目次

概要

攻撃者が不正アクセスを行う場合，攻撃者の身元を隠すために複数の踏み台ホストを経由する踏み台攻撃を行っている場合が多い．このような攻撃を検出する方式として，これまで踏み台ホストをはさんだリモートログインストロークの時間的な相関関係を見るという方法があった．しかし，この方法では踏み台ホストへのアクセスと踏み台ホストから被害ホストへのアクセスが共にリモートログインである場合に限定されることになり，踏み台ホストから被害ホストへのアクセスが FTP のような場合は検出できない．また，検出にある程度の時間が必要である．本稿では，踏み台攻撃が行われるとき，踏み台ホストに対するリモートログイン操作の終了と同期して踏み台ホストから被害ホストに対して TCP コネクションの確立要求が必ず送信されることに着目し，踏み台ホスト宛のリモートログインパケットと，踏み台ホストから送信される TCP の SYN パケットを監視することによって，リアルタイムに踏み台攻撃を検出するコネクション検出方式を提案する．提案方式をネットワークモニタ装置に実装して動作検証を行った結果，踏み台攻撃を確実に検出できることを確認した．

第1章 はじめに

企業ネットワークに対する不正アクセスはますます増加する傾向にある．外部からの不正アクセスに対しては，ファイアウォールや IDS などのセキュリティ機器の導入や，常に最新のセキュリティパッチを適用したり，不要なサービスを起動しないなど，ホストの要塞化によるセキュリティ対策が取られている．しかし，不正に入手したアカウント/パスワードを用いて被害ホストにアクセスするような攻撃を防ぐことは困難である．特にこのような攻撃を，踏み台ホストを介して実行されると（以下踏み台攻撃），管理者は攻撃者の身元を特定することすらできない．踏み台攻撃は，攻撃者から踏み台ホストに対しては必ずリモートログインでアクセスする．踏み台ホストから被害ホストへは，更にリモートログインでアクセスする場合が多いが，それ以外のあらゆるアクセス方法がありうる．管理者は管理目的のためにサーバ類にはリモートログインサービスを提供している場合が多いため，安易にサービスを停止させることはできない．しかも単なるリモートアクセス自体は正常な行為であり，上記のような既存のセキュリティ対策だけで踏み台攻撃を検出・防御することは難しいのが現状である．このようなことから管理者の意図に反して特定のホストが踏み台にされていることを検出することができれば有効である．

踏み台攻撃の検出に注力した研究には以下のようなものがある．いずれも踏み台ホストから被害ホストまでのアクセスがリモートログインである場合に限定されている．検出の手がかりにするものによって，コンテンツベース方式とタイミングベース方式の 2 種類に大別することができる．コンテンツベース方式には，リモートログインパケットのデータを直接比較する手法[1]や，リモートログインパケットに透かしを埋め込み，その透かしを比較する手法[2]があり，いずれも踏み台ホストをはさんだ 2 つのリモートログインパケットの内容が一致していることを検出する．この方式は検出にパケットの内容やサイズを手がかりにしているため，暗号化されたリモートログインに対応できないという課題がある．一方，タイミングベース方式[3]～[10]はリモートログインのキー入力ストロークには特徴があることに着目しており，踏み台ホストをはさんだ 2 つのリモートログインストロークに時間的な相関関係があることを検出する．検出にはパケットの到着時間情報を手がかりにしており，パケットの中身やサイズに依存しないため，暗号化されたリモートログインにも対応可能である．現在はタイミングベース方式を採用した研究が主流であり，検出精度を高めるために様々な提案がなされている．しかし，これらの手法はいずれもリモートログインが連鎖状態にあることを検出するもので，被害ホストへの最終アクセスが FTP のようなリモートログイン

以外の場合は検出することができない．また，タイミングベース方式は，相関関係を見る関係上ある程度の検出時間を必要とするという課題がある．

本研究，踏み台攻撃検出の一手法として，踏み台ホストに対するリモートログイン操作の終了と同期して，踏み台ホストから被害ホストに対して TCP コネクション確立要求が送信されることを検出するコネクション検出方式を提案する．この手法は，リモートログインパケットによる一連のコマンド情報を踏み台ホストが受信した直後に，踏み台ホストから被害者ホストに対して TCP コネクションの確立を要求するパケットが送信されることに着目したものである．提案方式を用いると，リモートログインが暗号化されていた場合や，被害者ホストへのアクセスがリモートログイン以外の様々なケースにおいても確実に踏み台攻撃の検出が可能になる．また，提案方式は特定の通信パケットをリアルタイム監視するため，踏み台攻撃が発生した直後にそのことを検出することができる．ネットワーク型監視ホストとして提案方式を実装し，動作確認を行った結果，踏み台攻撃を確実に検出できることを確認した．ネットワーク管理者はサーバが踏み台にされていることを早期に知ることができるため，提案方式は不正アクセスを防止する有効な手段となり得る．

以降，2 章で踏み台攻撃の定義と関連研究の紹介をし，3 章で提案方式の概要を説明する．4 章では実装について述べ，5 章で評価を行う．そして最後に 6 章でまとめる．

第2章 従来の踏み台攻撃検出方式

第2.1節踏み台攻撃の定義

本論文において対象とする踏み台攻撃モデルを**図 1** に示す．踏み台攻撃モデルの構成要素として，Attacker，Foothold，Target を定義する．Attacker（攻撃ホスト）は攻撃者が直接操作するホストである．Foothold（踏み台ホスト）は Attacker がリモートログインプロトコルを用いてアクセスするホストである．Target（被害ホスト）は Attacker が Foothold を介してアクセスするホストである．Attacker は何らかの方法を用いて，あらかじめ Foothold および Target のアカウントとパスワードを入手しているものとする．複数の Foothold を経由した攻撃もありうるが，本論文では Foothold が 1 台の場合について検討する．Foothold から Target までのアクセス方法は必ずしもリモートログインの必要はなく，FTP のようなプロトコルを用いることも可能である．このように，Foothold を経由して Target にアクセスする攻撃を全て踏み台攻撃と呼ぶ．Attacker から Foothold へのアクセスのフローを Flow X，Foothold から

Target へのアクセスのフローを Flow Y と呼ぶ．リモートログインプロトコルとしては，SSH，Telnet，Rlogin がある．

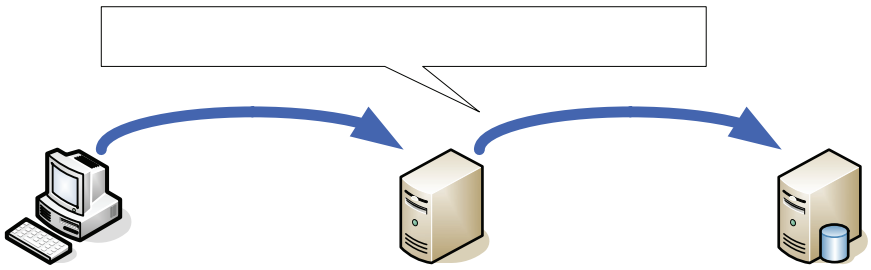

図 1　踏み台攻撃モデル

Fig. 1　A stepping-stone attack model.

Access to a Target thr

Flow X

Attacker

Foot

第2.2節従来研究とその課題

従来の踏み台攻撃検出方式は，Flow X および Flow Y がいずれもリモートログインである場合にのみ適用が可能である．初期の方式として，Flow X と Flow Y を流れるリモートログインパケットの内容を手がかりにするコンテンツベース方式が提案されている．

文献[1]では，踏み台攻撃が行われるとき，パケットのデータ内容が同一であるリモートログインパケットが Flow X と Flow Y を流れることに着目しており，Flow X と Flow Y のリモートログインパケットの内容を比較して，一致していることを検出する．

文献[2]では，Target に対してリモートログインによる不正侵入が行われた場合，Target から Attacker へと戻るリモートログインパケットに透かしを埋め込んでおき，Flow X と Flow Y で同じ透かしが埋め込まれていることを検出する．Foothold が踏み台にされた時点で踏み台攻撃を検出するのではなく，Target に対して不正侵入が行われたことを検出してからでなければ検出できない．

コンテンツベース方式はリモートログインパケットの内容を見る必要があるため，パケットが暗号化されると適用できない．このことから近年ではタイミングベース方式による検出が主流となっている．タイミングベース方式では，ユーザがリモートログインを利用するときのキー入力ストロークには特徴があることに着目しており，Flow X と Flow Y のリモートログインストロークの間に時間的な相関関係があることを検出する．例えば**図 2** のように，ある期間内に Foothold 前後で Flow X と Flow Y が発生したとき，各フローの

パケット発生タイミングを手がかりにし，2 つのフローに時間的な相関関係があるかどうかを発見する．この方式はパケットの内容を識別する必要がないため，SSH など暗号化されたリモートログインにも対応可能である．

文献[3]の ON/OFF ベースアプローチでは，トラフィックが発生していない期間，すなわちフローの各パケットの送信間隔時間を手がかりにして，異なるフローの相関関係を求めている．2 つのフローの各パケット送信間隔時間が同じであれば，踏み台攻撃であると判断する．しかし，あらゆるケースで検出率をあげるのは難しく，ネットワークトラヒックなどによって検出に要する適切なパラメータ値が異なるという課題がある．

この ON/OFF ベースアプローチ以外でも，検出精度やリアルタイム性の向上を図った手法，踏み台の連鎖がループ状になった踏み台攻撃の検出手法，踏み台の連鎖の長さを推測する手法など，踏み台攻撃検出に注力した研究が行われている[4]～[18].

