卒業論文　　2006年度 (平成18年度)

フロー順序に着目したボット検知手法の提案と検証

慶應義塾大学 環境情報学部

氏名：金井 瑛

指導教員

慶應義塾大学 環境情報学部

村井 純

徳田 英幸

中村 修

楠本 博之

高汐 一紀

湧川 隆次

平成19 年1 月24 日

卒業論文要旨 2006 年度 (平成 18 年度)

フロー順序に着目したボット検知手法の提案と検証

論文要旨

ボットウェアと呼ばれるソフトウェアに感染した不特定多数のホスト (ボット) からなるボットネットは，悪意を持つ攻撃者の命令によって DDoS 攻撃や SPAM メールの送信などの様々な脅威を引き起こしている。既存のボットネット対策の多くは，ボットウェアがウィルスと同様にホストの脆弱性を利用して感染することから，既存のセキュリティ対策手法を利用している。

しかし，ボットネットは二つの理由から既存のボットネット対策手法では有効な対策にならない．一点めは，亜種の頻繁な出現とボットの更新が容易であることから，多くの既存の手法が用いているパケットのヘッダやペイロードに着目した攻撃や感染活動の検出手法では，不十分であるという点である．二点めは，ボットがさまざまなネットワークに分散しているため，検知したボットの対応が困難な点である．

本手法ではこれらの問題を解決したボット検知手法を提案する．亜種の検知はトラフィックのフロー順序に着目し，特徴ある前後関係を抽出することで行う．さらに，管理ネットワークと外部ネットワークの境界において検知を行うことで，ネットワーク管理者に対してボットの特定を支援する手法を提案した．そして，提案手法に基づくシステムの設計及び実装を行った．

本提案手法により未知のボットウェアに対する検知が可能となり，ボットに対する効率的な対応が可能となる．

キーワード

1．インターネット，2．セキュリティ，3．ボットネット，4．フロー

慶應義塾大学 環境情報学部

金井 瑛

Abstract of Bachelor's Thesis

Academic Year 2006

The Proposal and Validation of Bot Detection Method based on Flow Sequences

Abstract

Bot is a host infected by malicious software, botware. A plenty of Bots construct a Botnet, and cause various threats, such as Distributed Denial-of-Service Attacks (DDoS attacks) and sending of spam mails, by the instruction of malicious attackers. The majority of countermeasures against these threats utilizes existing security techniques since botware infects hosts exploiting existing vulnerabilities like a computer virus does.

However, there are two reasons why the existing security measures are not effective against botnets. Firstly, due to the frequent appearance of subspecies and easiness of updating bots, existing detection methods which focus on the packet header or payload are insufficient against botnets. Secondly, because bots are distributed over the networks it is difficult to cope with the detected bots.

A method to detect bots which solves these problems is proposed in this paper. Subspecies are detected through focusing on the order of traffic flows and extracting the distinctive flow context. Furthermore, the method for network administrators to identify bots is proposed through the detection at the border between internal and external network. Finally, a system is designed and implemented based on the proposal.

Our research enables the detection of unknown botwares, and the effective treatment against bots become possible.

Keywords :
1. Internet, 2. Security, 3. Botnet, 4. Flow

Keio University , Faculty of Environmental Information

Akira KANAI

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

今日ではインターネットの利用範囲は，学術だけでなく行政や商業に広がり，社会において欠かせないインフラとなった．しかし，1998 年にコンピュータウィルスに感染した悪意を持つ攻撃者によって不正に利用される計算機（ホスト）の集合が出現した [1]．これは後に，**ボットネット**と呼ばれ，それ以降インターネット上の脅威として認識された．

ボットネットは経済的利益を得る目的としても利用されており，2006 年 5 月には攻撃者が大量のホストからクリック課金型の Web 広告を不正クリックし利益と得ている [2] と報告されている．今後のインターネットではより様々な場面で，ボットネットによる脅威が問題になると予想される．このため，ボットネットの対策が急務である．

1.2　ネットワークでのボット発見の必要性

ボットネットは社会的に影響を与える原因となる．被害を受けるのはボットネットによる攻撃を受けた側のみではない．多くの場合，ボットネットを構成するホスト（ボット）のホスト管理者はボットネットに加入していることを把握しておらず，他のホストやネットワークに対して悪意のある活動を行っていることを認識していない．ネットワーク内のボットの存在は，次に述べる理由により，ネットワーク管理者にとっても脅威となる．

機密性

ボットの存在はネットワーク内に含まれる機密情報の流失といった危険を含む．ボットの活動の中には，ホスト内に保存された情報を任意のホストに転送する活動や，ホストで入力された情報を記録する活動がある．これによって，ホスト管理者の個人情報が流失する．また，企業などのローカルネットワークにおいては，社内情報の流出などに結びつく．機密性が失われることは，経済的な損失や，さらなる攻撃の足がかりとなり，ネットワーク管理者にとって脅威である．

可用性

ボットはネットワークリソースを消費し．ネットワークの可用性を脅かす危険性を持つ．ボットは常に攻撃者からの命令を待ち受け，他のホストに対してボット感染のための調査と攻撃をする．また，ボットの活動として，DDoS 攻撃や不正なソフトウェ

アの共有といった，本来の目的外でネットワークの帯域が利用されることもある．ネットワークの帯域が増加すると，ネットワークの接続性が不安定になることがある．ネットワーク管理者にとって，定常的なネットワーク運用のためには可用性を保護しなければならない．

このように，ボットネットはネットワーク管理者の観点から今後何らかの対策が必要な脅威となっている．

従来，インターネット上で問題となっているコンピュータウィルスの脅威に対応するための対策が考案，実施されている．現在のボットの検知や対策は既存の技術の延長線上である．しかし，ボットネットは従来の脅威とは異なる特徴を有している．そのため，既存の対策手法による検知を避けるボットネットに対して十分な対応が期待できない．

1.3 本論文の目的

ネットワーク内のボットの対策には，内部のホストをボットネットに加入させない事前対策と，ボットが内部に存在する場合の事後対策として，ボットの検知と駆除が必要となる．しかし，ボットネットは攻撃や活動の種類を頻繁に変更するため，多くの既存の手法が用いている特長を利用した検知手法では十分な検知ができない．

ボットの事前対策や検知が難しい理由の１つとして，ボットの性質が随時アップデートされるというボットネットの特徴がある．現在多く用いられているコンピュータウィルスといった脅威に対応するための技術は，対象に関する特定の特徴を用いている．このため，ボットに対する特徴のアップデートが追いつかず，ボットの検知は対応が困難となっている．

そこで，本論文ではボットの通信順序を用いたボット検知に有効な手法を提案する．そして，提案手法のプロトタイプ実装を行い，有効なボット検知ができるか評価する．

1.4 本論文の構成

本論文は 8 章から構成される．2章では，本論文が対象とするボットネットの定義と現在行われている調査について述べる．また，実ネットワーク環境下におけるボットネットの実トラフィックを元に，ボットネットの通信について整理する．3章では，既存の対策と研究を挙げ，ボット検知実現の問題点を整理する．4章では，提示した既存の問題点を元に，新しいボット検知手法を設計する．5章では，提案手法の実装について述べる．6章では，実装を元に，提案した手法についての評価を行い，既存の技術では対応の難しかったボットに対して正当な検知が行われているかを評価する．最後に7章において，本論文についてまとめ，今後の展望について述べる．

第2章　ボットネット

本章では，ネットワークベースでのボット検知対策を検討するにあたり，ボットネットの概要ならびにボットネットを構成するボットの概要を述べ，ボットネットの特徴を整理する．

2.1　ボットネットの概要

現在発見されているボットネットは管理の仕組みによってClient/Server型 (C/S型) ボットネットとP2P型ボットネットに分類される．C/S型ボットネットは，現在ほとんどのボットネットで利用されているボットネットの制御手法である．ボットは攻撃者が接続している特定のホストに接続して命令を受けるまで待機し，攻撃者により管理される．P2P型ボットネットは，最近になって出現が確認されたボットネットの制御手法で，ボット間の命令の伝達でボットネットの管理が行われる．

現在最も多く発見されているIRC型ボットネットを解説する．IRC型ボットネットの構成要素を図2.1に示す．図中とボットネットで用いられる用語について述べる．**攻撃**

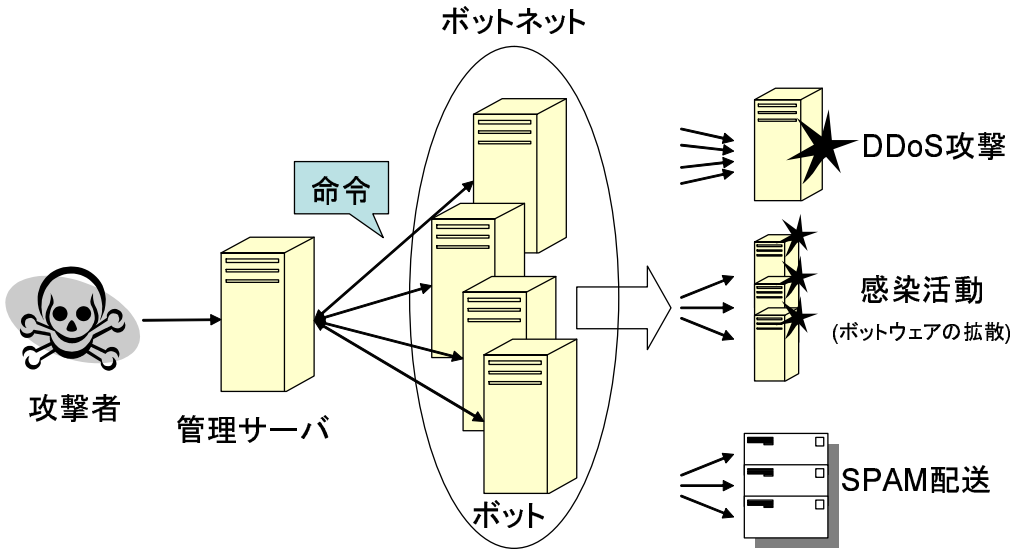

図 2.1: IRC型ボットネットの構成要素

者は，ボットネットを管理する悪意を持ったユーザである．**ボットネット**は，攻撃者により不正に利用されるホストを集合とするオーバレイネットワークである．**ボット**は，

ボットウェアに感染し，ボットネットに管理されるホストである．ボットは攻撃者からの命令に従って，ホストの管理者が意図しない通信や行動を実行する．**ボットウェア**は，ワームやウィルスといったマルウェアのうち，攻撃者の命令を実行するバックドア機能を有するソフトウェアを指す．**Command & Control(C&C) サーバ**は，攻撃者がボットの管理のための命令をボットに伝達するための中継ホストである．図 2.1 には C&C サーバが 1 台のみ存在するが，C&C サーバが複数台存在するボットネットや，接続が切れると予め用意しておいた予備の C&C サーバに接続しなおすボットネットも存在する．現時点において，C/S 型ボットネットでは，ボットの制御に IRC プロトコル [3]，あるいは IRC プロトコルをベースとした独自プロトコルを利用しているケースが一般的である．このボットネットを IRC 型ボットネットと呼ぶ．

管理ネットワークは，ネットワーク管理者によって管理されたインターネット上のネットワークである．ネットワーク管理者は検知したネットワーク内のボットに対してホストの通信制限や，ホスト管理者に連絡する．管理ネットワークの例としては，Internet Service Provider(ISP)，大学のキャンパスネットワークや企業のネットワークが挙げられる．図 2.2 に管理ネットワークとボットネットの関係を示す．図中では 2 つのボットネットが存在し，ボットネット上のボットはいずれかの管理ネットワーク上に所属する．しかし，あるボットネット上のボットがすべて同じ管理ネットワークに所属するとは限らない．

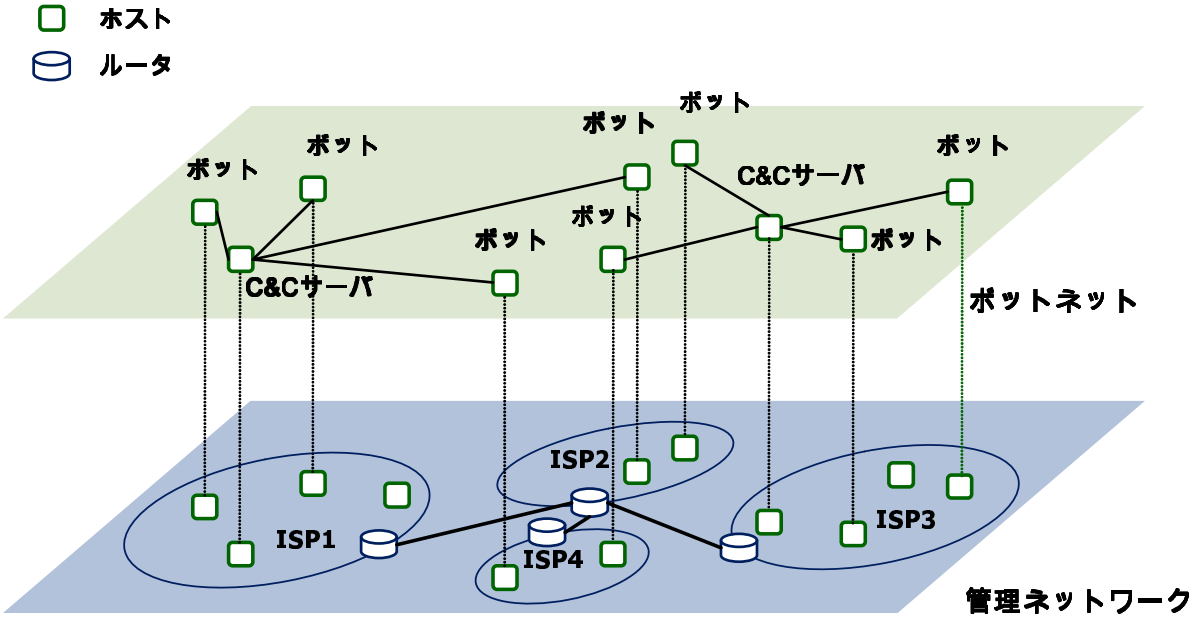

図 2.2: 管理ネットワークとボットネットの関係

2.2 ボットの動作

本節では，ボットネットを構成するボットに着目し，脆弱性を持つホストがボットウェアに感染し，ボットとしてどのように活動するかを述べる．ボットの動作を図2.3に示し，各活動について説明する．

