

平成 22 年度 (2010 年度)

博士学位論文

マルウェアの症例解析に基づいた ビヘイビア型ネットワークインシデント検知



水谷 正慶

慶應義塾大学大学院 政策・メディア研究科

Behavior-based Network Incident Detection with Analyzing Malware Infection Case

Copyright © 2010 by Masayoshi Mizutani

概要

本研究では悪意のあるソフトウェア(マルウェア)の対策として、利便性、運用の容易さを考慮した新しいネットワークセキュリティ監視システムを提案、構築した。本システムは98%以上の精度でネットワーク上のマルウェアの活動を検知しており、狡猾化を繰り返すマルウェアに対して有効な対策手法が確立できた。

インターネットが広く一般に普及した一方で、悪意ある人物はインターネットに接続している計算機をマルウェアによって乗っ取り、様々な経済的利益を得るようになった。悪意ある人物はネットワークや計算機上での検知を回避するために、大量のマルウェア亜種を発生させたり、検知を難しくする通信パターンを用いるなどの狡猾化を繰り返しているため、マルウェアへの対策はますます難しくなっている。

本研究ではマルウェアの共通する挙動に着目し、マルウェアの亜種やパターンの変化に対応できる強い耐性を持ったネットワークセキュリティ監視手法を提案、実装した。マルウェアの収集と解析を行う過程で、マルウェアの通信にはいくつかの共通する動作と振る舞いがあることが明らかになった。これを利用し、本システムはマルウェアの共通する動作や振る舞いを抽象化することで、マルウェアの動作を俯瞰して検知する手法を提案した。このシステムは頻出するマルウェアの亜種だけではなく、新しい通信モデルをもったマルウェアが出現しても適応できる。本システムによってマルウェアの検知を試みた結果、ネットワークサービスに対して感染元から攻撃コードを送信してくるタイプのマルウェアと、Webサイトの閲覧を契機として感染するマルウェアの両者を、4つのルールを用いることで98%以上検知することに成功した。既存のIDSは3,000以上のルールを用いて73.91%しか検知できなかったため、本手法に優位性があると言える。これによって、頻出するマルウェア亜種の高精度な検知が実現され、マルウェアによる直接的な被害を最小限に食い止めることができる。そして、本手法が普及し、マルウェアの全体数を減少させることにより、マルウェアによる犯罪インフラが縮小し、インターネット上に存在するリスクの軽減が期待される。

キーワード:

1. インターネット. 2. 情報セキュリティ. 3. マルウェア. 4. 侵入検知.

Abstract

This research achieved a new high accuracy network security monitoring system, named ROOK, designed with usability and effectiveness in mind. The system is able to detect malware activities on a network with over 98% accuracy and an effective countermeasure against wily malware is established.

Malware authors and users have strong economic incentives to improve their ability to target users' information assets. They continuously update their malware, making it hard to create effective counter measures. Moreover they create many variants of malware and use complicated communication pattern to evade from network security monitoring.

In this research, a new network security monitoring system was implemented for gaining an overview of malware behavior and analyzing sensitive correlations of malware communications. During our collection and analysis of malware, it became clear that malware have some common activities and behavior. ROOK is able to detect security incidents by abstraction of these malware common activities and can be adapted to not only many variants of malware but also those using communication models by updating a simple rule set. The system tries to detect two types of malware: 1) sending exploit codes to network services provided on target hosts, 2) infection when target hosts open web sites with exploit code. In the result, the system succeeded in detecting both type of malware with over 98% accuracy using only four rules, while existing IDS detected only 73.91% using a rule set of over 3,000 rules. The research established a method of countermeasure against frequently-occurring malware variants and the method minimizes damage by malware. The method will reduce cyber-criminal infrastructure and a consequent reduction of risk on the Internet is expected.

Keyword:

1. Internet. 2. Information Security. 3. Malware. 4. Intrusion Detection.

目次

概要	ii
Abstract	iii
目次	iv
図目次	viii
表目次	x
第1章 序論	1
1.1 情報通信技術と情報セキュリティ	1
1.2 インターネットにおける脅威の変遷	2
1.3 本研究の取り組み	4
1.4 本論文による貢献	6
1.5 本論文の構成	8
第2章 マルウェアによる情報ネットワーク上の犯罪インフラ形成	9
2.1 マルウェア	9
2.1.1 マルウェアの変遷	10
2.2 ボットネット	11
2.2.1 ボットネットの構成	11
2.2.2 ボットネットによる被害	12
2.3 ボットネットを取り巻く現状	15
2.3.1 ボットネットによる犯罪の分業化	16
2.3.2 マルウェア生成ツールと亜種発生間隔の短縮	16
2.3.3 マルウェアの延命技術の向上	17
2.3.4 マルウェアの相互連携	17
2.4 マルウェアの変化に伴う対策の困難化	17
2.4.1 ホスト上での対策	18
2.4.2 通信の制限による対策	19
2.4.3 通信の監視による対策	20

2.5	本研究で対象とするマルウェア	21
2.6	本章のまとめ	23
第3章	事前調査	25
3.1	多様な感染経路に対応したマルウェア収集および解析環境の構築	25
3.1.1	マルウェア解析および収集環境構築における課題	26
3.1.2	多様な感染経路に対応した収集・解析環境の実装	26
3.2	収集したマルウェアの解析	31
3.2.1	マルウェアの通信データ収集	31
3.2.2	受動攻撃型マルウェアの分析	32
3.2.3	分析結果の考察	36
3.3	外部データを用いたマルウェア共通動作の解析	39
3.3.1	マルウェアの活動モデル	39
3.3.2	CCC DATASET を用いたマルウェア活動の分析	41
3.3.3	考察	44
3.4	解析結果に基づいた対策手法の考案	45
3.5	本章のまとめ	46
第4章	ネットワークセキュリティ監視によるマルウェア対策とその課題	48
4.1	本研究で対象とするネットワーク環境	49
4.2	ネットワークセキュリティ監視によるインシデント検知	49
4.3	ネットワークセキュリティ監視によるマルウェア対策の問題点	51
4.3.1	特徴的な通信内容や通信傾向の排除	51
4.3.2	通信の発生規則の変化	52
4.3.3	新しい通信モデルの出現	52
4.4	関連研究	52
4.4.1	既存のIDSとその改良	53
4.4.2	イベント相関分析	53
4.4.3	マルウェアの活動モデルに基づいた検知手法	54
4.4.4	ホスト上でのマルウェアの挙動解析と検知	56
4.5	要求事項	57
4.5.1	検知精度	57
4.5.2	検知対象への適応性	58
4.5.3	追調査の容易さ	58
4.5.4	処理性能	58
4.6	本章のまとめ	59

第5章	ビヘイビア型ネットワークインシデント検知	60
5.1	マルウェアの挙動に基づいたネットワークセキュリティ監視手法の提案	60
5.1.1	マクロな相関関係	61
5.1.2	ミクロな相関関係	62
5.2	通信におけるソフトウェア挙動の抽象化	63
5.2.1	セッションとイベントの定義	63
5.2.2	通信の相関関係を利用した既存手法の分類	65
5.2.3	通信の挙動を表す相関関係の抽象化	67
5.3	挙動のモデルを用いた挙動追跡型ネットワークセキュリティ監視システムの構想	71
5.4	効果的なシナリオ	75
5.5	本章のまとめ	76
第6章	ネットワークセキュリティ監視システム: ROOK の設計と実装	78
6.1	システムの要求事項	78
6.1.1	処理性能	78
6.1.2	ルール記述の柔軟性	79
6.2	設計	79
6.2.1	想定するネットワークと観測地点	80
6.2.2	ルールの記述と運用	81
6.2.3	変数の設計	81
6.2.4	変数処理に基づく挙動のテンプレートの制御とマルウェアの検知	83
6.2.5	監視対象の分割	85
6.3	実装	85
6.3.1	データ構造と資源管理	86
6.3.2	ルール構成	90
6.3.3	検知エンジンの動作	95
6.4	本章のまとめ	98
第7章	評価	99
7.1	検知精度評価	99
7.1.1	評価用通信データの準備	100
7.1.2	評価用ルールの準備	101
7.1.3	能動攻撃型マルウェアの検知精度評価	107
7.1.4	受動攻撃型マルウェアの検知精度評価	107
7.2	誤検知率評価	109
7.2.1	評価環境	109
7.3	性能評価	111

Abstract	vii
7.4 関連研究との比較	113
7.5 考察	115
7.5.1 能動攻撃型マルウェアの検知	115
7.5.2 受動攻撃型マルウェアの検知	116
7.5.3 評価に用いたマルウェアの網羅性	117
7.5.4 変数の保持時間設定の変更に伴う検知精度の変化	119
7.5.5 運用コストの低減	120
7.6 本章のまとめ	121
第8章 結論	122
8.1 本論文のまとめ	122
8.2 今後の課題	126
8.2.1 本研究の発展	126
8.2.2 マルウェアからの防御	128
8.2.3 ボットネットの追跡	128
8.2.4 マルウェアの解析	129
8.3 今後の展望	131
参考文献	134
付録A 評価に用いた通信データに関連するマルウェアの分類	140
付録B 精度評価で利用したルール	146
付録C 本研究に関する業績リスト	150
C.1 査読有り論文誌	150
C.2 国内発表	150
C.3 国外発表	151
C.4 その他	152

目次

2.1	ボットネット構成	12
2.2	マルウェアの分類と本研究での着目	21
3.1	マルウェア収集・動的解析構成概要	27
3.2	VM ノード構成	30
3.3	単位時間あたりの送信データ転送量と受信データ転送量の分布	34
3.4	SYN パケットとその他の TCP パケットの観測数分布	35
3.5	HTTP 通信における IP アドレス数の累積度数分布	36
3.6	感染直後のマルウェア活動の状態遷移モデル	40
5.1	マクロな相関関係の例	61
5.2	ミクロな相関関係の例	62
5.3	通信におけるセッションの例	63
5.4	通信におけるイベントの例	64
5.5	既存の通信の相関関係を利用したネットワークセキュリティ監視手法	66
5.6	ボットの活動パターンのモデル例	68
5.7	イベント発生順序の抽象化	69
5.8	セッション間での通信内容比較例	70
5.9	挙動のテンプレート例	72
5.10	挙動のテンプレートにおける変数操作の例	73
5.11	変数を用いたイベント発生規則の処理例 2	74
5.12	本手法が有効なシナリオの例	75
6.1	本手法によるネットワーク監視点	80
6.2	本実装が記述できる基本的なイベント状態遷移の生成規則	83
6.3	ROOK のコンポーネント概要	86
6.4	解析コンポーネントのクラス図	87
6.5	ルールコンポーネントのクラス図	88
6.6	検知コンポーネントのクラス図	89
6.7	BNF によるルール文法	90
6.8	ルール記述例	92

6.9	マクロな相関関係を表すルール記述例	93
6.10	ミクロな相関関係を表すルール記述例	94
6.11	変数の変更に伴うルールの例	97
7.1	検知ルールにおける検査条件とパラメータの関係	105
7.2	False Positive 評価用通信データ取得環境概要	110
7.3	MAWI トラフィックデータを用いた処理性能の測定	112
7.4	v_{15} パラメータ持続時間設定の変更に伴う検知率の変化	119

目 次

3.1	分析に用いたデータセットの一覧	31
3.2	能動攻撃型マルウェアと受動攻撃型マルウェアの通信内容比較	32
3.3	マルウェアのダウンロード方法の比較	33
3.4	HTTP 通信で出現した要求パス (上位 5 件)	37
3.5	HTTP GET 要求で出現された変数名 (上位 5 件)	37
3.6	分割したデータセットの分類	42
3.7	各状態開始から遷移までの所要時間	43
3.8	S_2 における通信内容の内訳	44
5.1	図 5.12 のシナリオに基づいた本手法と既存手法の比較	76
6.1	変数に必要な機能一覧	82
6.2	ROOK 開発環境	85
6.3	filter に利用できる要素と引数	91
6.4	action に利用できる要素と引数	91
6.5	マクロな相関関係を表すルール記述例における各イベントの説明と対応する行	94
7.1	能動攻撃型マルウェアの検知用ルール概要	102
7.2	受動攻撃型マルウェアの検知用ルール概要	103
7.3	比較に用いた Snort のルール	106
7.4	能動攻撃型マルウェアの通信データを用いた False Negative 評価結果	107
7.5	検知された受動攻撃型マルウェアの通信データ数内訳	108
7.6	受動攻撃型マルウェアの通信データを用いた False Negative 評価結果	108
7.7	誤検知率評価用の通信データ一覧	109
7.8	False Positive 評価結果	111
7.9	関連研究との比較	114
7.10	D'_p には出現せず D'_q において新たに出現したマルウェア一覧	117
7.11	2008 年カスペルスキー社のセキュリティ対策ソフトによって検知されたマルウェアの種別 ([1] より抜粋)	118
A.1	各評価用データセットにおけるマルウェアの出現数	140

A.1 各評価用データセットにおけるマルウェアの出現数	141
A.1 各評価用データセットにおけるマルウェアの出現数	142
A.1 各評価用データセットにおけるマルウェアの出現数	143
A.1 各評価用データセットにおけるマルウェアの出現数	144
A.1 各評価用データセットにおけるマルウェアの出現数	145

第1章 序論

1.1 情報通信技術と情報セキュリティ

情報通信技術が発展する中でインターネットが普及し、世界中の人が使えるデジタル通信インフラが実現した。しかし、一方で悪意ある人物から情報資産を守るため、持続的に情報セキュリティ対策を整備する必要がある、特に不正アクセスによる情報資産の機密性、完全性、可用性の侵害はデジタルネットワーク利用における最大の脅威の一つとなっている。

研究ネットワークとして誕生したインターネットは1990年以降急速に普及し、世界全体を網羅する過去に例のないデジタル通信のインフラとなった。特にWorld Wide Webや電子メールは情報共有、コミュニケーションには欠かせない道具となり、その他のサービスも多様な場面でユーザに利用されている。また、金融・経済に関する情報交換や電力・水道などの重要インフラ制御も一部はインターネットを基盤としているため、その役割は日々大きくなっている。

インターネットが普及しユーザの利便性が向上する一方で、インターネットにおける様々なセキュリティの課題が発生している。インターネットの特徴の1つであるオープンコネクティビティは多くの利用者が相互に通信できるネットワークを実現したが、第三者の悪意ある活動によって情報資産に被害がおよぶ可能性も高くなった。代表的な被害としては、他者の計算機に無断でアクセスし情報資産を破壊や改ざん、あるいは盗難する不正アクセスや、ネットワーク帯域や計算機資源を圧迫することで意図した動作ができなくなるサービス不能攻撃(Denial of Service, DoS 攻撃)などが挙げられる。これらの問題に対して研究者やインターネットに関連するサービスを提供する事業者は、常に安全性を維持するための研究および対策に尽力してきた。

多くの研究者が情報セキュリティの研究や開発に貢献したことで、今日では多くのセキュリティ対策手法が普及している。例えばデータ暗号化手法の発達により、第三者に内容を知られずに通信をしたり、インターネット上に仮想的な専用通信路を作成できるようになった。また、ファイヤウォールなどの通信制御手法によって不要なアクセスを制限し、インターネットからの不正アクセスや内部からの情報漏洩の事故や事件を未然に防いでいる。さらに、計算機上で悪意ある活動をするプログラムを検知・排除する対策ソフトウェアも普及し、データの破壊や盗難を防いでいる。

しかし、どのような分野のセキュリティでも、新しい脅威の出現とそれに応じる対策の開発が繰り返されてきた。インターネットにおける情報セキュリティも例外ではなく、日々新

しい脅威が出現している。社会的なインターネットへの依存度が高まるほど、悪意のある人物にとっては魅力的な場となるため、研究者が考案したセキュリティ対策を回避、あるいは無効化する手法を考案している。これによって、既存のセキュリティ対策による安全性は低下しつつある。

1.2 インターネットにおける脅威の変遷

インターネットにおける脅威の変遷で着目すべき点は、悪意のある活動をする人物の動機が私怨や興味本位などの個人的な理由から経済的利益の追求に変化した点である。インターネットにおける脅威は、インターネットに関する技術の発展や社会的な影響を利用して変遷してきた。インターネット上におけるサービス妨害攻撃や不正アクセスはインターネット黎明期より発生していたが、これらの多くは私怨や愉快犯、技術的興味や名誉欲を満たすためのものであった。例えば、対象となるネットワーク・ソフトウェア・計算機などを私怨で破壊する事件、ディスプレイ上に異常な画像や映像を表示させてユーザを驚かせるプログラムを流布していた事件、あるいは自分が所属するコミュニティに対して自分の技術力を示すなどの目的で行われた事件が挙げられる。これらは一部で深刻な被害を引き起こしたが、検知や対応が容易だった点や資産価値の高い情報を計算機上やインターネット上で処理する場面が少なかったため、リスクとしては比較的小さかった。

しかし、2000年前後からインターネットや計算機の利用者が増加し、インターネットに関わるサービスが増加したことによって、徐々に被害が直接的なものになってきた。2000年2月にはAmazon.comやYahoo!などの大手Webサイトが大量の計算機から短時間にアクセスを殺到させ、数時間サービスを停止させる事件が発生した[2]。Webによる電子商取引をしているサイトでは数時間のサービス停止でも多額の損失が発生してしまう。また、2003年1月にはインターネット上で急速に拡散するワーム[3]が出現し、ワームが送信する攻撃コードによって韓国のネットワークが数日にわたって利用不能になった。韓国以外のインターネットも全体的に不安定となり、日米間の国際線も一時的に不通になった。すでに銀行の現金自動支払機の一部がインターネットによって通信をしていた米国では、サービスが受けられない状態となった。このように、インターネットに依存したサービスとその利用者が増加することによって、インターネットにおけるリスクも増加している。

このような背景の中、インターネット上のサービスや利用者が持つ情報資産を利用し、経済的利益を得ようとする人物が増加した。例えば、電子商取引を提供しているサービスが数時間停止した場合はサービス提供者の損失が多い一方で、同業者やサービス停止の予測に基づく株式の売買をした人物など、利益を得る人物も現れる。また、一般の利用者も資産価値の高い情報を計算機上やネットワーク上で処理する機会が増えている。近年はクレジットカード情報だけでなく、迷惑メールを送信するためのメールアカウントや、ゲームコンテンツを利用して現金取引をするためのオンラインゲームアカウント、企業や行政に関わる機密情

報の電子書類など、様々な情報が価値をもつようになった。インターネットは全世界を網羅するグローバルネットワークであり、多くの人が接続できるオープンネットワークとなっている。また、利用者の識別、認証を必要としない場合も多く、意図的に複数の計算機を経由させることで通信の発信元を隠蔽できる構造となっている。そのため、不正に多額の金銭を得ようとしている人物や、技術力はあるが合法的な仕事に従事できない人物が、このような構造を利用して、金銭的利益を得るための活動を続けてきた。