しかし，従来方式は Target へのアクセスがリモートログインである場合に限定されており，Target に対してそれ以外のプロトコルを利用するような踏み台攻撃を検出することはできない．また，Flow X と Flow Y が所定の時間を経過しないと相関関係があると判断することができない．攻撃者はログインストロークの間にディレイや余計なパケットを挿入するなどにより，相関関係の検出を困難にすることができることが指摘されている．

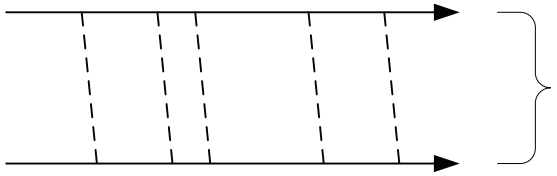

図 2　相関関係の発見

Fig. 2　Discovery of the correlation between two flows.

第3章 コネクション検出方式の提案

前述のように従来の踏み台攻撃検出手法は，Flow X と Flow Y がともにリモートログインを用いた場合にのみ適用可能である. しかし, 実際には Target に対するアクセスがリモートログインだけとは限らない. 例えば FTP によりファイルをダウンロード/アップロードすることも考えられる．ここで，Attacker が Foothold へとリモートログインした後，そこから Target に対して

あらゆる TCP 通信でアクセスする可能性があることを想定すると，踏み台攻撃発生時には必ず Foothold から TCP コネクションの確立が必要である．また，その直前には TCP コネクション確立要求送信のトリガとなるコマンドを乗せたリモートログインパケットを Foothold が受信しているはずである．本論文では，Foothold へのリモートログインパケットが送信されたのち，一定時間内に Foothold から TCP コネクションが確立されることを検出することにより踏み台攻撃を検出するコネクション検出方式を提案する．コネクション検出方式では，Foothold が存在するネットワークのゲートウェイの直下に通信を監視するホスト（以降，Detector と呼ぶ）を設置する（**図 3**）．Detector を設置したネットワークに存在するホストはどれが Foothold であってもかまわない．

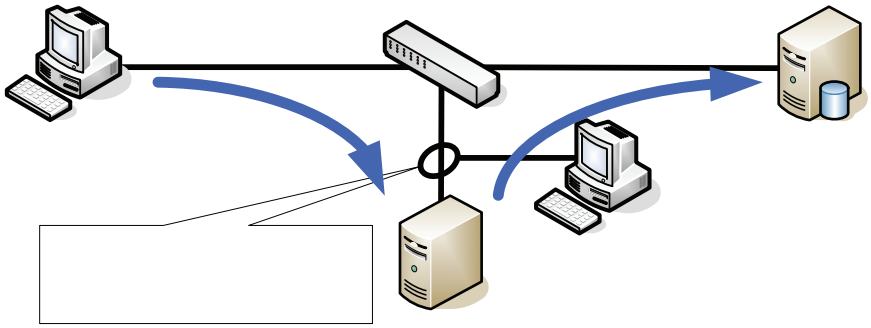

図 3　Detector の設置箇所

Fig. 3　Installation place of Detector.

図 4 にコネクション検出方式の原理を示す．まず Attacker が Foothold へリモートログインするために TCP コネクションの確立を行う．次に，Attacker は Foothold に対して Target へのアクセスを行うためのコマンドを投入する．コマンドの最後の文字が入力されると，Foothold はコマンドを解読して，Target に対して TCP コネクションの確立を行う．Detector はその間リモートログインパケットの監視を行いつつ，他のホストへ新たな TCP コネクションが確立されようとするのを監視する．リモートログイン通信パケットの受信とコネクション確立要求の送信との間の時間が一定時間内であれば，Detector はこの状況を踏み台攻撃と判断する．

Gate

Attacker

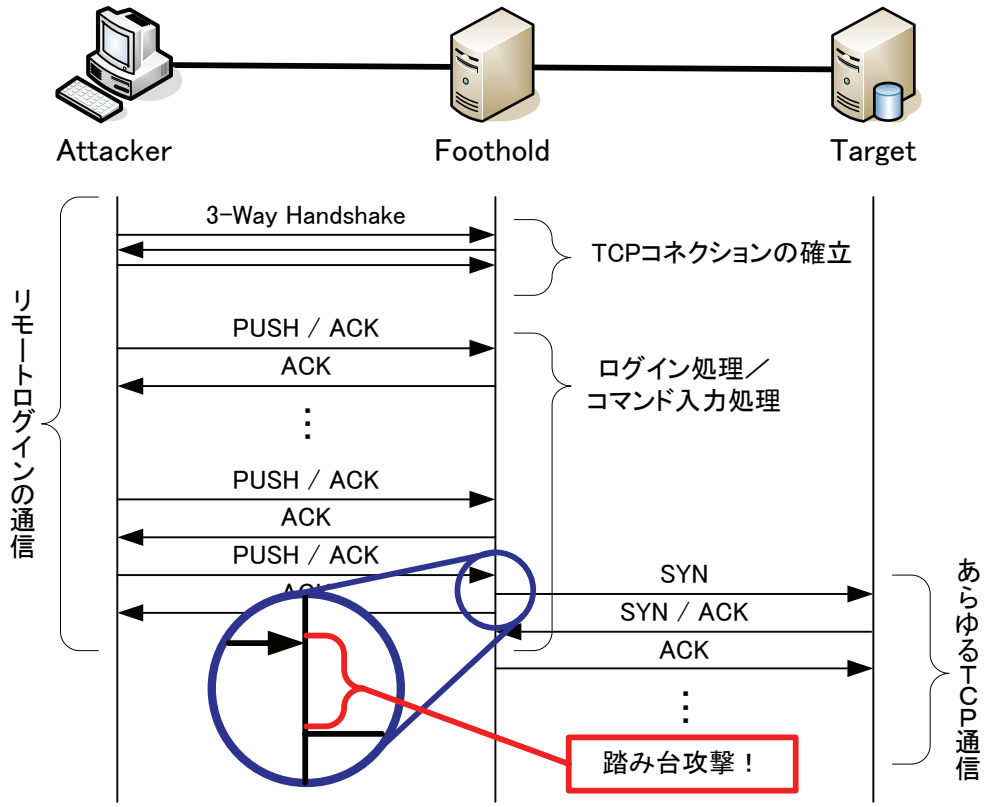

図 4　コネクション検出方式の原理

Fig. 4　The principle of Connection Detection Method.

以上の原理に従い，コネクション検出方式では Detector 上でキャプチャしたパケットに対して以下の処理を行う．

(1) リモートログイン通信パケットの監視

TCP パケットのうち，SSH，Telnet，Rlogin などのリモートログインの通信パケットでかつ PUSH フラグのセットされたパケットを検出する．検出する度にパケットの送信元・宛先 IP アドレスと送信元・宛先ポート番号，検出した時刻をリモートログイン受信記録として保存していく．PUSH フラグを見る理由は，リモートログインの通信パケットはローカルホストで入力された１文字分（あるいは 1 行分）のデータであり，受信したらすぐにアプリケーションに渡すため，PUSH フラグが必ずセットされているからである．

(2) TCP コネクション確立要求パケットの監視

TCP パケットのうち，あらゆる TCP サービスの SYN パケットを検出する．SYN パケット検出時にリモートログイン受信記録を参照し，リモートログイン通信パケットの宛先 IP アドレスと SYN パケットの送信元 IP アドレスが一致するものを検索する．一致するものがあれば，リモートログイン通信パケ

ットの検出時刻と SYN パケット検出時刻とを比較し，一定時間内に SYN パケットの送信が行われていれば，Foothold が踏み台になっているものと判断する．SYN フラグを見る理由は，Attacker からのコマンドを受けて Foothold が Target に対して TCP サービスを起動する場合, 必ず初めにコネクション確立要求を送信するので，これを検出すればよいからである．以降，リモートログインの PUSH パケットを検出してから TCP の SYN パケットを検出するまでの時間を踏み台検出時間あるいは単に検出時間と呼ぶ．また，踏み台検出時間になっていると判断するまでの最大待ち時間のことを監視時間と呼ぶ．検出時間が監視時間を超えている場合は，踏み台攻撃ではないと判断する．

コネクション検出方式ではリモートログイン通信パケットのデータ内容を参照する必要が無いので，リモートアクセスが暗号化されていれもかまわない．また，Target 宛送信パケットの SYN フラグだけを検出すればよいため，Foothold から Target へのあらゆる TCP 通信を検出対象とすることができ，かつ踏み台攻撃が発生したことを即座に検出できる．

