Telnetによる渡り歩きの検出方法の検討

竹 尾 大 輔† 渡 邊 晃†

近年，不正アクセスなどのイントラネット内部の犯罪が増加傾向にある．クラッカーが不正アクセスを行う場合，Telnetによる渡り歩きを行っているケースが多い．渡り歩きを検出することが出来れば，多くの不正アクセスを防止することが可能であると考えられる．不正アクセス対策技術の一つとして，IDS（Intrusion Detection System：侵入検知システム）が考えられるが，不正な渡り歩きを検出することは難しい．本論文では，正常か不正かをも判別できる渡り歩き検出方法について報告する．

Researches on Detection Method of Island Hop Using Telnet

DAISUKE TAKEO† and AKIRA WATANABE†

In recent years, the crime inside Intranets, such as illegal access, is increasing. When a cracker accesses illegally, Island Hop using Telnet is performed in many cases. If Island Hop is detectable, it will be thought possible to prevent many illegal accesses. As one of the illegal access measure technology, although IDS (Intrusion Detection System) can be considered, it is difficult to detect illegal Island Hop. In this paper, we report the Island Hop detection method which can distinguish whether it is legal or illegal.

1. はじめに

近年, 不正アクセスなどのイントラネット内部の犯罪が増加傾向にあり, その被害は外部からの侵入による被害に匹敵するとも言われている[1]. 不正アクセスの例としては, 調査（PINGスイープ，ポートスキャン，etc）・侵入（なりすまし，セキュリティホール，etc）・攻撃（バッファオーバーフロー，DoS攻撃，etc）・盗聴（パケットキャプチャ，etc）などが存在している[2]．企業などのネットワーク管理者にとっては, これらに対するセキュリティ対策が重要となっている.

ネットワーク犯罪に対し, セキュリティ対策技術も数多く提案・開発されている．検知・追跡の観点から見た技術には，IDS[2)~6)]やトラップ[7), 8)]などが挙げられる. IDSはセキュリティ侵害を検出して通知し, ログ情報を管理するシステムである. トラップは侵入者に偽情報を提供し, 侵入者の行動などの情報を収集するシステムである．また，これらから得られる各種ログから攻撃の兆候を発見したり, 侵入者の特定を試みたりして, 今後の対策を取ることができる. 防御の観点では, ファイアウォールやホストの要塞化などが挙げられる. ファイアウォールはパケットフィルタリングなどを行い，イントラネットに対するアクセスを制限するシステムである. ホストの要塞化は稼動するサービスを最小限に留めたり, 常に最新のセキュリティパッチを適用したりするなどして, セキュリティ侵害の発生する可能性を低くする対策方法である.

しかしながら, セキュリティ対策技術を導入していてもクラッカーは不正アクセスを試みる. 一般にクラッカーが不正アクセスを行う場合, 不正な渡り歩きを行っているケースが多く見られる. 渡り歩きとは, 一つ以上のホスト（踏み台）を介して連鎖状に多段にリモートログインすることを言う．リモートログインは，Telnet[9)]，rlogin，SSH[10)]（Secure SHell）などのコマンドで行うことができる．Telnet，rloginは通信が平文のままやり取りされるが，SSHはパスワード認証時も含めて全通信が暗号化されている. 渡り歩きが悪意を持って行われると, 踏み台攻撃となる. このような渡り歩きが多く行われている理由としては, クラッカーが自分の身元を隠すためであったり, 侵入したホストからさらに別のホストへ侵入するためであったりする. この渡り歩きを検出することが出来れば, 多くの不正アクセスを防止することが可能であると考えられる.

不正な渡り歩きに対するセキュリティ対策技術の一つとしてIDSがあるが，ネットワークを流れる通信やホストに対するアクションを監視しているだけの方式では, 不正な渡り歩き

† 名城大学理工学部
Faculty of Science and Technology, Meijo University

を検出することは難しい．本論文では，通信が平文で行われるTelnetを用いた渡り歩きを対象とし，それが正常か不正かをも判別できる渡り歩き検出方法について検討する[11), 19)]．そして渡り歩き検出方法の有効性を確認するための機能を開発・実装し，機能を評価する．

以降，2 章では既存のセキュリティ対策技術である IDS を取り上げ，その概要と渡り歩きを検出する上での問題点を述べ，不正な渡り歩き検出に必要な事項をまとめる．3 章で渡り歩きを検出する条件を述べた上で，本論文で提案する渡り歩き検出方法を説明する．4 章では提案方式の有効性を確認するための機能の設計を行い，5 章で IDS との比較などをして評価を行う．そして最後に 6 章でまとめる．

2. 侵入検知システム IDS

2.1. IDS の概要

IDS とは，ネットワークを流れる通信やホストに対するアクションを監視し，不正なアクセスを検知するシステムのことである．**図 1** に

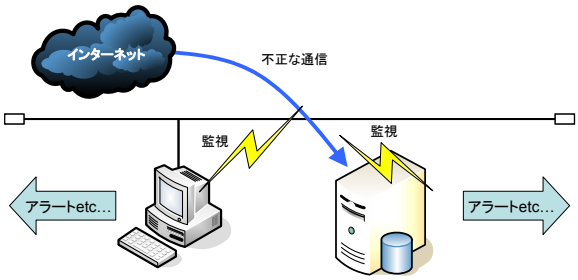


図 1 IDS の動作のイメージ図
Fig.1 Image figure of operation of IDS.

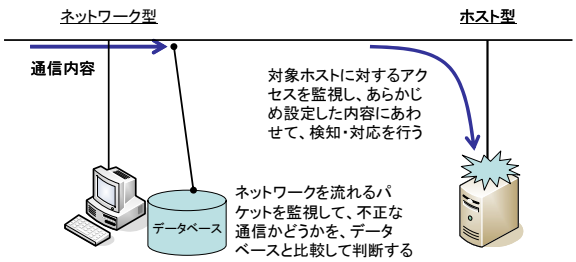


図 2 入力情報による分類
Fig.2 Classification by input.

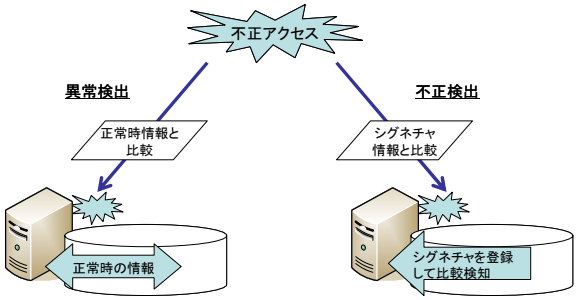


図 3 検出手法による分類
Fig.3 Classification by the detection technique.