インターネットで不正に利益を得るための手段として最も問題となっているのが、ボットネットと呼ばれるシステムである。ボットネットは1990年代後半にはすでに存在していたと見られているが、2004年頃から特に注目され始めた。ボットネットは悪意のあるプログラムである**マルウェア**によって乗っとられた計算機（ボット）の集合である。計算機上で不正に情報資産の完全性、機密性、可用性を侵害するマルウェアはコンピュータウィルス、ワーム、トロイの木馬などの名称でコンピュータネットワークが登場した時代から存在した。これらとボットネットを構成するマルウェアの違いは、ボットネットを管理する人物（ハーダ）が任意のタイミングで任意の命令を発行できる点である。これまでのマルウェアは自律的に感染を拡大させるプログラムが大多数だったが、ボットネットはインターネット上に独自のネットワークを構成し、ハーダが発行した命令をボットが一斉に実行する。ボットネットはいくつかの種類が存在するが、数万から数百万台のボットで構成されているものが少なくない。そのため、各計算機に保存されている情報資産の完全性、機密性、可用性を侵害するだけでなく、特定のサービスに対してサービス妨害攻撃をしかけたり、迷惑メールの大量送信やWebサービスに対する迷惑コメントの送信なども実行できる。2006年のTelecom-isacによる調査[4]では国内のインターネットに接続している5%程度のホストが感染している可能性があると報告している。いくつかのボットネットはハーダが逮捕されボットネットは解体されてきたが、依然として多数のマルウェアが登場し、ボットネット構築のために攻撃を続けている。ボットネットは大量のボットを任意に選択して攻撃に利用できるため、ハーダの存在を隠蔽しやすいという利点があり、ほとんどのサイバー犯罪がボットネットを利用して実行されるようになった。

利益を追求するためのボットネットを対策するにあたり、最も問題となるのは不正行為の効率化である。ボットネットは利益を追求するためのものが多く、ハーダの手間が多くなると得られる利益が少なくなってしまう。例えば、マルウェアに攻略されたボットは一般の利用者にとっては脅威であり、計算機上で発見された場合には駆除されるが、これが続くとボットネットの規模が縮小してしまう。そのため、ハーダは攻略後に発見されにくくなるような機能を追加したり、マルウェアを検知するセキュリティ対策ソフトの機能を停止させるなど、様々な手段を用いて延命させるようになった。このような取り組みにより、これまで有効であったマルウェア対策手法が有効に機能しない場面が多くなった。これによって、ボットネットによりインターネット上での犯罪インフラが形成されてしまった。

ボットネット運用のビジネス化と効率化によって、既存対策の多くは有効性が低下してい

る。これまで、ネットワーク型およびホスト型のファイヤウォールの導入と、セキュリティ対策ソフトの導入がマルウェアの感染を防ぐための効果的な方法として取り組まれてきた。マルウェアは短時間でより多くのホストに感染するために、感染したホストが新しい多数の攻撃対象に向けて通信を開始し、脆弱性のあるサービスに攻撃コードを送信していた。よって、ファイヤウォールで接続元と接続を許可するサービスを制限することにより、マルウェア感染の危険性が大幅に下がった。また、別の経路でマルウェアが感染したとしても、主にホスト上で動作するセキュリティ対策ソフトがホスト上のマルウェアを検知し、駆除が可能であった。しかし、ボットネットのビジネス化に伴い、効率的に運用しようとするハードがこれらの対策の有効性を低下させている。例えば、これまでは感染元ホストから通信を開始してOSのサービスなどに攻撃コードを送信していたのに対し、Webサイトに悪意のある攻撃コードをしかけてマルウェアの感染を誘発させる攻撃手法が一般化した。これはユーザが利用しているホストから通信を開始しているため、通常のファイヤウォールで防ぐのは難しい。また、亜種の発生頻度も急速に増加しており、既知のマルウェアのパターンなどを検知の拠り所とするセキュリティ対策ソフトでは、対応が間に合っていないという現状がある [5]。また、マルウェアに感染するとOSの機能を悪用するなどして感染した事実を利用者に隠し、長い期間潜伏しようとする。これを検知するために、様々な対策が考案されてきたが、検知精度の低さや新しいモデルのマルウェアの出現、さらに運用の困難さなどが原因となり、効果的な対策に至っていない。

1.3 本研究の取り組み

本研究ではマルウェアによる諸問題を解決するに当たり、ホストに感染しているマルウェアを検知、駆除し、減少させる方針を選択した。この方針による利点は2つ挙げられる。1つ目は、マルウェアに感染したホストを検知し、マルウェアを駆除することでホストを利用しているユーザへの直接的な情報資産の盗難や破壊といった被害を食い止めることができる点である。マルウェアは駆除されるのを防ぐため、感染したホスト上においてユーザに目立つ活動は避ける傾向がある。そのため感染ホストを特定し、駆除を促すことで、各ユーザごとのマルウェアによる被害を低減させられる可能性がある。2つ目は、感染したマルウェアの総数を減少させることによってボットネットの規模が縮小される点である。ボットネットの大きな利点は、多数の感染ホストを自由に操作して大規模な攻撃をしかけたり、ハードの身元を隠すことである。そのため、ボットネットの規模を縮小することで、ハードはボットネットをインフラとしたサイバー犯罪が困難になる。これにより、インターネット全体のリスクの低減につながると考えられる。

本研究ではまず、マルウェアの動向を把握に着手した。マルウェア本体やマルウェアによる攻撃に関するデータは、研究のためだけではなく悪用できるものも多いため、一般に公開されている情報は限られている。さらに、マルウェアに関する動向は変化を続けており、多

くの関連研究で解析されていない、新しい攻撃手法も出現を始めた。よって、新しく効果的な対策手法を検討するためには、独自の環境においてマルウェアの活動に関する詳細な情報を取得するだけでなく、多様な攻撃手法に対応した解析環境の構築が不可欠である。また、マルウェアの亜種の発生数は年々増加しているため、手動では十分な数の解析をするのは極めて難しい。さらに、なるべく多くのマルウェアを一度に解析する必要があるため、有限の資源を最大に利用する拡張可能な解析環境を設計、実装する必要がある。本研究では、これらの要件を満たす解析システムを構築、運用することで、感染経路によって異なる挙動を示すマルウェアの存在などを明らかにした。

本研究では対策の方針としてネットワークセキュリティ監視に着目した。ネットワークセキュリティ監視とはネットワークトラフィックに含まれる通信パターンを分析し、ネットワークに関連したインシデントに関連する情報を抽出する手法である。ネットワークセキュリティ監視はホスト型での監視と比較して、監視対象となるホストとは別の系で動作させることができる点と、ネットワーク全体を一括して監視できるという点が優れている。ホスト上で監視をする場合は、マルウェアによる隠蔽機能が働いていることを疑わなければならない。全てのハードウェア・ソフトウェアが正常に稼働しているかを検証した上で監視をしなければ、監視結果を信頼できないという問題がある。これは理論上可能だが、計算機資源に対する負荷が著しく大きくなると考えられるため現実的ではない。インターネットの性質上、暗号化通信などを用いても通信そのものを隠蔽するのは極めて困難である。したがって、検査をするために収集する情報が改ざんされる可能性は低くなり、完全性の高い情報に基づいて検査ができる。また、ネットワークの管理権限をもつユーザがネットワークセキュリティ監視をする場合、複数台のホストを同じ条件で一括して監視できる。よって、監視に必要なソフトウェアなどをホスト毎にメンテナンスする必要は少ないと言える。

一方で、ネットワークセキュリティ監視は検知精度と処理性能に関する課題がある。これまで、一般的なネットワークセキュリティ監視として、あらかじめ定義された検知ルールを元にインシデントに関する事象を検知するシグネチャ型侵入検知システム (Signature-based Intrusion Detection System, IDS) が使われてきたが、検知するべきではない事象を検知してしまう偽陽性 (False Positive) が問題となってきた。最終的にはネットワークセキュリティ監視を運用している人物が他のネットワーク情報や監視装置、記録を照らし合わせて、インシデントか否かを判断していた。また検知ルールの調整も必要であり、IDSの適切な運用には専門家が求められる。これは、インシデントの手がかりとなる事象ではなく、的確にインシデントである事象を抽出できていないのが原因であり、運用に専門知識・経験が求められるだけでなく、運用の負担も大きくしてしまうという問題がある。そして、一括して複数のホストを監視する場合には、ホスト上での監視と比べて高い処理性能か機能の分散化が必要となる。特に大規模なネットワークを監視する場合は、監視対象のホストが数千台から数万台の規模になるため、これらを適切に実時間で処理できなければならない。

既存の研究でも、様々な方法でネットワークセキュリティ監視によるマルウェア対策が検

討されてきた。2003年にSlammerやBlasterなど急速に拡散するマルウェアが出現してからには特にIDSが着目され、ボットネットで用いられるマルウェアの対策まで研究が取り組まれてきた。しかし第1.2節で述べた通り、マルウェアの作成やそれを用いた攻撃に対する動機の変化に伴って、マルウェアの活動傾向も大きく変化していることが本研究の調査によって明らかになっている。そのため、従来の手法とは異なる新しいネットワークセキュリティ手法を提案、実現する必要がある。

これらの議論に基づき、本研究ではマルウェアに感染したホストの共通した通信パターンに着目し、マルウェアの感染したホストの挙動をとらえて検知をするネットワークセキュリティ監視手法を提案、実装した。ボットネットで利用されているマルウェアは頻繁に亜種が発生するため、マルウェア本体のパターンに着目した検知方法では対応が追いつかない。さらに、通信も独特の通信プロトコルを使わずに、一般的にユーザが利用するプロトコルを使用したり、通信内容を部分的に変更するなどによって、ネットワークセキュリティ監視による検知を回避している。しかし、これらのマルウェアの活動は、細部の通信内容に着目するのではなく全体的な挙動を俯瞰すると、いくつかの共通した動作が見られる。このような動作は1つのホストに対して複数の通信チャンネルにまたがり俯瞰的に観測することで、挙動として認識できるようになる。従来のネットワークセキュリティ監視手法でも複数のチャンネルの相関関係を把握し検知に利用していたが、それぞれ実装ごとに限られた相関関係しか扱うことができず、マルウェアの複雑な動作に対応するのが困難であった。

本研究ではネットワークセキュリティ監視に、必要に応じてホストやセッションの状態や出現したデータを管理する変数のセットを組み込むことで、通信チャンネルにまたがった状態遷移や通信内容の比較を抽象化し、マルウェアの挙動を監視するビヘイビア型ネットワークインシデント検知を実現した。本論文では、これを挙動のモデルを示すテンプレートルールと変数と呼ぶ。通常のIDSで実現されているパターンマッチ型の検知機能に加え、通信チャンネルの発生順序や通信チャンネル上に発生した通信内容の比較などの相関関係を表現できるようになった。そして、これらをルールとして記述することによって、ホストの挙動における詳細な状態遷移や俯瞰的な動作を表現できるシステムを実装した。変数による通信チャンネル間の相関関係表現は、従来のネットワークセキュリティ監視手法で解釈できたものだけではなく、より複雑な相関関係をルールとして記述できるようになっている。本手法は通信の転送量やルールの増加に対し、時間計算量、空間計算量の両者が急速に増大することはない。さらに、問題領域として分割できる処理があるため、監視する計算機を増やすことで処理負荷も分散可能であり、効率的に実運用可能なシステムを実現できた。

1.4 本論文による貢献

本研究によって、ネットワークセキュリティ監視によるマルウェアの検知精度が向上した。本システムがマルウェア対策に有効であると示すために、筆者が収集したマルウェアの通信

データを用いて検知精度を評価したところ、本システムは98%以上のマルウェアの通信データを検知することに成功した。精度評価ではルールの作成と評価にそれぞれ異なる通信データを用いており、かつ高い検知精度を実現していることから、多くのマルウェア亜種に対して有効であると判断できる。既存のマルウェアから亜種を作成するのとは異なり、ハードも新しいモデルのマルウェアを開発し、拡散させるためには一定の時間を要する。そのため、今後も解析を継続することによって新しいモデルのマルウェアに対しても有効なルールの作成を続けられると考えられる。また、実運用ネットワークを監視したところ、誤検知である False Positive は発生しなかった。このことから、本システムは極めて高い検知精度のネットワークセキュリティ監視手法であると言える。さらに、処理性能の評価結果からも、実運用ネットワークにおいて利用可能であることを示した。

マルウェアに対して精度を保つことによって、ネットワークセキュリティ監視によるマルウェア対策の運用負担を低減させた。従来のネットワークセキュリティ監視では検知精度が十分ではないものが多く、ネットワークインシデントに関連した事象を発見した後はネットワークの運用者やセキュリティ管理者が追調査によってネットワークインシデント発生の有無を確認していた。しかし、本システムによって追調査の負担も大きく軽減されると期待される。これまでのネットワークセキュリティ監視の多くは検知精度の問題だけではなく、出力される結果が断片的なものであり、ネットワークインシデントに関連する事象を俯瞰しにくいという問題もあった。パケット、あるいはある通信チャネルの一方のみを監視している IDS は部分的なパターンマッチによって事象の発生を確認しており、どのような経緯で検知対象となった通信が発生したのかを把握しにくい。他にも、ネットワークトラフィックを統計的に処理して異常を検出するような手法があるが、出力結果からどの通信がどのような意味を持っていたかを判断するのは困難である。しかし、本手法はあるホストの通信を俯瞰するだけではなく、各通信がどのような順序や意図で発生したかを示すことができるため、検知結果からマルウェアの通信であるかどうかを判断するのが容易になっている。そのため、検知結果に対する追調査を最小限にとどめ、運用する人物の負担を軽減させられる。

運用負担の軽減は、より多くのネットワークでのネットワークセキュリティ監視の導入が期待できる。本論文執筆時点では、企業や行政などで一定規模のネットワークにネットワークセキュリティ監視を導入するのは一般的となっているが、規模の小さい企業や組織、自治体、教育ネットワークや家庭ネットワークなどでは、運用の負担が多く、専門知識や経験も不可欠であるため、導入や運用は難しい。しかし、検知精度が高く、運用負担も少ないシステムであれば、自動で稼働させることもできる。また処理負荷が少なければ、安価な装置で稼働させることも可能である。ネットワークセキュリティ監視は監視対象から独立した検査や、ネットワーク全体を網羅できるといった利点があるため、全体的なコストが低ければ導入のメリットは大きい。マルウェアを検知した後の調査や駆除、回復といった対応を自動化するのは難しいが、ユーザに対して早期に警告する、あるいは一時的にネットワークから遮断するなどによって被害は大幅に軽減できる。情報技術が発展し、ネットワーク上のサービ

スを活用したりホスト上に重要な情報資産を保持している状況を鑑みると、このような対応は今後必要であると考えられる。

そして、ネットワークセキュリティ監視がより一般に普及することによって、マルウェアが減少し、ボットネットが犯罪インフラとしての機能を低下させることが期待される。容易に導入・運用できるネットワークセキュリティ監視が普及し、多くのネットワークにおいて導入されることで、マルウェアの検知と駆除が促進されることが考えられる。さらに、マルウェアの全体数が減少することによって、ボットネットの規模が縮小し、インターネット上での犯罪が低減すると期待される。ボットネットが縮小すれば大量のボットを利用したサービス妨害攻撃や迷惑メールの送信などが困難になり、ボットネットを利用した犯罪も困難になると予想される。また、現在はボットネットの規模も大きく、犯罪が発生した場合でも使用されたボットをたどって犯人を検挙するのは困難な場合が多いが、規模の縮小によってハーダの検挙が容易になると考えられる。これによって、インターネットを利用した犯罪行為を減少させ、現在よりリスクの低いインターネットを実現できると考える。

1.5 本論文の構成

第2章ではボットネットを構成するマルウェアの現状と着目すべき機能や特徴について説明し、第3章では効果的な対策を考案するために、マルウェアの収集および解析環境を構築し、解析によってえられた結果と、対策に利用できるマルウェアの特徴について述べる。第4章で既存研究および既存技術による対策の難しさについて述べる。そして第5章では本研究で提案するネットワークセキュリティ監視手法の構想について述べ、第6章では具体的なシステムの要求、および設計と実装について述べる。第7章では本システムの検知精度、処理性能、および既存研究との比較について述べ、第8章で結論をまとめる。

第2章 マルウェアによる情報ネットワーク上の 犯罪インフラ形成

情報資産の破壊や改ざん、盗難、迷惑行為、不正利用などの情報犯罪はインターネットの登場と共に散見されていたが、インターネットが世界規模のオープンデジタルネットワークとして普及したことで、2000年以降にマルウェアを用いた情報犯罪のインフラがインターネット上に形成されていった。これによって、以前は私怨や愉快犯など個人によって引き起こされていた情報犯罪が組織的なものに変化し、ブラックマーケットが発生していった。また、悪意ある人物は営利を求めるためにより効率的な情報資産のセキュリティ侵害の手法を考案し、それに伴って対策が困難化していった。マルウェアの種類、感染状況、被害などは時々刻々と変化しているが、本章では本論文執筆時点でのマルウェアの現状についてまとめる。

2.1 マルウェア

マルウェアとは悪意のあるソフトウェアを意味する Malicious Software を短縮した造語 (Malware) である。従来、コンピュータやネットワークに被害を及ぼすソフトウェアはコンピュータウイルス、ワーム、スパイウェア、トロイの木馬など、各ソフトウェアが持つ性質と機能に応じて詳細に分類されていた。しかし、今日では新しい性質や機能を持つソフトウェアの増加や、異なる機能や性質を複数持っているソフトウェアなどが出現しているため、総称としてマルウェアという単語が使われるようになった。これはセキュリティ対策ベンダなどが自発的に使い始めた単語であり、定義は説明者によって異なる。本論文では Microsoft の技術情報における解説 [6] を参考し、マルウェアをコンピュータ、サーバー、コンピュータネットワークに被害を起こすように設計・実装されたソフトウェア全般と定義する。ソフトウェアは各 OS 上で実行可能な形式のファイルだけではなく、スクリプトなども広く含めるものとする。

現在、マルウェアは多機能化が進んでおり、マルウェア毎に機能を言及するのはあまり意味をなさない。これは1つのマルウェア検体が複数の機能を有したり、同時感染した複数のマルウェアが互いに機能を補完しあうためである。特に、近年では新しいマルウェアの亜種を作成するには専用の生成ツールを利用して、マルウェアが持つモジュールの組み合わせを自由に変更できる。そのため、同じ作成者によるマルウェアでも機能が大きく違う場合もある。

2.1.1 マルウェアの変遷

マルウェアの転機は大きく分けて3つある。まず1つ目は1990年代後半にウィルスやワームがメールなどインターネット上の通信手段を用いて感染を広めるようになったことである。初期のマルウェアはフロッピーディスクなどの持ち運ぶメディアを利用していたため、これによってより広範囲に対する感染の拡大が可能となった。特に電子メールを媒介とする1999年のBubbleboyや2000年のLove Letterウィルスは、電子メールクライアント (Mail User Agent, MUA) の脆弱性を利用して電子メールを閲覧するだけで感染したため、より高速に感染が拡大した。

しかし、2つ目の転機である2001年頃から発生したインターネット上を高速に伝搬するウィルスは、さらに高速にインターネット上のホストに感染する機能を持っていた。Codedred[7]やSlammer[3], Blaster[8]に代表されるウィルスは攻撃対象のホストが通信を受け付けているサービスに対して攻撃コードを送信し、攻撃対象のホストを攻略していた。これらによって24時間以内に全世界へ波及するウィルスが実現し、ネットワークに対する大きな脅威となった。また、これらのウィルスはユーザの動作を契機とせず、自律的に感染が継続されるという特徴があった。そのため、当時普及が進んでいたラップトップ型コンピュータがこのようなウィルスに感染し、コンピュータを持ち込んだネットワークが内部から感染を拡大させるという事例も多く発生した。しかし一方で、これらのウィルスは急速に感染を拡大させようとするため、短時間に大量のパケットを送信するものが多く、ネットワークセキュリティ監視による発見が容易であった。