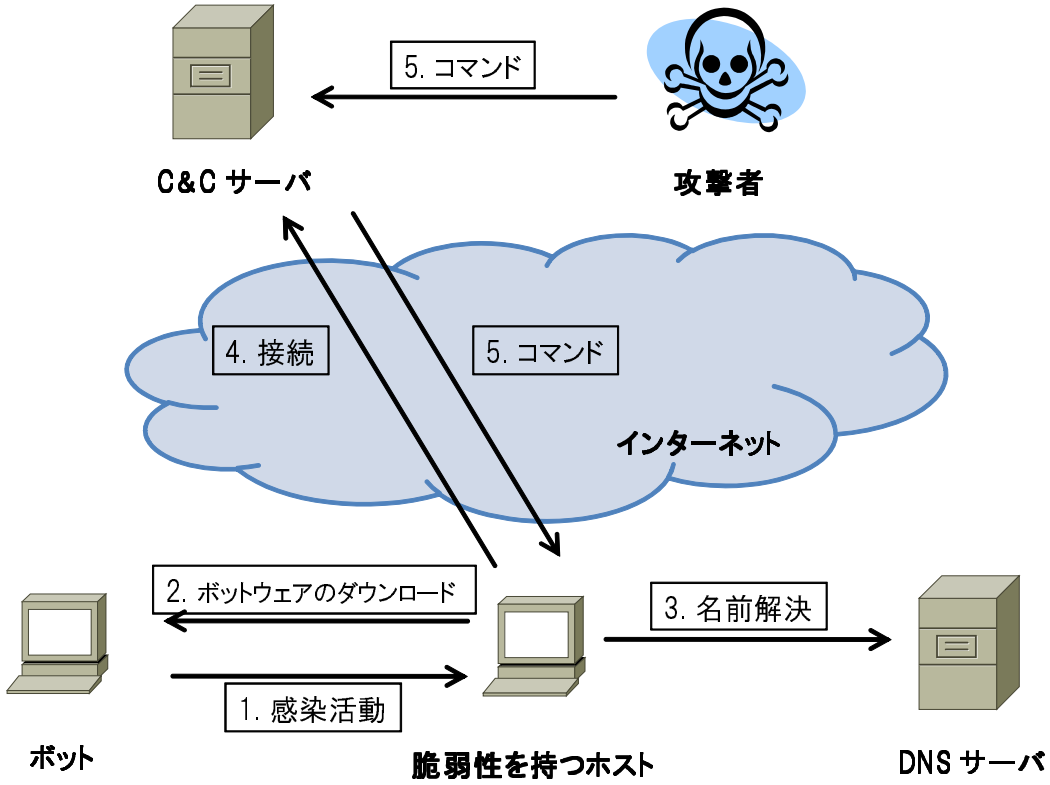

図 2.3: ボットのライフサイクル

感染活動

攻撃者は，しばしばボットネットの規模を拡大するために，ボットを操作して、脆弱性を持ったホストを探索する．探索により脆弱なホストが発見された場合，攻撃を行い，当該ホストにボットウェアをダウンロードさせる．

ボットウェアのダウンロード

脆弱性を持つホストは，脆弱性を利用した攻撃により，ボットウェアをダウンロードし実行させられる．ダウンロード元ホストは，感染活動元のボットとは限らない．しかし，多くの場合，感染活動元のボットと同じボットネットに所属するボットである．

名前解決

ホストにボットウェアがインストールされると，そのホストはボットとなり，攻撃者からの命令を待機するために C&C サーバと接続する．接続の際のドメイン名を解決するために，ボットは DNS サーバで名前を解決する．

接続
名前解決が成功するとボットはC&Cサーバに接続する．そして，ボットはC&Cサーバから命令を受け取るまで待機し続ける．

命令
攻撃者は任意にボットネットのボットに対して活動実行を命令する．ボットは攻撃者からC&Cサーバ経由で命令を受けると，2.3節で述べる様々な活動を開始する．命令実行後は再び，C&Cサーバからの命令を待機する．

基本的にボットは以上の動作を繰り返す．ボットはホスト管理者によってボットウェアが駆除されるか，あるいはC&Cサーバとの接続が不可能になるまで，攻撃者の命令を待つ．

2.3 ボットにおける活動

2.1節では，ボットネット全体の動作概要について述べた．本節ではボットネットを構成するボットが攻撃者からの命令に基づいて実行する活動について述べる．表2.1 に主な活動の例を示し，各例についてに説明する．

表 2.1: ボットの主な活動例

目的	活動	活動内容
情報取得	キーロギング	入力されたキーボードの内容を全て取得
	トラフィック盗聴	ボットのトラフィック盗聴
	記録情報の取得	記録されているパスワードや e-mail のアドレスを取得
攻撃者の利益	不正 Web 投票	Web 投票や Web 広告のクリックによる情報操作
攻撃行為	DDoS 攻撃	フラッディング攻撃によるサービス妨害やネットワーク輻輳
	SPAM メールの送信	SPAM メールの中継や送信による他のユーザへの迷惑行為
	攻撃者の踏み台	SOCKS サービスなどの提供による攻撃者の活動幇助
ボットネット運用	ボットウェアの拡散	他のホストへの攻撃によるボットネットの拡大
	C&C サーバへの昇格	ボットネットの C&C サーバ冗長化

キーロギング とは，ユーザの入力したキーボードの文字を収集する活動の一種である．保護されたシステムにアクセスするためのパスワードや，クレジットカード番号の入力など，第三者に対して秘匿にされるべき情報が記録される．
トラフィック盗聴 とは，感染ホストにおいて，そのネットワークトラフィックをすべて盗聴する活動である．暗号化されていないパスワードやメールの流出をはじめ，あらゆるネットワーク上での活動が記録されうる．

記録情報の取得とは，感染ホスト上に保存されている情報を取得する活動である．保存されている情報の例としては，Web ブラウザのキャッシュファイル，メールや写真等様々な情報である．

不正 Web 投票 とは，異なる多数のホストからアクセスされるべき Web 上のコンテンツにボットネットを用いてアクセスすることである．ボットから Web にアクセスし，ボットネットの管理者は WEB 投票に対して組織票を送り込むことができる．また，Web 上の広告へアクセスすることで利益を得るシステムを，ボットネットを用いて大量にアクセスし，攻撃者が経済的な利益を得ることができる．これらのことから，今後インターネット上での選挙や宣伝が考えられる上で深刻な問題となると予想される．

DDoS 攻撃 とは，特定のネットワークやホストに対してパケットを送る活動である．近年はユーザ回線の広帯域化が進み，少数のホストでも多くのトラフィックを発生させることが出来る．これによって，対象ネットワークの帯域を溢れさせたり，対象ホストのサービスを実質的に利用不可能にすることができる．DDoS 攻撃はインターネットを可用性を脅かす単純かつ大きな脅威であり，従来から問題視されているが，効果的な対策は難しい．

SPAM メールの送信 とは，SPAM と呼ばれる迷惑メールを不特定多数の人に送信する活動である．感染ホストにメールアドレスが保存されている場合に，あたかも正規の利用者からのメールのように見せかける場合もある．SPAM に対しては RBL[4] といった対策が存在するが，攻撃者はボットネットを用いて送信元を複数用意し，対策を困難にしている．

攻撃者の踏み台 とは，攻撃者が通信を被害ホストで中継することにより，通信をあたかも被害ホストから送信されたように見せかけるものである．例えば，SOCKS[5] を用いてすべての IP 通信を中継することで，攻撃者への対策や追跡を極めて困難にする．

ボットウェアの拡散 はボットネットの拡大のための活動である．ボットはボットウェアに感染あるいは攻撃者から命令を受けると，任意のアドレス範囲に対して脆弱性を持つホストを検索する．脆弱性を持つホストが見つかると，2.2節で述べたような感染活動を行う．感染活動が成功することにより，ボットネットに新しいボットが加入し，ボットネットの規模は拡大する．

C&C サーバへの昇格 とは，感染ホスト上でボットネットの C&C サーバのプログラムが起動され，ボットネットの C&C サーバとなる活動である．攻撃者はそのサーバをダイナミック DNS に登録し，随時更新する．それより，攻撃者は IP アドレスやドメイン名を用いたボットネットの対策を困難にする．

2.4 ボットネットの活動調査

現在，ボットネットはいくつかの組織によって調査が行われている．本節では，その中から主要な調査組織とその調査内容を述べる．

2.4.1 Honeynet Project

Honeynet Project はハニーポットを運用している非営利団体である．

2005 年 3 月に公開された Know your Enemy: Tracking Botnets[6] は，当時から現在まで主流となっている IRC[3] 型のボットについて述べられている．感染と C&C サーバとの接続について，実際のボットネットトラフィックを監視することでボットの通信について述べられている．

特に，**ボットの活動**，**ボットの原種 (亜種のもととなる) ボット**と**ボットと C&C サーバの通信内容**について述べられている．これらを踏まえた上で，Honeynet Project の実施しているボットネットの IRC 調査や，調査結果に基づいたネットワーク管理者への通達について述べられている．

2.4.2 An Inside Look at Botnets

An Inside Look at Botnets は 2006 年 4 月に公開されたボットネットに関する報告である．本報告は IRC 型ボットネットにおける C&C サーバとの通信内容について特に述べられている．本報告ではボットを **Agobot**, **SDBot**, **SpyBot** と **GT Bot** の 4 種類の原種に分類し，それぞれの特性を述べている．

2.4.3 Shadow Server

Shadow Server[7] はセキュリティ専門家のボランティアにより構成されたセキュリティ監視グループである．これは，2.4.1節の honeynet と同様に，ボットネットの活動を観測することで，更ににボットウェアの解析に注力している．

近年のボットウェアは解析を困難にするために，仮想マシン上での実行を不可能にしている場合が多々ある．本活動ではこれらの制限にも対応し，積極的なボットウェアの活動を分析している．トラフィック分析のみではなく，ボットウェアの挙動に着目した収集活動は，今後のボットネットの活動を追跡していく上で極めて有用である．

2.5 ボットネットの特徴

本節ではボットネットが持つ特徴について述べる．また，ボットネットの対応が困難である理由について整理する．

2.5.1 ボットウェアの亜種と更新機能

ボットネットの大きな特徴のひとつは，頻繁なボットウェアの亜種の出現と，ボットウェアの更新機能である．**亜種**は，あるボットウェアを元に攻撃者が必要に応じて一部の動作を改変したボットウェアを指す．2006 年 6 月時点では，毎日約 80 種類の亜種

が発生しているといわれており [8]，ボットの接続先，あるいは攻撃手法などを利用した検知を困難にする．多くのボットは，インターネット上から新しいボットウェアの亜種をダウンロードし，自分自身のボットウェアを更新する機能を持つ．ボットネットは C&C サーバによって一元的に管理されているため，攻撃者は新しい亜種を即座に配布できる．

表 2.2: 亜種の主な変更点と目的

変更点	目的
C&C 接続情報の変更	攻撃者毎の情報に変更
攻撃手法の変更	セキュリティ製品に対する対策等
Packer での圧縮	セキュリティ製品に対する対策

亜種によって変更される主な点を表 2.2に示し，以下で亜種における変更点とそれに伴う対応の難しさを述べる．

C&C 接続情報の変更

IRC 型ボットネットは，C&C サーバ，チャンネルとパスワードによって区別される．そのため，多くの攻撃者は自らのボットネットを生成する際は，これらの情報を書き換える．

また，C&C サーバの動的な更新によって，攻撃者は C&C サーバを対象とした対策を困難にする．攻撃者は 2.5.4節で述べるように C&C サーバの冗長化を図ることができる．攻撃者はある C&C サーバへの接続性が失われると，各ボットに対して別の C&C サーバに接続する亜種を送信することで，再度，冗長化を図る．それにより，C&C サーバを停止する対策は困難となる．

攻撃手法の変更

ボットウェアの利用する脆弱性はいずれセキュリティベンダやネットワーク管理者によって修正される．攻撃者は，利用する脆弱性がボットネットの拡大に効果的でなくなったと判断した際，各ボットに対して新しい脆弱性を利用した攻撃が可能なボットウェアに更新するよう命令を発行する．これにより，ボットネットは再度拡大が可能となる．

Packer での圧縮

Packer とは実行可能なファイルを圧縮するソフトウェアである．本来は，ファイルのサイズを縮小する目的か，逆アセンブルを困難にするための暗号化に用いられる．ボットが Packer で圧縮される目的は，既存手法の検知から逃れるためである．Packer で圧縮されたファイルは，ファイルの内容が圧縮前とは大きく異なる．そのため，Packer により圧縮されたボットウェアは，既存手法でよく用いられるファイルの特定の内容に特化した検知手法から逃れることができる．