第4章 実装

コネクション検出方式を実現するためには，ネットワークを流れる全通信パケットの監視を行う必要がある．パケットの監視は libpcap でキャプチャすることとし，FreeBSD 上で動作するアプリケーションとして本方式を実現した．コネクション検出方式による踏み台攻撃検出プログラムの試作モジュール構成図を**図 5** に示す．

ネットワークインタフェースが受信した全パケットのうち，TCP パケットのみを検出プログラムに渡すように libpcap にフィルタを掛ける．検出プログラムは，パケットがリモートログイン通信であれば，受信記録追加モジュールにパケットを渡す．TCP コネクション確立要求であれば，受信記録検索モジュールにパケットを渡す．その他の場合は無視し，次のパケットの受信を待つ．

受信記録追加モジュールは, リモートログイン通信パケットの送信元 IP アドレス，宛先 IP アドレス，送信元ポート番号，宛先ポート番号，受信時刻から成るレコードを，リングバッファにリモートログイン受信記録として保存する．

受信記録検索モジュールは，リモートログイン受信記録から，TCP コネクション確立要求パケットの送信元 IP アドレスと一致する宛先 IP アドレスを持つレコードを検索する．IP アドレスが一致するものが検出され，さらにリモートログイン通信パケットの受信時刻が TCP コネクション確立要求パケ

ットの送信時刻からさかのぼって監視時間内にあれば，踏み台攻撃が発生したことを示すメッセージを出力する．

本方式はパケットの IP ヘッダと TCP ヘッダの一部，およびリモートログイン受信記録を参照するだけでよいため，処理内容は極めて単純である．

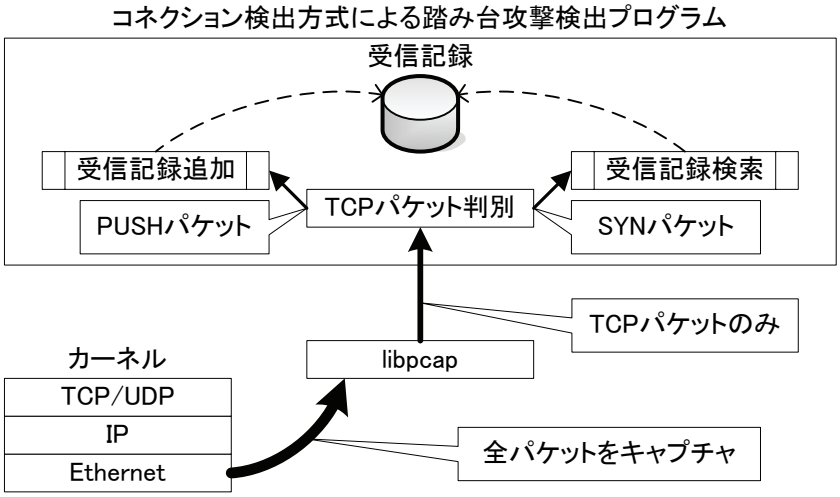

図 5　試作モジュール構成図

Fig. 5　Trial module construction.

第5章　評価

本章では，様々な評価実験を行うことで踏み台検出時間のばらつきを明らかにした．評価実験 1 として，Foothold に背景負荷を与えた場合の踏み台検出時間の測定を行い，評価実験 2 として，実際に運用されているサーバを Foothold とした場合の踏み台検出時間の測定を行った．そしてこれらの評価実験で明らかになったことを踏まえ，今後の課題について述べる．

第5.1節評価実験 1 の概要

評価実験 1 では，Foothold に背景負荷を与えた場合の踏み台検出時間を測定した．与える背景負荷の度合いを変化させていき，踏み台検出時間がどのように変動するのかを明らかにした．評価実験 1 の評価環境を**図 6** に示す．踏み台攻撃モデルを構成する Attacker，Foothold，Target に加えて，コネクション検出方式による踏み台攻撃検出プログラムを実装した Detector を用意した．Foothold と Detector は 100BASE のリピータハブで接続し，Detector が Foothold に流れる Flow A と Flow B の両方の通信を監視できるようにした．それ以外の機器はスイッチングハブを用いて接続した．Foothold では SSH,

Telnet，Rlogin，FTP サービスを，Target では Telnet，FTP サービスを起動させた．Detector，Foothold のマシンスペックは**表 1** の通りである．また，この評価環境において Foothold に背景負荷を与えない状態で実際に踏み台攻撃を行い，Detector で検出できることを確認した．Attacker から Foothold へのリモートログインには SSH，Telnet，Rlogin を，Foothold から Target への TCP 通信には Telnet，FTP を用い，いずれのリモートログインにおいても踏み台動作を確実に検出できることを確認した．

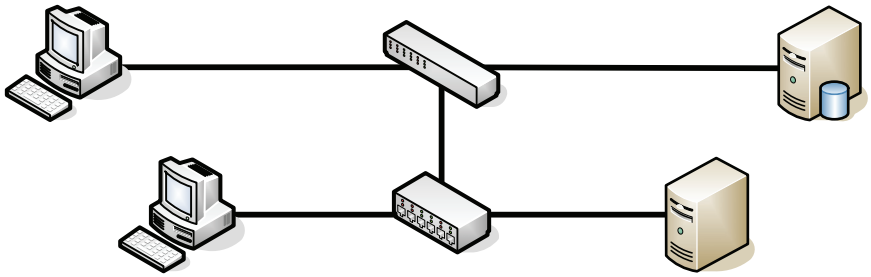

図 6　評価実験 1 の評価環境

Fig. 6　Valuation environment for Exp.1.

表 1　評価実験 1 で用いた Detector，Foothold のマシンスペック

Table 1　Machine spec. of Detector and Foothold for Exp.1.

	Detector	Foothold
CPU	Celeron 1.7GHz	Pentium4 2.4GHz
RAM	256MB	256MB
OS	FreeBSD 5.3-RELEASE	FreeBSD 5.3-RELEASE

第5.2節評価実験 1 の測定結果

踏み台検出時間を Foothold に背景負荷を与えない状態で測定した結果を**表 2** に示す．表 2 において起動時接続先指定とは，接続先を指定するパラメータとして Target を指定して Telnet クライアントまたは FTP クライアントを起動して接続した場合であり，起動後接続先指定とは，Telnet クライアントまたは FTP クライアントを起動した後に内部コマンドを用いて接続先を Target に指定して接続した場合である．Target への接続方法を 2 通り試行した理由

は, 全く同じ TCP のクライアントプログラムを用いても異なる挙動をすることを調査するためであり, 起動時接続先指定は起動後接続先指定に比べてプログラムを起動する処理の分だけオーバヘッドが多い.

表 2　背景負荷を与えない場合の踏み台検出時間測定結果

Table 2　Results of stepping-stone detection time without background load.

		Flow Y			
		Telnet		FTP	
		①	②	①	②
Flow X	SSH	7.73	2.62	3.92	0.39
	Telnet	7.69	2.55	3.81	0.35
	Rlogin	7.61	2.53	3.78	0.33

※表中の①は起動時接続先指定, ②は起動後接続先指定の意味

※値は 10 回試行の平均（ミリ秒）

Flow X に用いたリモートログインの種類によって若干異なるものの, Flow Y が Telnet 起動時接続先指定では 7.5 ミリ秒程度, Telnet 起動後接続先指定では 2.5 ミリ秒程度の間に SYN パケットが送信されていることがわかる. また Flow Y が FTP 起動時接続先指定では 4 ミリ秒程度, FTP 起動後接続先指定では 0.4 ミリ秒程度となり, Flow Y が Telnet である場合より短い結果が得られた.

踏み台検出時間が変動する要因を調査するために, Foothold に対して FTP 接続および HTTP 接続をそれぞれ行い, ファイル転送により Foothold に背景負荷を与えた状態で踏み台検出時間を測定した. 負荷となる FTP および HTTP のファイル転送数を 0 から 10 へ増加させていき, Flow X が Telnet, Flow Y が Telnet と FTP である場合の踏み台検出時間の測定を行った.

図 7 に FTP を背景負荷としてファイル転送数を変えた場合の踏み台検出時間を, **図 8** に HTTP を背景負荷としてファイル転送数を変えた場合の踏み台検出時間を示す. 図の横軸はファイル転送のセッション数を, 縦軸は踏み台検出時間を表している. 測定結果より, ファイル転送のセッション数が増加するにつれて踏み台検出時間も比例に近い関係で増加していることが分かる. 背景負荷が FTP と HTTP のどちらのときであっても踏み台検出時間はほぼ同じであった.