そして、3つ目はボットネットの出現である。マルウェアのソースコードを調査したところ、Agobotと呼ばれる系統のボットは1990年代から存在していたと見られる。第1.2節で述べた2000年のAmazonやYahooなどに対する大規模な攻撃もボットネットによるものと推測されている¹。実際にボットネットが注目され始めたのは2004年～2005年頃からである。本論文執筆時にも、形態を変化させながら依然としてボットネットによる被害が継続している。ボットネットを構成するボットも急速に感染を拡大させようとする場合もあるが、感染したホストをより長く継続させようとする傾向が強くなった。

これらの変遷において重要な要素は、マルウェア作成者の動機が営利目的に変化している点である。ボットネット以前のマルウェアは私怨や政治的主張、技術的な挑戦や興味本位といった、作成者の個人的理由から実行される場合が多く、マルウェアの作成と配布は作成者に対して実質的な利益を生むものではなかった。

しかし、マルウェアを利用して経済的利益を得られるようになり、多くのマルウェア作成者が営利目的の活動となった。例えば2005年11月には40万台のボットによってアフィリエイトを不正に利用し、月数千ドルを儲けていた人物が逮捕されている。この他にも迷惑メールの送信や分散型サービス妨害攻撃、個人情報の盗難などによって利益を得ているハッカー

¹ 当時は乗っとられたホストをゾンビホストなどと表現していた

は多く存在する。

このような状況に伴ってマルウェアの作成形態も変化した。特に大きな問題として、マルウェア作成者やハッカーがマルウェアに感染した事実を隠蔽し、感染状態を継続させる技術の導入や開発に注力を始めた点があげられる。これまで私怨や政治的主張、興味本位によって作成されたマルウェアは感染した事実や不具合が発生していることを意図的にユーザに主張するものも多くあった。しかし、経済的利益を得るために感染させたホストは、可能な限り長期的に感染した状態にした方が利益を得やすい。これは、同時に操作できる感染ホスト数を維持するだけでなく、長期的に感染させることでより多くの情報資産を盗むなどの理由もあげられる。また、個人的な動機ではなく、経済的利益を得ることを目的とした場合、複数人で目的を共有しやすいという点も異なる。直接的に目的を共有しなくても、マルウェアの開発や攻撃コードの生成などを金銭によって請け負うといった取引が成り立つ。さらに、技術的な挑戦の方向も変化した。これまでは自己改変や正規ソフトウェアに対するコードの埋め込みなど、高度な技術によって作成されたマルウェアが多く見られたが、今日では拡張性などを重要視し、隠蔽も rootkit などを別途用いるなど、必要な機能を必要な場所だけに用いるといった効率化が図られている。マルウェア作成も独自にコードを開発するのではなく、必要な機能に応じてマルウェアを生成するソフトウェアを利用している場合が多い。このように、経済的利益を得るために、より少ないコストで大きな利益をあげるための効率化が進み、攻撃そのものも巧妙化が加速した。

2.2 ボットネット

情報資産やネットワークに被害をおよぼすマルウェアは、コンピュータやネットワークの黎明期より発生してきたが、2000年以降に経済的な利益を追求する攻撃者が増加した。この動向にともなって、マルウェアに感染させた複数のホストを統括管理し、悪意ある活動をするためのシステムが出現した。このような統括管理のシステム全体を**ボットネット**と呼ぶ。

2.2.1 ボットネットの構成

本論文では、マルウェアに感染し、ハッカーによって操作可能な状態になっているホスト(ボット)の集合をボットネットとする。ボットネットの構成は大きく分類してサーバ・クライアント型とP2P型に分類できる。図2.1にボットネットの構成の例を示す。

サーバ・クライアント型では通常のホストとは別にコマンド&コントロール(C&C)サーバというホストがボットネットに存在する。ハッカーはC&Cサーバに対して命令を発行し、各ボットはC&Cサーバを経由して命令を受信する。C&Cサーバは1つのボットネットに複数配置されている場合が多く、C&Cサーバの役割を担うホストも頻繁に変化する場合が多い。これは、1つのC&Cサーバに多数のボットが接続しているため、該当するC&Cサーバの運

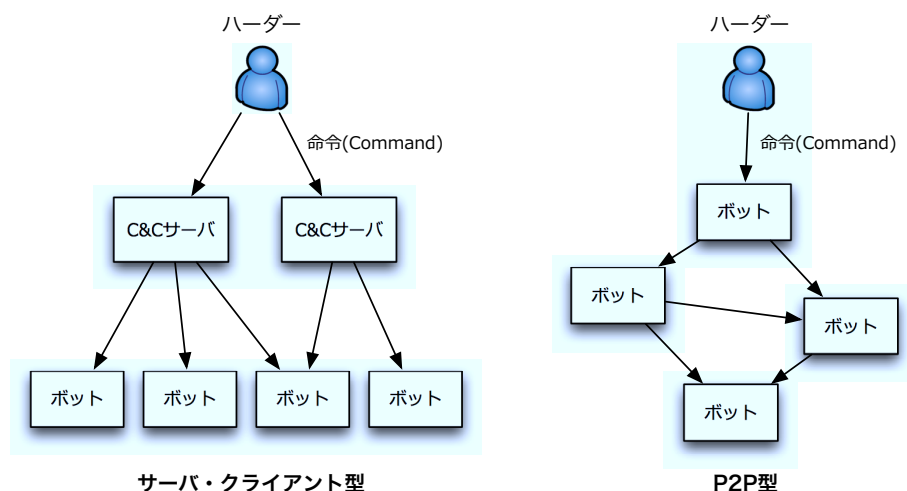


図 2.1: ボットネット構成

用を停止させたり，C&Cサーバへのアクセスを制限するなどの対応策によってボットネットの規模を縮小させることができる。しかし，ハーダはダイナミックDNSを利用するなどして，C&Cサーバへのアクセスを短時間で回復させたり，冗長化するなどの対処をしており，全てのC&Cサーバへの接続を阻止するのは難しい。C&Cサーバへの接続方法はInternet Relay Chat(IRC)やHyper Text Transfer Protocol(HTTP)を用いる場合が多い。

一方，P2P型は独自のオーバーレイネットワークを構築し，その上でハーダが発行した命令を交換する。これらは特定のC&Cサーバを持たないため，通信の宛先からボットネットへの参加を判断するのは難しい。P2P型では通信に独自の通信プロトコルを用いる場合が多い。オーバーレイネットワークに参加する方法としては，あらかじめ登録されているいくつかのランデブーポイントにアクセスを試みるものと，ランダムに接続先を探索するものが多く見られる。ハーダはいずれかのボットを経由して命令を発行し，オーバーレイネットワークを通じて命令を拡散させる。命令は命令文として発行される場合もあるが，特定の目的のために作成した実行形式ファイルを直接送信するか，あるいは特定の場所から実行形式ファイルを取得させ，そのファイルを実行させるものも存在する。

2.2.2 ボットネットによる被害

ボットネットは通常時には自律的に活動し，必要に応じて任意の命令を実行できる。さらに，ボットネットを利用して自ら犯罪行為をするだけでなく，第三者に対して一時的に利用権利を発行し，賃料を得るというビジネスに発展している。そのため，ハーダにとっては非常に効率的な犯罪用のインフラとなってしまう。

感染の拡大

ボットネットは自身を拡大させるために、様々な方法を用いてホストに攻撃をしている。同じシステムのボットでも感染経路を使い分けて攻撃をしている。セキュリティ対策の強度は最弱点問題となるため、対策が十分ではない部分から攻略されてしまう可能性が高い。主な感染経路を分類する。

- **開放しているネットワークサービスの脆弱性を利用した攻撃:** 外部のホストからの通信を受け付けているサービスの脆弱性を利用した攻撃。RPCや資源共有サービスなどは特に処理が複雑であるため、脆弱性を持つ場合が多い。一般のユーザが使うことを想定しているデスクトップ・ラップトップ用のOSでは資源の共有やネットワークサービスなどを特別な設定なしで使えるようにするため、OSの導入時からいくつかのネットワークサービスが自動的に起動している。これらのサービスが適切にアクセス制限されていないと、マルウェアが送信してきた攻撃コードで攻略されてしまう可能性がある。この手法を使う多くのマルウェアは自分の接続しているネットワークの近傍、特定のIPアドレスレンジ、無作為なIPアドレスなど、様々な宛先に対して、短時間で大量の攻撃コードを送信する。本論文では特にこのような攻撃を**能動型攻撃**と呼ぶ。
- **Webサイトの閲覧を契機とした攻撃:** 悪意あるWebサイトを閲覧した際、Webブラウザ本体の脆弱性やWebブラウザのプラグインの脆弱性、OSの描画エンジンの脆弱性などを利用して感染する攻撃。2006年頃から拡大をはじめ、2007年にはProvosらによる調査報告[9]によって紹介された。能動型攻撃はファイヤウォールの適切な設定や、Network Address and Port Transfer(NAPT)環境のネットワークに接続することで大部分の攻撃を防げたが、Webサイトの閲覧は一般のユーザが定常的に発生させる通信である。さらに、攻撃者はHyper Text Markup Language(HTML)のiframeタグや難読化されたJavaScriptなどを利用して、攻撃コードが設置されたサイトに誘導するなど、攻撃が巧妙化している。さらに、ハッカーはSQL injection攻撃などを利用して、悪意のない一般のサイトに攻撃コードや攻撃コードが設置されたWebサイトへの誘導コードを設置する場合が多い。そのため、対策が非常に難しい攻撃の1つとなっている。本論文では特にこのような攻撃を**受動型攻撃**と呼ぶ。
- **ファイル配布、共有システムを用いた攻撃:** ネットワークを介して他人にファイルを送付、もしくは共有するシステムを利用してマルウェアを配布する攻撃。配布の過程でシステムを破壊、攻略して任意のマルウェアを実行するタイプと、システムを破壊、攻略せずにユーザ自身に実行させる方法がある。例えば電子メールに悪意あるファイルを添付し、MUAの脆弱性を利用したり、ユーザを誘導してファイルを開かせる攻撃が挙げられる。電子メールに攻撃コードを含むファイルやマルウェア本体を添付する攻撃手法は以前からあったが、近年では記録内容の確認を促すようなメッセージによって添付

ファイルの開封を促す標的型攻撃など、従来の攻撃手法に加えて巧妙な技術が導入される傾向がある。また、P2Pなどに代表される不特定多数のユーザのファイルを共有するシステムにおいて、マルウェアを混入する手法もここに含まれる。

- **可搬型記憶媒体を利用した攻撃:** 持ち運びを想定した USB メモリなどの記憶装置を媒体として感染させる手法。ネットワークが十分に普及する以前はフロッピーディスクによるマルウェア感染が一般的だったが、ネットワークが普及した今日でも USB メモリによる感染拡大が深刻な問題となっている。Microsoft 社の Windows の一部のバージョンでは外付け記憶媒体のルートディレクトリに特定のファイルが存在する場合に、自動的に任意のファイルが実行されるという仕様がある。これによって、USB メモリをマウントするだけで感染するマルウェアが感染を拡大している。この攻撃手法は、閉鎖されたネットワーク内で脆弱なソフトウェアを使用している場合に多大な被害をもたらす傾向がある。

情報資産の盗難

マルウェアに感染したホストは様々な権限を奪われ、ホスト上で扱われる情報資産の機密性が侵害される。従来のマルウェアでは可用性、完全性の侵害としてデータの破壊や改ざんという事例が多かったのに対し、ボットとして活動するマルウェアは機密性を保持すべき情報をハードに送信することで、利益をあげている。

情報資産の中で最も被害に遭いやすいのは識別情報と認証情報である。識別情報としては金銭に直接関係するクレジットカードやオンラインバンキングのアカウント情報があげられる。また、直接金銭に関係しない場合でもメールアドレスや各種 Web サービスのアカウント情報、さらにオンラインゲームのアカウントが標的となる場合が多い。メールアドレスは迷惑メールを送る際、ボットネットを利用して不正に取得したメールアドレスではなく、正規のメールアドレスから送信した方が受け取り側で迷惑メールと判定される可能性が少なくなるという利点がある。各種 Web サービスのアカウント情報は電子商取引に利用し、アカウントの所有者に無断で商品を購入するなどが可能である。また、オンラインゲームのアカウント情報も、実際の金銭で取引する Real Money Trading(RMT) で、当該アカウントがオンラインゲーム上で所有しているアイテムなどを売却して利益を得られる。これらはホスト上に保存されている情報だけではなく、ユーザが入力したキーを記録するキーロガーや Web ブラウザで送受信した情報を記録するソフトウェアを動作させ、継続的に情報を収集、盗難する。

また、一方で機密情報の売買も行われている。Friess らの調査 [10] によれば、マルウェアに感染したホストから重要と思われる書類をあらかじめ調査し、リストを作成する。そして、そのリストを別の犯罪者に公開し、必要な情報だけを抽出する仕組みが動作していると述べられている。これは、無差別に感染を拡大させた場合は適切に稼働させるのが難しいが、標

的型攻撃などによって得られた情報については、得られる情報や情報の転売先がある程度明らかになるので、実質的な被害になりやすいと言える。

迷惑行為の中継

ボットネットに参加しているボットを中継し、迷惑メールやトラックバックスパムを送信されている。ボットネットに参加しているボットは非常に多いため、中継するボットを頻繁に変更できる。これによって、本当の送信者の特定を困難にした状態で迷惑メールやトラックバックスパムの送信が可能となる。ハード自身が迷惑メールやトラックバックスパムを送信することで直接利益を得るのは難しいが、多くのボットネットでは第三者が迷惑メールやトラックバックスパムを送信するために、ボットネットの利用権を発行している。ハードはボットネットを貸し出す際に時間あたりの貸出料を設定しており、これによって利益を得ることができる。現状、ほとんどの迷惑メールがボットネットから送信されていると見られており、日本国内のインターネットサービスプロバイダ (ISP) は Outbound Port 25 Blocking(OP25B) と呼ばれる対策を実施して。これは各 ISP 内部から外部に対して、メール送信プロトコルである Simple Mail Transfer Protocol(SMTP) が利用する Transmission Control Protocol(TCP) のポート 25 番を塞ぎ、各ホストが直接外部のメールサーバに対して迷惑メールを送信できないようにするための対策である。しかし、一方でボットネットも正規のメールアカウントを入手して送信するなどの対抗策を取り入れている。また、OP25B は主に日本国内での対応であるため、海外のホストを利用して送信され続けているという現状がある。

サービス妨害攻撃

ボットネットは命令の発行者がわからず、多くのホストを不正に操作できるという点から、サービス妨害攻撃にも頻繁に利用されている。これも迷惑メールの送信やトラックバックスパムの送信と同様に、第三者に対してボットネットの一部を貸し出し、賃料を得るというビジネスモデルが確立している。

詐欺行為による攻撃

フィッシング詐欺などによる攻撃では、短期間に偽のサーバを用意する必要があり、これもボットネットが悪意ある活動に使われる一つの形態である。

2.3 ボットネットを取り巻く現状

ボットネットが経済的な利益をえるための犯罪インフラとなったことによって、マルウェアに関する技術的、構造的なパラダイム転換がおきた。これはこれまでのマルウェア対策手法

にも影響を及ぼすものであり、今後のマルウェア対策を考えるにあたって重要な要素となる。

2.3.1 ボットネットによる犯罪の分業化

ボットネットが経済的利益を得られるようになり、ハードをはじめとするボットネットに関係している人物がそれぞれ異なる役割を担う構造が構築されている。過去のマルウェアは感染させるための攻撃コードの作成、マルウェアの配布、感染したホストの操作などは1人、もしくは極少数の人物が全ての役割をはたしていた。以前のマルウェアは私怨や技術的興味など、ごく一部の人物が共感する動機に基づくものが多く、動機を共有できる人物を探すのは困難だったと見られる。しかし、金銭的な利益を得る場合、動機の共有が容易なだけでなく、金銭的取引によって作業を分担できるというメリットが有る。

ボットネットに関する分業化は非合法市場で発展していると見られ、それぞれ専門に特化した活動をしている。例えば、あるソフトウェアの脆弱性を攻略する攻撃コードを開発する人物、ボットを機能拡張するために新しいモジュールを作成する人物、ボットネットを管理、拡大させる人物、ハードからボットネットを借りて迷惑メールなどを送る人物、ボットネットから収集した情報を転売する人物などである。通常の市場と同様に、それぞれが得意とする技能を専門として取り組めば、よりマルウェア開発や配布のコストを下げることができる。

さらに、金銭的利益をえることでより悪意ある活動に専念できる環境が整ってきていると言える。以前は個人的な動機によってマルウェアを配布していたため、マルウェア作成者が生活するためには別途金銭を得る必要があった。しかし、マルウェア作成や攻撃コード開発などによって金銭的な利益が得られれば、別途金銭を得るための活動をする必要がなくなる。以前もマルウェア作成者がクレジットカード情報の取引によって金銭的利益をえることはあったが、この場合は攻撃コードの開発やマルウェア配布、情報集取などを単独で作業する必要がある。敷居が高かった。現状では専門の活動によって利益が得られるため敷居も下がっており、より多くの人物がボットネット犯罪に関わっていると推測される。

2.3.2 マルウェア生成ツールと亜種発生間隔の短縮

ボットネット運用の効率化に当たり、マルウェアの作成の形式も大きく変化した。以前のマルウェアは個々のマルウェア作成者が独自に開発していたが、第2.3.1節で述べた通り攻撃コードや機能拡張をする開発者とマルウェアを配布する人物が分かれてきたため、容易にマルウェアを生成できるツールが流通している。これらのツールは必要な拡張機能を選択し、コードの難読化や自己解凍形式によるパッキングに関するパラメータを指定すれば、プログラミングの知識や経験がない人物でもの新しいマルウェアを生成できる。

このように新しいマルウェアの作成が容易になったことをうけて、マルウェアの亜種発生頻度が急上昇している。総務省や Telecom-ISAC が主催するサイバークリーンセンター [11]

が2008年1月に報告した調査結果によれば、PE_VIRUT.Aと呼ばれるマルウェアは1日あたり300種類以上の新しい亜種が生成されている。この他のマルウェアについても、同様に亜種生成の速度が急速に増加している。これは、新しい機能の拡張や傾向を変化させる目的だけではなく、マルウェア対策手法への対抗手段であると見られる。新しいマルウェアを生成するコストが下がれば、ハードは継続的にマルウェアの亜種を生成し続けることが容易になるが、対策側はこれに追従してマルウェアの解析や対策を続けなければならないため、マルウェア対策を回避したいハードにとって都合の良い状況だと言える。

2.3.3 マルウェアの延命技術の向上

ボットネットをつかって利益を得るにあたり、ハードにとってのリスクの一つはボットネットに参加しているボットが減少し、ボットネットの規模が縮小してしまうことである。ボットとして活動するマルウェアは不正にホストの資源を利用しているため、発見され次第駆除される場合が多い。そのため、ハードは様々な手段を講じてホストがマルウェアに感染した状況を継続させる[12]。