IDSの動作のイメージ図を示す．侵入検知を行うと，システムに対する攻撃を監視・記録して存在する脅威の程度を確認でき，得られた記録を分析することでセキュリティポリシーやシステムの設定を見直すことができる．それにより被害の発生の防止や軽減を図ることができ，ユーザに監視システムの存在を知らしめて不正行為を抑制することもできる．IDSや侵入検知方法に関して，現在も研究が行われている[12)~16)]．

IDS にはいくつかのタイプが存在し，入力情報による分類と検出手法による分類ができる．

入力情報による分類では，ネットワーク型（NIDS）とホスト型（HIDS）に分けられる（**図 2**）．NIDS はネットワークを流れるパケットを監視し，不正な通信かどうかをデータベースと比較して判断する．HIDS は対象ホストに対するアクセスを監視し，あらかじめ設定した内容にあわせて，検知・対応を行う．

検出手法による分類では，異常検出と不正検出に分けられる（**図 3**）．異常検出は正常な状況（プロファイル）を記憶・定義し，そこからの変化をチェックする．不正検出は不正な通信の情報（シグネチャ）を保持し，一致するものを検出する．

NIDS は不正検出の役割を，HIDS は異常検出の役割をすることが多い．

IDS が侵入を検知した後は，何らかの対応をする場合が多く，その対応例には，

- アラート
- メール
- ポケベル，携帯電話
- 通信の遮断処理
- 他のアプリケーションやプログラムとの連携
- SNMP トラップ

などがある．どの対応にも利点と欠点が存在し，ネットワークの管理体系やセキュリティポリシーに応じて慎重な設定が必要である．

2.2. IDS での渡り歩き検出の限界

Telnet による渡り歩きを考えた場合，ネットワーク上を流れる Telnet の動作自体は不正ではない為，NIDS で渡り歩きを検出することは出来ない．また，渡り歩きのパターン（Telnet データ）は不定であるので，固定パターンを定義して比較しても対応できない．HIDS では，踏み台となるホストのログやコマンドヒストリを監視することで渡り歩きを検出することができる．しかし，出力されたログから検出しているのでリアルタイム性に欠ける．

最も重要な問題として，NIDS，HIDS どちら

の場合でも，渡り歩きが正常か不正かは通常は判断できないという問題がある．一般的なネットワーク構成では，それを判断するための情報が無いからである．

2.3. 不正な渡り歩きの検出に必要な事項

Telnet による渡り歩きをリアルタイムで検出するためには，踏み台となるホストで直接パケットを監視する必要がある．また，渡り歩きが正常か不正かを判断するためには，通信開始者が，渡り歩きをしてアクセスしようとしている端末にアクセスしても良いか否かが分かる必要がある．

3. 渡り歩き検出方法

3.1. CCGI における渡り歩き

渡り歩きの正常・不正の判断には，判断の基準となる何らかの情報が必要である．そこで本提案では，閉域通信グループ[17)~21)]（Closed Communication Group for Intranet：CCGI）が構築されたネットワーク上でTelnetによる渡り歩きを検出する．

CCGI は，イントラネット内の各ホストを部門単位などでグルーピングすることで構築され，同一グループ内の通信を暗号化している（**図４**）．CCGI を構築することで他のグループからの不正アクセスや盗聴を防止し，グループ内の通信を保護することができる．CCGI 上の各ホストはグループ情報を所持しており，それを用いることで，どのホストと通信してよいか知ることができる．本提案ではグループ情報を利用することで，渡り歩きの正常・不正の判断が容易にできると考えられる．

CCGI 上での渡り歩きの例として，**図５**のようなグルーピングによってアクセスを制限された CCGI があったとする（ホスト X とホスト Y は部署 A に帰属しており，ホスト Y とホスト Z は一部の管理者がアクセス可能なグループ B に帰属している）．このネットワークでは，グループ A のみに帰属しているホスト X が，グループ B のみに帰属しているホスト Z に直接アクセスすることは出来ない．しかし同一グループに帰属しているホスト X とホスト Y は通信してよく，同様にホスト Y とホスト Z の間の通信は許可されている．よってホスト Y はホスト X，Z のどちらにもアクセスすることが可能である．ここでホスト X がホスト Y にリモートログインし，さらにホスト Z にリモートログインしてアクセスすると，不正な渡り歩きをしたことになる．

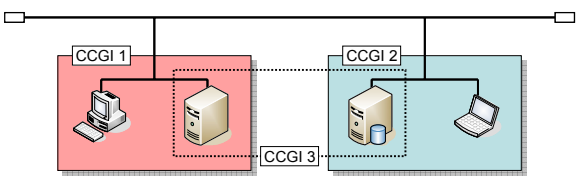


図４ CCGI が構築されたネットワーク
Fig.4 Network where CCGI was constructed.

3.2. 基 本 原 理

Telnet による渡り歩きでは，Telnet パケットがネットワーク上を流れることになるが，図５のホスト X が，送信元がホスト X，宛先がホスト Y の Telnet パケットを送信する場合を考える．このパケットをホスト Y が受信すると，ホスト Y では送信元がホスト Y，宛先がホスト Z，しかしデータは同一の Telnet パケットが生成され，ホスト Z へと送信される（**図６**）．本提案では，IP アドレスは異なるが，データは同じである送受信パケットが踏み台となるホストでほぼ同時に発生していることに着目し，踏み台となる可能性のあるホスト自身，或いはその直前に設置される機器において送受信パケットを監視することで渡り歩きを検出する．

3.3. 渡り歩き検出処理の流れ

監視対象となるホストでは，パケットの監視は IP 層で直接行う．受信パケットはデータリンク層からトランスポート層に渡される前に，送信パケットはトランスポート層からデータリンク層に渡される前に横取り・差し戻しを行う（**図７**）．こうすることでリアルタイムに渡り歩きを検出できるようになり，不正な渡り歩きが検出された際にはパケットを破棄することも可能となる．

提案方式では，Telnet パケットを受信してから，ごく短い一定時間内に Telnet パケットが送信される場合を渡り歩きと判断する．よってパケットの監視は，初期状態では受信パケットのみ行う．そして Telnet パケットを受信したら，

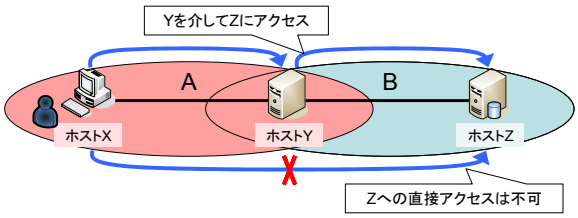


図５ CCGI 上での渡り歩きの概念図
Fig.5 Conceptual figure of Island Hop on CCGI.

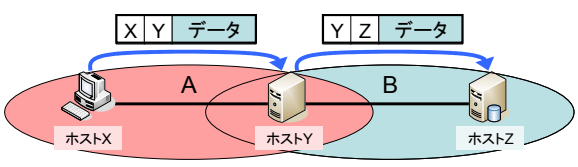


図６ Telnet パケットの動き
Fig.6 Motion of Telnet packets.

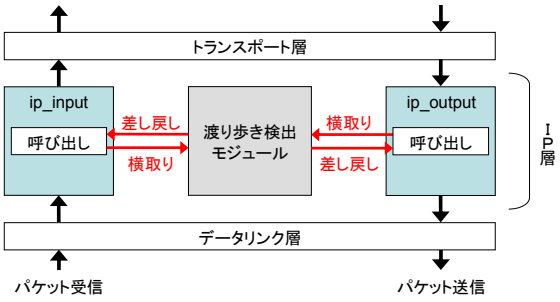


図７　IP 層での動き
Fig.7　Motion on IP Layer.

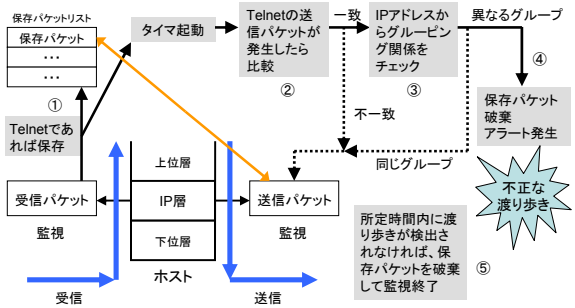


図８　渡り歩き検出処理の流れ
Fig.8　Island Hop Detection process flow.