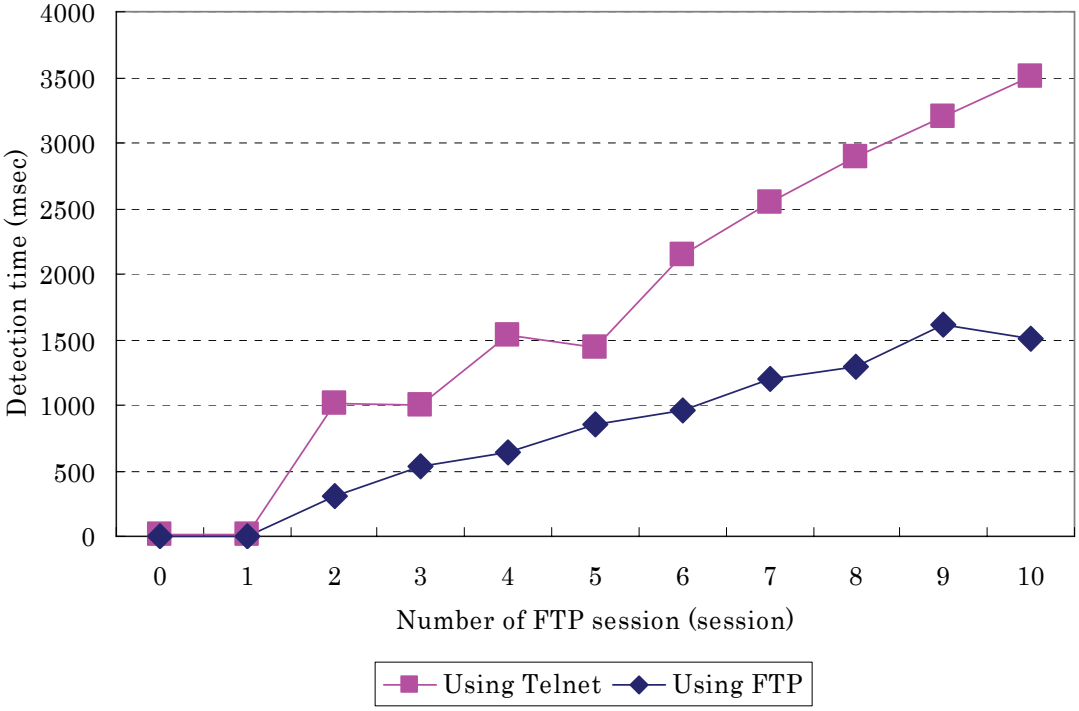

図 7　FTP による背景負荷を与えた場合の検出時間変化

Fig. 7　Change in detection time when background load using FTP is given.

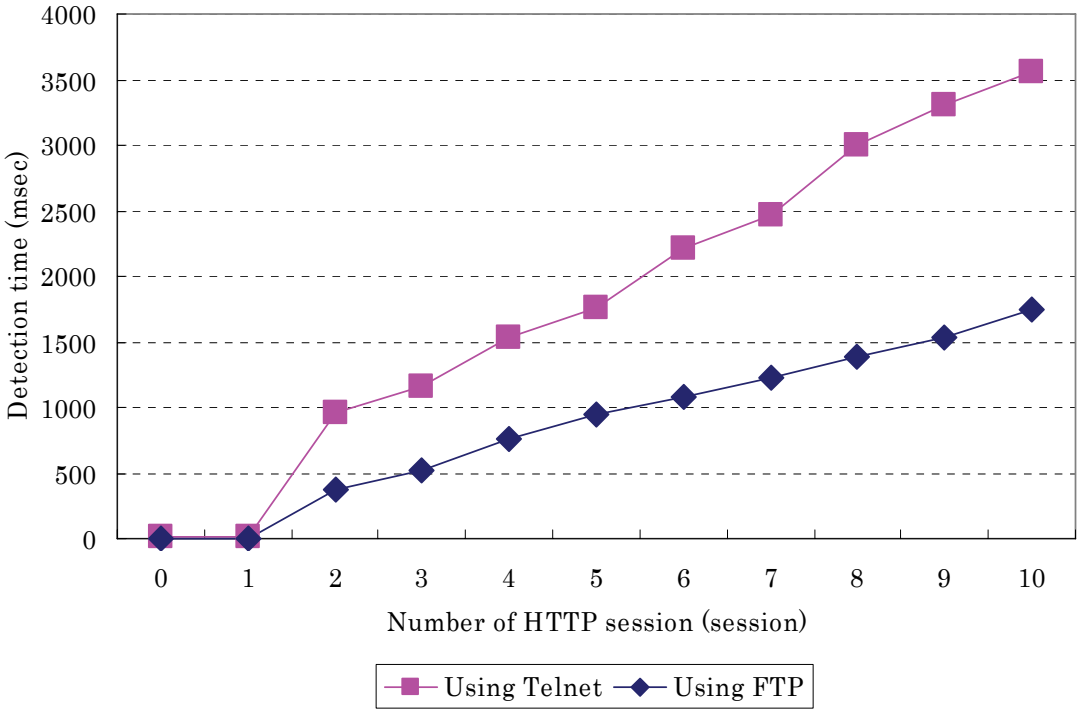

図 8　HTTP による背景負荷を与えた場合の検出時間変化

Fig. 8　Change in detection time when background load using HTTP is given.

踏み台検出時間の変化に影響を与えたものが，CPU の処理負荷なのかネットワークインタフェースの処理負荷なのかを明らかにするために，Foothold に背景負荷を与えたときと同一の条件において，Foothold の CPU 全体の負荷状態，FTP デーモンおよび HTTP デーモンの 1 プロセス当たりが与える CPU 負荷がどの程度になるかを測定した（**図 9**，**図 10**）．CPU 全体の負荷としては，FTP を背景負荷とした場合，セッション数が 2 つ以上になると 85～90%を占めていることが分かる．HTTP を背景負荷とした場合，セッション数が 2 つ以上になっても 25%前後で済んでおり，それ以上には増加していないことが分かる．これに対して，FTP デーモンの 1 プロセス当たりが与える CPU 負荷としては，背景負荷が FTP と HTTP のどちらであってもセッション数が増加するにつれて減少していく．また，背景負荷が FTP の場合に比べて HTTP の場合は CPU 負荷が遥かに小さいことが分かる．図 9，図 10 には各ファイル転送数における FTP デーモンおよび HTTP デーモンが与える CPU 負荷の合計も示した．複数起動している FTP デーモン全体が与える CPU 負荷は 55～65%で推移しているが，HTTP デーモンの場合は合計しても 5%以下である．背景負荷が HTTP の場合の CPU 負荷が 25%前後で済んでいるのは，HTTP デ

ーモンに割り当てられる CPU 処理能力の合計が小さいことが理由として考えられる.

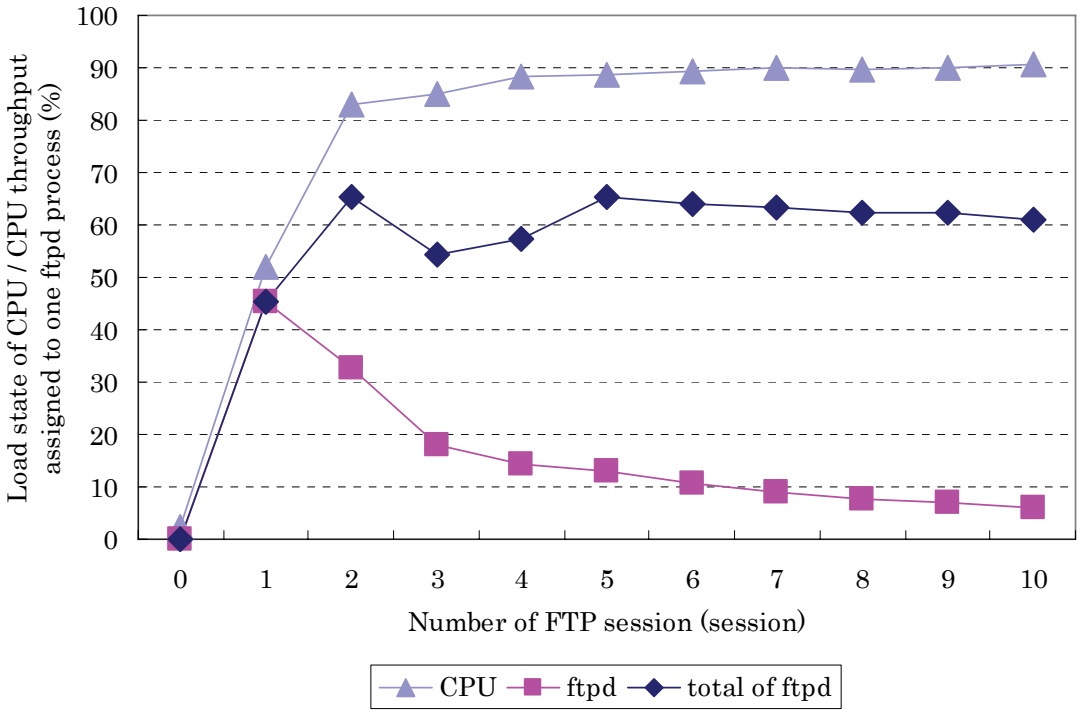

図 9　FTP による背景負荷を与えた場合の Foothold の CPU 状態

Fig. 9　State of Foothold's CPU when background load using FTP is given.