2.3.4 マルウェアの相互連携

ボットネットがシステムの的に運用されるようになったことで、個々のマルウェアの機能も変化している。従来は1つのマルウェアが自律的に活動し、自己増殖による感染拡大や発病による破壊行為などを担っていた。しかし、マルウェアに感染してボットになったホストに対しては、ハードが任意のタイミングで任意のマルウェアを導入できる。2005年前後に感染が拡大したボットネットは1つのマルウェアに複数の機能が搭載されていたが、本論文執筆時点では1つのマルウェアをなるべく単機能化する傾向が強くなった。それぞれ、感染活動をするマルウェア、口座番号やクレジットカード番号、ネットワークサービスのアカウントを盗むマルウェア、rootkitとして動作してセキュリティ対策ソフトを無効化するマルウェアなど、必要に応じて特定の機能を持つマルウェアが導入されるようになった。そのため、これまでは個々のマルウェアに大して注目されていた悪意ある活動は、複数のマルウェアによる有機的な動作の連携として目を向ける必要がでてきた。

2.4 マルウェアの変化に伴う対策の困難化

セキュリティ管理者がネットワーク内のボットを検知する場合、ネットワークトラフィックを監視してボットの通信を発見するのが効果的である。これは、ボットが感染したホスト上で動作するセキュリティ対策ソフトの機能を阻害する点や、OSの機能を改ざんしてボットの活動を隠蔽する点などが挙げられる。しかし、従来の不正な通信のパターンを検知するネッ

トワークトラフィック監視手法を用いたIDSでは、亜種の発生が頻繁であり、悪意ある活動の通信の巧妙化が進むボットの検知が困難となっている。

2.4.1 ホスト上での対策

ボット作成者はボットの機能拡張やボットの検知、駆除防止のため、頻繁に亜種を作成し配布しており、2008年8月のサイバークリーンセンターの調査[13]では収集したボットを含むマルウェア検体のうち18%以上が市販のセキュリティ対策ソフトでは検知できなかったと報告されている。これは、現在のIDSやセキュリティ対策ソフトは検知精度を上げるために、個々のマルウェアの種類毎にバイナリパターンを定義したシグネチャを作成しているためである。新しい亜種の出現からシグネチャを作成し検知可能になるまでの時間差が発生してしまうと考えられる。筆者や金井らの調査[5]では、新しいマルウェアの亜種が発生してからセキュリティ対策ソフトが検知のためのシグネチャを作成するまで、およそ3日間ほどかかっていると推測される。その間にも新しい亜種が次々に発生し、攻撃コードとともに送信されている現状では、セキュリティ対策ソフトだけでの感染防止は事実上限界となっている。

ホスト上でのセキュリティ対策ソフトのもう一つの検知方法として、プログラムの活動パターンからマルウェアの類似度を推測するヒューリスティック検知、ビヘイビア型検知の手法が挙げられる。これらは、セキュリティ対策ソフトの開発元が詳細な情報を公開していないため、詳細についての言及はできないが、ファイルアクセス方法、OSに関するデータへのアクセス方法やアクセス内容をもとに、マルウェアと疑わしいプログラムを検知する方法である。しかし金井の調査[14]によれば、ヒューリスティック・ビヘイビア型検知の機能が搭載されているSymantec社のNorton AntiVirus2009によって無作為に抽出した42種類のマルウェアの検知を試みたところ、33件は検知に成功したが、9件は検知に失敗した。この調査では検証のための母数が十分に多いとは言えないが、それでも20%以上のマルウェアの検知に失敗していることから、既存のセキュリティ対策ソフトで感染を防ぐのは困難になっていることが明らかになっている。

マルウェアはホストに感染した直後に、ホスト内のデータを改ざんし、自身を隠蔽、延命化するため、ホスト上における感染後の事後対策はさらに困難になっている。例えば、竹森らの調査[12]によれば、マルウェアは感染後に多数のマルウェアを別途ダウンロードし、ホスト内の様々なディレクトリやファイルに設置する。マルウェアの亜種が頻繁に発生していることから、セキュリティ対策ソフトでファイルシステムを検査しても、全てのマルウェアを正確に検知するのは困難であると考えられる。検知されなかったマルウェアが再度新しいマルウェアをダウンロードし感染ホストに設置するという活動を繰り返せば、感染ホストはボットとして延命することとなる。

また、感染したマルウェアはホスト上のセキュリティ対策ソフトを無効化する機能を持つ場合がある。これはセキュリティ対策ソフトのプロセスを停止させるだけでなく、セキュリ

ティ対策ソフトが更新されたシグネチャを取得するサーバのドメイン名の名前解決を妨害したり、セキュリティ対策ソフトの一部の機能を無効化するというものが挙げられる。これによって、ユーザからはセキュリティ対策ソフトが動いているように見えるにも関わらず、ホスト内部ではマルウェアが活動しているという状況が起きてしまう。

さらに、OSの機能やハードウェアを改ざんするマルウェアも存在する。感染したOSのファイルアクセスに関するAPIを改ざんするなどして、ファイルシステム上に存在するマルウェアの存在を隠蔽したり、ハードディスクのマスタブートレコードにマルウェアの起動用コードを書き込み、OSが再インストールされても再度新しいマルウェアをダウンロードして感染させるものもStone-Grossらの調査[15]によって明らかになっている。また、Basic Input Output System(BIOS)やネットワークインターフェースカードの記憶領域にマルウェアのコードを埋め込むなどの手法も、技術的には実現されてきている。

このようなマルウェアを機能から、既存の対策のみを用いてホスト上のみで対策するのは非常に難しくなっている。そのため、セキュリティの原則に基づき、多層的な対策を提案していく必要がある。

2.4.2 通信の制限による対策

通信の一部を制限してネットワーク上の脅威から守る代表的な手法としてファイヤウォールが挙げられる。ファイヤウォールは主にOSI参照モデルにおけるネットワーク層とトランスポート層の情報に基づいてネットワークの通信を制御する。主な役割としては、外部ネットワークから内部ネットワークへのアクセスを拒否する、外部ネットワークから公開サーバへのアクセスの中から不要と判断されるアクセスを拒否する、などが挙げられる。また、ファイヤウォールにはネットワーク全体の通信を制御するネットワーク型ファイヤウォールと、各ホスト上で動作するホスト型ファイヤウォール(あるいはパーソナルファイヤウォール)の2種類に分類できる。

2003年以降、感染したホストが他のホストに対して攻撃コードを送信するマルウェアが流行した際、ファイヤウォールは有効な防御手法として導入が進んだ。これらのマルウェアを防ぐためには外部ネットワークからの攻撃だけではなく、内部の近傍ホストからの攻撃も防ぐ必要があったため、ネットワーク型のファイヤウォールだけではなく、ホスト型ファイヤウォールも利用されるようになった。しかし、ファイヤウォールはあくまで外部から内部、あるいは自ホストへのアクセスを制御するのが目的であり、内部から外部にアクセスを制御する際に、悪意ある通信のみを遮断するのは極めて難しい。そのため、Webサイトの閲覧を契機として攻撃コードを送信する受動攻撃型マルウェアに対しては無効である。これは、ファイヤウォールがネットワーク層とトランスポート層の情報に基づいてアクセス制限をしているためであり、ファイヤウォールの構造上の限界であると言える。

2.4.3 通信の監視による対策

通信を監視することで、セキュリティに関連する事象を発見するための手法として、もっとも普及しているのはネットワーク型侵入検知システム (Network-based Intrusion Detection System, IDS) である。IDS は統計的処理によってセキュリティに関連する事象を発見するアノマリ型とあらかじめ指定した通信パターンを発見するシグネチャ型の2種類に分類できるが、本論文ではシグネチャ型のみをIDSと呼ぶ。

IDS はあらかじめ攻撃コードやマルウェアに関する活動の通信パターンが定義されたシグネチャを読み込み、ネットワークの通信に含まれるパケットを解析する。読み込んだシグネチャと通信パターンが一致した場合、悪意ある活動を検知したとみなす。ここでの通信パターンとは、Internet Protocol(IP), User Datagram Protocol(UDP), Transmission Control Protocol(TCP), Internet Control Message Protocol(ICMP) のヘッダ情報や、各プロトコルのデータセグメント、ペイロードのデータ列パターンなどを指している。あるサービスに対する攻撃コードを含んだパケットを発見したい場合は、そのサービスのプロトコル、使用しているポート番号、攻撃コードの一部を指定しておけば、条件に合致したトラフィックを検知した際に、管理者に対してアラートが発行される。

しかし一般のIDSは、検知のために利用している情報が部分的であるため、誤検知を多発しやすいという問題がある。IDSは分類システムであるため、検知するべきものを見逃してしまう False negative と検知するべきではないものを検知してしまう False positive の2種類の誤検知があるが、IDSでは特に False positive が問題になることが多い。また、外部からの攻撃コードを検知できたとしても、すでに攻撃コードが利用しようとしている脆弱性がすでに修正されていたり、ホストベースのファイアウォールによって保護されていた場合、その攻撃コードにはリスクがないと判断出来る。このような検知を対策済み検知と呼ぶが、IDSが検知した攻撃のリスクをその都度判断するのは難しい。よって、マルウェアの活動に対し、IDSは有効性が低いと言える。

既存手法の検知精度を示すため、筆者らの環境において2008年10月1日から同月31日までNepenthes 0.2[16]で収集した188種類のマルウェア検体を実験環境下のVirtualBox 2.0[17]上でWindows XPを動作させ、検体を実行し、検体実行後、それぞれ1時間ずつ発生するトラフィックを収集した。筆者らの調査[18]によってボットによるものと認められた71件のトラフィックデータを既存のIDSであるSnort 2.8.1.2[19]によって検査した。検査には2008年10月30日版のSourcefire VRT Certified RulesとEmerging Threats[20]で配布されているシグネチャを用いた。この結果、41件のトラフィックデータからはボットの活動を特定し検知したが、残りの30件は検知できなかった。各通信からはなんらかのセキュリティ侵害の可能性を示すイベントとしては検知されたが、これらのイベントからボットの活動を特定、検知するためにはボットに関する知識に基づく追調査が必要であり、多くの管理者はこのような知識や経験がないため、出力された情報をボット対策として有効に活用するのは困難であると考えられる。

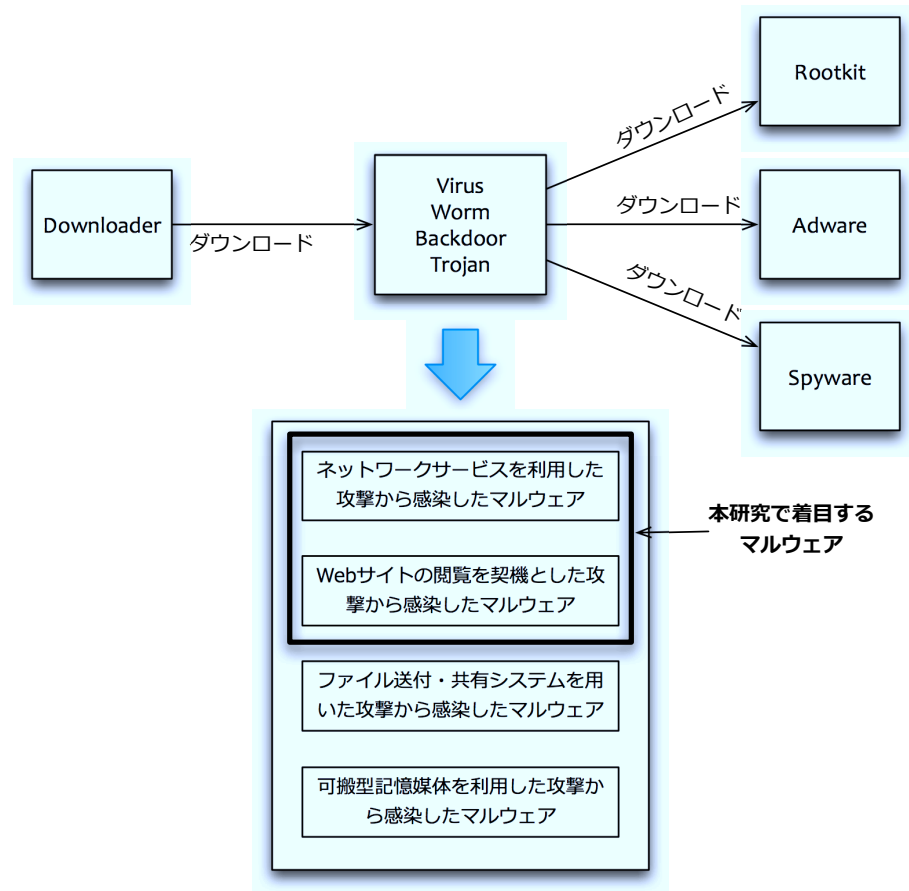


図 2.2: マルウェアの分類と本研究での着目

さらに、セキュリティ対策ソフトでボットが検知できない場合が多くなる理由として、悪意ある通信の巧妙化が進んでいる点も挙げられる。Paul らの調査 [21] によれば、C&C サーバからのコマンドを受信するためのコントロールチャネルには Internet Relay Chat(IRC) が多く利用されており、通信内容にも特徴的な文字列を避ける傾向が見られる。既存手法でネットワーク通信からマルウェアを見つける場合は、マルウェア独自の通信プロトコルや通信内容に着目し、シグネチャを作成していたが、ボットが一般に用いられるプロトコルを利用したり、特徴的な文字列による通信を避けた場合、既存手法では誤検知が発生しやすくなる。そのため、現在普及している技術では、ネットワーク監視によるマルウェア対策も困難となっている。

2.5 本研究で対象とするマルウェア

本論文では、これまで取り上げてきたマルウェアの中から、特に被害の中心となるマルウェアに対象を絞る、対策を論じる。第 2.1 節で述べたとおり、マルウェアの定義は非常に広く、

多様なソフトウェアを指す。そこで本論文では、ボットネットにおいて感染ホストの制御や悪意ある活動の活動起点となるマルウェア（図 2.2 中の Virus, Worm, Backdoor, Trojan）を対象として、対策を考案した。これらのマルウェアを対処することによって、マルウェアによる全体的な被害を軽減していけると考えられる。

本論文執筆時にはマルウェアの多様化が進み、マルウェア分類基準の設定が困難になっている。そのため、セキュリティ対策ベンダごとに呼び名が異なっているが、本論文では各セキュリティ対策ベンダが用いているマルウェアの命名規則などを参考にマルウェアを分類し、その関係を図 2.2 に示している。一般的な定義を用いて説明すると、Virus は正常なソフトウェアのプログラムそのものを改ざんして自らを寄生させながら感染を広げるマルウェア、Worm は単独のソフトウェアとして感染を広めていくマルウェア、Backdoor は感染後に命令を受信して悪意ある活動をするマルウェア、Trojan は悪意のないソフトウェアに見せかけて感染するマルウェア、Rootkit は感染ホスト上の OS 機能を改ざんし攪乱するマルウェア、Adware は不要な広告などを表示するマルウェア、Spyware は情報資産を盗みだすマルウェアと分類できる。本論文では、これらの中から悪意ある活動の中心になると見られる Virus, Worm, Backdoor, Trojan に着目した。

図 2.2 では説明でとりあげたマルウェアの関係を示している。Virus, Worm などは自律的に感染を拡大させるものも多く見られるが、近年のボットネットにおいて活動しているマルウェアは、複数のソフトウェアによる連携が多く見られる。ボットネットでは各ソフトウェアの役割がはっきりと分かれている場合が多く、各機能ごとにソフトウェアを用意して動作させる。これはハードにとって、マルウェアの静的解析によってボットネットにおけるマルウェアの連携手順や新しい機能を隠蔽する目的があると推測される。また、ボットネットが大規模化したことで機能を分離させた構造的な管理に移行したとも考えられる。例えば、マルウェアの攻撃が成功して任意のコードの実行が可能になったホストでは、最初に C&C サーバなどと通信するようなマルウェア（Virus, Worm, Backdoor, Trojan）が実行される事は少なくなっている。ボットネットを運用するにあたり、最新のマルウェアに感染させる必要がある、あるいはマルウェアの配布サーバが転々としているなどの状況により、感染拡大のために攻撃をしてくるホストと感染するマルウェアを送信するホストは異なる場合が多くなった。これにともない、マルウェアをダウンロードするだけの機能を持つ Downloader（あるいは Dropper）が最初に実行され、攻撃者が指定した手順にもとづいて Virus, Worm, Backdoor, Trojan などを取得し、実行している。また、Virus, Worm, Backdoor, Trojan は感染後に必要に応じて新しいマルウェア（Rootkit, Adware, Spyware）をダウンロードして、不正にインストールする。Rootkit, Spyware, Adware も単独で感染することも多いが、Virus, Worm, Backdoor, Trojan などがハードの命令に基づいてインストールすることで各マルウェアが連携し、より深刻な被害になってしまう。

よって、本研究ではこれらの悪意ある活動の中心となる Virus, Worm, Backdoor, Trojan に着目し、さらに第 2.2.2 節で述べた感染経路からネットワークサービスを利用した攻撃に

よって感染するマルウェア（能動感染型マルウェア）と Web サイトの閲覧を契機とした攻撃によって感染するマルウェア（受動感染型マルウェア）の2つに着目している。計算機の黎明期にはネットワークが普及しておらず、フロッピーディスクをはじめとする可搬型記憶装置による感染が多かったが、その後ネットワークのインフラが整備されるにしたがってネットワークを経由した攻撃へと変化していった。ネットワーク普及の初期では常時ネットワークに接続しているホストが少なかったため、感染元ホストと攻撃対象のホストが非同期に感染を拡大させやすいファイル配布や共有システムを用いた攻撃が多かった。しかし、2000年以降はネットワークの普及にともないネットワークサービスの直接攻撃による感染が急増し、さらに2006年以降はWebサービスの普及とともにWebサイトの閲覧を契機とした攻撃が増加している。特にネットワークサービスの直接攻撃とWebサイトの閲覧を契機とした攻撃による感染は即時性が高いことから急増しており、大きな被害を及ぼしている [1]。よって、本研究ではネットワークサービスの直接攻撃によって感染するマルウェアを能動攻撃型マルウェア、Webサイトの閲覧を契機とした攻撃によって感染するマルウェアを受動攻撃型マルウェアとし、これらを中心とした対策に取り組む。これにより、ボットネットにおいて多く活動しているマルウェアへの対策となり、マルウェアによる犯罪行為の低減につながると考えられる。

2.6 本章のまとめ

コンピュータネットワークと計算機が爆発的に普及し、様々なサービスが計算機上から利用可能になったことによって、悪意ある人物もこれを利用した犯罪行為を開始した。多くのサービスを利用するに当たり、ユーザが使用する情報資産や計算機資源は様々な側面から悪意ある人物にとって魅力のあるものとなり、それを利用、盗用することによって経済的利益をえることが可能となった。

グローバルネットワークであるインターネット上では様々な悪意ある活動が発生しているが、ボットネットの出現により大部分の犯罪がボットネット上で行われるようになった。マルウェアに感染したホストはボットとなってボットネットに参加し、感染したホスト内の情報資産の機密性を侵害したり、計算機資源やネットワーク資源を無断で利用して、様々な犯罪行為をおこなう。近年はこれによって金銭的利益を得ることが可能なり、様々な側面から犯罪行為が加速している。

特に金銭的利益が得られるようになったことで、ボットネットによる犯罪のビジネスモデルが誕生し、様々な悪意ある活動の効率化が図られるようになった。金銭的利益が得られるため、金銭に基づいた取引が犯罪者間で成立するようになり、各自の担当の専門化が進んでいる。さらに、マルウェア作成についても作業の分担や生成ツールの流通などによって、マルウェアの生成や配布に関する敷居も下がっている。さらに、継続的に利益を得るためにボットネット規模の維持と拡大はハッカーにとって重要であり、マルウェア延命技術の汎用化や効率化、そして攻撃行為の拡大が社会的な問題に発展している。このような現状に対処するた