そのパケットと対になる送信パケットが発生するかどうかを一定時間だけ監視する．

一つの受信パケットに対する渡り歩き検出処理の流れは以下の通りである（**図８**）．これにより不正な渡り歩きを検出することができる．

① 受信パケットを監視しておき，パケットが Telnet であればその内容を“保存パケットリスト”に保存し，それと同時にタイマを起動する．
② 受信パケットと同様に送信パケットも監視し，タイマ起動中に Telnet の送信パケットが発生したとき，保存しておいた Telnet 受信パケットの内容と比較する．
③ パケットの内容が一致した場合，受信パケットの送信元 IP アドレスと送信パケットの宛先 IP アドレスからグループ関係をチェックし，正常なログインによる渡り歩きであるか不正な渡り歩きであるかを判断する．パケットの内容が一致しなければ渡り歩きではないと判断し，送信パケットの監視を続ける．
④ グループが異なっていた場合，不正な渡り歩きであると判断し，保存しておいた受信パケットを破棄して送信パケットの監視処理を終了し，アラートを発生する．同一グループであれば正常な渡り歩きであると判断し，送信パケットの監視を続ける．
⑤ タイマを起動してから一定時間内に渡り歩きが検出されなかった場合，保存しておいた受信パケットを破棄して送信パケットの監視処理を終了する．

上記の渡り歩き検出処理の⑤を補足するが，タイマを起動してから一定時間内に Telnet パケットを受信した場合，そのパケットも“保存パケットリスト”に追加保存され，処理②から処理⑤を行う．

一方，一つの送信パケットに対して見てみると，送信時に保存されている受信パケット全てと比較することになる．

“保存パケットリスト”に保存するのは受信パケットのデータ全てではなく，送信パケットとの比較に必要なデータのみである．以下に保存するデータを示す．

- 送信元 IP アドレス
- 宛先 IP アドレス
- 送信元ポート番号
- グループ番号
- Telnet データ
- 受信時刻

送信元 IP アドレスは攻撃者となるホストを識別するものである．

宛先 IP アドレスは踏み台となるホストを識別するものである．

送信元ポート番号は Telnet のセッションを識別するものである．

グループ番号は送信元と宛先のホストが帰属するグループの番号であり，送信パケットのグループと比較するのに用いる．

Telnet データは TCP ペイロードの先頭 1 バイトのデータであり，送信パケットの Telnet データと比較するのに用いる．

受信時刻は Telnet パケットを受信した時刻であり，タイマ確認に用いる．

現段階ではアラート処理の内容は決定していないが，“渡り歩き検知リスト”として渡り歩き検出時の情報を記録している．アラート処理の候補として，

- 渡り歩き検出時の送信パケットの破棄
- 管理者への通知
- 攻撃者への警告

などを考えている．しかしこれらのアラートを一回だけの不正な渡り歩き検出で発生させてしまうと，誤報であった場合に問題である．そこで，“渡り歩き検知リスト”に検知回数を記録するフィールドを設け，攻撃者とターゲット

の組が同一である不正な渡り歩きが既定回数に達した場合のみ, アラートを発生するようにすればよいと思われる. 以下に“渡り歩き検知リスト”に記録するデータを示す.

- 攻撃者 IP アドレス
- 踏み台 IP アドレス
- ターゲット IP アドレス
- 攻撃者ポート番号
- 踏み台ポート番号
- 攻撃者グループ番号
- ターゲットグループ番号
- 検知回数
- 初回検知時刻
- 最終検知時刻

攻撃者 IP アドレスは受信パケットの送信元 IP アドレスである.

踏み台 IP アドレスは受信パケットの宛先 IP アドレス（送信パケットの送信元 IP アドレス）である.

ターゲット IP アドレスは送信パケットの宛先 IP アドレスである.

攻撃者ポート番号は攻撃者と踏み台との間の Telnet セッションを識別するものである.

踏み台ポート番号は踏み台とターゲットとの間の Telnet セッションを識別するものである.

攻撃者グループ番号は攻撃者ホストが帰属するグループ番号である.

ターゲットグループ番号はターゲットホストが帰属するグループ番号である.

検知回数はこのパターンで検知された不正な渡り歩きの回数であり, アラートを発生するかの判断に用いる.

初回検知時刻は初めてこのパターンで不正な渡り歩きが検知された時刻である.

最終検知時刻は最後にこのパターンで不正な渡り歩きが検知された時刻である.

4. 渡り歩き検出機能の実装

本提案による渡り歩き検出方法は既存の IDS とは仕組みが異なるため, 提案方式を実現するには新規にプログラムを作成しなければならない. ここでは渡り歩き検出機能の開発と実装方法について述べる.

4.1. 開　　発

開発は IP 層の処理に関する情報が多い FreeBSD で行っている. IP 層の処理は FreeBSD のカーネルに組み込まれているため, 渡り歩き検出機能のプログラムの作成も FreeBSD と同じ開発言語である C 言語を使用する. **表 1** に開発環境を示す.

表 1　開発環境
Table.1　Development environment.

OS	FreeBSD 5.1
CPU	Intel Pentium4 2.4GHz
メモリ	192MB
開発言語	C言語
開発ツール	gcc

4.2. 実　　装

カーネルプログラミングは, プログラムにバグがあり問題が発生した場合, OS 自体が停止してしまうため, デバッグ作業が大変である. よって現段階ではテスト開発として, 提案方式の渡り歩き検出機能の有効性を確認するためのプログラムを作成した（**付録**）. 動作が確認された後に IP 層への実装（カーネルへの組み込み）を行う予定である.

試作したプログラムは, BPF[22)]（BSD Packet Filter）というデータリンク層へ直接アクセスできるインターフェースを使用してパケットキャプチャしている. FreeBSDならば特別なライブラリを導入する必要がないが, キャプチャしたパケットはデータリンク層でコピーされただけのものであり, オリジナルパケットはそのまま送受信処理されてしまう. 提案方式で行いたい動作を厳密には再現できないが, アラートとしてパケット破棄などをしなければ, リアルタイム検出は可能である. 試作プログラムでは渡り歩きを検出した場合はコンソールに渡り歩き情報を表示する.

プログラムの処理は大きく二つのモジュールに分けられ, 一つはパケットのキャプチャとプロトコルの判別処理のモジュール, もう一つは渡り歩き検出処理のモジュールである. 渡り歩き検出モジュールは, パケットのプロトコルが TCP である場合にのみ呼び出され, 送受信パケット比較などの処理を行う.

5. 機 能 評 価

5.1. 実 験 環 境

試作したプログラムを実装して, 提案方式による渡り歩き検出が有効であるか確認する実験を行った. 実験には三台の PC を使用した. **表 2** に使用した各 PC のスペックを, **図 9** に実験ネットワーク構成を示す.

無線ルータ間はブリッジとして設定されており, それ以外の有線は 100BASE-TX で接続されている. 踏み台ホストとターゲットホストでは Telnet サービスを稼動させており, それぞれ適当なユーザアカウントを作成しておいた.

実験手順は, 踏み台ホストで試作プログラム

表 2　実験に使用した PC のスペック
Table.2　Spec. of PCs used for the experiment.