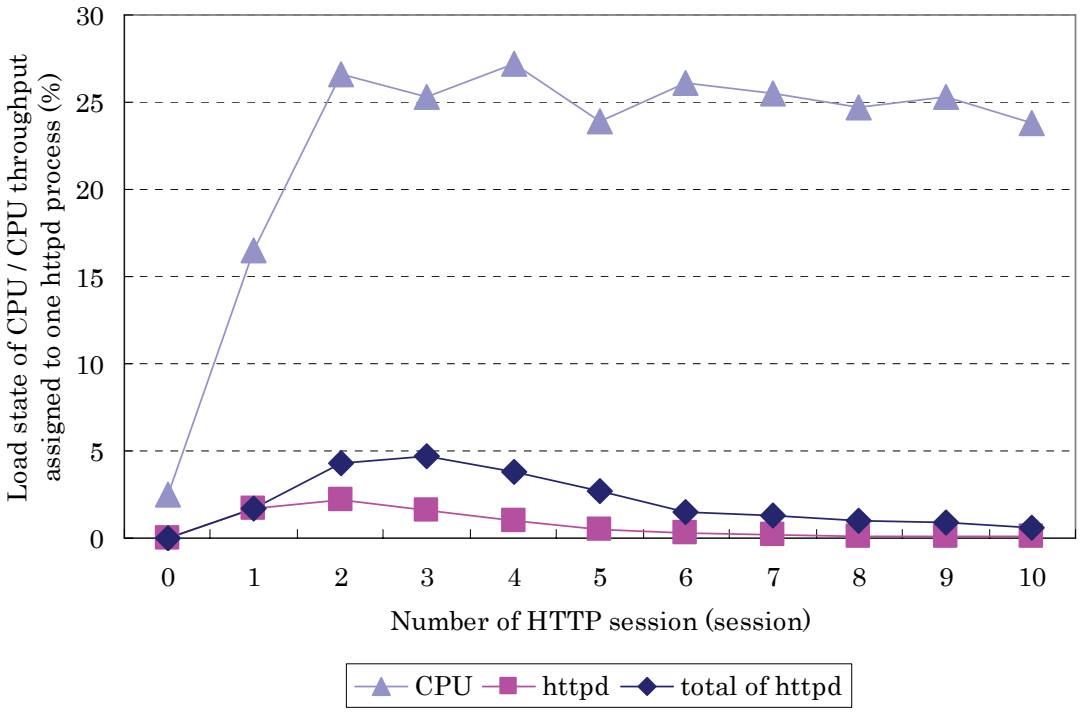

図 10　HTTP による背景負荷を与えた場合の Foothold の CPU 状態

Fig. 10　State of Foothold's CPU when background load using HTTP is given.

以上のことから，ファイル転送数と CPU 負荷との間に相関は見られず，踏み台検出時間は背景負荷のファイル転送のセッション数，すなわちネットワークインタフェース処理負荷に大きく依存していることが分かる．

Foothold に対する通信負荷が大きい状態において，踏み台攻撃と判断するまでの時間の閾値である監視時間が短いと，踏み台攻撃が検出できない可能性がある（フォールスネガティブの増加）．逆に監視時間を長くすれば必ず検出できるが，一定時間内にネットワーク上を流れるリモートログインの PUSH パケットが多くなり，Attacker の候補が増加することによって Attacker を特定しづらくなる可能性がある（フォールスポジティブの増加）．この課題の対策案としては，Foothold に対する通信量を同時に監視しておき，踏み台攻撃と判断するまでの監視時間を動的に設定するという方法が考えられる．また，踏み台検出時間はマシンスペックによっても変化する可能性があり，これらの状況に合わせて監視時間を設定する必要がある．

第5.3節 評価実験 2 の概要

評価実験 2 では,実際に学内で運用されているサーバを Foothold にみたて,どのような踏み台検出時間が得られるかを測定した．評価実験 2 の評価環境を**図 11** に示す．Foothold-1 は外部・内部接続用中継サーバであり，グローバル IP アドレスが割り当てられている．Foothold-2 は学生用メールサーバであり，プライベート IP アドレスが割り当てられている．これらには学内に設置した Detector から学内ネットワークを通じてアクセスした．本来であれば Detector は Foothold-1 および Foothold-2 と同じネットワークに設置するべきであるが，Detector で Attacker と Target を兼用することにより，Foothold-1 および Foothold-2 に流れる Flow A と Flow B の両方の通信を監視できるため,等価の実験結果が得られるものと判断した. Foothold-1, Foothold-2 では Telnet サービスが起動しており，Detector でも Telnet サービスを起動させ，自分自身が Attacker と Target を兼ねる構成とした．これらのマシンスペックは**表 3** の通りである．実験を行った時期は 2005 年 12 月 22 日から 28 日までの 1 週間で，1 時間当たり 5 回の踏み台攻撃を Foothold-1 および Foothold-2 に対して行った．Flow X と Flow Y にはともに Telnet を使用した．

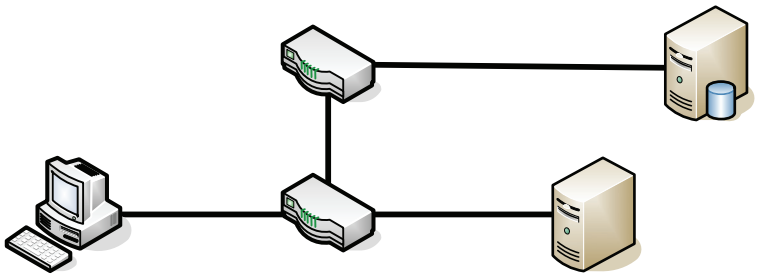

図 11　評価実験 2 の評価環境

Fig. 11　Valuation environment for Exp.2.

表 3　評価実験 2 で用いた Detector，Foothold-1,2 のマシンスペック

Table 3　Machine spec. of Detector, Foothold-1 and Foothold-2 for Exp.2.

	Detector	Foothold-1	Foothold-2
CPU	Celeron 1.7GHz	Xeon 3.06GHz	Xeon 3.06GHz x 2
RAM	512MB	2048MB	2048MB
OS	FreeBSD 5.3-RELEASE	Red Hat Enterprise Linux ES release 3	Red Hat Enterprise Linux ES release 3

Route

第5.4節評価実験 2 の測定結果

Foothold-1 に対する踏み台検出時間の測定結果を**図 12** に，Foothold-2 に対する踏み台検出時間の測定結果を**図 13** に示す．図の横軸は日付を，縦軸は検出時間を表しており，1 日毎に得られた踏み台検出時間の平均値，最大値，最小値の変動を示している．Foothold-1 に対する測定結果から，1 週間通しての最大値は 210 ミリ秒であったが，平均すると 5 ミリ秒程度であり，最小値である 4 ミリ秒に近い値となっていることが分かる．Foothold-2 に対する測定結果においても，最大値は 78 ミリ秒であったが，平均すると 7 ミリ秒程度であり，最小値である 4 ミリ秒に近い値となっている．また，10 ミリ秒以上の突出した検出時間が発生した確率は，Foothold-1 で約 1%，Foothold-2 で約 7%であった．以上のことから，評価実験 1 で行ったような極端に高い負荷が Foothold に与えられていなければ，監視時間を短くすることができると考えられる．

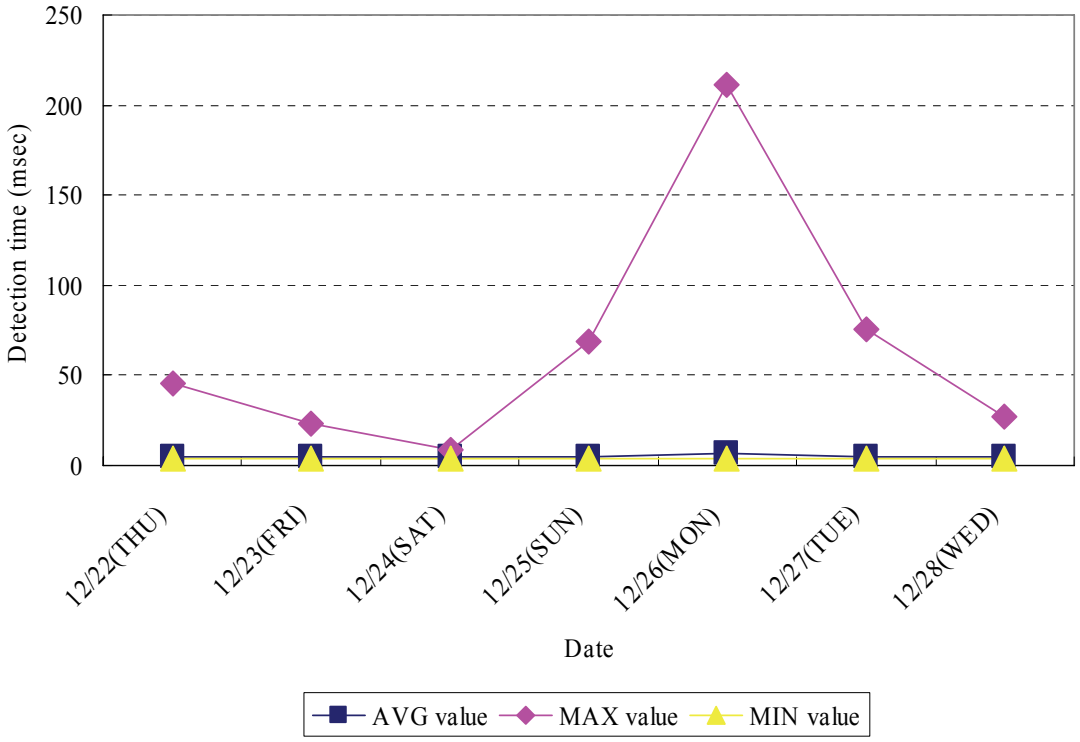

図 12　Foothold-1 に対する踏み台検出時間

Fig. 12　Detection time of Foothold-1.