めには、マルウェアの活動傾向について熟知し、新しい対策に取り組んでいく必要がある。

第3章 事前調査

本章では多様な感染経路に対応したマルウェアの収集、解析環境の構築と、構築した環境によって得られた解析結果について述べる。マルウェアの脅威が深刻化することによって、マルウェア対策の研究に取り組む研究者が多くなった。しかし、研究に対して有用性が高いマルウェアの実行ファイル本体やマルウェアの通信データ、マルウェアのソースコードなどは攻撃コードなどを含む可能性が高く、悪用される危険性がある。そのため、マルウェアについては表面的な情報しか公開していない研究機関が多い。

そのため、本研究ではマルウェアを収集、解析する環境を独自に構築することでマルウェアに関する情報の収集を試みた。構築にあたってはマルウェアに感染したホストが自律的に感染活動を開始する能動攻撃型マルウェアだけではなく、Webサイトの閲覧を契機として感染する受動攻撃型マルウェアの収集、解析にも着目した。従来、マルウェアの研究では能動攻撃型マルウェアが着目されており、マルウェアの検知に関する研究の多くはこれらの調査に基づいた手法が提案されている。これに対し、本論文では実験用の環境で能動攻撃型と受動攻撃型のマルウェアに関する通信データをそれぞれ収集、分析した。

分析によって得られた能動攻撃型マルウェアと受動攻撃型マルウェアの相違点を明らかにし、既存研究による対応の難しさについて述べる。本環境を用いることで能動攻撃型マルウェアと受動攻撃型マルウェアの両者に関する通信データを収集できた。これを比較した結果、それぞれの活動パターンが大きく異なることが明らかになった。既存研究におけるマルウェアの検知手法では能動攻撃型マルウェアの活動パターンに着目したものが多かったため、既存研究の有用性が低下していると言える。

3.1 多様な感染経路に対応したマルウェア収集および解析環境の構築

マルウェアによる脅威が拡大する中、マルウェアの解析や対抗策の研究が急務となっている。マルウェアによる犯罪行為から経済的利益が得られるようになり [10, 15]、マルウェア作成者は多様な手段を用いて検知の回避や耐性の向上を図っている [12]。そのため、従来以上に感染が拡大し、新種のマルウェアの出現頻度や、傾向の変化が著しくなっている。マルウェアの対策を実施する際には、その活動形態の調査が必須である。急速に変化するマルウェアに対し、マルウェアの活動を知らなければ効果的な対策を提案するのは難しい。

マルウェアの解析には、感染対象となる OS 上でマルウェアを動作させて情報を収集する動的解析と、バイナリコードを解析して動作を把握する静的解析がある。静的解析はマルウェア

アに関する詳細な情報を入手できるが、近年のマルウェアは意図的に難読化されており、解析には多大な時間を要する。さらに、新しいマルウェアの出現頻度が高いため [13]、短時間でマルウェアの動作に関する情報を得られる動的解析と併用するのが効果的である。

本節では、マルウェアの収集・解析環境構築における要件を定義し、設計・実装した多様な感染経路に対応する解析環境について述べる。これまで、収集と解析は攻撃を待ち受けるハニーポットを利用したものが多く、積極的に通信を開始し感染を拡大させるマルウェアのみに対応しているものがほとんどであった。これに対し、近年は Web サイトの閲覧などによって感染するマルウェアが増加しており、マルウェアの収集・解析環境の多様化が求められている。本稿では今後もマルウェアの活動形態が変化し続ける可能性を考慮し、複数の感染経路に対応できるマルウェア収集・解析環境を構築した。また、研究者が小規模な環境から構築する点を考慮することで、拡張性や柔軟性などの要件を定義し、これを満たすよう実装した。

3.1.1 マルウェア解析および収集環境構築における課題

従来、マルウェアの収集・解析は能動攻撃型のものが注目されてきた。能動攻撃型マルウェアとは、攻撃元ホストから通信を開始し、攻撃コードを送信することによって感染を拡大させるマルウェアである。能動攻撃型マルウェアは 2003 年に発生した Blaster[8]などを筆頭に大量に発生しており、擬似的なサービスを提供することで攻撃を待ち受けて攻撃コードやマルウェアを収集するハニーポットが多く利用されてきた。

しかし、2007 年の Provos らの調査 [9] から受動攻撃型マルウェアが注目されはじめた。受動攻撃型マルウェアは攻撃元ホストから一方的に攻撃コードが送信されるのではなく、ユーザの動作が起因となって攻撃が発生する。Provos らの調査では Web ブラウザであらかじめマルウェア本体をダウンロードさせ、Web ブラウザやプラグインの脆弱性を利用して感染する攻撃 (drive by download) に着目している。この攻撃手法はファイヤウォールや Network Address Translator(NAT)によって外部からの通信が遮断されているネットワークのホストでも、マルウェアに感染させることができ、マルウェア感染の危険性が高まっている。

このような感染経路をはじめとするマルウェアの活動形態の変化は、マルウェアの対策にも大きな影響を及ぼすため、新しいマルウェアに対応した解析環境が必要となる。筆者らの調査でも、能動攻撃型マルウェアと受動攻撃型マルウェアでは異なる傾向があることが明らかになっている。一部のマルウェア対策の研究では、通常の通信とマルウェアによる通信の傾向の違いを利用した検知手法が提案されているが、通信傾向が変化することで検知が困難になってしまう。そのため、継続的にマルウェアの活動を観測しつつ、多様な感染経路に対応できるマルウェアの収集・解析環境が必要である。

3.1.2 多様な感染経路に対応した収集・解析環境の実装

マルウェアの収集・解析における要求事項を挙げる。

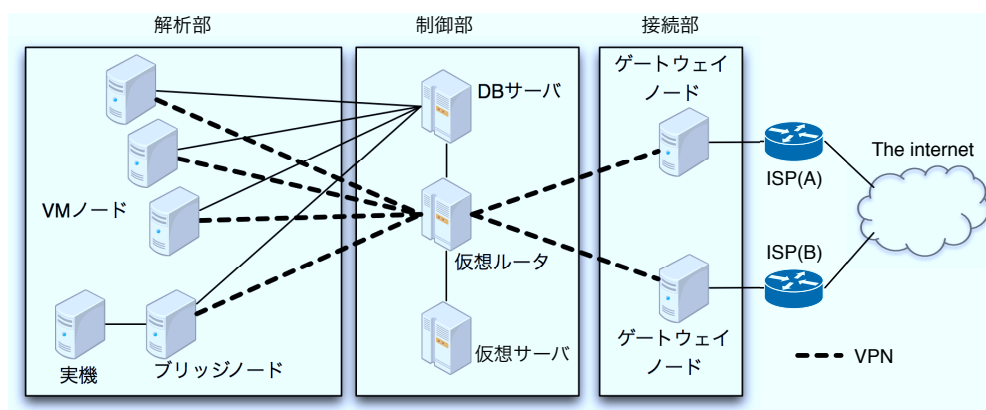


図 3.1: マルウェア収集・動的解析構成概要

- **並列性:** 動的解析は短時間で多くのマルウェアを解析できるのが利点であるため、複数のノードを同時に動作させる構成が望ましい。また、収集に関しても並列して実行することによって、より多くのマルウェアを収集できる可能性が高まる。
- **完全性:** マルウェアを実行した際に、意図しないデータや通信が改ざんされる事を防ぐ必要がある。また、同一の解析環境にある別のマルウェアによって、収集するデータが干渉されないように配慮しなければならない。
- **頑健性:** 一部の解析処理に問題が発生しても、全体の系は処理を継続できるように実装する必要がある。
- **拡張性:** 今後、マルウェアの増加速度はより加速するものと予想されるため、収集・解析環境を任意に拡張できる実装が必要である。特に、システム全体を停止させずに拡張・変更できる環境構成が望ましい。
- **柔軟性:** 近年、受動型攻撃が増加したように、今後も感染経路が変化していく可能性がある。感染経路や活動形態にあわせて、接続する機器やネットワークを選択できるような柔軟性が求められる。
- **通信の制御:** マルウェアの収集や一部の解析では、インターネットに接続しなければならない。しかし、マルウェアを実際に実行する動的解析では、他のネットワークに対する迷惑行為や攻撃が発生する可能性がある。そのため、必要に応じて通信を遮断する、外部ネットワークに見せかけた疑似サービスへ誘導する、通信量を制限するなどの措置が適宜必要となる。

これらの要求に基づき、マルウェアの収集・解析システムを実装した。本実装ではマルウェアの収集において、攻撃元ホストから通信を開始する能動型攻撃だけではなく、悪意がある

Webサイトの閲覧時に感染する受動型攻撃についても対応できるシステムを構築した。本システムでは、主に仮想マシンを利用してマルウェアの収集・解析を実施しており、仮想マシンの起動時に任意の命令を発行できるよう構成されている。外部ネットワークからの攻撃通信をシステム内のホストに転送すれば、従来の能動攻撃型マルウェアに対応したハニーポットとして動作し、起動時に攻撃コードが含まれると予想されるWebサイトを自動的に閲覧する指示を発行すれば、受動攻撃型マルウェアに対応したハニーポットとなる。この構成を応用すれば、より新しい感染経路が発生しても柔軟に対応できると考えられる。

システム全体の構成

本システムは汎用的な大規模テストベッドではなく、必要に応じて部品を増設する専用のシステムとして設計されている。そのため他の用途に使うのは難しいが、比較的安価に設置・運用できるのが特徴の1つである。

図3.1は実装した収集・解析環境の概要を示している。本システムはVMノードとブリッジノードおよび実機からなる解析部、仮想ルータ、仮想サーバからなる接続部、ゲートウェイノードからなる表層部の3つのパートで構成されている。

- **VMノード**: 仮想マシンを用いて、マルウェアが感染対象とするOSを動作させ、マルウェアへ感染させる、あるいはマルウェアを動作させることで、マルウェア本体や活動に関する情報を収集するノード。各ノード上では複数の仮想マシンを動作させ、マルウェア収集・解析の並列処理を実現する。本システムではホストOSとしてUbuntu Linux, 仮想マシンにVirtualBox[17]を用い、VirtualBox上でWindows XP(R) Professional(パッチ未適用)を基本として利用した。
- **ブリッジノード・実機**: 実機上にインストールしたマルウェア感染対象のOSを動作させ、収集・解析を実施する。自身の解析を防ぐために、仮想マシン上での動作を検知して動作を変更させる機能を有するマルウェア[22]が存在するため、同一のOSやソフトウェアを使っている場合、仮想マシン上での動作と実機上での動作が異なる可能性があり、実機での収集・解析環境も必要となる。実機で動作させた場合、特殊な制御を当該ホスト上で実施するのは難しいため、実機上で発生した通信をブリッジノードを介してネットワークに接続させる。
- **仮想ルータ**: VMノードとブリッジノードの解析部とゲートウェイノードを仲介する本システムの基幹ルータ。ただし、各VMノード、ブリッジノード、ゲートウェイノードとは全てVPNを利用して接続している。仮想ルータはVMノード、ブリッジノードからきた通信を収集・解析内容に応じてルーティングし、必要に応じて特定のサービスを仮想サーバに誘導する。さらに、ファイアウォールを用いることで外部へ送信してはならない通信を遮断し、IDSを用いることで通信の概要を把握できる。本実装ではOSに

Ubuntu Linux, VPN に OpenVPN[23], ファイアウォールに iptables, IDS に Snort[19] を利用した。

- **仮想サーバ:** マルウェアが外部のホストに対して送信するリクエストの宛先を仮想サーバに改ざんし、あたかも外部のホストが応答しているかのように見せかけるためのサーバ。通信の誘導には宛先 Network Address Translation (NAT) を利用している。例えば DNS の場合、偽のアドレスを応答したり、DNS サーバを経由して実際の問い合わせ結果を応答するのも可能である。また、SMTP の場合は大量の迷惑メールが外部に送信されないように、受信したメールを捨て続けるブラックホール SMTP サーバとして動作させる事も可能である。また、TCP の SYN パケットに対して偽の SYN-ACK パケットを送信する dumnet[24] を利用することで、擬似的に TCP のセッションを確立させ、マルウェアにパケットを送信させるようにも設定できる。本実装では OS に Ubuntu Linux, DNS サーバに bind9, SMTP サーバに Postfix を利用した。
- **DB サーバ:** マルウェアに関する情報を一元的に管理するサーバ。収集したマルウェアの検体や、悪意のある URL とドメイン名、解析結果であるマルウェア通信などの情報が蓄積されている。VM ノード上の仮想マシンの多くは自動で実行され、その際にどのような解析をするか、どの検体を利用するか、どの URL に接続するかなど、動作に関する指示を発行する役割も DB サーバが担う。さらに解析結果に関する情報も自動的に DB サーバに送信される。
- **ゲートウェイノード:** インターネットへと接続するため、外部接続性を提供するノード。マルウェアの振る舞いは感染対象のネットワークによって異なるという仮説があるため、これらの検証のために複数の外部ネットワークと接続性を用意した。また、冗長性などの点からも複数の回線を持つ構造が望ましいと考えられる。

一定規模のテストベッドを構築する場合、専用のホスト設置場所やネットワークを用意し運用しなければならないが、本システムは規模に関わらず容易な構築が可能である。インターネット上の保護された通信上に環境を構築しているため、インターネットが利用できれば物理的な設置場所やネットワーク構成などに依存しない。また、システム構成を変更する場合も、比較的容易に VM ノードやブリッジノードを追加できる。

VM ノードの構成

本システムの収集・動的解析は VM ノードが主体となっている。村上らの調査 [22] で述べられている通り、仮想マシン上では動作が制限されるマルウェアも存在するが、本システムでは効率的な収集・動的解析を優先している。図 3.2 に VM ノードの構成を示す。VM ノードは複数の仮想マシンを有し、それぞれを並行して動的解析を実行できる。仮想マシン数は

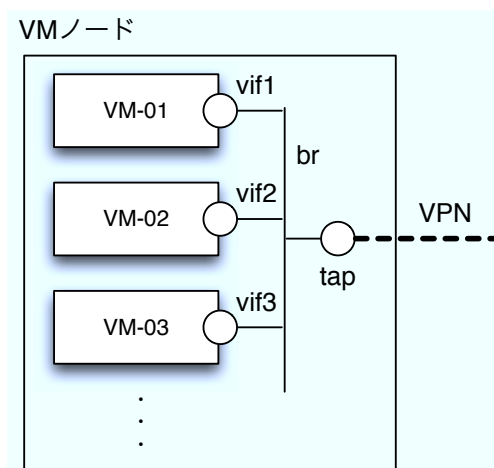


図 3.2: VM ノード構成

VM ノードの性能とゲスト OS の推奨性能を考慮して設定するのが望ましい。各仮想マシンには仮想ネットワークインターフェース (vif_n) を有し、そのインターフェースと VPN によって作成される仮想ネットワークインターフェース (tap) をブリッジすることで、直接インターネットに接続するのを防いでいる。各仮想マシンのネットワークインターフェース vif_n では通信内容を記録すると同時に、iptables と ebtables[25] を用いて同一セグメント内の別の仮想マシンに送信される通信を遮断し、複数種類のマルウェアが混在しないよう配慮している。

各 VM ノードは仮想マシン上のゲスト OS を起動する時に DB サーバから収集・解析に関する指示、収集・解析の継続時間を取得する。現在、1) 取得済みのマルウェアを実行する、2) 悪意のあると考えられる Web サイトを Internet Explorer で閲覧する、の 2 種類の動作が実装されており、各 VM は HTTP 経由で DB サーバから指示を得る。各 VM はホスト OS との共有ディレクトリが設定されており、DB サーバから指示された内容を元に共有ディレクトリにバッチファイルを生成する。バッチファイルには Internet Explorer で参照する URL や、実行するマルウェア検体のファイル名が記述されており、ゲスト OS は起動時に共有ディレクトリに設置されているバッチファイルを自動的に起動する。また、とくにゲスト OS に指示を与えず、任意のゲートウェイノードに送信される攻撃通信を転送することによって、従来の能動型攻撃を受け付けるハニーポットとしても動作する。指定された収集・解析の継続時間を経過した後はゲスト OS を終了させ、感染前の状態に復元する。

各 VM ノードは自立分散的に動作しており、システムへの追加や離脱がシステム全体に及ぼす影響は小さい。また、何らかのトラブルで 1 部の VM ノードが停止しても他の VM ノードは継続して動作し続ける。これによって、拡張性や頑健性が実現されている。

表 3.1: 分析に用いたデータセットの一覧

	データ収集時期	期間	データ数	説明
D_p	2008 年 7 月～11 月	20 分	255	受動攻撃型マルウェア
D_a	2008 年 5 月～10 月	60 分	484	能動攻撃型マルウェア
D_c	2008 年 4 月	–	144	CCC DATAsset 2008
D_m	2009 年 1 月	15 分	250	MAWI Traffic Archive[26]

3.2 収集したマルウェアの解析

第 3.1 節で述べたマルウェア収集・解析環境を運用したことで得られたデータを解析した。

3.2.1 マルウェアの通信データ収集

受動攻撃型マルウェアの活動を調査するため、Web ブラウザの脆弱性を利用して感染するマルウェアの通信データを収集した。さらに、比較対象とするために能動攻撃型マルウェアなどの通信データを用意した。各データセットの情報を表 3.1 にまとめ、各データセットの収集方法について述べる。

検体および通信データの収集

受動攻撃型マルウェアは、Web ブラウザの脆弱性を利用して攻撃コードを送信するものを収集した。Malware Domain List[27] では悪意のある Web サイトの URL をまとめており、攻撃コードが含まれる Web サイトも公開されている。この一覧に含まれる URL を 2008 年 7 月 15 日から同年 11 月 14 日までの間に第 3.1 節において示した仮想マシン上の Internet Explorer (Version: 6.0.2600.0000) によって閲覧し、20 分間通信を監視した。790 件の URL を調査し、攻撃が成功したと判断した 255 件を分析対象とした。攻撃成功の判断では、Windows の実行形式ファイルのパターン [28] のダウンロードに着目した。Windows の実行形式ファイルがダウンロードされなかった通信データには、対象とする Web サイトが削除されていたものや、Web サイト閲覧後に通信が発生しなかったものが含まれる。攻撃が成功したと見られる通信データセットを D_p とする。

比較用の能動攻撃型マルウェアは、筆者らが運用しているハニーポットによって検体を収集し、第 3.1 節で示した実験環境において動的解析を実施した。2008 年 5 月から同年 10 月までの間にクラス C のネットワークを 16 個用意し、Nepenthes 0.2.0[16] を実行して検体を収集した。収集された検体は 935 種類で、これらを 2008 年 11 月 1 日から同月 6 日の間に実験環境で実行した。各検体は仮想マシンの起動直後に自動的に実行し、1 時間分の通信データを収

表 3.2: 能動攻撃型マルウェアと受動攻撃型マルウェアの通信内容比較

観測された通信	$D_a(N = 484)$	$D_p(N = 255)$
複数ホストに対する感染活動	278 (57.44 %)	5 (1.96 %)
感染後の IRC による通信	342 (70.66 %)	2 (0.78 %)
感染後の HTTP による通信	150 (30.99 %)	234 (91.76 %)
感染後の SMTP による通信	9 (1.86 %)	40 (15.69 %)

* 割合は小数点第3位以下を四捨五入

集した。実行した検体のうち、マルウェアによる通信が観測されたのは484件であった。これによって収集された通信データセットを D_a とする。

その他のデータ

一般的なクライアントによる通信とマルウェアによる通信の傾向を比較するため、MAWI Working Group Traffic Archive[26]において公開されているデータを利用した。通信データは[26]における samplepoint-F の2009年1月8日14:00-14:15のデータから、TCPのポート80番にアクセスしていた250件のIPアドレスを無作為に抽出した。これらのIPアドレスによる通信データの集合を D_m とする。

また、著者らが以前に実施したCCC DATASet 2008による能動攻撃型マルウェア調査データも利用する。CCC DATASet 2008はサイバークリーンセンター[11]によって運用されているハニーポットで収集されたデータである。脆弱性を修正していないWindows XPをインターネットに接続し、能動攻撃型マルウェアからの攻撃と感染によって発生する通信データを144件収集している。このデータセットを D_c とする。

3.2.2 受動攻撃型マルウェアの分析

受動攻撃型マルウェアと能動攻撃型マルウェアの振る舞いの観測結果を比較したところ、両者の間には異なる傾向が確認された。受動攻撃型マルウェアは能動攻撃型マルウェアと比較して、大量の他ホストへの感染活動やIRCによるC&Cサーバと通信するマルウェアが少数であることが観測された。

能動攻撃型マルウェアとのプロトコル比較

表3.2は能動攻撃型マルウェアと受動攻撃型マルウェアの通信方式を比較した結果を示している。表ではNetBIOSなどの脆弱性を利用した他ホストへの感染活動が観測された検体数、感染後にIRCによる通信が観測された検体数、感染後にHTTPによる通信が観測された検体

表 3.3: マルウェアのダウンロード方法の比較

	D_a	D_c	D_p
HTTP による転送	465 件	337 件	1,075 件
TFTP による転送	0 件	1 件	0 件
独自プロトコルによる転送	137 件	351 件	0 件
合計	602 件	689 件	1,075 件

数, 感染後に SMTP による通信が観測された検体数と, 各データセットの全数に対する割合を示している. 各プロトコルの判定にはアプリケーション毎のセッション開始時のペイロードデータのパターン [29] を利用した. D_p における感染後の通信とは, Windows 実行形式のファイルがダウンロードされた後の通信を対象としている.