攻撃者ホスト	
OS	Windows XP Professional
CPU	Intel Pentium4 2.4GHz
メモリ	512MB
踏み台ホスト	
OS	FreeBSD 5.1
CPU	Intel Pentium4 2.4GHz
メモリ	512MB
ターゲットホスト	
OS	Red Hat Linux 8.0
CPU	Intel Pentium4 1.8GHz
メモリ	256MB

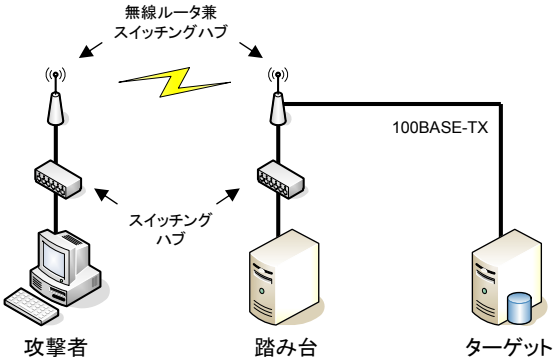


図 9　実験ネットワーク構成
Fig.9　Experiment network composition.

を起動した後, 攻撃者ホストから踏み台ホストへ Telnet によるリモートログインを行い, さらに踏み台ホストからターゲットホストへ Telnet によるリモートログインを行った.

5.2.　実験結果の評価

実験を行った結果, 提案方式で不正な渡り歩きを検出することができた. 実験手順の中で最初に攻撃者ホストからターゲットホストへと Telnet パケットが送られるのは, ターゲットホストへログインするときのユーザ名入力であるが, ここで初めて渡り歩きが検出された. 以降, キーを入力する度に渡り歩きを検出することができた. このように, 提案方式の有効性が確認できた.

表 3 に IDS と提案方式の渡り歩き検出に関する機能比較を示す. 既存の IDS で不正な渡り歩きを検出できないことや, 実験で数値的な結果を得ることが難しかったため, 今回は具体的な数値比較を諦め, 機能比較に留めることにした.

表 3　IDS と提案方式の機能比較
Table.3　Function comparison between IDS and proposal method.

	NIDS	HIDS	提案方式
リアルタイム性	高い	低い	高い
渡り歩き検出	不可	可能	可能
正常・不正の判断	不可	可能※	可能

※ 提案方式と同じ条件下の場合

NIDS では, リアルタイム性は高いものの渡り歩きを検出することはできない.

HIDS では, 渡り歩きを検出することが可能であるが, 普通のネットワークにおいては渡り歩きの正常・不正の判断ができない. 提案方式と同じように CCGI が構築されたネットワークであれば, グループ情報を用いて正常・不正の判断が可能になるとも考えられるが, いずれにせよリアルタイム性は低い.

提案方式では, 送受信パケットを監視することで渡り歩きを検出でき, CCGI のグルーピング情報を用いることで正常・不正の判断ができる. また, IP 層で直接パケットを監視するので検出までの時間が速い.

6.　お わ り に

本論文では, Telnet による渡り歩きを検出する方法を検討した. これにより渡り歩きが正常か不正かをも判別できるので、不正アクセスを防止することが可能となる.

また, 提案方式による渡り歩き検出機能の有効性を確認するためのプログラムを試作した. 実験により不正な渡り歩きを検出できることを確認した.

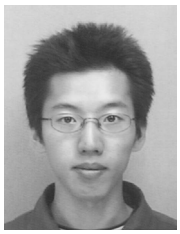
今後の課題としては, 渡り歩き検出処理の効率化と, カーネルへ組み込んだときのIP層本来の機能との整合性を図ることが必要である. また, 通信が暗号化されているSSHを用いた渡り歩きをも検出可能な方法も考案していくべきである. そして, 提案方式で得られる“渡り歩き検知リスト”を用いて, 多段リモートログインの大元が誰であるのかを探すトレースバック[23)~27)]への応用も検討している.

参　考　文　献

1) 2003 CSI/FBI Computer Crime and Security Survey, Computer Security Institute. http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2003.pdf
2) 白井雄一郎, 白濱直哉, 又江原恭彦, 柳岡裕美:インターネットセキュリティ　不正アクセスの手法と防御, ソフトバンクパブリッシング (2001).
3) 武田圭史, 磯崎　宏:ネットワーク侵入検知, ソフトバンクパブリッシング (2000).
4) Snort. http://www.snort.org/
5) 日本 Snort ユーザ会. http://www.snort.gr.jp/

6) RealSecure Network Sensor, http://www.isskk.co.jp/product/RS_NSensor.html, Internet Security Systems K.K. product.
7) 竹森敬祐, 力武健次, 三宅　優, 中尾康二 : Intrusion Trap System における安全で有効なログ収集のための動的切替え機能の実装, 情報処理学会論文誌, Vol.44 No.8 pp.1838-1847 (2003).
8) Decoy Server Solution, http://www.atsweb.it/Images/Documenti/TOP_DS_Decoy%20Server%20Solution.pdf, TOP Layer Networks Product.
9) J. Postel, J.K. Reynolds : Telnet Protocol Specification, RFC854, http://www.ietf.org/rfc/rfc0854.txt (1983).
10) Tatu Ylonen, Tero Kivinen, Markku Juhani Saarinen, Timo Rinne, Sami Lehtinen : SSH Protocol Architecture, Internet-Drafts, http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-15.txt (2003).
11) 竹尾大輔, 渡邊　晃 : TELNET による渡り歩きの検出方法の検討, 2003 年度電気関係学会東海支部連合大会, 一般講演 360 (2003).
12) Intrusion Detection Exchange Format (idwg) Charter, IETF. http://www.ietf.org/html.charters/idwg-charter.html
13) Adriano M. Cansian, Artur R. A. de Silva and Marcelo de Souza : An attack signature model to computer security intrusion detection, MILCOM 2002 - IEEE Military Communications Conference, Vol.21 No.1 pp.1368-1373 (2002).
14) Edward L. Witzke, Thomas D. Tarman, Sumit Ghosh and Gerald Woodard : A novel scaleable architecture for intrusion detection and mitigation in switched networks, MILCOM 2002 - IEEE Military Communications Conference, Vol.21 No.1 pp.394-398 (2002).
15) Ramkumar Chinchani, Shambhu Upadhyaya and Kevin Kwiat : Towards the scalable implementation of a user level anomaly detection system, MILCOM 2002 - IEEE Military Communications Conference, Vol.21 No.1 pp.1503-1508 (2002).
16) Constantine Manikopoulos and Symeon Papavassiliou : Network intrusion and fault detection: A statistical anomaly approach, IEEE Communications Magazine, Vol.40 No.10 pp.76-82 (2002).
17) 渡邊　晃, 厚井裕司, 井手口哲夫, 横山幸夫, 妹尾尚一郎 : 暗号技術を用いたセキュア通信グループの構築方法とその実現, 情報処理学会論文誌, Vol.38 No.4 pp.904-914 (1997).
18) 鈴木秀和, 渡邊　晃 : GSCIP を構成する DPRP の仕組みの検討, 情報処理学会 第 66 回全国大会, 5V-1 (2004).
19) 竹尾大輔, 渡邊　晃 : GSCIP を構成する渡り歩き検出機能の仕組みの検討, 情報処理学会 第 66 回全国大会, 5V-2 (2004).
20) 増田真也, 渡邊　晃 : 閉域通信グループにおける暗号通信方式の検討, 情報処理学会 第 66 回全国大会, 5V-3 (2004).
21) 保母雅敏, 渡邊　晃 : 多段構成ネットワークにおける鍵配送方式の一検討, 情報処理学会 第 66 回全国大会, 6V-2 (2004).
22) 村山公保 : 基礎からわかる TCP/IP ネットワーク実験プログラミング, pp.177, オーム社 (2001).
23) Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent and W. Timothy Strayer : Single-packet IP traceback, IEEE/ACM Transactions on Networking, Vol.10 No.6 pp.721-734 (2002).
24) Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson : Network support for IP traceback, IEEE/ACM Transactions on Networking, Vol.9 No.3 pp.226-237 (2001).
25) Dawn Xiaodong Song and Adrian Perrig : Advanced and authenticated marking schemes for IP traceback, IEEE INFOCOM 2001 - The Conference on Computer Communications, No.1 pp.878-886 (2001).
26) Kihong Park and Heejo Lee : On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack, IEEE INFOCOM 2001 - The Conference on Computer Communications, No.1 pp.338-347 (2001).
27) Brian Carrier and Clay Shields : A recursive session token protocol for use in computer forensics and TCP Traceback, IEEE INFOCOM 2002 - The Conference on Computer Communications, Vol.21 No.1 pp.1540-1546 (2002).