2.5.2 規模性の高さ

規模性とは，ボットネットを構成するボット数の大きさを示す．規模が大きくなると，個々のボットに対策を施すことが困難となっている．ボットネットを構成するボットの規模は数百台から数千台のものが多い [9]．今までに検出された最大規模のボットネットは 50 万以上のホストから構成されている．インターネット全体におけるボットの感染率は 5%とも 10%以上とも言われている [10]．平成１７年通信利用動向調査の結果[11] によると，平成 17 年末における日本でのインターネット利用者は 8529 万人である．ボットの感染率が 5%と仮定すると約 400 万以上ものユーザが知らずにボットに感染していることとなる．一度，ボットウェアに感染したホストは，ネットワークから切り離し，アンチウィルスソフトなどで駆除，もしくはシステムを再インストールしなければならない．インターネット上の全てのホストに対しての対応はその規模から困難であるため，ボットネット自体への対応は非常に困難となる．

2.5.3 分散性の高さ

分散性とは，ボットネットを構成するボットが，様々なネットワークに分散している様子を指す．実際に分散性の高いボットネットではボットが世界中に分散しており，さらに，それらのネットワーク管理者が異なるため，個々のボットへの対策が困難となる．

まず，図 2.2のように，あるボットネットのボットが全て単一の管理ネットワークに所属していない．そこで，あるボットに対策しようとするとそれぞれのボットの管理者あるいはネットワークの管理者に通知をしなければならない．多くの場合被害者が得られるボットの情報だけではホストの管理者に連絡は取れないため，ネットワークの管理者に通達されるが，対応に時間がかかる場合が多い．ホスト管理者に連絡をしても対応がされない問題が多い．更に，ネットワークの管理者への通達に対応が行われない事例もある．

2.5.4 C&C サーバの冗長化

ボットネットは C&C サーバを予め複数接続可能しておき冗長性を保っている．冗長化によりボットが接続している C&C サーバが利用不可能になって，別のサーバに切り替えることでボットネットの運用を継続できる．冗長化の方法は大きく，DNS を用いた冗長化と，バックアップ C&C サーバを用意しておく２つの方法がある．また，この２つの冗長化の方法は組み合わせて利用される．

- DNS を用いた冗長化
  ボットは接続先ホストの IP アドレスを DNS によって得る．まず，攻撃者は DDNS のサービスを購入する．予め，一定規模のボットネットを用意しておき，そのうちの数台を DDNS にドメインのネームサーバとして登録する．これらのネーム

サーバには C&C サーバの IP アドレスが含まれる．ボットは感染後にこの名前を用いて C&C に接続する．ドメイン名に対する IP アドレスを変更することで，冗長化した別の C&C サーバに接続することが可能となる．

- バックアップ C&C サーバの用意
  予め攻撃者はボットウェアに C&C サーバの情報を 2 つ以上含ませておく．C&C サーバが停止した場合，ボットは他の C&C サーバとの接続を試みる．

C&C サーバとの接続は冗長的に保証されており，C&C サーバへの接続を停止することは難しい．またボットとして活動しているホストを C&C サーバに昇格させ，現在の C&C サーバを停止させる事も可能である．これらのことから，C&C サーバの停止によるボットネットの機能停止は困難である．

2.6 ボットネット調査

本節では実ネットワーク環境におけるボットの活動調査と収集されたトラフィックについて述べる．

まず，ボットネット調査の必要性について述べる．次に，調査に使用したトラフィックの収集環境を示し，本環境下はホストがボットウェアに感染し，ボットとして活動を始めるトラフィックを 2 例収集した．そしてサンプルのトラフィックからボットトラフィックの特徴を見つける．

2.6.1 ボットネット調査の必要性

近年では，2.4節で述べたようにボットの動作に関する調査やそれらを基にした研究がいくつも発表されている．しかし，実際にボットネットの行っているトラフィックに関する情報は少なく，ボットネットに対する効果的な対策の考察するための十分な情報が記載されているとは言えない．本研究において，効果的な対策を提案するために，実ネットワーク環境において実際にボットに感染したマシンのトラフィックを分析して，特徴を見つけ出す必要がある．

2.6.2 調査用ネットワークトポロジ

本調査に用いたトポロジを図 2.4 に示す．本調査は日本国内の ISP 利用者の協力を得て，ユーザが一般に利用しているネットワークのトラフィックを監視した．トラフィックの監視は外部ネットワークとの境界で行っているため，外部と内部で行われる全ての通信を記録している．ネットワーク内部には 1 台のホストが存在している．特に，192.168.0.100 のアドレスが付与されたホストはルータで NAT[12] されており，外部からのアクセスは全てこのホストに転送される．

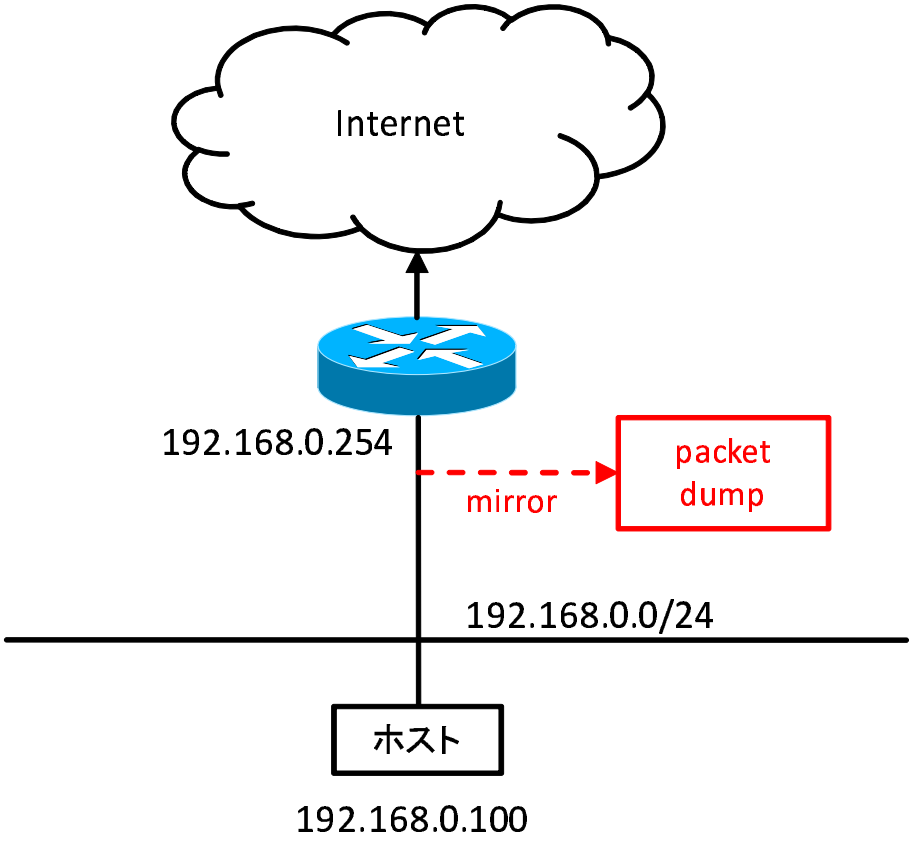

図 2.4: 調査に用いたトポロジ

2.6.3 トラフィック調査の手法

本節では取得したトラフィックに関する調査の手法について述べる．まず，収集したトラフィックに対して tcpdump による出力結果から，著者の目視によって，正常ではない通信を発見する．これまでに述べた調査により，一般的なボットの感染プロセスは以下のように区別されているため，本調査でも，その区別に従って，該当する通信のフローがどのような動作をしているかについて述べる．

- 攻撃
- ボットのダウンロード
- C&C サーバとの接続

2.6.4 調査結果

本節では，調査の結果収集できたボット感染のトラフィックの tcpdump 出力結果の抜粋を示し，通信内容を示す．ただし，通信内容の IP アドレスとドメイン名は変更が加えてある．

第１のサンプル

１つめの感染トラフィックデータにおける tcpdump 出力結果の抜粋を図 2.5 に示す.

```
1: /* 感染 */
2: 422.685616 218.x.x.x 192.168.1.100 TCP 2035 > epmap [SYN]
3: 422.685991 192.168.1.100 218.x.x.x TCP epmap > 2035 [SYN, ACK]
4: 422.800352 218.x.x.x 192.168.1.100 DCERPC
5: 422.800675 192.168.1.100 218.x.x.x DCERPC
6: 422.852140 218.x.x.x 192.168.1.100 MGMT
7: 422.853104 192.168.1.100 218.x.x.x MGMT
8: /* ダウンロード */
9: 423.781109 192.168.1.100 218.x.x.x TFTP Read Request, File: WindowsLogon.exe
10: ···TFTP データ転送···
11: 455.381920 192.168.1.100 218.x.x.x TFTP Acknowledgement, Block: 455
12: 455.577860 192.168.1.100 218.x.x.x TCP epmap > 2144 [FIN, ACK]
13: /* 接続先 IRC サーバのアドレス解決 */
14: 474.218393 192.168.1.100 192.168.1.254 DNS Standard query A vr00m.example.be
15: 474.255759 192.168.1.254 192.168.1.100 DNS Standard query response A 72.x.x.x
16: /* C&C サーバとの接続 */
17: 474.257812 192.168.1.100 72.x.x.xx TCP 1033 > 6667 [SYN]
18: 474.408865 72.x.x.x 192.168.1.100 TCP 6667 > 1033 [SYN, ACK]
```

図 2.5: ボット感染時のトラフィック例１

サンプルトラフィック１の解説

line 1-7: Windows NT 系列の RPC DCOM 脆弱性を利用した攻撃
line 9-11: TFTP を用いたボットネットのダウンロード
line 12: 攻撃に用いられたセッションが終了
line 14-15: C&C サーバのドメイン名解決
line 17-18: IRC を用いた C&C サーバとの接続
C&C サーバとの通信内容を図 2.6 に示す.

サンプルトラフィック１の IRC サーバとの通信内容解説

line 1-10 : C&C サーバへのログインと制御用チャンネルへの参加
line 11-13 : 調査機がボットネット拡大のために ClassB のネットワークをスキャンしていると攻撃者に通知
line 14 : 調査機がチャンネルに接続 *line 15-19 :* 他のボットが C&C と接続・切断

第２のサンプル

1: >>**NICK [‘]—081832**
2: >>**USER vrwiphvp 0 0 :[‘]—081832** *3:* >>**JOIN ##damn## fucked** *4:* >>**USERHOST [‘]—081832**
5: >>**MODE [‘]—081832 -xt**
6: >>**JOIN ##damn## fucked**
7: *<<2 MODE [‘]—081832 :+iwx*
8: *<<:Nasty.example.net 302 [‘]—081832 :[‘]—081832=+ host@mydomain.jp*
9: *<<:[‘]—081832 MODE [‘]—081832 :-x*
10: *<<:[‘]—081832!host@mydomain.jp JOIN :##damn##*
11: >>**PRIVMSG ##damn## :.n.z.m. (root.p.l.g) .... Random Port Scan started on 192.168.x.x:445 with a delay of 5 seconds for 0 minutes using 250 threads.**
12: >>**PRIVMSG ##damn## :.n.z.m. (root.p.l.g) .... Random Port Scan started on 192.168.x.x:1433 with a delay of 5 seconds for 0 minutes using 250 threads.**
13: >>**PRIVMSG ##damn## :.n.z.m. (root.p.l.g) .... Random Port Scan started on 192.168.x.x:135 with a delay of 5 seconds for 0 minutes using 250 threads.**
14: *<<:Nasty.example.net 404 [‘]—081832 ##damn## :You need voice (+v) (##damn##)*
15: *<<:[‘]—791597!brhvcjdq@58.x.x.x JOIN :##damn##*
16: *<<:[‘]—217134!uijvmmjg@221.x.x.x QUIT :Connection reset by peer*
17: *<<:[‘]—738136!pmlaixd@otherdomain.jp JOIN :##damn##*
18: *<<:[‘]—349563!dcjydfo@221.x.x.12 QUIT :Ping timeout*
19: *<<:[‘]—086447!jnmcino@221.x.x.16 JOIN :##damn##*

図 2.6: C&C サーバとの通信例 1

2つめの感染トラフィックデータにおける tcpdump 出力結果の抜粋を図 2.7 に示す.