表 3.2 からは, 多くのマルウェアが感染直後から外部の複数ホストに対して, 調査と攻撃を実施していると分かる. 自律的に感染活動を開始する検体もあるが, 多くの能動攻撃型マルウェアは IRC によって C&C サーバに接続し, 送信された命令に応じて活動している. これに対し, 受動攻撃型マルウェアでは 98% 以上の検体で複数ホストに対する感染活動や IRC による C&C サーバへの接続が発生していない. 一方で, 能動攻撃型マルウェアでは約 30% しか HTTP による通信が観測されなかったのに対し, 受動攻撃型マルウェアは感染後も 90% 以上が HTTP で通信していた.

表 3.3 は $D_{a,p,c}$ について, マルウェア本体のダウンロードにおける通信方式の比較を示している. この表におけるダウンロード数は延べ件数であり, 通信データ数とは異なる. 転送方式は HTTP による転送, TFTP による転送, 独自プロトコルによる転送の 3 種類を挙げた.

能動攻撃型マルウェアを示した D_a と D_c では HTTP による通信の他にも独自プロトコルによる転送が多く見られる. 特に D_c では独自プロトコルによる転送が HTTP による転送を上回っている. また, D_c では TFTP による転送も見られる. これに対し, D_p では全てのマルウェアを HTTP によってダウンロードしており, 他の通信方式によるマルウェア転送は見られなかった.

能動攻撃型マルウェアとの通信傾向比較

図 3.3 では, 各通信データの単位時間あたりの送信データ転送量と受信データ転送量を分布図によって表している. X 軸が 1 秒あたりの送信データ転送量 (bps), Y 軸が 1 秒あたりの受信データ転送量 (bps) をそれぞれ対数で表している. 各点が 1 つの通信データを表しており, $D_{p,a}$ に関する分布を示している.

単位時間あたりの受信データ転送量と送信データ転送量の関係から見ると, D_p は主に 1 つのグループに集約されているのに対し, D_a はおよそ 2 種類のグループに分かれている. 受動

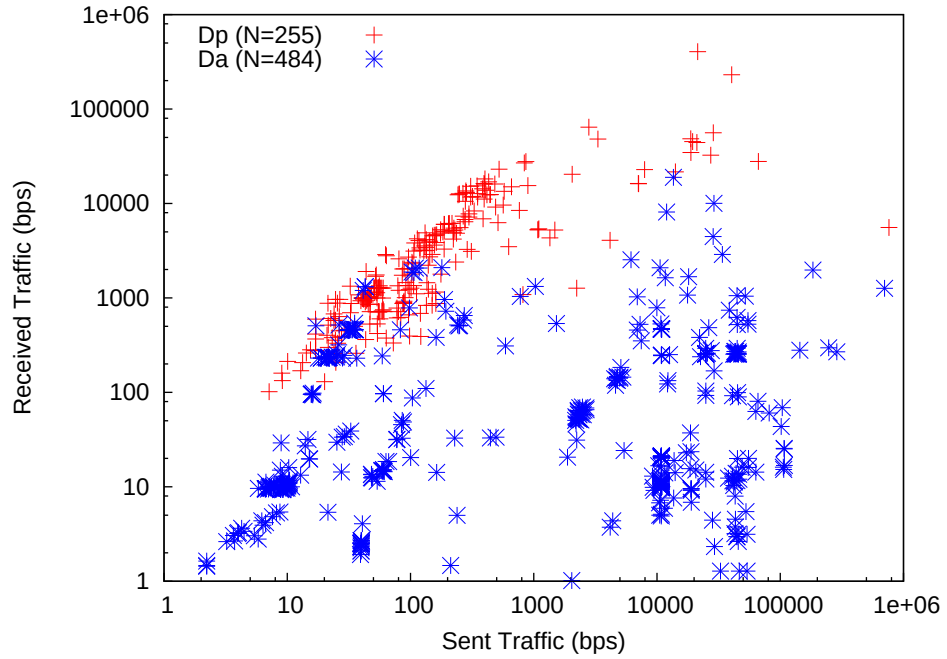


図 3.3: 単位時間あたりの送信データ転送量と受信データ転送量の分布

攻撃型マルウェアは送信が 10bps 以上、受信が 100bps 以上で、単位時間あたりの受信データ転送量が送信データ転送量の 10 倍程度の通信データが多い。これは表 3.2 や表 3.3 に示されているとおり、HTTP による通信で Web サイトからのデータ受信が多いためである。これに対して能動攻撃型は 2 つのグループに分かれている。1 つ目は D_p と同じく HTTP を多用する、もしくは IRC のみの通信しかしないグループである。2 つ目は送信データが 1000bps 以上のグループである。これは外部ホストに対して感染活動をするグループであり、送信データ転送量に対して受信データ転送量は少ない傾向にある。

図 3.4 は $D_{a,p,m}$ に対し、各通信データで観測された SYN パケットとそれ以外の TCP パケットの比率を示した分布図である。X 軸は通信データに含まれる SYN パケットの 1 分間あたりの観測数、Y 軸はそれ以外の TCP パケットの 1 分間あたりの観測数である。 $D_{a,p}$ は NAPT 環境にあり、SYN パケットはマルウェアによる送信と判断できる。

図 3.4 から、 $D_{p,m}$ は類似する通信傾向にあるのに対し、 D_a には $D_{p,m}$ とは異なる傾向を持つグループがあるのがわかる。 D_p は主に SYN パケットの送信数が毎分 10 回以下のグループによって構成されている。多くの受動攻撃型マルウェアは多くのセッション数を発生させず、1 分間に数回程度の HTTP 通信で悪意ある活動に必要な情報を送受信している。さらに、SYN 以外の TCP パケット数が SYN パケットの約 10 倍程度という傾向も見られる。これは D_m でも同様の傾向が見られ、受動攻撃型マルウェアによる通信傾向と一般的なクライアントによる通信傾向は類似していると言える。これに対し、 D_a は主に 2 つのグループに分かれている。

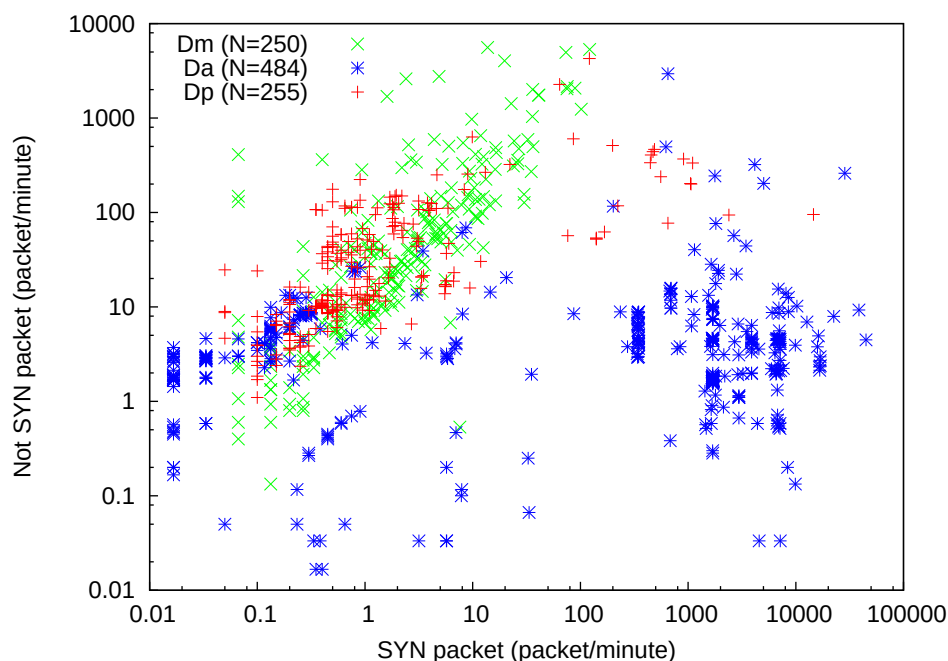


図 3.4: SYN パケットとその他の TCP パケットの観測数分布

一方は SYN パケットの送信が毎分 100 回以下のグループである。これは IRC による通信が多いマルウェアであり、 $D_{p,m}$ と同様の傾向が見られる。一方は SYN パケットの送信が毎分 100 回以上観測されているグループである。このグループは SYN パケットの流量が多いだけではなく、SYN 以外の TCP パケットの流量が少ない傾向が見られ、感染活動のための調査や攻撃が多いと分かる。

HTTP 通信の分析

図 3.5 は $D_{p,m}$ の各通信データにおける HTTP 通信のサーバ側 IP アドレス数を累積度数分布によって表している。X 軸が通信データごとの IP アドレス数、Y 軸が累積度数を割合で示している。 D_p はペイロードから HTTP と判定された通信、 D_m は TCP ポートが 80 番の通信を抽出し、各通信データの IP アドレスを数えた。

D_p と D_m を比較すると、 D_p の方が全体的に通信先 IP アドレスが少ないと分かる。10 種類以下の IP アドレスと通信している通信データは D_p が 80% 以上なのに対し、 D_m では 60% 程度にとどまっている。このように、受動攻撃型マルウェアによる HTTP 通信は一般的なクライアントによる HTTP 通信より通信先ホスト数が少ない傾向がある。短い間隔で大量のホストへ接続すると一般のクライアントとは異なる通信傾向として検知されてしまう可能性があるため、それを防ぐための挙動だと考えられる。

表 3.4 と表 3.5 は、 D_p の HTTP 要求において出現したパスおよび GET 要求に含まれる変数

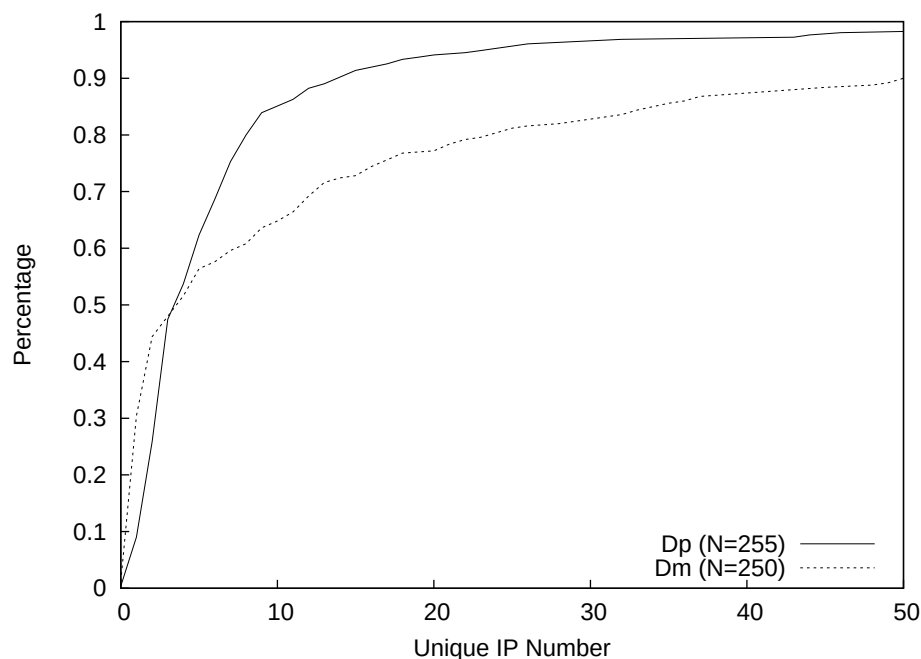


図 3.5: HTTP 通信における IP アドレス数の累積度数分布

名の種類と、出現した通信データ数の上位 5 件を示している。大部分のパスや変数名は全ての通信データにおいて 1 度しか出現していない。最も多くの通信データで出現しているルートディレクトリを示す "/" も全ての通信データの 33% 程度であり、受動攻撃型マルウェアに共通した動作であるとは言い難い。

HTTP の要求におけるパスや変数名は、マルウェア作成者がネットワーク監視によるマルウェア検知を防ぐために、一般的に使用される頻度が高いパスや変数名を多用していると考えられる。特徴的な要求パスや変数名を利用した場合、ネットワーク型侵入検知システムのシグネチャに記述することで、セキュリティ管理者は容易にマルウェアを検知できる。しかし、受動攻撃型マルウェアは悪意のない Web サイトで多用される HTTP の要求パスや変数名を利用することで、マルウェアの活動の検知を困難にしている。

3.2.3 分析結果の考察

感染経路に基づいたマルウェアの活動比較

第 3.2.2 節の分析から、受動攻撃型マルウェアと能動攻撃型マルウェアでは通信の傾向に違いが見られる。これはマルウェア作成者が感染経路によって被害ホストのネットワーク環境を予想し、セキュリティ管理者による検知・駆除を回避するためと考えられる。

能動攻撃型マルウェアに感染したホストが接続しているネットワークは、ファイヤウォー

表 3.4: HTTP 通信で出現した要求パス (上位 5 件)

HTTP 要求のパス	出現した通信データ数
/	86 (33.72%)
/cgi-bin/index.cgi	61 (23.92%)
/forum.php	47 (18.43%)
/in.cgi	29 (11.37%)
/soft3/common/16.gif	22 (8.63%)

* 割合は小数点第 3 位以下を四捨五入

表 3.5: HTTP GET 要求で出現された変数名 (上位 5 件)

変数名	出現した通信データ数
id	109 (42.75%)
v	39 (15.29%)
t	37 (14.51%)
ad	35 (13.73%)
ver	28 (10.98%)

* 割合は小数点第 3 位以下を四捨五入

ルや侵入防止システムによるセキュリティ対策が運用されていない場合が多い。このようなネットワークでは、ネットワーク監視によるセキュリティ対策も適切に運用されていない可能性が高く、定常時と異なる通信が発生してもネットワークインシデントに関するイベントとして検知されない。そのため、IRC による C&C サーバとの通信や、他ホストへの感染活動、迷惑メールの大量送信など、通常と異なる傾向の通信をしても検知・駆除されにくく、マルウェアとしての活動を継続できる。

一方、本論文で調査対象とした受動攻撃型マルウェアは Web ブラウザの脆弱性を利用した感染方式であり、一定以上のセキュリティ対策が導入、運用されているネットワークでも感染する可能性がある。マルウェアによる IRC 通信や独自プロトコルによる通信を防ぐために、ファイアウォールによって外向きの通信を遮断する運用方法もあるが、HTTP と HTTPS によるアクセスは全面的に許可している場合が多い。そのため、ネットワーク監視などのセキュリティ対策が運用されているものと仮定し、一般的なクライアントに見せかけることで、マルウェアが検知・駆除されるのを回避していると見られる。

通信の隠蔽化

受動攻撃型マルウェアは大部分の通信がHTTPであり、一般的なWebブラウザなどによる通信との区別を困難にしている。 D_p は複数ホストに対する感染活動や感染後のSMTP通信を実施するグループ(G_1)とHTTP、DNS、一部のインスタントメッセージサービスの通信のみを実施するグループ(G_2)に分類できる。 G_1 は48件、 G_2 は207件である。 G_1 はマルウェアとしての特徴が見られるのに対し、 G_2 は一般的なクライアントと同様の振る舞いをするため、ネットワーク監視による検知が難しくなっている。

G_1 はマルウェア独自のプロトコルによる通信も見られるため、検知が容易である。例えばMydoom[30]に感染したホストは独自プロトコルによってWindowsの実行形式ファイルを受信し、実行する。Mydoomのプロトコルは現在も利用されており、表3.3における D_a の独自プロトコルによる転送は、大部分がこのプロトコルによるものである。このプロトコルはEmerging Threats[20]などでネットワーク型侵入検知システムのシグネチャとして登録されている。

しかし、 G_2 はマルウェアとしての特徴が少なく、検知が難しい。第3.2.2節や第3.2.2節で示した通り、HTTPによる通信は一般的なクライアントと類似する傾向がある。さらに、HTTPの要求に含まれるパスや変数名もマルウェアとしての特徴が見られず、検知のてがかりにするのは難しい。これは、マルウェア作成者がマルウェアの検知・駆除を防ぎ、生存時間をのばすための実装であると考えられる。

受動攻撃型マルウェアに対する既存検知手法の有効性

ボットを中心としたマルウェアの検知手法の研究では、他のホストへの感染活動やIRCによるC&Cサーバとの通信など、能動攻撃型マルウェアによる通信の特徴や傾向に着目している研究が多い。そのため、既存研究で提案されているネットワーク監視型の検知手法は、受動攻撃型マルウェアに対して有効性が低いものが多い。

Binkleyらの検知手法[31]はあるホストが送信するパケットの種類の割合を調べ、他ホストへの感染活動を検知する手法である。しかし、第3.2.2節の分析結果ではTCPパケットの送信傾向に違いが見られ、[31]の有効性は低い。また、他のホストへの感染活動が発生しやすい能動攻撃型マルウェアではフロー(送信元IPアドレスと宛先IPアドレスの組)が大量に発生するため、フローの傾向分析によってマルウェアの活動を検知するNetwork Behavior Analysis[32]が有効であった。しかし、受動攻撃型マルウェアの多くはHTTPのみの通信であり、HTTPのフローも一般的なクライアントの通信と類似しているため、受動攻撃型マルウェアの検知は難しい。

BotHunter[33]はボットの活動の状態遷移をモデル化し、汎用的な検知を実現する手法である。BotHunterの状態遷移モデルでは、ボットの動作をマルウェア本体の転送、C&Cサーバとの接続、悪意ある活動など、抽象化した状態を定義しており、各状態に該当するイベントを