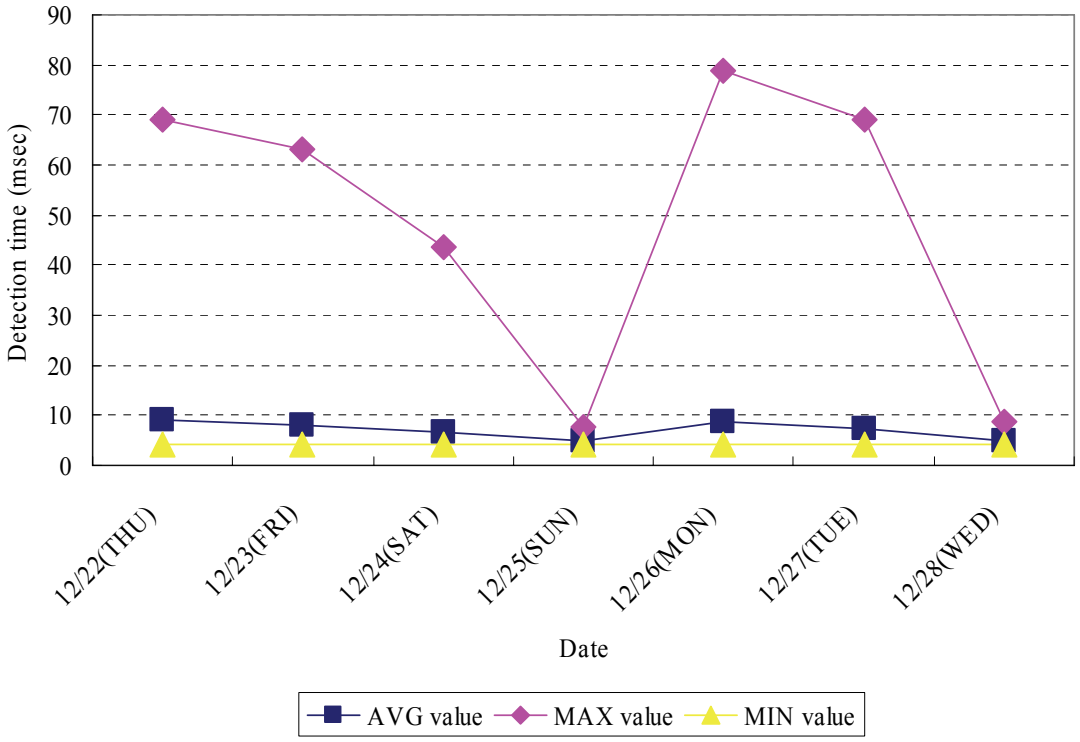

図 13　Foothold-2 に対する踏み台検出時間

Fig. 13　Detection time of Foothold-2.

第5.5節コネクション検出方式の課題

コネクション検出方式の課題としては，監視対象パケットの発生時刻の時間差を手がかりとしているため，大量の監視対象パケットが発生すると Attacker の特定が困難になる可能性がある．そこで，踏み台攻撃検出機能で Attacker を特定せず，疑わしい Attacker のリストを管理者に報告するなど，最終的な判断は人間に任せる必要があると考えられる．また，リアルタイム性を持たせつつ検出精度を向上させる手法についても検討していく必要がある．

なお，本方式では以下のようなケースは踏み台攻撃の検出対象とはならない．即ち，Target に対して，UDP による攻撃を行う場合がありうる．この場合は Target 上で UDP によるサービスを起動していることが条件となるが，重要なサーバ上ではこのようなサービスは起動しない等の処置が必要である．次に，事前に Foothold に bot のようなリモートコントロールプログラムを仕込んでおき，例えば Target に対して時間差をつけて攻撃をしかける方法も可

能である．また，UNIX では指定時間だけコマンドの実行を停止できる Sleep というコマンドがあり，Attacker が Sleep を用いて攻撃に時間差をつける方法もありうる．このような攻撃はリアルタイムで Target の状況を把握できないので攻撃には高度な技術が必要となる．今回の方式ではこのように Foothold 上で大きな時間差のある攻撃を検出することは困難であり，別の手段を適用する必要がある．例えばリモートコントロールプログラムを Foothold に仕込む攻撃であれば，そのようなプログラムを仕込む機能を有したコンピュータウィルスやトロイの木馬プログラムを検出･駆除することで，事前に攻撃を防ぐ等の処置を行う必要がある．

踏み台動作は正規のユーザが正規の手段として行う場合も考えられる．悪意のある攻撃者が行う踏み台攻撃を検出するためには，本方式に加えて踏み台の正常・不正の判断をしなければならない．事前に正常・不正パターンを定義した IP アドレスリストなどを用意することで，踏み台の正常・不正の判断を行う必要があると考えられる．

第6章　まとめ

本稿では，踏み台攻撃発生時に Foothold から Target に対して TCP コネクション確立要求が必ず送信されることに着目し，Attacker から Foothold に対するリモートログイン通信パケットが発生してから一定時間内に TCP コネクション確立要求パケットが発生することを検出するコネクション検出方式を提案した．提案方式では TCP ヘッダのコントロールフラグを参照する方式であるため，Foothold に対するリモートアクセスに用いられるプロトコルはどのようなものであってもよい．また Target に対するアクセスはどのような TCP 通信であってもかまわない．検出方法が TCP コントロールフラグを参照するだけの簡単なアルゴリズムであることから，踏み台攻撃をリアルタイムに検出することができる．

提案方式をネットワーク型機器として実装して動作確認を行い，踏み台攻撃を検出できることを示した．評価実験を行い，踏み台検出時間のデータを示した．そして，Foothold に対する通信量や TCP 通信を行うプログラムの種類などによって踏み台検出時間が変化することを明らかにし，適切な監視時間の設定が必要であることが分かった．今後はフォールスポジティブを低減させる方法や，踏み台の正常，不正を判断し，攻撃者を適切に特定する方法などについて検討を進める予定である．

謝辞

本研究に関して，研究の方向や進め方など終始ご熱心なご指導とご教示を賜りました，名城大学理工学部情報工学科　渡邊晃教授に心より厚くお礼申し上げます．

本研究を進めるにあたり，研究内容に関して終始ご熱心なご指導とご教示を賜りました，名城大学理工学部情報工学科　小川明教授，高橋友一教授，宇佐見庄五講師に心より厚くお礼申し上げます．

本研究を進めるにあたり，適切なご指導及びご助言を頂いた，宮崎大学工学部情報システム工学科　岡崎直宣助教授に心より厚くお礼申し上げます．

最後に，本研究を行うにあたり，適切なご検討を頂いた，名城大学理工学部情報工学科渡邊研究室の皆様に心より感謝致します．

參考文獻

[1] S. Staniford-Chen and L. T. Heberlein, "Holding Intruders Accountable on the Internet," 1995 IEEE Symposium on Security and Privacy, pp.39-49, May 1995.

[2] X. Y. Wang, D. S. Reeves, S. F. Wu and J. Yuill," Sleepy Watermark Tracing: An Active Intrusion Response Framework," the 16th International Information Security Conference (IFIP/Sec'01), June 2001.

[3] Yin Zhang and Vern Paxson, "Detecting Stepping Stones," 9th USENIX Security Symposium, pp.171-184, Aug. 2000.

[4] Kunikazu Yoda and Hiroaki Etoh, "Finding a Connection Chain for Tracing Intruders," 6th European Symposium on Research in Computer Security (ESORICS 2000), pp.191-205, Oct. 2000.

[5] X. Wang, D. S. Reeves and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," 7th European Symposium on Research in Computer Security (ESORICS 2002), pp.244-263, Oct. 2002.

[6] Xinyuan Wang and Douglas S. Reeves," Robust Correlation of Encrypted Attack Traffic through Stepping Stones by Manipulation of Interpacket Delays," 10th ACM conference on Computer and communications security (CCS 2003), pp.20-29, Oct. 2003.