（平成 16 年 2 月 13 日評価用提出）
（平成 16 年 3 月 15 日製本用提出）



竹尾　大輔（渡邊研究室 B4）
2004 年名城大学理工学部情報科学科卒業予定．同年，同大学院理工学研究科情報科学専攻修士課程進学予定．インターネットセキュリティの研究に従事．2002 年度情報科学科プログラミングコンテスト特別賞受賞．ソフトウェア同好会部員．情報処理学会学生会員．

渡邊　　晃（教授）
1974 年慶応大学電気工学科卒業．1976 年同大学院修士課程修了．同年三菱電機（株）入社．以来，同社情報技術総合研究所にて LAN，ネットワークセキュリティ等の研究開発に従事．新エネルギー・産業技術総合開発機構（NEDO）を経て，現在，名城大学理工学部教授．情報処理学会会員．電子情報通信学会会員．

付録

- 今回提案したTelnetによる渡り歩きの検出方法の有効性を確認するためのプログラムを作成したので，そのソースコードを付録として添付する．
  - ihdump.c
  - gscip.h
  - ih_dtct.c
  - ih_dtct.h

```
/*********************************************************************
 *  Telnetによる渡り歩きモニタリングプログラム (ihdump.c)
 *       Ver 1.0 2004年 1月 25日
 *                              制作・著作 竹尾大輔 (Daisuke Takeo)
 *
 *  使用許諾書
 *    本プログラムは、Telnetによる渡り歩きをモニタリングするために、
 *    「TCP/IPパケットモニタリングプログラム (ipdump.c)」(制作・著作
 *    村山公保) を基に改造したものです。本プログラムについて、法律で
 *    禁止されているか、または、公序良俗に反するような改造、及び、使
 *    用を禁止します。本プログラムは無保証です。制作者は本プログラム
 *    によって発生したいかなる損害についても責任を取ることはできませ
 *    ん。
 *    また、本プログラムには不必要な余分なipdump.cのコードが残ってい
 *    ますが、渡り歩きモニタリングの動作には問題ないため、そのままに
 *    してあります。
 *********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>


#include "gscip.h"
#include "ih_dtct.h"


#include <net/ethernet.h>
#define  __FAVOR_BSD
#include <netinet/if_ether.h>
#include <arpa/inet.h>

#include <sys/ioctl.h>
#include <net/bpf.h>
#include <net/if.h>
#include <fcntl.h>

#define MAXSIZE 4096
#define OPTNUM  8
#define ON 1
#define OFF 0

enum {ETHER, ARP, IP, TCP, UDP, ICMP, DUMP, ALL};

int open_bpf(char *ifname);

void print_ip(struct ip *ip);
void print_tcp(struct tcphdr *tcp);

void dump_packet(unsigned char *buff, int len);

char *tcp_ftoa(int flag);
char *ip_ttoa(int flag);
char *ip_ftoa(int flag);
void help(char *cmd);


SP *sptop = NULL ;
SP *spend = NULL ;
IHD *ihdtop = NULL ;
int timer = 0 ;
int pkts = 0 ;
int telnets = 0 ;
int discards = 0 ;
int receives = 0 ;


int main(int argc, char* argv[])
{
  struct ether_header *eth;  /* Ethernetヘッダ構造体             */
  struct ether_arp *arp;     /* ARPパケット構造体                */
  struct ip *ip;             /* IPヘッダ構造体                   */
  struct icmp *icmp;         /* ICMPパケット構造体               */
  struct tcphdr *tcp;        /* TCPヘッダ構造体                  */
  struct udphdr *udp;        /* UDPヘッダ構造体                  */
  int s;                     /* ソケットディスクリプタ           */
  int len;                   /* 受信したデータの長さ             */
  int c;                     /* getopt()で取得した文字           */
  int disp;                  /* 画面に出力したかどうかのフラグ   */
  char buff[MAXSIZE];        /* データ受信バッファ               */
  char *p;                   /* ヘッダの先頭を表す作業用ポインタ */
  char *p0;                  /* パケットの先頭を表すポインタ     */
  char ifname[256] = "lnc0"; /* FreeBSDのインターフェース名      */
  int opt[OPTNUM];           /* 表示オプションのフラグ           */
  extern int optind;         /* getopt()のグローバル変数         */

  int bpf_len;               /* BPFでの受信データの長さ          */
  struct bpf_hdr *bp;        /* BPFヘッダ構造体                  */


  int mode = 0 ;
  PID pid ;
  int val = 0 ;
  char ownip[16] = "172.18.16.57" ;

  if(argc>=2 && strcmp(argv[1], "lnc0")!=0)
    strcpy(ifname, argv[1]) ;
  if(argc==3 && strcmp(argv[2], ownip)!=0)
    strcpy(ownip, argv[2]) ;
  if(argc >= 4)
    fprintf(stderr, "Too many parametas!¥n") ;

  /* 表示するパケットの種類の初期値 */
  opt[ETHER] = OFF;
  opt[ARP]   = ON;
  opt[IP]    = ON;
  opt[TCP]   = ON;
  opt[UDP]   = ON;
  opt[ICMP]  = ON;
  opt[DUMP]  = OFF;
  opt[ALL]   = OFF;

  if ((s = open_bpf(ifname)) < 0)
    exit(EXIT_FAILURE);
  bpf_len=0;

  while (1) {
    /*  BPFからの入力 */
    if (bpf_len <= 0) {
      /* 複数のパケットを一括取りだし */
      if ((bpf_len = read(s, buff, MAXSIZE)) < 0) {
        perror("read");
```

```
        exit(EXIT_FAILURE);
      }
      bp = (struct bpf_hdr *)buff;
    } else {
      /* BPFの次のパケットへポインタを移動 */
      bp = (struct bpf_hdr *)((char *) bp + bp->bh_hdrlen + bp->bh_caplen);
      bp = (struct bpf_hdr *)BPF_WORDALIGN((int)bp);
    }
    /* Ethernetヘッダの先頭にポインタをセット */
    p = pO = (char *)bp + bp->bh_hdrlen;
    len    = bp->bh_caplen;
#ifdef DEBUG
    /* BPFヘッダ構造の値を表示 */
    printf("bpf_len=%d,",   bpf_len);
    printf("hdrlen=%d,",    bp->bh_hdrlen);
    printf("caplen=%d,",    bp->bh_caplen);
    printf("datalen=%d¥n", bp->bh_datalen);
#endif
    /* 次のwhileループのための処理 */
    bpf_len -= BPF_WORDALIGN(bp->bh_hdrlen + bp->bh_caplen);
    /*
    * パケット表示ルーチン
    */
    disp = OFF; /* 画面に出力したかどうかのフラグ */

    /* Ethernetヘッダ構造体の設定 */
    eth = (struct ether_header *) p;
    p = p + sizeof (struct ether_header);

    if (ntohs(eth->ether_type) == ETHERTYPE_IP) {
      ip = (struct ip *) p;
      p = p + ((int)(ip->ip_hl) << 2);

      if (ip->ip_p == IPPROTO_TCP) {
        tcp = (struct tcphdr *) p;
        p = p + ((int)(tcp->th_off) << 2);

        pid.g_no = htons(0) ;
        if(ip->ip_src.s_addr == inet_addr(ownip)) {
          mode = 0 ;
          if(ip->ip_dst.s_addr == inet_addr("172.18.16.34")) {
            pid.g_no = htons(2) ;
          }
        }
        else if(ip->ip_dst.s_addr == inet_addr(ownip)) {
          mode = 1 ;
          if(ip->ip_src.s_addr == inet_addr("172.18.16.47")) {
            pid.g_no = htons(1) ;
          }
        }

        printf("src:%-15s --> ",
          inet_ntoa(*(struct in_addr *)&(ip->ip_src))) ;
        printf("dst:%-15s / ",
          inet_ntoa(*(struct in_addr *)&(ip->ip_dst))) ;

        val = ih_dtct(mode, &pid, ip, tcp, p) ;
        if(val == 0)
          printf("not telnet¥n") ;
        else if(val == 1)