サンプルトラフィック 2 の解説
*line 1-4:*LSASS の脆弱性を利用した攻撃
*line 6-9:*FTP コントロールセッション．ポート番号は Well-known ポート番号とは異なり 15228 番ポートを使用
line 11-12: TFTP を利用した他の攻撃によるボットウェアのダウンロード
line 14-15: C&C サーバのドメイン名解決
line 17-20: IRC を用いた C&C サーバとの接続．ポート番号は Well-known ポート番号とは異なり 3000 番ポートを使用

サンプルトラフィック 2 では，FTP を用いたダウンロードが試行された．このトラフィック内容を図 2.8 示す.
トラフィック内容より，FTP を用いていると推定できるが，ボットウェアのダウン

1: 87.622934 59.x.x.x 192.168.1.100 TCP 1209 > microsoft-ds [SYN]
2: 87.623752 192.168.1.100 59.x.x.x TCP microsoft-ds > 1209 [SYN, ACK]
3: 87.659608 59.x.x.x 192.168.1.100 TCP 1209 > microsoft-ds [ACK]
4: 87.869841 59.x.x.x 192.168.1.100 TCP 1209 > microsoft-ds [FIN, ACK]
5: **/* FTP コントロールセッション */**
6: 88.508550 192.168.1.100 59.x.x.x TCP 1035 > 15228 [SYN]
7: 88.543588 59.x.x.x 192.168.1.100 TCP 15228 > 1035 [SYN, ACK]
8: 88.543913 192.168.1.100 59.x.x.x TCP 1035 > 15228 [ACK]
9: 89.569675 59.x.x.x 192.168.1.100 TCP 15228 > 1035 [PSH, ACK]
10: **/* TFTP ダウンロードセッション */**
11: 287.995906 192.168.1.100 59.x.x.x TFTP Read Request, File: Windows-spyware.exe
12: 312.307350 192.168.1.100 59.y.y.y TFTP Acknowledgement, Block: 454
13: **/* コントロールサーバの名前解決 */**
14: 330.010369 192.168.1.100 192.168.1.254 DNS Standard query A ircserver.example.biz
15: 330.047128 192.168.1.254 192.168.1.100 DNS Standard query response A 24.x.x.x
16: **/* コントロールサーバとの接続 */**
17: 330.048926 192.168.1.100 24.x.x.x TCP 1039 > 3000 [SYN]
18: 330.229591 24.x.x.x 192.168.1.100 TCP 3000 > 1039 [SYN, ACK]
19: 330.229872 192.168.1.100 24.x.x.x TCP 1039 > 3000 [ACK]
20: 330.229990 192.168.1.100 24.x.x.x TCP 1039 > 3000 [PSH, ACK]

図 2.7: ボット感染時のトラフィック例 2

1: *<<220 Reptile welcomes you..*
2: >>**USER 1**
3: *<<331 Password required*
4: >>**PASS 1**
5: *<<230 User logged in.*
6: >>**PORT 192,168,1,100,4,12**
7: *<<200 PORT command successful.*
8: >>**RETR eraseme_64865.exe**
9: *<<150 Opening BINARY mode data connection*
10: >>**425 Can't open data connection.**

図 2.8: FTP コントロールセッション例 2

ロードは失敗している．

サンプルトラフィック 1 の IRC サーバとの通信内容解説
C&C サーバとの通信内容を図 2.9 に示す．

```
1: >>NICK XP-3517925
2: >>USER nowiitojd 0 0 :
3: >>PONG :A1D1B3E6
4: >>JOIN #B.t[r1]N.t Dragon
5: >>USERHOST XP-3517925
6: >>MODE {XP}-3517925 +i
7: >>JOIN #B.t[r1]N.t Dragon
8: >>JOIN #ole1,#ole2
9: >>PONG :irc.example.com
10: >>PONG :irc.example.com
11: >>PONG :irc.example.com
12: ···以下, PONG の繰り返し···
```

図 2.9: C&C セッション例 2（2 つ目）

サンプルトラフィック 1 の IRC セッションとは異なり，この IRC セッションではサーバからの通信がない．通常の IRC プロトコルでは，*line 3* や *line 9-11* の PONG コマンドは IRC サーバからの PING コマンドに対する応答に用いられる．だが，この IRC セッションでは評価機から能動的に PONG コマンドが送られている．このため，サンプルトラフィック 2 のボットネットは，IRC プロトコルをベースとした独自のプロトコルで制御されていると言える．

2.6.5 ボットトラフィックのまとめ

本節ではボットネットの効果的な対策を思案するにあたり，実ネットワーク環境下におけるボットのトラフィックを収集し，解析した．その結果，主なボットトラフィックの特徴ととして以下の点を挙げる．

- ポート番号を用いたボット検知は困難
  多くの既存手法の対策は，サービスごとのよく知られたポート番号 (Well-Known ポート)[13] を用いて攻撃を防止することができる．しかし，多くのボットによる通信は Well-Known ポートとは異なるポート番号を用いている．

- 通信内容によるプロトコルの判別は困難
  ポート番号を用いることによるサービスの特定が困難であることを述べたが，同様に，通信内容を分析することによるプロトコルの判別も困難である．IRC 型ボットネットの場合，IRC 型という名称ではあるが，実際は RFC で規定されている IRC の通信とは異なる．これにより，通信内容を一定のプロトコルに照らし合わせることは困難である．

- フローの類似性
  本調査で扱った 2 つのトラフィックは，図 2.3で示したボットネットの感染順序に従って通信をしている．図 2.10 に 2 つのトラフィックで観測したボットネット加入までのフローを示す．ボットの通信ヘッダあるいはペイロードを用いて通信の性質を判断することは難しいが，通信フローの順序には類似性があると推測できる．

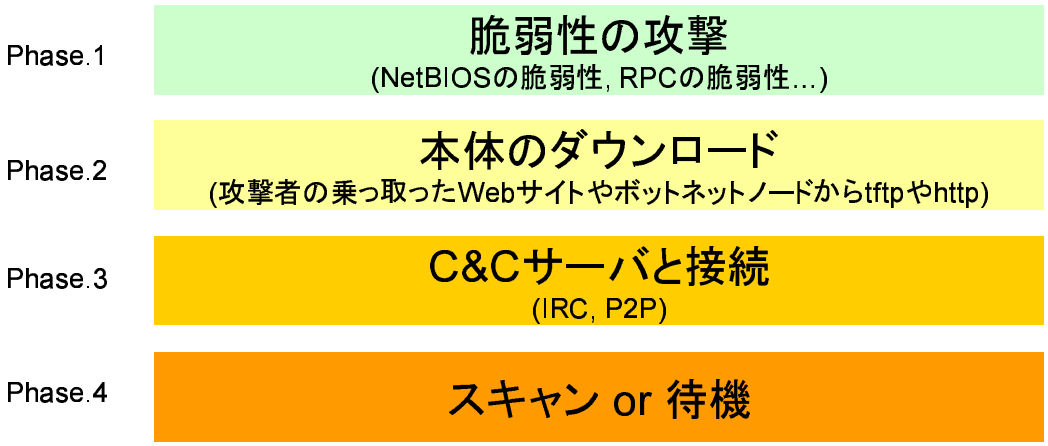

図 2.10: 調査で観測した定型フロー

2.7 まとめ

本章では，ボットネットに関する概要を説明し，特徴やインターネットにおけるトラフィック観測の例を示した．ボットネットは多くの操作が可能であり，C&C サーバの冗長化やボットウェアのアップデートなどの機能を持ち，既存の対策を困難にしている．ボットの通信するトラフィックの解析より，ボットの通信は様々であるが，通信フローの順序性に特徴があることを発見した．

第3章　ボット検知と課題

本章では，ボットネット対策の既存手法を整理し，各手法ごとの課題を述べる．整理した手法を踏まえたうえで，既存手法をミスユース型とアノーマリ型に分類し，それぞれの問題点を述べる．

3.1　既存のボット検知手法

本節ではボットネット対策の既存手法を整理する．

3.1.1　DNS による検知

DNS による検知は，多くのボットウェアが特定のホストとの接続にドメイン名を利用している特性を利用したボット検知方法である．近年のマルウェアの多くは，C&C サーバやボットウェア更新元の外部ホスト間の接続にドメイン名を使用している．そして，攻撃者の多くは動的な DNS のアップデートの仕組みを用いたダイナミック DNS[14] と呼ばれるシステムを用いることで，冗長化を図り，また対策を困難にしている．

そこで，特定サーバとの接続にドメイン名を利用する特性を利用し，ボットの検知を試みる研究が行われている．大学や教育・研究機関へのネットワークプロバイダである SURFnet[15] では，DNS 通信を調査することでマルウェアに感染したホストの検出を試みている [16]．この調査では，以下のような DNS の通信が観測されるホストはマルウェアに感染している傾向が強いと述べられている．

- マルウェアに利用されているドメイン名解決
- 頻繁に名前解決が行われるドメイン名解決
- 提供されているリゾルバ以外でのドメイン名解決
- 一般的に用いられないレコードでのドメイン名解決

この報告では，ネットワーク内部の該当ドメイン名に対するホストを摘発したり，ダイナミック DNS の組織に該当レコードの削除を要請できると述べられている．

3.1.2 ミスユース型 NIDS

管理ネットワークに接続される攻撃やスキャン活動の検知，あるいは遮断を目的としたシステムは，セキュリティデバイスと呼ばれる．Network Intrusion Detection System (NIDS) は，特に内部ネットワークにおけるネットワークトラフィックを監視するセキュリティデバイスである．

ミスユース型 NIDS は NIDS でも特に悪意のあるトラフィックに対してのシグネチャと呼ばれる特徴データ郡を予め用意しておき，トラフィックがそれらに一致するかを比較するセキュリティデバイスである．シグネチャを用いるミスユース型は，シグネチャが登録されている活動に対しては有効であり，適切にチューニングされたシグネチャを用いることで正常なパケットの誤検知なしに既存の攻撃を検知する．このため，既知のボットウェアの感染や，特定のシグネチャが一致する活動に関しては有効に利用できる．

ただし，既存のシグネチャ型 IDS では未知の攻撃の検知が困難である．未知の攻撃に対して，honeycomb[17] や EarlyBird[18][19] といった研究をはじめ，シグネチャ型 IDS のためのシグネチャを動的に生成する機構の研究 [20] が活発に行われている．これらの手法を用いることによって，従来のシグネチャ型 IDS では困難であった未知の攻撃検知が可能になり，ボット検知への応用にも利用できる可能性がある．しかし依然としてシグネチャを用いるという性質は変わらないため，亜種に十分な対応を行えない．

3.1.3 セッションベース型 NIDS

セッションベース型 NIDS[21] とは，ミスユース型 NIDS に加えて，攻撃の成否を判定する機能を付与した NIDS である．従来のミスユース型 NIDS は，IP パケットに着目し，攻撃の成否に関わらず警告する．しかし，失敗した攻撃は管理者にとって優先順位の高いイベントではなく，攻撃の成否を判断するにはログの解析という段階が必要であった．セッションベース型 NIDS では，OSI 基本参照モデル第 4 層トランスポート層 (L4) のフローに着目し，攻撃に対する応答パケットも監視する．これにより，攻撃が成否が判断できるイベントに関しては，重要性の高いイベントのみを得ることが可能となった．セッションベース型 NIDS によって，管理者の付加を軽減した効率的なボットの検知が可能となる．

3.1.4 アノーマリ型 NIDS

アノーマリ型とは異常検知型とも呼ばれ，様々なネットワークのパラメータを用いることで，ボットの検知を可能とする．例えば，トラフィックのパケット数に注目し，サービス毎の平均的なアクセス傾向を保持しておくことで，外部ネットワークから特定のサービスに対してのアクセスが急激に増えたことなどを検知できる．この例では，ボットの DDoS 攻撃活動や感染活動を検知できる可能性がある．また，ペイロード内

のデータバイト自体の分布に注目する研究 [22] も行われている．アノーマリ型 NIDS はミスユース型 NIDS に比べて，パケット内の特定の情報に着目するのではなく，トラフィックのデータに含まれるバイトに着目しており，亜種に対応できる可能性を持つ．

しかし，いずれのアノーマリ型も実ネットワーク上で正常時と異常時を明確に区別することが難しいと言われている．一時的なアクセス集中，新しいアプリケーションの普及や，サービスの開始時などに，ネットワーク管理者が本来は正常と判断するトラフィックを攻撃として検知してしまうことが懸念される．また，既知のボットに対しても確実な検知ができるとは言えない．

これらの理由からアノーマリ検知は，現在でも実用的な手法とはいえず，有効なボット検知手法とはいえない．

3.1.5 パケットフィルタ型ファイアウォール

パケットフィルタ型ファイアウォールは，IP ヘッダと L4 のプロトコルヘッダを用いてパケット単位での通信を制御するシステムである．それにより，特定のサービスや特徴的な攻撃や活動を防ぐことができる．また，特定のルールに一致したログを保持することで，異常な通信傾向のホストを発見することも可能である．

しかし，ネットワーク内の不特定ホストに対してパケットフィルタ型ファイアウォールを適応すると，正常なサービスの利用に影響が生じる場合がある．

3.1.6 パターンマッチ型アンチウィルスソフト

アンチウィルスソフトはエンドホストにおいて動作するソフトウェアである．特にパターンマッチ型アンチウィルスソフトでは，ファイルがホストに保存されるときや，実行時に，ファイルが定義されたマルウェアのシグネチャと一致するかを監視する．シグネチャを用いたマルウェアの一部は，ミスユース型 NIDS でも検知可能である．しかし，エンドホストの検知は圧縮されたファイルなどでも，メモリに展開された際の監視ができるなど，物理ネットワークにおける検知に比べて，ホストの詳細な情報を利用した攻撃の監視ができる．

3.2 既存手法の問題点

本節では既存手法を大きく**ミスユース型検知**と**アノーマリ型検知**の 2 つに分類し，それぞれの問題点を述べる．

ミスユース型検知の問題点

ミスユース型検知は，ボットが通信する際のパケットに含まれる特定の情報に着目して検知する．DNS による検知，ミスユース型 NIDS，セッションベース型 NIDS，パターンマッチ型アンチウィルスソフトがこれに該当する．

ボットの通信に含まれる特定の情報に着目した検知手法は，通信の内容が変化しない対象には有効である．例えばコンピュータウィルスは自らの複製を拡散させるのみなので，活動の手順や通信内容は常に同一である．亜種の発生によって通信内容が変化するが，亜種の発生頻度は低い．

これに対して多くのボットネットでは，ボットウェアの亜種が高い頻度が生成されている．亜種は感染活動のための攻撃手法を変更や，C&C との通信プロトコルの変更，ボットウェア自身の暗号化がなされている．ミスユース型検知ではこれらの特徴を手がかりに検知しているが，亜種の発生によって特徴がわずかに変化しただけでも，検知ができなくなってしまう．これに対応するためには，検知のための特徴を示したシグネチャの更新が必要だが，シグネチャを作成するための調査にかかる時間や，配布開始からシグネチャの受信までの時間差があるため，確実な検知が望めない．

また，ボットウェアの配布では Packer をボットウェアの暗号化に用いている．暗号化されたボットウェアの特徴をシグネチャとして定義するのは困難であり，ミスユース型の手法による検知は難しい．

