックの提案と評価, 2004 年度卒業論文, 南山大学数理情報学部情報通信学科 (2005).

[2] 櫨山寛章, 若狭　賢, 門林雄基：実証実験に向けた IP トレースバックシステム導入シナリオに関する一考察, 信学技報 IA, Vol. 108, No. 120, pp. 25–30 (2008).

[3] 福井麻美：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, 2009 年度修士論文, 南山大学数理情報研究科数理情報専攻 (2010).