          printf("rcv telnet¥n") ;
        else if(val == 2)
          printf("snd telnet¥n") ;
        else if(val == 3)
          printf("island hop!¥n") ;
      }
    }
  }
  return EXIT_SUCCESS;
}
```

```
/*
 * void print_ip(struct ip *ip);
 * 機能
 *     IPヘッダの表示
 * 引き数
 *     struct ip *ip;    IPヘッダ構造体へのポインタ
 * 戻り値
 *     なし
 */
void print_ip(struct ip *ip)
{
  printf("Protocol: IP¥n");
  printf("+-----+-----+----------+------------------------+¥n");
  printf("| IV:%1u| HL:%2u| T: %8s| Total Length: %10u|¥n",
    ip->ip_v, ip->ip_hl, ip_ttoa(ip->ip_tos), ntohs(ip->ip_len));
  printf("+-----+-----+----------+------+-----------------+¥n");
  printf("| Identifier:        %5u| FF:%3s|FO:         %5u|¥n",
    ntohs(ip->ip_id), ip_ftoa(ntohs(ip->ip_off)),
    ntohs(ip->ip_off) &IP_OFFMASK);
  printf("+----------+----------+-------+-----------------+¥n");
  printf("| TTL:    %3u| Pro:    %3u| Header Checksum:   %5u|¥n",
    ip->ip_ttl, ip->ip_p, ntohs(ip->ip_sum));
  printf("+----------+----------+-------+-----------------+¥n");
  printf("| Source IP Address:                %15s|¥n",
    inet_ntoa(*(struct in_addr *)&(ip->ip_src)));
  printf("+-----------------------------------------------+¥n");
  printf("| Destination IP Address:           %15s|¥n",
    inet_ntoa(*(struct in_addr *)&(ip->ip_dst)));
  printf("+-----------------------------------------------+¥n");
}

/*
 * char *ip_ftoa(int flag);
 * 機能
 *     IPヘッダのフラグメントビットを文字列に変換
 * 引き数
 *     int flag;   フラグメントフィールドの値
 * 戻り値
 *     char *      変換された文字列
 */

char *ip_ftoa(int flag)
{
  static int f[] = {'R', 'D', 'M'}; /* フラグメントフラグを表す文字 */
  static char str[17];              /* 戻り値を格納するバッファ */
  u_int mask = 0x8000;              /* マスク */
  int i;                            /* ループ変数  */

  for (i = 0; i < 3; i++) {
    if (((flag<< i) & mask) != 0)
      str[i] = f[i];
    else
      str[i] = '0';
  }
  str[i] = '¥0';

  return str;
}

/*
 * char *ip_ttoa(int flag);
 * 機能
 *     IPヘッダのTOSフィールドを文字列に変換
 * 引き数
 *     int flag;    TOSフィールドの値
 * 戻り値
 *     char *       変換された文字列
 */
```

```
char *ip_ttoa(int flag)
{
  static int f[] = {'1', '1', '1', 'D', 'T', 'R', 'C', 'X'};
                          /* TOSフィールドを表す文字 */
  static char str[17];    /* 戻り値を格納するバッファ */
  u_int mask = 0x80;      /* TOSフィールドを取り出すマスク */
  int i;                  /* ループ変数 */

  for (i = 0; i < 8; i++) {
    if (((flag<< i) & mask) != 0)
      str[i] = f[i];
    else
      str[i] = '0';
  }
  str[i] = '¥0';

  return str;
}

/*
 * void print_tcp(struct tcphdr *tcp);
 * 機能
 *     TCPヘッダの表示
 * 引き数
 *     struct tcphdr *tcp;  TCPヘッダ構造体
 * 戻り値
 *     なし
 */
void print_tcp(struct tcphdr *tcp)
{
  printf("protocol: TCP¥n");
  printf("+-------------------------+-------------------------+¥n");
  printf("| Source Port:       %5u| Destination Port:  %5u|¥n",
    ntohs(tcp->th_sport), ntohs(tcp->th_dport));
  printf("+-------------------------+-------------------------+¥n");
  printf("| Sequence Number:                          %10lu|¥n",
    (u_long)ntohl(tcp->th_seq));
  printf("+-------------------------------------------------+¥n");
  printf("| Acknowledgement Number:                   %10lu|¥n",
    (u_long)ntohl(tcp->th_ack));
  printf("+------+---------+-------+-------------------------+¥n");
  printf("| DO:%2u| Reserved|F:%6s| Window Size:       %5u|¥n",
    tcp->th_off, tcp_ftoa(tcp->th_flags), ntohs(tcp->th_win));
  printf("+------+---------+-------+-------------------------+¥n");
  printf("| Checksum:          %5u| Urgent Pointer:    %5u|¥n",
    ntohs(tcp->th_sum), ntohs(tcp->th_urp));
  printf("+-------------------------+-------------------------+¥n");
}

/*
 * char *tcp_ftoa(int flag);
 * 機能
 *     TCPヘッダのコントロールフラグを文字列に変換
 * 引き数
 *     int flag       TCPのコントロールフラグ
 * 戻り値
 *     char *         変換された文字列
 */
char *tcp_ftoa(int flag)
{
  static int f[] = {'U', 'A', 'P', 'R', 'S', 'F'};
                          /* TCPのフラグを表す文字 */
  static char str[17];    /* 戻り値を格納するバッファ */
  u_int mask = 1 << 5;    /* TCPのフラグを取り出すマスク */
  int i;                  /* ループ変数 */

  for (i = 0; i < 6; i++) {
    if (((flag << i) & mask) != 0)
      str[i] = f[i];
    else
      str[i] = '0';
  }
  str[i] = '¥0';

  return str;
}


/*
 * void dump_packet(unsigned char *buff, int len);
 * 機能
 *     Ethernetフレーム先頭から16進数ダンプ(アスキー文字表示)
 * 引き数
 *     unsigned char *buff:  ダンプするデータの先頭アドレス
 *     int len;              ダンプするバイト数
 * 戻り値
 *     なし
 */
void dump_packet(unsigned char *buff, int len)
{
  int i, j;  /* ループ変数 */

  printf("Frame Dump:¥n");
  for (i = 0; i < len; i += 16) {
    /* 16進数ダンプ */
  for (j = i; j < i + 16 && j < len; j++) {
    printf("%02x", buff[j]);
    if (j % 2 == 1)
      printf(" ");
  }

  /* 最後の行の端数を整列 */
  if (j == len && len % 16 != 0)
    for (j = 0; j < 40 - (len % 16)*2.5; j++)
      printf(" ");
    printf(": ");

    /* アスキー文字表示 */
    for (j = i; j < i + 16 && j < len; j++) {
      if ((buff[j] >= 0x20) && (buff[j] <= 0x7e))
        putchar(buff[j]);
      else
        printf(".");
    }

    printf("¥n");
  }
  fflush(stdout);
}

/*
 * int open_bpf(char *ifname);
 * 機能
 *     BPFをオープンする
 * 引き数
 *     char *ifname;  インタフェース名
 * 戻り値
 *     int            ファイルディスクリプタ
 */
int open_bpf(char *ifname)
{
  char buf[256];    /* 文字列格納用 */
  int bpfd;         /* ファイルディスクリプタ */
  struct ifreq ifr; /* インタフェース属性構造体 */
  int i;            /* ループ変数 */

  /* BPFデバイスファイルのオープン */
  for (i = 0; i < 4; i++) {
    sprintf(buf, "/dev/bpf%d", i);