このように，多くのボットネットでは攻撃者によりミスユース型検知への対策がされている．そのため，ミスユース型検知手法による効果的なボット検知は期待できない．

アノーマリ型検知の問題点

アノーマリ型検知は通常時との差分に注目することで，ボットを検知する．特定の情報に依存しない点はボットの亜種に対して検知できる可能性がる．また，3.1.1節で述べた SURFnet の DNS を用いたマルウェアの検知手法は，多くのボットネットがダイナミック DNS を用いていることからも有効な手法と考えられる．

しかし，アノーマリ型検知の問題点として，様々なアノーマリ型の検知手法が研究されているにも関わらず，誤検知が多発してしまう問題点が解決されていない．多くの研究で評価実験に用いられるネットワークは，当該研究にとって理想的な環境である．しかし実際のネットワークは運用形態やネットワークの構成が様々であり，アノーマリ型検知を有効に活用するためにはパラメータの調整が非常に難しい．また，一時的なアクセス集中やサービスの開始時などで，ネットワークのトラフィック傾向も変化する可能性がある．したがって，正常時と異常時を明確に区別することが難しく誤検知を多発しやすい．そのため，アノーマリ型の検知を実際のネットワークのボット対策手法として用いることは難しい．

3.3 まとめ

本章では，ボット検知の手法を 3 つに分類し，その上で具体的な検知に用いられる手法の概要を述べた．しかし，いずれの手法も，何らかの課題を抱えており，ボットネットに対して効果的な対応を取れるとはいえない．

次章では，本章で述べた課題をもとに，ボット検知で必要な要件について述べる．

第4章　設計

本章では，これまでに挙げた既存手法の問題点を解決するためのボット検知の要件を定義する．また，要求を満たすアプローチを提案し，アプローチを用いた機構を設計する．

4.1　ボット検知の要件

本節ではボット検知手法を設計するために必要な要件を定義する．

4.1.1　専門的な知識を必要としない運用

ボット検知はボットに関する専門的な知識がなくとも可能である必要がある．既存の手法を用いたボット検知には，ボットネットに関する知識と経験が必要となる．例えば，ネットワーク管理者が自らトラフィックを監視し分析する手法であれば，ボットのトラフィックに熟知している必要がある．だが，全てのネットワーク管理者がボット検知のための知識，経験や技術があるわけではない．ボット検知は特殊な知識をもたなくとも検知をできなければならない．

4.1.2　シグネチャを用いないボットトラフィックの捕捉

ボットトラフィックの捕捉とは，ボットの通信を検知できることである．ボットは亜種が頻繁に発生し，既存の手法によるボット検知を困難にしてきた．そのため，ボットの通信はパケットのヘッダやペイロードのシグネチャ情報による判断は困難である．あるボットは，C&C サーバとの接続に IRC プロトコルに従ったものを用いる．しかし，2.6節での調査で示したように IRC プロトコルに従わないボットネットも存在する．このように，シグネチャを用いた検知手法では，ボットネットの通信を捕捉することができない．そのため，シグネチャを用いないボットネット通信の捕捉手法が必要となる．

4.2 アプローチ: フロー順序を用いるボット検知

本節では，アプローチについて述べる．本研究で提案する手法の概念図をを図 4.1に示す．図中左は，いくつかのボットが感染から C&C サーバに接続するまでの通信を示

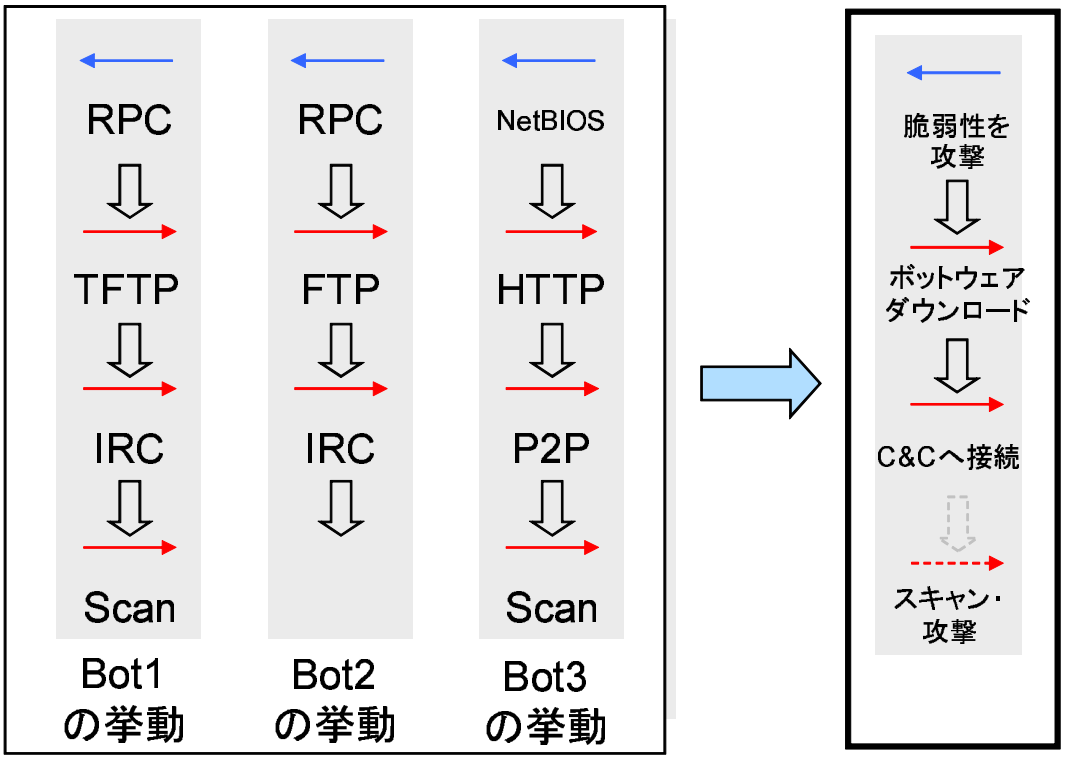

図 4.1: ボットフローの抽象化

している．図に示された 3 つのボットは通信の性質が異なるボットであり，それぞれ異なった通信でボットウェアに感染する．図中右は，これらのボットの通信方向と意味を示している．2.6.5節と図から分かるように，フローが異なっても，各ボットの感染活動や C&C サーバの接続といった通信内容は同じである．

また，従来の対策手法の問題点より，特定のパターンを用いる検知手法はボットの検知に不向きである．そこで，実ネットワークのトラフィックをフローとして解釈する際には，通信の方向や通信の送受信量の割合など，具体的なトラフィックの内容を用いない．

本研究では，柔軟性を持ったボットネット通信の捕捉を実現するために，フロー順序を用いたボット検知を提案する．2.6.5節において**フロー順序の類似性**について述べた．ボットネットのトラフィックは，フロー単体での識別は困難である．しかし，連続したフロー順序として捉えることで，ボットネットの特徴を持ったトラフィックとして検知できる．本論文ではボットの検知手法として，次節以降で述べるフロー順序を用いたボット検知手法を用いる．

4.3 全体概要

本節ではボット検出機構の概要を述べる．図 4.2にシステムの概要図を示す．本機構

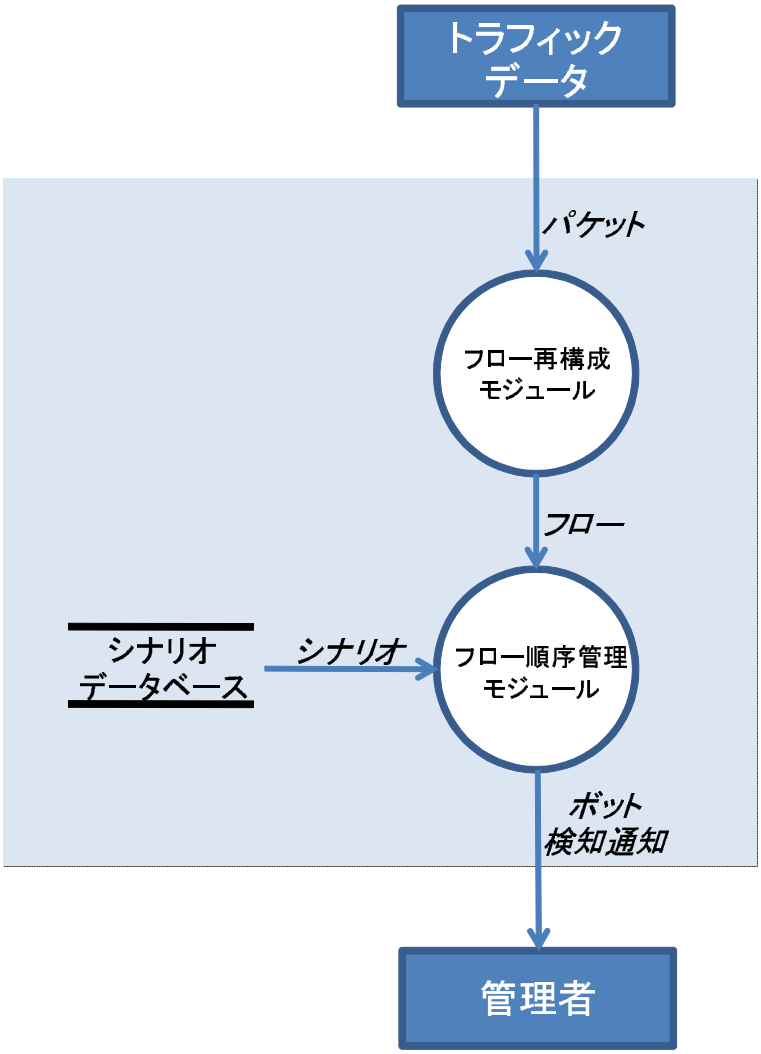

図 4.2: 本機構の設計概要

はフロー再構成モジュール，フロー順序管理モジュールとシナリオ定義によって構成されている．**フロー再構成モジュール**は，蓄積されたトラフィックデータあるいはリアルタイムトラフィックデータの入力を受け，パケットをフローとして再構成する．再構成されたフローはホストごとに分類され，フロー順序管理モジュールに渡される．**フロー順序管理モジュール**は，フロー再構成モジュールからホスト毎に分類されたフローとシナリオ定義のシナリオを比較する．シナリオと完全一致すれば，ボットを検出したと管理者に通知する．**シナリオ定義**は，フロー順序管理モジュールでフロー順序の比較に用いるシナリオを定義する．本機構では予めボットの感染時や活動時に発生するフローの順序をシナリオとして定義する．

4.4 フロー再構成モジュール

本節では，フロー再構成モジュールの設計について述べる．本モジュールは，管理ネットワークのトラフィックデータを取得し，パケットを，内部の各ホスト毎にフローとして再構成する．再構成を行ったフローの情報はフロー順序管理モジュールに通達される．

4.4.1 各プロトコルによるフローの扱い

本節では，フロー再構成モジュールにおけるフローの扱いについて述べる．図 4.3 にパケットからフローへの変換例を示す．図中左では hostA が宛て先ポート 80 番の TCP

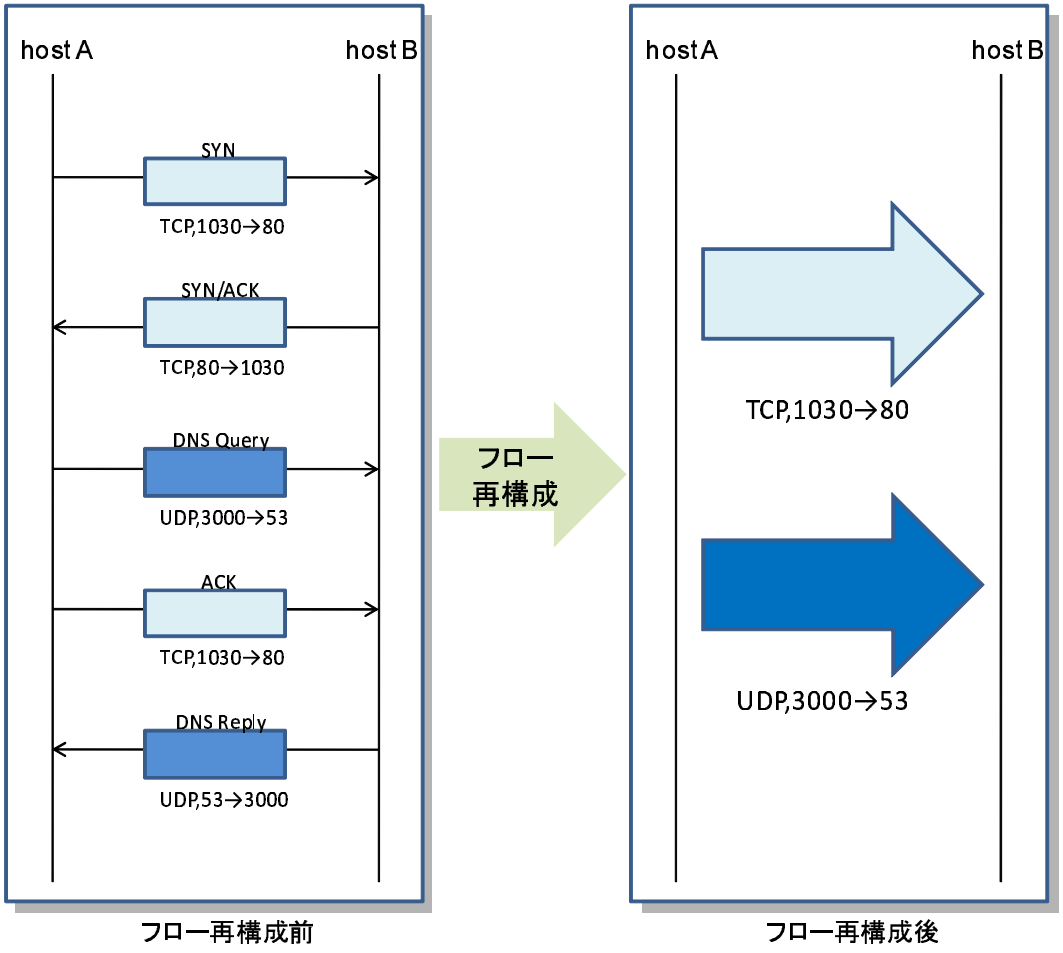

図 4.3: パケットからフローへの変換

セグメントと宛て先ポート 53 番の UDP データグラムの 2 つの通信をしている．図中の四角はパケットを示す．トラフィックデータでは 2 つの通信はパケットの集合でしかないが，フロー再構成モジュールではこれらのパケットをフローとして再構成する．図中右ではそれぞれの通信をフローとして再構成されたようすを示す．