検出することで状態遷移を発見する。この状態遷移モデルを利用した検知ではイベントの種類や発生順序が変化しても汎用的にマルウェアを検知できるという利点がある。しかし、分析結果で示した通り、受動攻撃型マルウェアが発生させる個々の事象はC&Cサーバとの接続や悪意ある活動が一般ユーザによる通信に類似しており、マルウェアの通信であると判断するのが困難である。また、BotHunterの実装では迷惑メールの送信や、他ホストへの感染活動などを検知のよりどころとしているが、受動攻撃型マルウェアではこのような動作が少なく、状態遷移モデルが有効に機能しないと考えられる。

3.3 外部データを用いたマルウェア共通動作の解析

本節ではマルウェアの通信の類似性に着目し、複数のマルウェアに共通する活動の存在を示す。マルウェアは短期間で亜種が発生するが、基本的な動作の流れは大きく変更されない傾向が見られる。個々の動作は一般的な通信プロトコルやマルウェア独自のプロトコルを利用するなど、マルウェア毎の固有のパターンや組み合わせが存在し、亜種の発生によって多様に変化している。しかし複数の動作を、一連の目的にそった状態遷移として捉えた場合、多くのマルウェアに共通する動作の流れがあると考えられる。マルウェアは感染後にCommand & Control(C&C)サーバから命令を受信し、マルウェアネット管理者が自由に感染ホストを操作するものが多い。これを実現するために実行ファイルのダウンロードやC&Cサーバへの接続が必要であり、これらの動作の流れの共通項を調査した。

3.3.1 マルウェアの活動モデル

本論文ではまずマルウェアの感染直後の活動を表す状態遷移のモデルを示し、研究用データセット CCC DATASet 2008 の攻撃通信データ (以下, CCC DATASet) を用いてモデルの正当性を検証する。CCC DATASet の通信データにはセキュリティ対策が十分ではないホストをインターネットに接続し、マルウェアの感染と感染後の活動が含まれる。感染後の活動記録は限られた時間しか記録していないため、感染の成功と感染直後のモデルに限定して議論する。

本論文では実行ファイルを利用しC&Cサーバから指示をうける悪意あるプログラムをマルウェアと定義する。個々のマルウェアが複雑な動作をするため、ホストの脆弱性を利用して不正利用するために送信される攻撃コードとは別に専用の実行ファイルが別途ダウンロードされる場合が多い。そのため、感染活動だけを実行する実行ファイルを持たない Slammer Worm[3] などは含まない。また、自律的に他ホストへの感染活動を繰り返すプログラムもマルウェアに含めない。

上記の条件に基づき作成したマルウェアの活動モデルを図3.6に示す。図3.6ではマルウェア感染後の通信に関する状態遷移を示している。状態は S_0) 開始状態, S_1) 攻撃コードの受信と感染, S_2) 実行ファイルのダウンロード, S_3) インターネットへの疎通確認, S_4) C&C サー

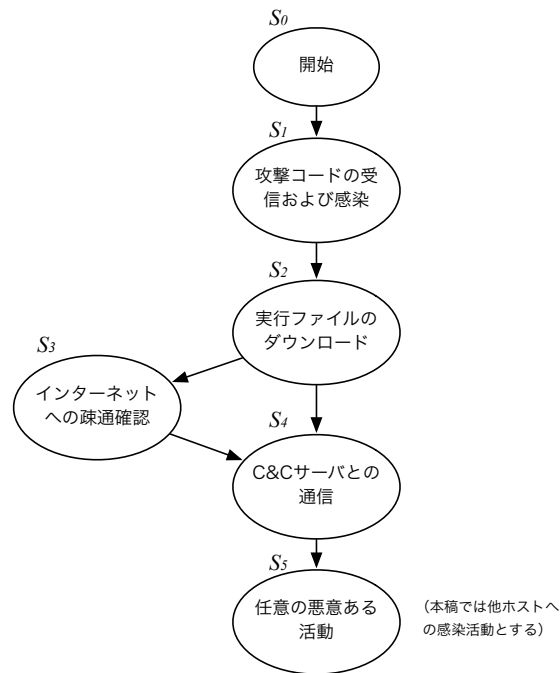


図 3.6: 感染直後のマルウェア活動の状態遷移モデル

バとの通信, S_5) 任意の悪意ある活動の5種類とした。各状態におけるマルウェアの具体的な通信内容は、マルウェアの種類に応じて異なるものとする。例えば、実行ファイルのダウンロードでは HTTP, FTP, TFTP, あるいはマルウェア独自のファイル転送用プロトコルが挙げられるが、各通信がマルウェアとして活動するための実行ファイルをダウンロードするという共通の目的であるとする。

今回使用する通信データは、感染元のホストが監視対象のホストと通信することでマルウェアに感染する。そのためマルウェアに感染した場合は、感染後にはなんらかの方法で実行ファイルをダウンロードする。実行ファイルをダウンロードした後は C&C サーバと通信する。ただし、近年はマルウェア研究のために情報セキュリティ関係の研究者が閉鎖されているネットワーク上でマルウェアを実行する場合がある。C&C サーバとの通信前にインターネット上の著名な Web サービスなどへ接続することで、疎通を確認するマルウェアが存在する。また、 S_5 はスパムメールやトラックバックスパムの送信、感染ホストに関する情報の報告、Web サービスのアカウント不正取得などが挙げられるが、本論文では C&C サーバとの通信直後に発生しやすい活動として、他ホストへの感染活動に着目した。

3.3.2 CCC DATASET を用いたマルウェア活動の分析

データセットの分割と解析

第3.3.1節で作成したモデルを検証するために、マルウェアによる通信データを調査した。調査に利用した CCC DATASET の通信データには Windows 2000 と Windows XP のホストをそれぞれ1台ずつインターネットに接続し、攻撃活動に関する通信が含まれる。両ホストは定期的にクリーンな状態にリセットされている。今回のモデルは正常状態からマルウェアへの感染、マルウェアとしての活動開始までの遷移に着目するため、OS が初期状態に復元された時刻を正確に把握しなければならない。一方の対象ホスト (以降、 H_1) は Windows XP であり、起動時に time.windows.com. への問い合わせ、および NTP による時間同期を実行する。そのため、通信データから H_1 が送信元もしくは宛先ホストとなっているパケットを抽出し、前回のリセット予想時刻から一定時間以上経過した後に発生する time.windows.com を問い合わせる DNS パケットと、NTP のパケットによって通信データを分割した。これによって 144 個の通信データセットが生成された。

CCC DATASET2008 を復元時刻によって分割した全てのデータセットについて、状態遷移を調査した。調査した結果を表3.6に示す。複数のマルウェアに感染したと見られるデータセットもあるが、表の状態遷移では最初にマルウェアに感染した際の状態遷移を示している。それぞれのデータセット群を D_N として分類した。ただし、 D_0 では注目すべき通信が発生せず、 D_1 は攻撃が成功しなかった可能性が高いため除外して議論する。また、状態遷移のモデルに従わなかったものを D_e とした。これは C&C サーバとの通信をせずに自律的に感染活動を実施しており、本論文におけるマルウェアの定義とは異なる。データセットの状態遷移では、各データセットは図3.6で示したどの状態を遷移したのかを示している。結果の欄ではマルウェアが最後の状態から最終的にどのような活動に至ったかを示している。最後の状態以降にマルウェアとしての活動と思われる通信が観測されなかった場合が**終了**、観測されたデータセットが**継続**、終了に加えて最後の状態での活動に不備があったと見られる痕跡が観測されたデータセットが**失敗**としている。

状態遷移に基づいた分析

表3.7では各状態が遷移する際の時間間隔として、各状態遷移のサンプル数、平均時間、標準偏差、サンプル中の最小時間、サンプル中の最大時間を示している。今回の分析では状態遷移の時間を、遷移前の状態における最後の動作の時刻と遷移後の状態における最初の動作の時刻の差としている。これは各状態では複数回類似した動作が繰り返される場合があり、通信データだけではどの動作が次の状態遷移へのきっかけになったかを判別するのが難しいためである。例えば、実行ファイルがダウンロードされる前に攻撃コードの受信する回数は、各データセットにおいて平均3.2回発生しており、成功した攻撃コードを特定するのは難しい。

表 3.6: 分割したデータセットの分類

	データセットの状態遷移	結果	件数
D_0	S_0	終了	11
D_1	$S_0 \rightarrow S_1$	終了	9
D_2	$S_0 \rightarrow S_1 \rightarrow S_2$	失敗	26
D_3	$S_0 \rightarrow S_1 \rightarrow S_2$	終了	5
D_4	$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_4$	失敗	1
D_5	$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_4$	継続	13
D_6	$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_4 \rightarrow S_5$	継続	51
D_7	$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5$	継続	27
D_e	モデルから外れたデータセット	----	1
		合計	144

そのため、このような状態遷移を分析する場合には今後も検討が必要となる点に留意するべきである。

表では $S_1 \rightarrow S_2$, $S_2 \rightarrow S_3$, $S_2 \rightarrow S_4$, $S_3 \rightarrow S_4$ の状態遷移はそれぞれ平均時間が 15 秒未満となっており、最大時間もそれぞれ 60 秒程度である。これは、マルウェアがあらかじめ定められたアルゴリズムに従って単独で動作しているため、次の状態に遷移するための待ち時間を設ける必要がないためだと考えられる。各遷移において遷移時間が比較的長いサンプルがいくつか存在した。これらはホスト内部における情報の検索にかかる時間か、もしくは通信の間隔を長くしセキュリティ管理者に発見されにくくしていると予想される。

一方 $S_4 \rightarrow S_5$ は平均遷移時間が 137.091 秒、最大遷移時間が 489.683 秒であり、他の状態遷移と比較して間隔が長い。これは単独で実行しているプログラムに従った動作ではなく、IRC プロトコルによって C&C サーバから非同期的に送信されるコマンドに従っているためである。IRC による他ホストへの感染活動の実行命令例は [34] などに示されている。今回のデータセットでは活動時間が限られていたため、長期間の観測では遷移までの時間が長くなる可能性がある。

表 3.7: 各状態開始から遷移までの所要時間

状態遷移	数	平均遷移時間	標準偏差	最小遷移時間	最大遷移時間
$S_1 \rightarrow S_2$	96	5.116	11.401	0.041	66.416
$S_2 \rightarrow S_3$	27	2.688	9.765	0.219	51.451
$S_2 \rightarrow S_4$	63	4.618	11.050	0.282	66.416
$S_3 \rightarrow S_4$	27	11.334	16.190	0.006	59.280
$S_4 \rightarrow S_5$	78	137.091	147.392	1.096	489.683

(遷移時間の単位は秒，各計算結果の小数点第4位以下は切り捨て)

各状態における観測事象

第3.3.1節で述べたとおり， $S_{1,2,3,4,5}$ の各状態は具体的な通信内容を限定しない．そのため，CCC DATASETにおいて観測された具体的な通信内容について述べる．

まず， S_1 ではWindows OSで標準的に備わっているNetBIOSやDCERPCの脆弱性を利用した攻撃コードが送信され，感染している．ただし，CCC DATASETの通信データではホストの状態変化を知るためのデータがないため，どの攻撃によって感染したかを特定するのは難しい．そのため， S_2 へ遷移する直前の攻撃を S_1 の成功と見なした．

S_2 の通信内容について解析した結果を表3.8に示す．実行ファイルのダウンロードを発見するためにはWindowsで利用される実行ファイルのフォーマット[28]のパターンを検知するルールを作成し，検査した．このルールによってTCPおよびUDPによる通信のペイロードを全て検査したため，暗号化されていない通信中の実行ファイルを検知できる．また，実行ファイルが検知されなかったデータセットについても目視で確認したが，暗号化などによって検知を回避した形跡は発見されなかった．独自プロトコルによるダウンロードおよびアップロードは全て同一のプロトコルであると見られる．バックドアによる実行ファイルのダウンロードでは，Windowsのコマンドプロンプトを利用してFTPコマンドを発行しているが，実行ファイルのダウンロードは確認されなかった．TFTPによる実行ファイルのダウンロード失敗では，サーバ側のホストが応答せず通信が終了していた．

S_3 では著名なWebサイトやメールサーバに接続を試み，インターネットへの到達性を確認していると予想される．観測された確認方法は主に2種類ある．1つは著名なメールサービスのMXレコードを問い合わせた後，DNSの応答に含まれるメールサーバに対してSMTP[35]の標準ポート番号である25番へ接続を試みている．ただし，CCC DATASETの環境では内部から外部へのTCPポート25番がフィルタリングされていたと見られ，全てのメールサーバから応答が観測されなかった．もう1つの確認方法はWebサイトから特定のファイルをダウ

表 3.8: S_2 における通信内容の内訳

ダウンロード方法	結果	件数
独自プロトコルによる攻撃元ホストからの実行ファイルのダウンロード	成功	83
	失敗	0
独自プロトコルによる攻撃元ホストからの実行ファイルのアップロード	成功	7
	失敗	5
DNS による名前解決および HTTP による実行ファイルのダウンロード	成功	7
	失敗	0
バックドアによって攻撃元ホストが指示した実行ファイルのダウンロード	成功	0
	失敗	20
TFTP による実行ファイルのダウンロード	成功	1
	失敗	1

ンロードする手法である。HTTP によるインターネットへの到達性確認には AOL のインスタントメッセージャーや ICQ のインストールファイルが利用されていた。筆者が独自に観測したマルウェアの活動では、ポート 25 番への到達性がある場合に迷惑メールの配信を開始したものがあため、到達性確認後の活動にも影響していると推測される。

S_4 における C&C サーバとの通信は全て IRC プロトコルを利用していた。データセット $D_{4,5,6,7}$ において IRC 接続は合計 216 件発生していたが、TCP のサーバ側ポート番号は IANA で定められている標準のポート番号 [36] を利用していたのは 29 件のみであった。 D_4 での接続失敗は DNS によって問い合わせた IRC サーバのアドレスから応答が無かったためである。ただし、別のデータセットでは同一のサーバとポート番号に対して IRC の接続が確立していたことから、サーバ側ホストの一時的な問題によって接続が失敗したと見られる。

S_5 における他ホストへの感染活動は S_1 での攻撃と同様に、Windows の NetBIOS や DCERPC の脆弱性を利用するためポート開放の確認と攻撃となっている。最初に観測されたスキヤンの宛先ポート番号は TCP の 135 番が 71 件、139 番が 2 件、445 番が 6 件であった。

3.3.3 考察

調査した結果、マルウェアの活動が途中で不備を起こさない限りは図 3.6 にそって状態が遷移していることを確認した。 $D_{5,6,7}$ のデータセットでは作成したモデルの状態遷移が確認さ

れた。攻撃コードの受信や実行ファイルの取得、インターネットへの接続性確認、C&Cサーバへの接続はそれぞれ以前より確認されていたマルウェアの活動だが、調査したマルウェアではこれらが同一のモデルに従って動作する事が明らかになった。この事実は、状態遷移に従って活動するマルウェアは亜種であっても検知できる可能性を示唆している。各状態の具体的な通信方法が異なっている、これらを抽象化できる表現力を持ったネットワーク監視手法であれば、大幅に活動内容を変更しない限り検知できる。そのためシグネチャを作成していないマルウェアの検知も可能となり、マルウェア対策として意義があると考えられる。

一方で、途中で動作が失敗してしまうデータセットやモデルに沿わないデータセットも存在する。各状態において失敗した動作については第節で述べたが、これらは総じてマルウェアネットが他者の資源を無断で利用している動的なシステムであることに起因する。多くの場合、マルウェアは発見されるとそのホストから除去されてしまうため、C&Cサーバや実行ファイル配布サーバが頻繁に変化する。マルウェアネット全体にこうした変更の情報が伝播せず、動作の失敗が起こりやすいと考えられる。しかし、今回のデータセットでは活動時間が限られており、通信データはマルウェアが感染した直後の通信に限られている。そのため、感染から長時間経過した場合に新しいモデルを作成できる傾向が発見できる可能性がある。また、感染直後ではなく感染後にネットワークの変更や再起動などが実施された場合の活動に対しても新たな動作が見られる可能性があり、より様々な状況での通信データ収集が必要になると考える。

3.4 解析結果に基づいた対策手法の考案

Guofei の研究 [33] でもマルウェアの活動パターンにおける共通性が示されているが、ネットワーク監視により受動攻撃型マルウェアを検知するためには、マルウェアの活動を状態遷移のシナリオとして表したルールによる検知手法が有効と考えられる。サイバークリーンセンターによる報告 [37] によれば市販のウィルス対策ソフトでは検知できないマルウェアが月ごとに 200~1,000 種類発見されている。多くのウィルス対策ソフトではマルウェア本体の特徴を示すシグネチャによって検知しているが、個別のマルウェア毎に特徴を調査して検知する手法は限界である。CCC DATASET の調査結果 [18] や BotHunter では活動の状態遷移の類似点を利用することで、全てのマルウェアの特徴を調査することなく大部分のマルウェアを検知する手法を提案している。状態遷移による検知では、遷移前の状態と遷移後の状態の両方を監視するため、必然的に検知精度が向上し、マルウェアの活動の断片的な特徴でも状態の特定に利用できる。

マルウェアの活動の状態遷移を利用した検知手法としては NetSTAT [38] をはじめとするイベント相関分析や、BotHunter が挙げられるが、受動攻撃型マルウェアの検知は困難である。まず、多くのイベント相関分析では IDS のイベントの発生順序に着目しており、複数イベントが任意の順序で発生する条件や、イベントの発生回数などを条件として設定するのが困難

であり、柔軟性に欠ける。一方、BotHunterは第3.2.3節で述べたとおり、攻撃やマルウェアの転送、マルウェアの管理といった状態毎にイベントを抽象化しているため、汎用性に優れる反面、個々のイベント毎に相関関係を定義できないという構造上の弱点がある。例えば、送信されたGETリクエストのパラメータとダウンロードされたファイルの形式の関係や、あるサーバから指示されたドメイン名と接続先のHTTPサーバのドメイン名との関係などのように、個々のイベント間の具体的な相関関係を定義し、検知に利用できない。第3.2.2節で示したように、受動攻撃型マルウェアの通信を断片的に見てもマルウェアに関連した活動だと判断するのは難しく、能動攻撃型マルウェアとは異なる判断基準が必要となる。

そのため能動攻撃型マルウェアだけではなく、受動攻撃型マルウェアを検知するためには、マルウェアの活動を汎用的かつ柔軟にシナリオ化した検知手法が有効であると考えられる。表3.4、3.5で示したように、受動攻撃型マルウェアにも部分的には共通する通信パターンが見られる。このような通信を利用して検知を試みた場合、マルウェアの通信なのか悪意のない一般ユーザによる通信なのかを判断するのは難しい。しかし、共通する通信パターンの発生順序や通信内容に着目すると、それぞれ通信に相関関係があることが分かった。これらはマルウェア自身の行動アルゴリズムやC&Cサーバからの命令に基づいた行動であり、マルウェアの活動の基礎となる部分である。このようなシステムはマルウェアネット構築に深く関連するため、短期間で著しく変化する可能性は低いと考えられる。もちろん相関関係の細部を変更するのは容易だが、厳密なシナリオではなく、複数の分岐を許容するシナリオを記述できれば、検知の回避に対して耐性が高くなる。このように、個々のイベントの関係性を具体的かつ柔軟に捉えることで、汎用性の高いシナリオで精度の高い検知が実現できると見込まれる。