    if ((bpfd = open(buf, O_RDWR, 0)) > 0)
```

```
      goto bpf_ok;
  }
  fprintf(stderr, "cannot open BPF¥n");
  return -1;

bpf_ok :
  /* インタフェース名の設定 */
  strcpy(ifr.ifr_name, ifname);
  if (ioctl(bpfd, BIOCSETIF, &ifr) < 0) {
    sprintf(buf, "ioctl(BIOCSETIF, '%s')", ifname);
    perror(buf);
    return -1;
  }
  fprintf(stderr, "BPF read from '%s' (%s)¥n", ifr.ifr_name, buf);

  /* promiscuousモード */
  if (ioctl(bpfd, BIOCPROMISC, NULL) < 0) {
    perror("ioctl(BIOCPROMISC)");
    return -1;
  }

  /* 即時モード */
  i = 1;
  if (ioctl(bpfd, BIOCIMMEDIATE, &i) < 0) {
    perror("ioctl(BIOCIMMEDIATE)");
    return -1;
  }

  return bpfd;
}
```

```
/*****************************************
    Grouped Secure Communication for IP
                gscip.h
*****************************************/

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>



/*
 * Structure of a process information data.
 */
typedef struct proc_info_data {
  u_long saddr ;    /* Src IP Address    */
  u_long daddr ;    /* Dest IP Address   */
#if BYTE_ORDER == LITTLE_ENDIAN
  u_char proc_b:4,  /* Process Info B    */
    proc_a:4 ;      /* Process Info A    */
#endif
#if BYTE_ORDER == BIG_ENDIAN
  u_char proc_a:4,  /* Process Info A    */
    proc_b:4 ;      /* Process Info B    */
#endif
  u_char proc_c ;   /* Process Info C    */
  u_short g_no ;    /* Group Key Number  */
  u_short key_ver ; /* Group Key Version */
  u_char timer ;    /* Timer             */
} PID ;

/*
 * Definitions for process information.
 */
#define PROC_INI 1
#define PROC_MID 2
#define PROC_END 3

#define PROC_ENC 1
#define PROC_DEC 2
#define PROC_FWD 3
#define PROC_DSC 4
#define PROC_MAK 5

/* GSCIP Package(temporary) for EEA/N */
int gscip_een(
  int    mode,     /* 1:Rcv, 0:Snd */
  struct ip  *ip,  /* IP Header */
  struct mbuf  *m  /* mbuf */
  ) ;

int isdprp(
  struct icmp  *icp  /* ICMP Header */
  ) ;
```

```
/*****************************************
      Detection of Island Hop by Telnet
                  ih_dtct.c
*****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include "gscip.h"
#include "ih_dtct.h"

#define _IHDDEBUG_ 3


extern SP *sptop ;
extern SP *spend ;
extern IHD *ihdtop ;
extern int timer ;
extern int telnets ;
extern int discards ;

/*
 * Island Hop Detection function.
 */
int ih_dtct(int mode, PID *pid, struct ip *ip, struct tcphdr *th, u_char
*data)
{
  struct timeval ctm ;
  SP *sp = sptop ;

  /* Judgmentation whether TCP protocol is Telnet */
  isTelnet(th) ;    /* If a packet isn't Telnet, processing will be
ended.(return:0) */

  telnets++ ;
  /* Recording called time */
  reccalledtm(&ctm) ;

  /* Judgmentation of called mode */
  switch(mode)
  {
  case 0 :
    goto SND ;
  case 1 :
    goto RCV ;
  default :
    return -1 ;
  }


SND :

#if _IHDDEBUG_ == 1
#endif

  /* Checking whether timer is on. */
  if(!timer_chk())
    return 2 ;

  /* Checking Stored Packet data */
  while(sp != NULL)
  {
    /* Checking whether Stored Packet data is alive. */
    if(chk_sptl(sp, &ctm))
    {
    /* Comparison Telnet data(To be next if these datas agree) */
      if(*(data) == sp->data)
      {
    /* Comparison received packet's source IP address and send packet's dest
IP address(It is Island Hop if these addresses agree) */
        if(ip->ip_src.s_addr == sp->daddr)
        {
    /* Comparison group number(It is unjust Island Hop if these numbers
disagree) */
          if(pid->g_no != sp->group)
          {
            ihdtop = (struct ihd_list *)calloc(1, sizeof(struct
ihd_list)) ;
            add_ihd_list(sp, ip, th, &(pid->g_no), &ctm) ;
            print_ih(sp, ihdtop, data) ;
            free(ihdtop) ;
            return 3 ;
          }
        }
      }
    }
    else
    {
      sptop = sp->n ;
      free(sp) ;
      sp = sptop ;
      discards++ ;
      continue ;
    }
    sp = sp->n ;
  }

  return 2 ;    /* Processing is end(return:2) */


RCV :

  /* Storing received packet. */
  store_rcv_pkt(ip, th, &(pid->g_no), data, &ctm) ;
  /* Starting timer. */
  timer_on() ;

  return 1 ;    /* Processing is end(return:1) */
}

/*
 * Printing Island Hop situation.
 */
void print_ih(SP *sp, IHD *ihd, u_char *data)
{
  struct tm *tmp = NULL ;

printf("¥n+-----------------------+-----------------------+----------
----------------+¥n") ;
  printf("|        Attacker         |        Foot Hold        |
Target          |¥n") ;

printf("+-----------------------+-----------------------+----------
----------------+¥n") ;
  printf("|%17s:%-7d", inet_ntoa(*(struct in_addr *)&(ihd->aaddr)),
ntohs(ihd->aport)) ;
  printf("|%17s:%-7d", inet_ntoa(*(struct in_addr *)&(ihd->fhaddr)),
ntohs(ihd->fhport)) ;
  printf("|    %-21s|¥n", inet_ntoa(*(struct in_addr *)&(ihd->taddr))) ;

printf("+-----------------------+------+----------+------+----------
----------------+¥n") ;
  tmp = localtime((time_t *)&(sp->rtm.tv_sec)) ;
  printf("|   %04d/%02d/%02d %02d:%02d:%02d %06d   |", tmp->tm_year+1900,
tmp->tm_mon+1, tmp->tm_mday, tmp->tm_hour, tmp->tm_min, tmp->tm_sec,
sp->rtm.tv_usec) ;
  printf("  %07d  |", (ihd->ltm.tv_sec-sp->rtm.tv_sec)*1000000 +
(ihd->ltm.tv_usec-sp->rtm.tv_usec)) ;
```

```
  tmp = localtime((time_t *)&(ihd->ltm.tv_sec)) ;
  printf("   %04d/%02d/%02d %02d:%02d:%02d %06d    |¥n", tmp->tm_year+1900,
tmp->tm_mon+1, tmp->tm_mday, tmp->tm_hour, tmp->tm_min, tmp->tm_sec,
ihd->ltm.tv_usec) ;

printf("+-------------------------------+----------+-----------------
--------------+¥n") ;