IP 通信の多くは TCP, UDP, ICMP であるが，プロトコル毎に通信の特徴があり，フロー化の手順が異なる．本モジュールでの各プロトコルの扱いについて述べる．

TCP は，コネクション型のプロトコルであるため，セッションの確立である 3-way ハンドシェイクからセッションの終了である FIN フラグまでを取得することでフローとして再構成できる．だが，いくつかの状態や特定のプロトコルでは，フローの開始から終了までを把握するのには工夫が必要となる．例えば，ホストやネットワークの異常によって対向ホストへの接続性が失われてしまい，正常な終了手順を踏まずに終了することが考えられる．異常な TCP 通信時には最後に監視できたパケットからの経過時間が一定時間を過ぎた際に，通信がタイムアウトしたと判断する必要がある．

UDP は，コネクションレス型の通信であるため，通信を監視するだけではセッションの開始と終了を判断することが出来ない．そのため，常に最後に監視できたパケットからの経過時間が一定時間を過ぎた際に，通信が終了したと判断する．

ICMP は，ICMP ヘッダに含まれる**タイプ**と**コード**によって，通信の性質が異なる．表 4.1に主な ICMP メッセージとフローの扱いについて示す．例えば ping アプリケーションで用いられる ICMP メッセージでは，応答と要求をフローとして扱う．宛先到達不可や TTL 超過といった例外ための ICMP メッセージは，受け取ったメッセージを１フローとして扱い，また，例外が発生したフローに関する情報を破棄する．

表 4.1: 主な ICMP メッセージとフローの扱い

タイプ	コード	説明	フローの扱い
0	0	ICMP Replay	それぞれのパケットを利用した ping アプリケーションのフロー
8	0	ICMP Request	として扱う
3	全て	送信先に到達不可	フローとしての処理は行わない．ただし，過去の TCP あるいは UDP フローを異常終了させる
11	0	TTL 超過 (traceroute)	フローとしての処理は行わない．ただし，過去の TCP あるいは UDP フローを異常終了させる．

4.4.2 フローとして保持される情報

本節では，フロー再構成モジュールにおいて，フローが保持する情報について述べる．本機構では，ボットトラフィックの検知に用いる情報をフローが保持する．パケットに含まれる情報を適切に保持するために，フローが保持すべき情報を表 4.2に示す．

通信元，送信先 IP アドレスは，特に通信の方向を判断するのに用いる．本機構を設置するネットワークをあらかじめ定義することで，通信がネットワーク内部との通信なのか，外部との通信なのかを判断できる．**プロトコルタイプ**は，L4 のプロトコルを示し，他の項目と組み合わせることによって，ボットの検知に利用できる．**通信元，送信先ポート番号**は，プロトコルタイプと合わせてサービスの特定に用いる．亜種の

表 4.2: フローが保持すべき情報

ヘッダ	主な利用目的
送信元 IP アドレス	通信ホストの特定
送信先 IP アドレス	通信ホストの特定
プロトコルタイプ	サービスの判断
送信元ポート番号	通信のサービス特定
送信先ポート番号	通信のサービス特定
フローの継続時間	長期間にわたるセッションの発見
送信バイト数	転送量による通信種別の判断
受信バイト数	転送量による通信種別の判断
送信パケット数	転送量による通信種別の判断
受信パケット数	転送量による通信種別の判断
フロー開始時間	前のフローからの経過時間の把握
フローの状態	不正終了したフローの把握

存在により，ポート番号がサービスの特定に利用できるとは限らない．しかし，ポート番号はいくつかのサービスの特定を可能とする．１つはドメイン名の解決の解決に使われる DNS である．多くのボットにおいても，ドメイン名の解決にはシステムのドメイン名解決機構が用いられているため，ドメイン名の解決には DNS の Well-known ポート番号が用いられている．また，現在の多くのボットウェアが，そのダウンロードに FTP と TFTP の Well-known ポート番号が用いられている事が調査により明らかになっている．**フローの継続時間**は，フローの内容やポート番号によらない検知に利用する．フローの継続時間を用いることで，C&C サーバとの通信の特定を可能にする．**送信，受信バイト数**は，フローの内容やポート番号によらない検知に利用する．例えば，ダウンロードフローなどは外部から内部に対してのデータ転送が多いと予想されるため，外部から内部への転送のバイト数や割合を比較することでダウンロードフローの推測が可能である．**送信，受信パケット数**は，送信，受信バイト数と同様の目的に利用する．**フローの開始時間**は，また，フローの継続時間とあわせることで，フローの終了時間を把握するためにも用いる．**フローの状態**は，特に TCP フローで有効な項目である．セッションが正常に終了されたか，異常終了をしたかといった情報を用いる．ただし，UDP においては，正常なフローにおいても終了が明確ではないためこの項目は利用できない．

4.5 フロー順序管理モジュール

本モジュールは，フロー管理モジュールから通達されたフロー情報を元に，各ホストが持つシナリオとの一致状況を管理する．シナリオと完全に一致した場合，その情

報は管理者に通達される．

図 4.4 にシナリオの例を示す．本モジュールでは各ホストごとにシナリオの状態を保

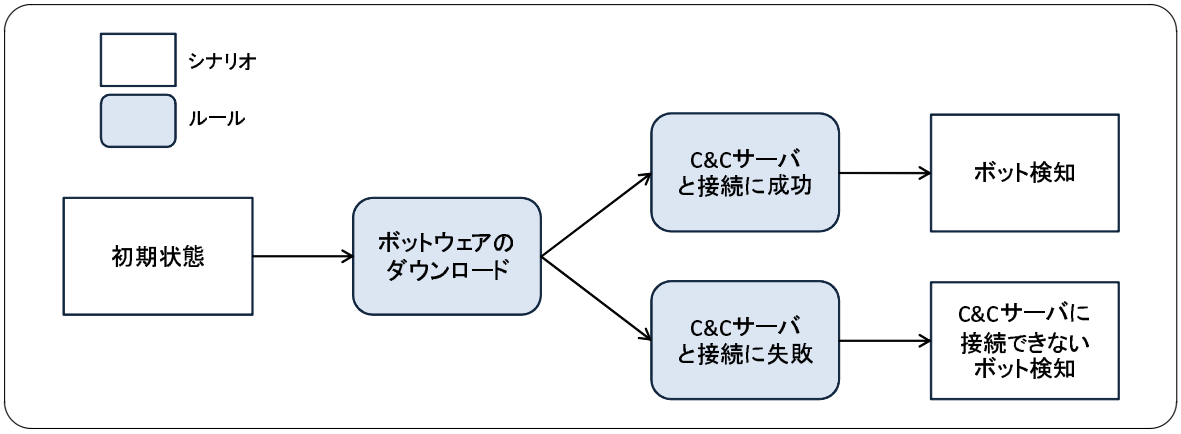

図 4.4: パケットからフローへの変換

持する．はじめは全てのホストのシナリオは初期状態にあり，フローを受け取ると，次のルールに一致するかを比較する．**ルール**とは条件となるフローの集合である．ルールはボットの各段階で行われるフローを集約するために用いる．ルールの例として，ボットウェアのダウンロードルールは TFTP か FTP のフローと定義される．

4.5.1 内部ネットワークの情報管理

本手法では，内部ネットワークのホスト毎に現在の**シナリオステータス**を保持する必要がある．シナリオステータスとは，ホストがどのシナリオの段階に到達しているかを指す．

シナリオステータスは次のルールとの一致したときに加え，タイムアウトによっても変更が生じる．このタイムアウトの判断に用いる時間も保持しておく必要がある．

4.5.2 フロー定義との比較

フロー管理モジュールから通達されたフロー情報には，表 4.2 の情報が含まれる．**フロー定義**はこれらの情報の条件を組み合わせた定義である．シナリオの各段階では，次のシナリオの段階に遷移するための条件として，フロー定義が設定されており，通達されたフローは現在の状況に適切なフローであるかを判断する．

第5章　実装

本論文では，提案手法に基づくプロトタイプを実装した．本章では，まず5.1節で実装の概要を述べてから，各モジュールの詳細について述べる．

5.1　実装概要

本機構は，表5.1 に示す環境で実装を行った．本手法のプロトタイプはPHP 言語とzsh のシェルスクリプトを用いて約550 行で実装した．

表 5.1: 実装環境

OS	FreeBSD 5.5
プログラミング言語	PHP 4.4.4 zsh 4.3.2
パケット収集ライブラリ	libpcap 0.8.3
パケット収集ソフトウェア	tcpdump 3.8.3
フロー集約ソフトウェア	Argus 2.0.6.fixes.1

本実装の概要を図5.1に示す．本機構は，tcpdump により，pcap 形式で保存されたトラフィックデータを入力として受け付ける．受け付けたトラフィックデータは，フロー集約ソフトウェアにより，フローとして再構成され，PHP とシェルスクリプトによって実装されたシナリオ順序管理モジュールに渡される．シナリオ順序管理モジュールで定義されたシナリオに一致すると，本機構は，結果を標準出力に表示する．

5.2　フロー再構成モジュール

本節では，フロー再構成モジュールの実装について述べる．フロー再構成モジュールはパケット単位でのネットワークトラフィックを元にフロー集約を行い，フロー順序管理モジュールにフローを通達するモジュールである．

本実装では，フロー再構成モジュールとしてフロー集約ソフトウェア argus[23] を用いる．argus はリアルタイムのトラフィックデータ，あるいは蓄積されている pcap 形式のネットワークトラフィックデータからフロー集約を行い，フローの出力や集計をするソフトウェアである．argus の主な特徴としては，以下の点が挙げられる．

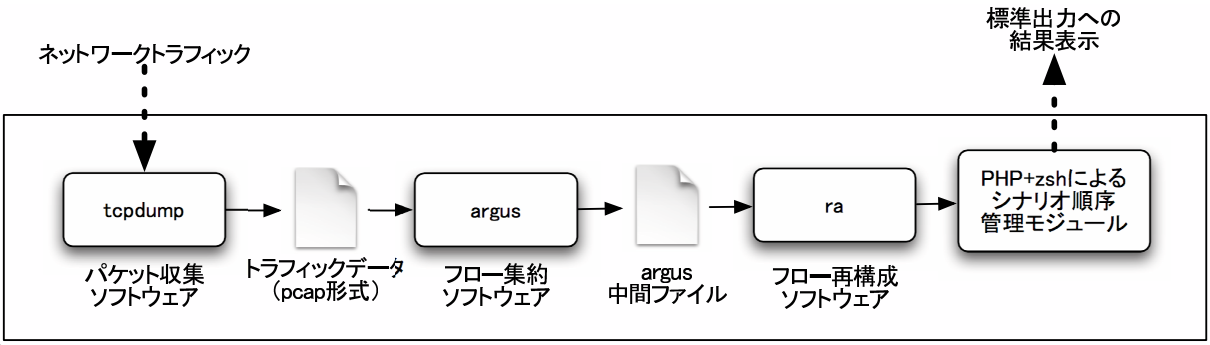

図 5.1: プロトタイプ実装の概要

- GNU ライセンスで開発が行われているオープンソースプロジェクト
- フロー集約が可能
- 集約した結果をリアルタイムに TCP ソケットで転送可能

argus はフローの再構成時に，フローの終了ステータスもフロー順序管理モジュールに通達する．プロトコルごとの主な argus の処理内容と，フロー順序管理モジュールに通達する状態は次の通りである．**TCP** の場合，フロー開始後最初の状態は INT として通達する．その後，継続したフローは CON として通達され，正常な終了は FIN と通達する．一方のノードが切断されるなど通信が不正に終了した場合は TIM と通達する．**UDP** の場合，フローの開始と終了は TCP に比べて明確でない．argus が現在管理していないフローが観測された場合を通信の開始として INT と通達し，一定時間が経過した通信は終了したとして TIM を通達される．**ICMP** の場合は，そのタイプとコードにより処理が異なる．ping アプリケーションで用いられる echo 要求と echo 応答はそれぞれ 1 つのフローと定義される．また，宛て先到達不可メッセージでは，メッセージを通達し，該当するフロー情報を保持していれば破棄する．

5.3 フロー順序管理モジュール

本節では，フロー順序管理モジュールの実装について述べる．フロー順序管理モジュールはフロー管理モジュールからフロー情報を受け取り，ホストごとのシナリオ段階を管理するモジュールである．

5.4 シナリオ記述言語

本機構を用いるネットワーク管理者がシナリオを柔軟に編集できるために，本実装ではシナリオを設定ファイルとして外部に記述できる．付録中の図 B.1に本実装におけるシナリオ記述言語の完全な ABNF 表記を記載する．