[7] W. T. Strayer, C. E. Jones, I. Castineyra, J. B. Levin and R. R. Hain, "An integrated architecture for attack attribution," BBN Technologies, Tech. Rep. BBN REPORT-8384, Dec. 2003.

[8] David L. Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit and Stuart Staniford, "Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay," 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002) LNCS-2516, pp.17-35, Oct. 2002.

[9] Avrim Blum, Dawn Xiaodong Song and Shobha Venkataraman, "Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds," 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004), pp.258-277, Sept. 2004.

[10] Linfeng Zhang, Anthony G. Persaud, Alan Johson and Yong Guan, "Stepping Stone Attack Attribution in Non-Cooperative IP Networks," Technical Report 2005-02-1, Department of Electrical and Computer Engineering, Iowa State University, 2005.

[11] Xinyuan Wang, "The Loop Fallacy and Serialization in Tracing Intrusion Connections through Stepping Stones," 2004 ACM symposium on Applied computing, pp.404-411, Mar. 2004.

[12] K. Wong and H. Yung, "Detecting Long Connecting Chains of Interactive Terminal Sessions," 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002) LNCS-2516, pp.1-16, Oct. 2002.

[13] Jianhua Yang and Shou-Hsuan Stephen Huang, "A Real-Time Algorithm to Detect Long Connection Chains of Interactive Terminal Sessions," 3rd international conference on Information security, pp.198-203, Nov. 2004.

[14] Jianhua Yang and Shou-Hsuan Stephen Huang, "Matching TCP Packets and Its Application to the Detection of Long Connection Chains," 19th International conference on Advanced Information Networking and Applications (AINA 2005), pp.1005-1010, March 2005.

[15] Jianhua Yang and Shou-Hsuan Stephen Huang, "Correlating Temporal Thumbprints for, Tracing Intruders," The Third International Conference on Computing, Communications, and Control Technologies, pp.236-241, July 2005.

[16] Jianhua Yang and Shou-Hsuan Stephen Huang, "Improved Thumbprint and Its Application for Intrusion Detection," The Third International Conference on Computer Network and Mobile Computing (ICCNMC), pp.433-442, Aug. 2005.

[17] X. Wang, D. S. Reeves, P. Ning and F. Feng, "Robust network-based attack attribution through probabilistic watermarking of packet flows," Technical Report TR-2005-10, Department of Computer Science, NC State Univ., Feb. 2005.

[18] P. Peng, P. Ning, D. S. Reeves and X. Y. Wang, "Active Timing-Based Correlation of Perturbed Traffic Flows with Chaff Packets," The 2nd International Workshop on Security in Distributed Computing Systems (SDCS-2005), June 2005.

研究業績

1.学術論文

なし

2.国際会議

なし

3.口頭発表

[1] 竹尾大輔，渡邊晃，“ TELNET による渡り歩きの検出方法の検討 ”，電気関係学会東海支部連合大会，Oct. 2003.

[2] 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，Mar. 2004.

[3] 竹尾大輔，渡邊晃，“ FPN における渡り歩きの検出方法の検討 ”，情報処理学会研究報告，Vol.2004，No.75，2004-CSEC-26，pp.275-280，Jul. 2004.

[4] 竹尾大輔，渡邊晃，“ 渡り歩き検出方法の検討 ”，CSS2004 論文集，Vol. Ⅰ，pp.103-108，Oct. 2004.

[5] 竹尾大輔，渡邊晃，“ ネットワーク型渡り歩き検出手法の検討 ”，DICOMO2005 シンポジウム論文集, Vol.2005, No.6, pp.385-388, Jul. 2005.

[6] 竹尾大輔，岡崎直宣，渡邊晃，“ コネクション検出方式による踏み台攻撃検出手法の提案 ”，SCIS2006，Jan. 2006.

[7] 播磨宏和，竹尾大輔，渡邊晃，“ MAC アドレスを用いた IP トレースバック技術の提案 ”，情報処理学会　第 67 回全国大会，Mar. 2005.

[8] 播磨宏和，竹尾大輔，渡邊晃，“ MAC アドレス情報に基づく IP トレースバック技術の提案 ”, DICOMO2005 シンポジウム論文集, Vol.2005, No.6, pp.605-608，Jul. 2005.

[9] 播磨宏和, 竹尾大輔, 渡邊晃,“ MAC-Based トレースバック方式の実装 ”, 電気関係学会東海支部連合大会，Sep. 2005.

4.表彰

[1] DICOMO2005 ヤングリサーチャ賞．竹尾大輔，渡邊晃，“ ネットワーク型渡り歩き検出手法の検討 ”, DICOMO2005 シンポジウム論文集, Vol.2005, No.6，pp.385-388，Jul. 2005.

付録

付録 A　踏み台検出プログラムのソースコード

コネクション検出方式を用いた踏み台攻撃検出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．ここで添付したソースコードは平成 18 年 2 月 19 日現在のものである．

現時点の実装では，SYN パケットを受信してから最も近い PUSH パケットの送信元のみを Attacker の候補とするようにしている．踏み台攻撃検出時には，検出時刻，踏み台検出時間，Attacker・Foothold・Target の IP アドレス，Flow A・B の通信プロトコル情報をログファイルに出力し，同時にコンソールにも表示する．今後は検出時の動作について改良を行っていく予定である．