  printf("|                                      0x%02x
|¥n", *data) ;

printf("+-------------------------------------------------------------
--------------+¥n") ;
}


/***********************************************
  Receive Mode Process Functions
***********************************************/

/*
 * Storing received packet.
 */
int store_rcv_pkt(struct ip *ip, struct tcphdr *th, u_short *group, u_char
*data, struct timeval *tv)
{
  SP *sp ;

  sp = (struct str_pkt *)calloc(1, sizeof(struct str_pkt)) ;

  sp->saddr = ip->ip_src.s_addr ;
  sp->daddr = ip->ip_dst.s_addr ;
  sp->sport = th->th_sport ;
  sp->group = *group ;
  sp->data = *data ;
  memcpy(&sp->rtm, tv, sizeof(struct timeval)) ;
  sp->n = NULL ;

  if(sptop == NULL)
  {
    sptop = spend = sp ;
    printf("First rcv.¥n") ;
  }
  else
  {
    spend->n = sp ;
    spend = sp ;
  }

  return 1 ;
} ;

/*
 * Starting timer.
 */
int timer_on(void)
{
  timer = 1 ;
  return 1 ;
}


/***********************************************
  Send Mode Process Functions
***********************************************/

/*
 * Checking whether timer is on.
 */
int timer_chk(void)
{
  return timer ;
}

/*
 * Checking whether Stored Packet data is alive.
 */
int chk_sptTl(SP *sp, struct timeval *tv)
{
  int diftm ;
  diftm = (tv->tv_sec-sp->rtm.tv_sec)*1000000 +
(tv->tv_usec-sp->rtm.tv_usec) ;
  if(diftm > SPTTL)
    return 0 ;
  return 1 ;
}

/*
 * Add new Island Hop Detection list.
 */
int add_ihd_list(SP *sp, struct ip *ip, struct tcphdr *th, u_short *group,
struct timeval *tv)
{
  ihdtop->aaddr = sp->saddr ;
  ihdtop->fhaddr = sp->daddr ;
  ihdtop->taddr = ip->ip_dst.s_addr ;
  ihdtop->aport = sp->sport ;
  ihdtop->fhport = th->th_sport ;
  ihdtop->agroup = sp->group ;
  ihdtop->tgroup = *group ;
  ihdtop->dtctnum = 1 ;
  memcpy(&ihdtop->ftm, tv, sizeof(struct timeval)) ;
  memcpy(&ihdtop->ltm, tv, sizeof(struct timeval)) ;
  ihdtop->n = NULL ;

  return 1 ;
}
```

```
/*****************************************
      Detection of Island Hop by Telnet
                  ih_dtct.h
*****************************************/


/*
 * The number of times of detection
 * untill it generates an alert.
 */
#define ALERT 6

/*
 * Time to live for Stored Packet data.(usec)
 */
#define SPTTL 10000 /* 10ms */




/*
 * Structure of a Stored Packet data.
 */
typedef struct str_pkt {
  u_long saddr ;        /* Src IP address  */
  u_long daddr ;        /* Dst IP address  */
  u_short sport ;       /* Src port number */
  u_short group ;       /* Group number    */
  u_char data ;         /* Telnet data     */
  struct timeval rtm ;  /* Received time   */
  struct str_pkt *n ;   /* Next packet     */
} SP ;

/*
 * Structure of an Island Hop Detection list.
 */
typedef struct ihd_list {
  u_long aaddr ;        /* Attacker's IP address   */
  u_long fhaddr ;       /* Foot Hold's IP address  */
  u_long taddr ;        /* Target's IP address     */
  u_short aport ;       /* Attacker's port number  */
  u_short fhport ;      /* Foot Hold's port number */
  u_short agroup ;      /* Attacker's group number */
  u_short tgroup ;      /* Target's group number   */
  int dtctnum ;         /* Number of detection     */
  struct timeval ftm ;  /* First detection time    */
  struct timeval ltm ;  /* Last detection time     */
  struct ihd_list *n ;  /* Next list               */
} IHD ;


/*
 * Island Hop Detection function.
 */
int ih_dtct(
  int mode,             /* 1:Rcv, 0:Snd             */
  PID *pid,             /* Process information data */
  struct ip *ip,        /* IP header                */
  struct tcphdr *th,    /* TCP header               */
  u_char *data          /* Telnet data              */
  ) ;

/*
 * Judgmentation whether it is Telnet (Macro function)
 * struct tcphdr *th (TCP header)
 */
#define isTelnet(th)
if(ntohs((th)->th_dport)!=23||(u_char)(*data)==0xff){return(0);}

/*
 * Recording called time (Macro function)
 * struct timeval *calledtm (Called time)
 */
#define reccalledtm(calledtm) gettimeofday((calledtm),NULL)

/*
 * Printing Island Hop situation.
 */
void print_ih(
  SP *sp,        /* Stored Packet data        */
  IHD *ihd,      /* Island Hop Detection data */
  u_char *data   /* Telnet data               */
  ) ;


/*****************************************************
          Receive Mode Process Functions
*****************************************************/

/*
 * Storing received packet.
 */
int store_rcv_pkt(
  struct ip *ip,       /* IP header     */
  struct tcphdr *th,   /* TCP header    */
  u_short *group,      /* Group number  */
  u_char *data,        /* Telnet data   */
  struct timeval *tv   /* Received time */
  ) ;

/*
 * Starting timer.
 */
int timer_on(
  void  /* No param */
  ) ;


/*****************************************************
             Send Mode Process Functions
*****************************************************/

/*
 * Checking whether timer is on.
 */
int timer_chk(
  void  /* No param */
  ) ;

/*
 * Checking whether Stored Packet data is alive.
 */
int chk_sptl(
  SP *sp,              /* Stored Packet data */
  struct timeval *tv   /* Called time        */
  ) ;

/*
 * Add new Island Hop Detection list.
 */
int add_ihd_list(
  SP *sp,              /* Stored Packet data */
  struct ip *ip,       /* IP header          */
  struct tcphdr *th,   /* TCP header         */
  u_short *group,      /* Group number       */
  struct timeval *tv   /* Detected time      */
  ) ;
```