5.5 結果表示

本節では，今回の実装において検知されたシナリオをネットワーク管理者に通達するメッセージについて述べる．フロー順序管理モジュールでシナリオが達成されると，本実装は各フローに関する情報を出力する．ある実際のボット検知際の出力例を図 5.2に示す．

```
1: Detected [BotDetect]
2: REPORT BEGIN
3:
4:  Detected Scenario [download] by Flow[tftp]
5:  ->StartTime: Thu Nov 9 21:54:47 JST 2006
6:  ->udp 192.168.1.100:1034->10.1.1.1:69 (17sec)
7:  ->Incoming Traffic: 153606bytes [9036byte/s] 278pkts [17pkt/s]
8:  ->Outgoing Traffic: 12766bytes [751byte/s] 277pkts [17pkt/s]
9:
10:  Detected Scenario [connection] by Flow[port6667]
11:  ->StartTime: Thu Nov 9 21:57:00 JST 2006
12:  ->tcp 192.168.1.100:1037->10.100.100.100:6667 (0sec)
13:  ->Incoming Traffic: 255bytes [0byte/s] 4pkts [0pkt/s]
14:  ->Outgoing Traffic: 217bytes [0byte/s] 3pkts [0pkt/s]
15:
16: REPORT END
```

図 5.2: シナリオ一致時の結果表示例

図の出力結果について説明する．

line 1: 検知されたシナリオ名．図中では BotDetect.

line 4-8: 1 番目のフロー．download フローとして TFTP を検知した．

line 10-14: 2 番目のフロー．connection フローとして port6667 を検知した．

各フローに関する出力は図 5.3に従う．フローの出力には，4.2に示したフローが保持する情報が含まれている．

Detected Scenario [一致したフロー] by Flow[一致したルール]
->StartTime: フロー開始時間
->tcp 内部ホストの IP アドレス:ポート番号 <-外部ホストの IP アドレス:ポート番号 (フローの継続時間)
->Incoming Traffic: 外部からの転送バイト数 外部からの転送パケット数
->Outgoing Traffic: 外部への転送バイト数 外部への転送パケット数

図 5.3: 結果表示のシナリオ一致出力例

第6章　評価

本章では，本機構が十分なボット検知の機能を有するかを評価する．

6.1　評価環境

6.1.1　機器環境

本評価に使用した機器の環境を表6.1に示す．

表 6.1: 評価に用いた機器の環境

プロセッサ	Intel(R) Xeon(TM) CPU 3.06GHz
メモリ	4Gbyte
OS	FreeBSD 5.5
ネットワーク	Broadcom BCM5703 Gigabit Ethernet

6.1.2　シナリオ定義ファイル

本評価でボット検知に用いたシナリオファイルを図6.1 に示す．

シナリオの解説

本評価で用いるシナリオ定義ファイルには４つのシナリオが含まれる．１つ目のシナリオの *botdetect* は，明らかにボットと推測できるフロー順序を定義しており，*download* フロー群の後に *connection* フロー群が発生するものとする．２つ目のシナリオの *inactive-botdetect* は DNS 障害による C&C サーバへ接続できないボットの検知を定義しており，*download* フロー群の後に *connection_fail* フロー群が発生するものとする．他の２つのシナリオである *downloaded-botdetect* と *downloaded-inactive-botdetect* は，既に用意されたボットウェアの検知を試行するためのシナリオであり，それぞれ，*connection* フロー群と *connection_fail* フロー群が発生するものとする．

download フロー群は *ftp* フローあるいは *tftp* フローの発生を条件とする．現在，把握しているボットのダウンロードにはそれぞれ Well-Known ポートが使用されているため，ポート番号を明示的に指定した．*ftp* フローは TCP のポート 21 番の外向けフローであり，*tftp* フローは UDP のポート 69 番の外向けフローである．

```
1: internal = 192.168.0.0/255.255.255.0
2:
3: scenario[botdetect] = download -> connection
4: scenario[inactive-botdetect] = download -> connection_fail
5: scenario[downloaded-botdetect] = connection
6: scenario[downloaded-inactive-botdetect] = connection_fail
7:
8: rule[download] = ftp, tftp
9: rule[connection] = port6667, outtcp_over30sec
10: rule[connection_fail] = dns_excess
11:
12: flow[ftp] = tcp and port 21 and OUT
13: flow[tftp] = udp and port 69 and OUT
14: flow[dns_excess] = udp and dport 53 and timeout 30 and count 20 and OUT
15: flow[port6667] = tcp and dport 6667 and OUT
16: flow[outtcp_over30sec] = tcp and port 6667 and duration < 30 and OUT
```

図 6.1: 評価に用いたシナリオ定義ファイル

connection フロー群は *port667* フローあるいは *outtcp_over30sec* の発生を条件とし，*port6667* フローは TCP のポート 6667 番の外向けフローである．*outtcp_over30sec* フローは TCP の 30 ポートに関わらない 30 秒以上継続されている外向けフローである．

connection_fail フローは *dns_exccess* フローを条件とする．*dns_exccess* フローは UDP ポート 53 番のフローが 30 秒以内 20 フロー発生するものをいう．

6.2 評価 1: 未知のボット検知

本節では，本機構が未知のボットの検知が可能かを評価する．本評価の手順を示す．

未知のボット収集

本評価では，インターネット上で脆弱性を持ったホストをエミュレーションすることで，実際のボットウェアをインターネットから評価機上にダウンロードする．ボットウェアの採取には Nepenthes[24] を用いる．Nepenthes は，よく知られた脆弱性をエミュレートし，一部の ShellCode のエミュレーションを行い，そして，マルウェアの採取や他のホストからのマルウェアのダウンロードが可能な低インタラクション型のハニーポットである．Nepenthes は脆弱性のエミュレートに加えてマルウェアをダウンロードする．

今回，ボット収集ホストは，/24 に分割された研究ネットワークに所属しており，ドメイン名の登録は行われているが，Web サービスやメールサービスなど，対外的なサービスは行っていない．

ボット検知

次に述べる既存の検知機構および本実装を用いて収集したボットウェアに対してボット検知あるいはボットウェア検知をする．ボット検知の手順については次に述べるとおりである．

- Snort
  Snort[25] は代表的なミスユース型 NIDS である．Snort での検知に用いるシグネチャには Snort で標準に用意されているシグネチャファイルに加えて，BLEEDING-EDGE[26] を用いる．BLEEDING-EDGE はインターネットの脅威に対するシグネチャデータベースで，セキュリティ専門家グループにより，維持されている．このシグネチャデータベースは Snort の運用に広く使われている．

  Snort はネットワークトラフィックを用いてボット検知をする．そのため収集したボットウェアを直接ボットかどうか判定することができない．本評価ではネットワークに接続された実機上でボットウェアを動作させ，ボットウェアのトラフィックを再現し，Snort でのボット検知を試みる．

- Norton Internet Security 2004
  Norton Internet Security 2004(NIS) は，Symantec[27] から販売されている Windows 上の総合セキュリティソフトウェアである．NortonIS には，パターンマッチング型アンチウィルスとパーソナルファイアウォールの機能が同梱されている．

  パターンマッチング型アンチウィルスの機能はホスト上の既存のソフトウェアを定期的に検査し，またホストに新しく作成されるファイルを検査する．本評価では，採取したボットウェアを NortonIS の評価用ホストにコピーした際や，実行した際にボットの検知が行われるかを確認する．

  パーソナルファイアウォール機能はホストで動作しているソフトウェア毎にネットワークへのアクセス権を設定できるソフトウェアである．未知のソフトウェアがネットワークへのアクセスを試みた場合，NortonIS のパーソナルファイアウォールはホスト管理者にネットワークの接続を許可するかを確認する．

  以上の 2 つの機能についてボット検知が可能か評価する．

- KASPERSKY Free online virus scan
  KASPERSKY Free online virus scan[28](KAV) は，Web を介して KASPERSKY 社のウィルススキャンサービスを受けることの出来るサービスである．本サービスを利用することで，Web を介してパターンマッチ型アンチウィルスソフトのサービスを利用することが出来る．KAV の評価は，収集したマルウェアを Web 上からアップロードし，マルウェアとして検知が行えるかを評価する．

- 本機構
  本機構はこれまでに述べたように，ネットワークベースで動作するボット検知機構である．ボット検知のシナリオには，6.1.2節に前述した定義を用いる．ただし，

定義の解説で述べたように既にダウンロードされたボットを検知対象とするために，*downloaded-botdetect* あるいは *downloaded-inactive-botdetect* の検知をもって，検知がされたと判断する．

本手法はネットワークトラフィックを利用したネットワークベースの検知であるため，Snort と同様に，ネットワーク上の実機で実際にボットウェアを動作させ，そのネットワークトラフィックに対してボット検知を試みる．

6.2.1 評価結果

評価結果を示す．本評価でのボット収集は 2007 年 1 月 13 日 0 時から 24 時にかけて行った．また，NIS 及び KAV のシグネチャ更新と調査は 2007 年 1 月 14 日 6 時に行った．本評価の結果を表 6.3に示す．

表 6.2: 未知のボット検知

	本機構	NIS	KAV	Snort
サンプル 1	downloaded-botdetect	×	×	×
サンプル 2	downloaded-inactive-botdetect	×	Packed.Win32.CryptExe	×
サンプル 3	downloaded-botdetect	×	Trojan-Spy.Win32.Agent.ct	○

各項目について説明する．

ファイル名

Nepenthes によって収集されたマルウェアの名称．

本機構

本機構によるボット検知の結果を示した．表中には検知が成功したものに関しては，検知されたシナリオ名を示した．

NIS

NIS によるボット検知の結果を示した．欄中に文字列が記入されているものは，欄中のボットウェアとアンチウィルスによって検知が行われたことを示す．パーソナルファイアウォールにより検知が行われたボットに関してはその旨を示すが，今回は該当するボットは収集されなかった．検知が行われなかったものに関しては✕と示した．

KAV

KAV によるボット検知の結果を示した．欄中に文字列が記入されているものは，欄中のボットウェアとアンチウィルスによって検知が行われたことを示す．検知が行われなかったものに関しては✕と示した．

Snort Snort によるボット検知の結果を示した．本結果はトラフィックデータを解析した結果のログファイルを目視することで確認した．Snort では，多くのシグネチャが用意されており，それぞれのシグネチャについて優先度が定められている．本表では，ボットウェア実行後に発生したイベントの中で最も優先度の高い数値を示す．また，検知が行われなかったものに関しては×と示した．

各ボットウェアの検知結果について述べる．

サンプル１
本ボットウェアは既存手法では検知が行えず本機構でのみ検知した．しかし，長期の TCP セッションを C&C サーバと確立していたことから，機能するボットネット (downloaded–botdetect) として本機構の検知した．尚，本論文執筆時点では既存研究での検知ができ，サンプル１は SDBot の亜種と検知された．

サンプル２
本ボットウェアは既存手法でも検知された．サンプル２は，接続先 C&C サーバと接続ができず，拡散活動も行わなかったため，Snort では検知が行われなかった．本機構では，短時間に多数の DNS 要求フローが検知されたため，機能しないボットネット (downloaded-inactive-botdetect) として検知された．

サンプル３
本ボットウェアは既存手法でも検知された．また，長期の TCP セッションを C&C サーバと確立していたことから，機能するボットネット (downloaded–botdetect) として本機構においても検知した．

6.3 評価 2: 誤検知率と検知率

評価目的
本節では，ボット検知の容易さを示すために，ボット検知を定量的に評価する．また，同環境において同時に既存の手法を用いた場合にどのような出力が得られるかを調べる．

6.3.1 評価方法

本評価では，2.6節の調査時に収集した pcap 形式のトラフィックデータを利用する．各評価トラフィックデータに含まれる**ボットトラフィック数**はボットに感染し C&C サーバに接続するまでの流れを１つと数える．ボットトラフィック数は著者が手動で解析した．本機構の検知数は 6.1.2 節のシナリオ定義ファイルを用いた本実装に pcap 形式のトラフィックデータを入力し，その中でシナリオが成立した出力の数である．本評価で用いた Snort の環境を表 6.3 に示す．

表 6.3: 評価に用いた Snort の環境

バージョン	2.6.0.2
シグネチャ	Snort 標準シグネチャ(20051219)
	Bleeding Snort(20051219)

6.3.2 評価結果

本評価の結果を表 6.4 に示す.

表 6.4: トラフィックに対する検知数

評価トラフィックデータ名	ボットトラフィック数	本機構	Snort
サンプル４	1	2	45
サンプル５	0	0	2
サンプル６	2	2	975
サンプル７	1	1	2164
サンプル８	1(活動しないボット)	0	354
サンプル９	0	0	258
サンプル１０	1	2	3661

6.3.3 考察

評価結果と出力結果のログより，本機構では用意した評価トラフィックデータの中から，適切にボットに感染し，C&C サーバと接続を行ったイベントを検知できた．本機構での検知数がボットトラフィック数を上回るトラフィックデータでは，含まれるボットの C&C との接続が，TCP のポート 6667 番ポートを用いており，かつ，30 秒以上セッションを継続していた．そのため，*connection* フロー群で *port6667* フローと *outtcp_over30sec* の両方に一致した．そのため，本機構では正常なフローをボットとは検知していない.

対して Snort による検知では，含まれるボットトラフィック数に対して大きく上回るイベントを検出した．イベントを確認したところ，ボットに感染した原因となったトラフィックも，ボットの感染とは関係ないトラフィックも同じシグネチャで検知が行われた．結論としては，Snort でもボット感染の原因となったトラフィックは検知できたものの，管理者にボット検知を伝えるには至らない.