使用言語：C 言語
プラットフォーム：FreeBSD
C ソースファイル名：ssd_cdm.c（単一ファイルのみで構成）
ソースファイル行数：429 行

ーーーーここからーーーー

```
/*
 * $ProjectSEC: ssd_cdm.c,v 1.000 2006/02/16 02:14:30 Daisuke Takeo Exp $
 *
 * Copyright (c) 2004-2006 Watanabe Laboratory, Meijo University.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the laboratory nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE LABORATORY AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE LABORATORY OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <net/if.h>
#include <net/ethernet.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <pcap.h>

#define DEBUG  1

#define ERRBUF 1024

/* definition of Remoto Login port # */
#define SSH     22      /* ssh          SSH Remote Login Protocol */
#define TELNET  23      /* telnet       Telnet */
#define RLOGIN  513     /* login        remote login a la telnet; */

/* definition of RRI size # */
#define MAXRRINFO       1024    /* max rri size */

/* definition of interval time */
#define OBS_TIME        10000000        /* usec */

/*
 * Structure of a Remote-login Receive Information.
 */
typedef struct rmt_rcv_info {
        u_long  saddr;          /* Src IP address */
        u_long  daddr;          /* Dst IP address */
        u_short sport;          /* Src port number */
        u_short dport;          /* Dst port number */
        struct timeval rtm;     /* Received time */
} RRI;

struct rmt_rcv_list {
        RRI rri[MAXRRINFO];
```

```
        int head;
        int tail;
        int flag;
} rrl;

char errbuf[PCAP_ERRBUF_SIZE];

FILE *fp;
char fname[64] = "./ssd_2006-02-17-000000.dat";

pcap_t *pd;

static char *isremotelogin(u_short dport)
{
        char *s;
        switch (dport) {
        case SSH:
                s = "SSH";
                break;
        case TELNET:
                s = "TELNET";
                break;
        case RLOGIN:
                s = "RLOGIN";
                break;
        default:
                s = NULL;
                break;
        }

        return s;
}

pcap_t *init_pcap(char *device,
    int snaplen, int pflag, int to_ms,
    int Oflag, char *filter)
{
        pcap_t *pd;
        struct bpf_program fcode;
        bpf_u_int32 localnet, netmask;

        if (device == NULL){
                device = pcap_lookupdev(errbuf);
                if (!device){
                        printf("pcap_lookupdev: %s\n", errbuf);
                        return(NULL);
                }
        }

        pd = pcap_open_live(device,
            snaplen, pflag, to_ms, errbuf);

        if(NULL == pd){
```

```
                printf("pcap_open_live: %s\n", errbuf);
                return(NULL);
        } else if(filter == NULL) {
                /* no filter */
                return(pd);
        } else if(pcap_lookupnet(device,
                      &localnet, &netmask, errbuf) < 0){
                printf("pcap_lookupnet: %s\n", errbuf);
                return(NULL);
        } else if (pcap_compile(pd,
                       &fcode, filter, 0flag, netmask) < 0) {
                printf("pcap_compile(): %s\n", pcap_geterr(pd));
                return(NULL);
        } else if (pcap_setfilter(pd, &fcode) < 0) {
                printf("pcap_setfilter(): %s\n", pcap_geterr(pd));
                return(NULL);
        }

        return(pd);
}

#define isether(type) ((type) == DLT_EN10MB)

void print_pktdir(struct ip* ip, struct tcphdr *tcp)
{
        printf(" %d.%d.%d.%d --%d->",
                (int)(ip->ip_src.s_addr & 0x000000FF),
                (int)((ip->ip_src.s_addr & 0x0000FF00) >> 8),
                (int)((ip->ip_src.s_addr & 0x00FF0000) >> 16),
                (int)((ip->ip_src.s_addr & 0xFF000000) >> 24),
                ntohs(tcp->th_dport));
        printf(" %d.%d.%d.%d",
                (int)(ip->ip_dst.s_addr & 0x000000FF),
                (int)((ip->ip_dst.s_addr & 0x0000FF00) >> 8),
                (int)((ip->ip_dst.s_addr & 0x00FF0000) >> 16),
                (int)((ip->ip_dst.s_addr & 0xFF000000) >> 24));
#if DEBUG == 0
        if (tcp->th_flags & TH_SYN)
                 printf("(syn)");
        if (tcp->th_flags & TH_PUSH)
                 printf("(psh)");
        if (tcp->th_flags & TH_ACK)
                 printf("(ack)");
#endif

        printf("\n");
        return;
}

void add_rrl(struct ip *ip, struct tcphdr *tcp, struct timeval *ts)
{
        if (rrl.tail == rrl.head) {
                /* rrl is full. */
```

```
                if (rrl.flag != 0) {
                        rrl.head = (rrl.head + 1) % MAXRRINFO;
                        if (rrl.head == 0) rrl.flag = 0;
                }
                /* rrl is empty. */
        }
        rrl.rri[rrl.tail].saddr = ip->ip_src.s_addr;
        rrl.rri[rrl.tail].daddr = ip->ip_dst.s_addr;
        rrl.rri[rrl.tail].sport = tcp->th_sport;
        rrl.rri[rrl.tail].dport = tcp->th_dport;
        memcpy(&rrl.rri[rrl.tail].rtm, ts, sizeof(struct timeval));

#if DEBUG == 1
        printf("add: rrl[%04d]", rrl.tail);
        /* Print Attacker's info. */
        print_pktdir(ip, tcp);
#endif

        rrl.tail = (rrl.tail + 1) % MAXRRINFO;
        if (rrl.tail == 0) rrl.flag = 1;
}

/*
void search_rrl_before_add(struct ip *ip, struct tcphdr *tcp, struct timeval
*ts)
{
        int i, j;
        u_long diftm;

        if (rrl.flag == 0) return;
        j = rrl.head;

        for (i = rrl.tail + 1; i != j;) {
                i = (i - 1) % MAXRRINFO;
                if (rrl.rri[i].daddr != ip->ip_src.s_addr) continue;
                else {
                        diftm = (ts->tv_sec - rrl.rri[i].rtm.tv_sec) *
                        1000000 + (ts->tv_usec - rrl.rri[i].rtm.tv_usec);
                        if (diftm <= OBS_TIME) {
                                printf("[Detected Island-Hop,"
                                        " interval: %u.%03ums]\n",
                                        diftm/1000, diftm%1000);
                                break;
                        }
                }
        }
}
*/

void search_rrl(struct ip *ip, struct tcphdr *tcp, struct timeval *ts)
{
        int i, j;
        u_long diftm;
```

```
        j = rrl.head;
        i = rrl.tail;
        printf("search: ");
        do {
                i = (i - 1) % MAXRRINFO;
                printf("[%d]", i);

                diftm = (ts->tv_sec - rrl.rri[i].rtm.tv_sec) * 1000000 +
                        (ts->tv_usec - rrl.rri[i].rtm.tv_usec);

               /* If detection time is longer than observation time, then return.
*/
                if (diftm > OBS_TIME) return;

                if (rrl.rri[i].daddr == ip->ip_src.s_addr) {
                        struct tm *tp;
                        tp = localtime((time_t *)(&ts->tv_sec));
                        /* Print detected time. */
                        printf("\n%04d/%02d/%02d %02d:%02d:%02d.%03d",
                                tp->tm_year + 1900, tp->tm_mon + 1,
                                tp->tm_mday, tp->tm_hour, tp->tm_min,
                                tp->tm_sec, ts->tv_usec/1000);
                        printf(", %u.%03u ms",
                                diftm/1000, diftm%1000);

                        /* Write detected time to log-file. */
                        fwrite(ts, sizeof(struct timeval), 1, fp);

                        /* Write detection time to log-file. */
                        fwrite(&diftm, sizeof(u_long), 1, fp);

                        /* Write Attacker and Foothod info to log-file. */
                        fwrite(&rrl.rri[i].saddr, 8, 1, fp);

                        /* Write Target info to log-file. */
                        fwrite(&ip->ip_dst, 4, 1, fp);

                        /* Write port info of Flow A,B to log-file. */
                        fwrite(&rrl.rri[i].sport, 4, 1, fp);
                        fwrite(&tcp->th_sport, 4, 1, fp);

                        /* Print Attacker's info. */
                        printf(", %d.%d.%d.%d --%d->",
                                (int)(rrl.rri[i].saddr & 0x000000FF),
                                (int)((rrl.rri[i].saddr & 0x0000FF00) >> 8),
                                (int)((rrl.rri[i].saddr & 0x00FF0000) >> 16),
                                (int)((rrl.rri[i].saddr & 0xFF000000) >> 24),
                                ntohs(rrl.rri[i].dport));

                        print_pktdir(ip, tcp);

                        break;/*continue;*/
```

```
                }
        }
        while (i != j);
}

void
ssd_cdm(userdata, h, p)
    u_char *userdata;
    const struct pcap_pkthdr *h;
    const u_char *p;
{
        int i, len;
        u_long diftm;
        char *rl_name;
        u_char *c;
        struct ether_header *eh;
        struct ip *ip;
        struct tcphdr *tcp;

        /* verify */
        len = h->caplen;

        if (len < sizeof(struct ether_header)) return;
        len -= sizeof(struct ether_header);

        eh = (struct ether_header *)p;
        c = ((char *)eh + sizeof(struct ether_header));

        ip = (struct ip *)c;

        if (len < sizeof(struct ip)) return;
        if (len < ((int)(ip->ip_hl) << 2)) return;

        c = ((char *)ip + ((int)(ip->ip_hl) << 2));
        tcp = (struct tcphdr *)c;

        /*
         * Check TCP Control Flag Field, syn or push
         */
        if (tcp->th_flags == TH_SYN) {
                /**printf("[Detected tcp-syn packet]\n");**/

                /* If rrl is empty, then only print packet direction. */
                if (rrl.head == rrl.tail && rrl.flag == 0)
                        goto PRINT;
                /* Search Island-Hop Receive List */
                search_rrl(ip, tcp, (struct timeval *)&h->ts);
        } else if (tcp->th_flags & TH_PUSH) {
                if (rl_name = isremotelogin(ntohs(tcp->th_dport))) {
                        /**printf("[Detected %s]\n", rl_name);**/
                        add_rrl(ip, tcp, (struct timeval *)&h->ts);
                } else return;/**/
        } else return;/**/
```

```
        /*else if (rl_name = isremotelogin(ntohs(tcp->th_sport)))
                printf("[<-%s] ", rl_name);*/

PRINT:

        /**print_pktdir(ip, tcp);**/
        return;
}


void ssd_cdm_close(int unused)
{
        printf("\nEnd stepping-stone detection program.\n");
        pcap_close(pd);
        fclose(fp);
        exit(0);
}


int main(int argc, char *argv[])
{
        int snaplen, pflag, to_ms, Oflag;
        int cnt;
        char *device, *filter;
        struct timeval tv;
        struct tm *tp;
        /**pcap_t *pd;**/

        gettimeofday(&tv, NULL);
        tp = localtime((time_t *)(&tv.tv_sec));
        /* Generate log-file name. */
        sprintf(fname, "./ssd_%04d-%02d-%02d-%02d%02d%02d.dat",
                tp->tm_year + 1900, tp->tm_mon + 1,
                tp->tm_mday, tp->tm_hour, tp->tm_min,
                tp->tm_sec);
        if ((fp = fopen(fname, "wb")) == NULL) {
                printf("ERROR: cannot open file [%s]\n", fname);
                exit(1);
        }

        rrl.head = 0;
        rrl.tail = 0;
        rrl.flag = 0;

        /*
         * default value
         */

        snaplen = 100;
        pflag = 1;
        to_ms = 500;
        Oflag = 0;
```

```
        cnt = -1;

        filter = "tcp";

        if (argc == 2)
                device = argv[1];
        else
                device = NULL;

        /*
         * main
         */

        pd = init_pcap(device, snaplen, pflag, to_ms, Oflag, filter);

        if (pd == NULL)
                exit(1);

        if (!isether(pcap_datalink(pd))) {
                printf("ERROR: not ethernet\n");
                exit(3);
        }
        signal(SIGINT, ssd_cdm_close);
        printf("Start stepping-stone detection program on %s.\n",
                device ? device : "autoselect");
        printf("log-file name is [%s]\n", fname);
        if (pcap_loop(pd, cnt, ssd_cdm, NULL) < 0) {
                printf("ERROR: pcap_loop: %s\n",
                    pcap_geterr(pd));
                exit(3);
        }

        pcap_close(pd);

        exit(0);
}
```

ーーーーここまでーーーー