6.4 まとめ

本研究では，本機構と既存のセキュリティ対策機構を用いて，新しいボットウェアの検知と，誤検知率・検知率について評価を行った．その結果，新しいボットウェアの検知では，既存手法として挙げた手法では検知されないボットを検知した．また，ボットに感染した際のトラフィックからの検知では，他の手法に比べて，より効率的な検知が行える事が分かった．

これにより，本手法は既存の手法では行えなかったボットの発見を行い，管理者に的確にボットの検知を通達できる手法であると言える．

第7章　結論

本章では本研究のまとめと今後の課題について述べる．

7.1　まとめ

本研究は，フロー順序を定義したシナリオにより管理ネットワーク内のボット検知機構の設計し実装した．

現在主流のセキュリティ対策手法の多くはシグネチャを必要とする．シグネチャを必要とする検知手法では常に亜種の出現するボットウェアへの対策は困難である．また，ボットネットに着目した調査やDNSを用いた対策手法は，ボットの規模性やC&Cサーバの冗長化といった問題を抱えている．これらの既存手法は世界に拡散し，常に亜種の出現するボットネットに対して十分な成果が期待できない．

この問題に対して，本機構は，ネットワークトラフィックのフロー順序に着目し，既存の手法に比べ効果的にボットネットを検知できる．フロー順序を用いることで正確なボットの検知ができるため，ネットワーク管理者は本機構の運用にボットについてのな知識が必要ない．また，シナリオ定義には独自の記述形式で柔軟な設定ができ，新たなボットに対応できる．

7.2　今後の課題

本節では，本研究の今後の展望として，以下の事項を挙げる．

7.2.1　実運用に向けた実装

本研究ではフロー順序に着目した手法のプロトタイプを実装した．また，評価より，本手法はボットの検知に効果的であると言える．しかし，今回はプロトタイプ実装のために，PHPとシェルスクリプトによって実装したが，実際のネットワークで運用するには，十分な実装方法とは言えない．本手法を用いた今後の研究のためには，C言語による再実装と実運用を進める必要がある．

7.2.2 さまざまなネットワークにおける本機構の検証

今回の調査及び既存の調査から，ClassB のアドレス空間に感染活動するボットネットが多いと分かった．本研究では１つのネットワークにおいて，ボットを収集し評価した．だが，他のネットワークにおいては，今回のネットワークとは異なる性質を持ったボットネットが存在すると考えられる．そのため，異なるネットワークにおけるボットトラフィックの解析を進める必要がある．

7.2.3 他の検知手法との連携

本機構は評価より，他の検知手法では検知できないボットウェアを検知できた．しかし，既存のボットトラフィックとは大きく異なるボットウェアは検知することが難しい．ボットネット対策は 3章で述べたように，手法によって利点と欠点がある．本研究は未知のボットウェアに対して成果を挙げたが，他の手法の検知結果を用いることで，より効果的な成果が期待できる．

謝辞

本研究を進めるにあたり、ご指導をいただきました慶應義塾大学 環境情報学部教授 村井純博士、同学部教授 中村修博士に感謝いたします。また、日々厳しく助言を頂いた慶應義塾大学 環境情報学部助教授楠本博之博士、環境情報学部専任講師 重近範行博士、政策・メディア研究科講師 南政樹氏に深く感謝いたします。

執筆にあたり絶えずご指導を頂いた白畑真氏，水谷正慶氏に感謝いたします。また，本論文の執筆にあたりご意見，ご助言を頂きましたNTT 情報流通基盤総合研究所 豊野剛氏，株式会社ラック 村上純一氏に感謝いたします．慶応義塾大学村井研究室SING/IAグループの小原泰弘氏，小椋康平氏、空閑洋平氏，奥村佑介氏，尾崎隆亮氏，波多野敏明氏，佐藤龍氏に感謝いたします。

最後に私の研究を影ながら支えてくれた両親と妹，多くの友人・知人に感謝し，謝辞と致します。

2007年1月24日
金井 瑛

参考文献

[1] JPCERT コーディネーションセンター. ボットネットの概要〜報告書〜, July 2006.

[2] Swa Frantzen. Clickbot, May 2006. http://isc.sans.org/diary.php?storyid=1334.

[3] J. Oikarinen, D. Reed. RFC 1459: Internet Relay Chat Protocol, May 1993.

[4] MAPS LLC. Introduction to the Realtime Blackhole List (RBL).

[5] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. RFC 1928: SOCKS Protocol Version 5, March.

[6] Paul Bacher, Thorsten Holz, Markus Kotter, Georg Wicherski. Know your Enemy: Tracking Botnets -Using honeynets to learn more about Bots-, March 2005.

[7] Nicholas Albright. Researching Botnets, Febrary 2006. http://www.shadowserver.org/whitepapers/Botnets.pdf.

[8] Impress Watch Corporation. INTERNET Watch 「シグネチャベースのボット対策は限界」BOT ネット対策 2006 , Jun 2006.

[9] Inc. Nikkei Business Publications. Nikkeibp: Telecom-isac japan が「ボットネット」の実験結果を報告, April 2006.

[10] Inc. Nikkei Business Publications. Nikkeibp:［network 調査隊］「ボットネット」の正体を探る, January 2006.

[11] 日本国総務省. 平成１７年「通信利用動向調査」の結果, May 2006.

[12] K. Egevang, P. Francis. Request for Comments: 1631: The IP Network Address Translator (NAT), May 1994.

[13] J. Reynolds, J. Postel. Request for Comments: 1700: ASSIGNED NUMBERS, October 1994.

[14] P. Vixie, S. Thomson, Y. Rekhter and J. Bound. RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE), April.

[15] SURFnet. Welcome to the SURFnet website. http://www.surfnet.nl/info/en.

[16] Antoine Schonewille, Dirk-Jan van Helmond. The Domain Name Service as an IDS, Feb 2006.

[17] Christian Kreibich and Jon Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (Hotnets II)*, Boston, Nov 2003.

[18] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. The EarlyBird System for Real-time Detection of Unknown Worms. August 2003.

[19] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. Automated Worm Fingerprinting. December 2004.

[20] 金井 瑛, 水谷 正慶, 白畑 真, 南 政樹, 村井 純. IDS と連携した高速に伝播するワームのシグネチャ自動生成機構の設計と実装. **第** *13***回マルチメディア通信と分散処理ワークショップ**, November 2005.

[21] 水谷正慶, 白畑真, 南政樹, 村井純. Session Based IDS の設計と実装, July 2004.

[22] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection, 2004.

[23] C. Bullard. Argus Homepage. http://www.qosient.com/argus/.

[24] Nepenthes Development Team. Nepenthes - finest collection - Web Page. http://nepenthes.mwcollect.org/.

[25] Martin Roesch. SNORT-LIGHTWEIGHT INTRUSION DETECTION FOR NETWORKS. *USENIX LISA ‘ 99 Conference*, 1999.

[26] Bleeding Edge Threats. Bleeding Edge Threats Web Page. http://www.bleedingsnort.com/.

[27] Symantec Corporation. *Symantec WWW page.*

[28] Kaspersky Lab. KASPERSKY - Free online virus scan - Web Page. http://www.kaspersky.com/scanforvirus.

付 録A　設定ファイルにおけるフロー条件

本節では，シナリオ記述に用いるフロー条件式示す.

太文字：条件
斜体：引数
＝演算子：直前の変数が直後の数値と一致する
＜演算子：直前の変数が直後の数値よりも小さい
＞演算子：直前の変数が直後の数値よりも大きい

図 A.1: 条件式の凡例

flow tcp
flow tcp の構文を図 A.2に示す．tcp 条件はフローが TCP であるかを確かめる.

構文: **tcp**

図 A.2: flow tcp 条件

flow udp
flow udp の構文を図 A.3 に示す．ucp 条件はフローが UDP であるかを確かめる.

構文: **udp**

図 A.3: flow udp 条件

flow icmp
flow icmp の構文を図 A.4 に示す．icmp 条件はフローが ICMP であるかを確かめる.
flow sport
flow sport の構文を図 A.5 に示す．sport 条件はフローにおける送信元のポート番号が *port* と一致するかを確かめる.

構文: **icmp** p

図 A.4: flow icmp 条件

構文: **sport** *port*
第一引数 *port*: フローの送信元ポート番号 (0-65535 の整数)
注意*:* 本条件は *tcp* あるいは *udp* 条件が設定されているときのみ有効である

図 A.5: flow sport 条件

flow dport
flow dport の構文を図 A.6 に示す．dport 条件はフローにおける送信先のポート番号

構文: **dport** *dort*
第一引数 *port*: フローの送信先ポート番号 (0-65535 の整数)
注意*:* 本条件は *tcp* あるいは *udp* 条件が設定されているときのみ有効である

図 A.6: flow dport 条件

が *port* と一致するかを確かめる．

flow sbyte
flow sbyte の構文を図 A.7 に示す．sbyte 条件はフローを開始した側から送信されたトラフィックのバイト数を *byte* と演算子 *eval-expr* で比較する．

flow dbyte
flow dbyte の構文を図 A.8 に示す．dbyte 条件はフローを開始した側が受信したトラフィックのバイト数を *byte* と演算子 *eval-expr* で比較する．

flow spkts
flow spkts の構文を図 A.9 に示す．spkts 条件はフローを開始した側が送信したパケットの総数を *pkts* と演算子 *eval-expr* で比較する．

flow dpkts
flow dpkts の構文を図 A.10 に示す．dpkts 条件はフローを開始した側が受信したパケットの総数を *pkts* と演算子 *eval-expr* で比較する．

flow in
flow in の構文を図 A.11 に示す．in 条件はフローが外部ネットワークから内部ネットワークに対して開始されたものかを確かめる．

flow out
out 条件は flow out の構文を図 A.12 に示す．out 条件はフローが内部ネットワークから外部ネットワークに対して開始されたものかを確かめる．

構文: **sbyte** *eval-expr byte*
第一引数 *eval-expr*: <, >, = のいずれかの演算子
第二引数 *byte*: 比較対象のバイト数

図 A.7: flow sbyte 条件

構文: **dbyte** *eval-expr byte*
第一引数 *eval-expr*: <, >, = のいずれかの演算子
第二引数 *byte*: 比較対象のバイト数

図 A.8: flow dbyte 条件

flow count
flow count の構文を図 A.13 に示す．count 条件はそのフローを検知した回数を *times* と演算子 *eval-expr* で比較する．

flow timeout
flow timeout の構文を図 A.14 に示す．timeout 条件はフローが発生しなければならない時間を *time* であらわす．単位は秒．

flow duration
flow duration の構文を図 A.15 に示す．duration 条件はフローの継続時間を *time* と演算子 *eval-expr* で比較する．

構文: **spkts** *eval-expr pkts*
第一引数 *eval-expr*: <, >, = のいずれかの演算子
第二引数 *pkts*: 比較対象のパケット数

図 A.9: flow spkts 条件

構文: **dpkts** *eval-expr pkts*
第一引数 *eval-expr*: <, >, = のいずれかの演算子
第二引数 *pkts*: 比較対象のパケット数

図 A.10: flow dpkts 条件

構文: **in**

図 A.11: flow in 条件

構文: **out**

図 A.12: flow out 条件

構文: **count** *eval-expr times*
第一引数 *eval-expr*: <, >, = のいずれかの演算子
第二引数 *times*: 対象の繰り返し回数

図 A.13: flow count 条件

構文: **timeout** *time*
第一引数 *time*:

図 A.14: flow timeout 条件

構文: **duration** *time*
第一引数 *eval-expr*: $<$, $>$, $=$ のいずれかの演算子
第二引数 *times*: 対象のセッション継続時間

図 A.15: flow duration 条件

付 録B　設定ファイルのABNF表記

```
config = *( *WSP line 1*CR )

line = “internal” *WSP “=” *WSP internal_expr /
“scenario[“ id-name “]” *WSP “=” *WSP scenario_expr /
“rule[“ id-name “]” *WSP “=” *WSP rule_expr /
“flow[“ id-name “]” *WSP “=” *WSP flow_expr

id-name = 1*(ALPHA / DIGIT / “-” / “_” )

internal_expr = IPv4address-with-subnet *( *WSP “,” IPv4address-with-subnet )

scenario_expr = scenario_expr *( *WSP “->” *WSP scenario_expr) /
id-name

rule_expr = rule_expr *( *WSP “,” *WSP rule_expr) /
id-name

flow_expr = flow_expr *( 1*WSP (and / or) 1*WSP flow_expr) /
“tcp” / “udp” / “icmp” /
“count” 1*WSP eval-expr 1*WSP 1*DIGIT /
(“s” / “d”) “byte” 1*WSP eval-expr 1*WSP 1*DIGIT /
(“s” / “d”) “pkts” 1*WSP eval-expr 1*WSP 1*DIGIT /
(“s” / “d”) “port” 1*WSP portnum /
(“s” / “d”) “host” 1*WSP IPv4address /
(“s” / “d”) “net” 1*WSP IPv4address-with-subnet /
“in” / “out” /
“timeout” 1*WSP 1*DIGIT /
“duration” 1*WSP eval-expr 1*WSP 1*DIGIT
```

図 B.1: 設定ファイルの ABNF 表記 (1/2)

```
eval-expr = "=" / "<" / ">"

IPv4address = dec-octet "." dec-octet "." dec-octet "." dec-octet
IPv4address-with-subnet = IPv4address "/" IPv4address

portnum = 4DIGIT ; 0-9999
/ %x31-35 4DIGIT ; 10000-59999
/ "6" %x30-34 3DIGIT ; 60000-64999
/ "65" %x30-34 2DIGIT ; 65000-65499
/ "655" %x30-32 DIGIT ; 65500-65529
/ "6553" %x30-35 ; 65530-65535

dec-octet = DIGIT
/ %x31-39 DIGIT
/ "1" 2DIGIT
/ "2" %x30-34 DIGIT
/ "25" %x30-35
```

図 B.2: 設定ファイルの ABNF 表記 (2/2)