3.5 本章のまとめ

本章ではマルウェアの収集・解析環境の構築とそれを利用した解析結果、そして外部のデータを用いた調査によって、マルウェア対策の指針を考案した。マルウェアの対策を研究するうえで、マルウェアに関連するデータの収集は必須である。これまでShadowserver Foundation[39]やCyber-TAでデータ収集は行われてきたが、これらの解析結果は非公開であったため他の研究者が利用しにくい状態にあった。そのため、本章では小規模でも効率的にマルウェアを収集するためのフレームワークを構築した。これは従来の能動攻撃型マルウェアだけではなく、Webサイト閲覧を契機に感染する受動攻撃型マルウェアの収集も実施してきた。このシステムによって2009年7月2日から同年9月3日までの2ヶ月間で1,515件の悪意があると考えられるWebサイトを調査できた。これらの通信データにおけるHTTPの通信を解析したところ、174種類のWindowsの実行形式ファイルを取得した。これらをAVG Free Edition 8.5.409[40] (ウィルスDBバージョン270.13.76/2345)を用いて検査したところ、131件をマルウェアとして検出した。さらに検知できなかった43件についても、ハッシュ値をMalware

Hash Registry[41] で検索したところ、マルウェアとして登録されている事が明らかとなった。この結果から、受動攻撃型マルウェアの収集システムとして有効に機能していると言える。

これらの通信データを解析した結果、能動攻撃型マルウェアと受動攻撃型マルウェアの活動パターンの違いが明らかになった。これまで、能動攻撃型マルウェアは感染後に他のホストに対する感染活動を実施するため、一般的なユーザが利用している場面とは異なる傾向のネットワークトラフィックが発生していた。しかし、本章での調査により、受動攻撃型マルウェアは主に一般的なユーザの利用に類似したネットワークトラフィックを発生させることが明らかになった。マルウェア対策の既存研究では能動攻撃型マルウェアが他のホストに感染する特徴を利用していたが、大部分の受動攻撃型マルウェアではこの特徴が見られず、検知に大きな影響を与えると見られる。

しかし、本環境による解析結果と外部データを用いた解析から、マルウェア対策の手がかりが得られる。本章では Cyber Clean Center から提供された CCC Dataset 2008 を用いてマルウェアの活動に共通するパターンが有ることを明らかにした。これは Cyber-TA などの研究[33]においても同様の指摘がされており、一般性がある事実だと言える。本章では先行研究で取り組まれてきた能動攻撃型マルウェアに加えて、受動攻撃型マルウェアでも通信の状態遷移を適切に利用することによって効果的に対策ができる可能性を示唆している。ただし、既存の状態遷移モデルを利用した対策などでは、受動攻撃型マルウェアの巧妙な動作を捉えるのが困難であると考えられる。よって、本研究ではこれらの解析結果に基づいたより効果的な対策を提案する。

第4章 ネットワークセキュリティ監視によるマルウェア対策とその課題

マルウェアによってより効率的に経済的利益を得るために、ハードウェアやマルウェアの開発者は既存のマルウェア対策の回避や無効化に注力してきた。ネットワークを経由して感染するマルウェアの対策手法としては、ホスト上での対策やサーバ上での対策、通信の監視や制御による対策などが挙げられるが、マルウェアの改変により多くの対策の有効性が低下している。これによって、マルウェアに感染する危険性は上がっており、さらに感染してしまった後にマルウェアを検出するのも難しくなっている。ハードウェアはマルウェアに感染したホストを長期間にわたって支配し続けることでより多くの利益を得られるため、ユーザやネットワーク管理者などに気づかれにくい活動をとる場合が多い。感染したホストのユーザは、これによって不利益を被る危険性がより一層高くなる。

本研究ではマルウェア対策の中でも、ネットワークトラフィックを監視することでセキュリティに関する情報を収集し、ネットワークに関連したインシデントを発見する**ネットワークセキュリティ監視**に着目する。ネットワークセキュリティ監視は他のセキュリティ対策手法と比べてユーザへの干渉が比較的少なく、利便性の低下が少ない。また、感染対象となるホストから独立して監視ができるため、ホストがマルウェアに感染した場合でも監視に必要な情報の完全性に影響が少なくなる。さらに、一斉に多数のホストを監視できる利点がある。

ただし、マルウェア対策にネットワークセキュリティ監視を用いる際、検知精度、追調査の容易さ、処理性能の3点が課題となる。ネットワークセキュリティ監視ではホスト上でマルウェアの活動を監視するのに比べ、検知に利用できる情報が少ないと言う不利点があり、検知精度が低下しやすい。また、検知結果がどのような活動を検知したのかが明確に分かることと、どのような経緯で検知したのかを追調査できる容易さも問題となる。検知結果が出力された場合、対象のホストを調査、回復しなければならないが、その際にマルウェアのどのような活動が検知されたかが示されることで、調査、回復の事後対応の手がかりとなる。そして、ネットワーク全体を一括して監視できるだけの処理性能の高さである。ホスト上での監視と比べると扱う情報の種類は少ないが、多くのホストを一括して監視する場合には、監視するホスト数や通信料に比例して処理の負担が増加するため、一定以上の処理性能と分散可能な処理システムが必要になる。マルウェア対策においてこれらの問題点を十分に解決する既存手法はなく、新しいネットワークセキュリティ監視手法の提案が求められる。

4.1 本研究で対象とするネットワーク環境

本論文ではマルウェア対策を考えるに当たり、企業や大学のようにネットワーク内のセキュリティに責任をもつ人物がいることを想定する。この他のネットワーク構成についても対応できるような対策を考案するがもっとも優先して対応するのはこのような管理ネットワークとする。特に大学のネットワークはネットワークや計算機資源が豊富にあるため、悪意あるユーザに攻撃対象にされやすい傾向がある。また、企業や行政機関などでは情報流出による被害などが大きく、守るべき資産が比較的重要であると考え、そのため、対象とするネットワーク環境はこれらの構成に近いものを想定する。

ネットワークの規模はクラス C からクラス B 程度のネットワークとする。ネットワークはユーザセグメントやサーバセグメントなど、必要に応じて分割されているものとする。接続するホストは数台から 60,000 台程度でユーザ用 PC やサーバなどを含むものとする。ユーザ数は数人から 10,000 人程度を想定する。ネットワークは厳しく制御されている企業ネットワークから構成するユーザが比較的自由にホスト・ネットワークを利用できる大学ネットワークまでを想定する。

管理構成としては、ネットワーク管理者とセキュリティ管理者がネットワークの構成・運用とセキュリティの維持に責任をもつものとする。ただし、ネットワーク管理者がセキュリティ管理者を兼ねる場合もある。セキュリティ管理者はネットワーク内部のセキュリティを保つことに責任をもつ。もっとも優先すべき資産は帰属組織の利益に関するものとするが、ネットワーク利用者に対する被害を最小限にすることも2次的な目的とする。セキュリティ管理者はセキュリティ対策を導入、運用するのに関してある程度の知識や技能をもつ人物とする。

このようなネットワーク環境において、セキュリティ管理者がマルウェアへの対策を講じることを想定し、本研究に取り組む。

4.2 ネットワークセキュリティ監視によるインシデント検知

本論文ではネットワークトラフィックをルータやスイッチなどの中継点において監視し、セキュリティに関する情報を収集するネットワークセキュリティ監視を採用する。ネットワークセキュリティ監視は Herberlein による“A Network Security Monitor”[42]において提唱され、その後に IDS として一般的なセキュリティ対策となった手法である。Roesch による“SNORT-LIGHTWEIGHT INTRUSION DETECTION FOR NETWORKS”[43]において提案、実装された Snort[19]は最も著名なシグネチャ型のオープンソース IDS であり、本論文執筆時も広く利用されている。Snort 以降も様々なネットワークセキュリティ監視手法が提案されたが、現実によく利用されているネットワークセキュリティ監視手法は Snort と同じシグネチャ型 IDS となっている。

ネットワークセキュリティ監視によるネットワークインシデント検知は、ネットワーク管

理者が責任を持つべきネットワーク全体を監視し、インシデントの早急な発見と被害の極小化を促すための対策である。ネットワークインシデントの対策は、大きく事前対策と事後対策の2つに分けることができる。事前対策はネットワークインシデントの発生を防ぐ対策であり、防御、抑止、回避、予防など挙げられるが、いかに対策を多重化しても全ての脅威を網羅できるわけではない。また、対策が有効に機能しない場合もある。事後対策はネットワークインシデントの兆候や発生を確認した際の対策であり、反応、軽減、調査、回復などが挙げられる。事前対策によりネットワークインシデントを完全に防ぐのが理想だが、完全に防ごうとすると膨大なコストが必要となると同時に、利便性が犠牲となる。そのため、事前対策と事後対策を並行して適用する必要があるが、被害発生後に事後対策を開始するためのきっかけが重要となる。検知はネットワーク管理者に被害の発生を通知し、迅速な事後対応を遂行するために不可欠である。ネットワークインシデントの発生を早急に検知、対応できれば、実質的な損害を出さない事も可能である。対応までの時間が短ければ重要な情報資産を保護できる可能性が高くなり、最も深刻な状況を避けられる。

マルウェアによる活動は主に計算機上、ネットワーク上、可搬型記憶媒体上の3種類に分類することができる。第2.1節において、マルウェアの定義はコンピュータ、サーバー、コンピュータネットワークに被害を起こすように設計・実装されたソフトウェア全般であると述べた。このソフトウェアは主に計算機上でプロセスやスクリプト、マクロなどとして動作し、ネットワークや可搬型記憶媒体を感染経路としている。そのため、第2.4節で述べたとおり、システムによってマルウェアを対策する場合はこれらのいずれか、または複数の媒体に出現する情報に基づいて防御、検知する必要がある。

マルウェア対策においてネットワークセキュリティ監視を採用する利点は、主に3つに分けられる。第一の利点は、検知が他のセキュリティ対策手法と比べて利便性の低下が少ないという点である。不正侵入を含むセキュリティ対策では、通信やシステムへのアクセス制御による防止対策やあらかじめ脆弱性を修正する予防対策が挙げられる。これらは有効に機能した場合、マルウェアの感染を防ぐことができる。しかし、これらは守るべきシステムに直接干渉するため、ユーザに対する影響が小さくない。例えば、設定された内容とユーザのポリシーが不一致を起こすとアクセス制限はユーザの意図した動作を阻害する場合がある。また、脆弱性の修正などは正規のソフトウェアにも影響を及ぼす可能性があり、大規模な企業や組織では安易に脆弱性の修正を実施できない場合がある。これに対して、検知の場合はユーザへの干渉が比較的少ない。特にネットワークトラフィックを中継点で監視する場合、ユーザは検知対策が導入されているのを意識する必要はない。多くの場面で情報技術が使われるようになっているため、検知対策の導入によってネットワークやシステム、ユーザへの影響が少ないことは利点となりうる。

2つ目の利点は感染対象となるシステムから独立している点である。マルウェアを検知する際には、ホスト上から収集できるプロセスやコンピュータ資源の利用状況が有用な手がかりとなる。そのため、より詳細なマルウェアの検知が可能となる場合があるが、その一方で第

2.4節で示した通り、マルウェアは感染ホスト上で延命するための技術を多く導入しており、一度感染してしまうと完全な駆除が非常に困難である。もっとも問題になるのはウイルス対策ソフトを無効化したり、自身をファイルシステムから隠蔽してしまい、ユーザが感染の事実気づかないことである。マルウェア感染の発見が遅れば、被害の多様化や拡大が加速し、より深刻な被害になる可能性が高い。これはOSの動作を起動直後から検査し、わずかな異常でも発見するようなシステムの開発によって解決できる可能性があるが、CPUの動作やメモリの状態などをすべて継続的に監視し続ける必要があり、現実的ではないと考えられる。これに対してネットワークトラフィックを監視した場合、感染の対象となるシステムと監視するシステムをそれぞれ独立させた状態で動かすことができる。同じホスト上で監視した場合と比べて、ネットワークトラフィックから得られる情報は少ないという難点はあるが、否認防止が成り立つ情報は検知において重要な手がかりになると言える。

そして、3つ目の利点は一斉に多数のホストを監視できるという点である。2つ目の利点であげた監視システムの独立性は、仮想マシンなどを利用すれば同一ホスト上でも実現できる。ただし、監視対象となるホストはネットワーク上に多数存在する場合が多い。そのため同一の対策を導入、運用するためには一括して管理できるのが望ましい。

これら3つの理由から、本論文ではネットワークセキュリティ監視に着目した対策を提案、実現する。

4.3 ネットワークセキュリティ監視によるマルウェア対策の問題点

これまで、ネットワークセキュリティ監視の分野ではIDSを中心とした手法によってインシデント検知に取り組んできた。しかし、マルウェアの狡猾化にともなってネットワークセキュリティ監視による検知も困難になってきている。

4.3.1 特徴的な通信内容や通信傾向の排除

単純なパターンマッチによってマルウェアの活動を検知されないようにするため、多くのマルウェアでは標準化されたプロトコルを用いたり、特徴的な文字列、データ列を排除するようになった。多くのマルウェアではIRCやHTTPといった標準化されたプロトコルによってC&Cサーバやマルウェア配布サーバと通信し、コマンドの受信や新しいマルウェアの取得をおこなっている。さらに第3.2節でも取り上げている通り、Webサイトの閲覧を契機として感染する受動攻撃型マルウェアでは通常のWebサービスで見られるようなHTTPのURLや変数名を用いて通信している。マルウェアの通信に出現する特殊な文字列やデータ列に基づいて検知をしていたネットワークセキュリティ監視手法は、シグネチャを調整しても検知精度に影響がでてしまう。

4.3.2 通信の発生規則の変化

近年のマルウェアでは通信内容の発生する規則が状況によって変化するため、単純なパターンマッチによる検知手法と同様に検知が難しいという問題点がある。マルウェアは亜種の生成が容易になっているため、非常に頻繁に新しい亜種が発生する。亜種は全体的な機能に大幅な変化はないものの、細部の機能が調整されていたり、必要に応じたモジュールが組み込まれているなどの変化がある。さらに、ボットネットに参加しているマルウェアはC&Cサーバから受け取ったコマンドによって、活動する内容が異なり、それにともなって通信内容も変化する。したがって、通信の発生規則は亜種の発生や状況に応じて変化すると言える。

パターンマッチによるネットワークセキュリティ監視手法を補助するため、複数の文字列やデータ列の出現規則に基づき、インシデントを検知する手法が挙げられる。しかし、このような通信の発生規則が変化した場合に追従できなければ、検知精度を維持するのは難しく、False negativeの原因となってしまう恐れがある。

4.3.3 新しい通信モデルの出現

マルウェア開発者も持続的に新しい種類のマルウェアを開発しているため、新しい通信モデルのマルウェアも徐々に出現している。顕著な例の1つが第3.2節の能動攻撃型マルウェアと受動攻撃型マルウェアの違いだと言える。能動攻撃型マルウェアは半数以上が高速にTCPのSYNパケットを送信したり受信データサイズに対する送信データサイズの量が多いなど、特徴的な傾向があるのに対して、受動攻撃型マルウェアは一般のユーザと類似した通信内容に変化している。能動攻撃型マルウェアが2005年から“*Know your enemy*”[44]で解説されていたのに対し、受動攻撃型マルウェアは2007年のProvosらの調査報告である“*The ghost in the browser analysis of web-based malware*”[9]によって注目を集め始めたことから、能動攻撃型マルウェアから受動攻撃型マルウェアへの通信モデルの変化が起きていることを示している。

こうした通信モデルの変化は、ネットワークセキュリティ監視による検知にも大きな影響を及ぼす。これまでに提案されてきたネットワークセキュリティ監視によるマルウェアの検知手法では、定義したマルウェアの通信モデルに基づいて検知する手法も多い。しかしそれらの手法では、新しい通信モデルが出現する度に新しい検知手法を考案する必要がある。そのための調査だけでなく、モデルの作成などにかかる時間や手間を考慮すると、近年のマルウェアに対応していくのは困難であると考えられる。

4.4 関連研究

ネットワークセキュリティ監視による対策を提案するに当たり、関連研究を調査した。Network Security Monitor以降、SnortをはじめとするIDSの実装や関連する技術が研究、開発されて

きた。しかし、現状のマルウェアを対策する上では様々な問題がある。また、ネットワークセキュリティ監視の他にもホスト上でマルウェアの挙動を監視する研究についてまとめる。

4.4.1 既存のIDSとその改良

IDSは最も一般に普及しているネットワークセキュリティ監視手法である。ルールによって検知すべきネットワークトラフィックを具体的かつ簡易に記述でき、検知の要因がどのような事象であったかを容易に判断できるため、運用が比較的容易だったという背景がある。しかし、現在普及している最も基本的なIDSは、単一のパケットや単一のセッションに出現するパターンを検査するシステムとなっている。そのため、定型のパターンが通信に含まれるソフトウェアや攻撃コードの検知は容易だが、わずかでもパターンが変化すると検知できなくなるというFalse negativeの問題がある。また、監視しているホストの性質や通信の前後関係などとは関係なく、パターンと合致した場合には悪意ある活動を検知したと解釈して、警告を発行する。そのため、偶然に検知対象のパターンがネットワークトラフィックに出現した場合はFalse positiveが発生してしまう。このようなFalse negative, False positiveが発生しやすいことから、現在のIDSはマルウェア対策に適した対策手法であるとは言い難い。

このような問題を解決するため、これまでに、通信の状態遷移を利用したシグネチャを利用して検知精度を高める手法としてはNFR[45]やNetscreen-IDPのStateful signature[46]において実装されている。また、ネットワークサービスに対する要求と応答の組み合わせを見ることで検知した事象のリスクを評価できる手法を藤田が“侵入検知ポリシーの記述性向上によりログ出力量の低減を可能とした不正アクセス処理システムの開発”[47]において提案している。さらに、Sommerらは“Enhancing Byte-Level Network Intrusion Detection Signature with Context”[48]においてプロトコル上の要求と応答をルールに組み込めるようにしている。これらは個々のセッション毎の状態遷移と、各状態において検知に利用する不正な通信のパターンを定義できるため、通信に特徴が少ないマルウェアを検知する場合でも複数の条件を組み合わせ設定可能となり、検知の精度が向上する。しかし、定義できるパターンは個々のセッションに限られるため、対応できるパターンに限界がある。

4.4.2 イベント相関分析

従来のIDSやその他の機器から得られた検知結果を1つのイベントとして扱い、発生順序によってイベントを関連づけし、新しいイベントを検知する手法をイベント相関分析と呼ぶ。関連づけする方法はルールに基づく方法と、アルゴリズムに基づく方法がある。ルールに基づく方法としてはログファイルからイベントの発生順序を抽出する“SEC - a Lightweight Event Correlation Tool”[49]や、ネットワーク上のイベントを関連付ける。“STATL: An Attack Language for State-based Intrusion Detection”[50], “NetSTAT: A Network-based Intrusion Detection

“Approach”[38]が挙げられる。アルゴリズムに基づく方法はイベントをクラスタリングする“Clustering intrusion detection alarms to support root cause analysis”[51]が挙げられる。これはOSSIM[52]やArcSight[53]などにおいて、実現されている機能である。相関分析では予め設定したルールやアルゴリズムに基づいてイベントを検知する。Argus[54]で取得した通信の送信元・宛先IPアドレス、プロトコル、ポート番号のような部分的な情報を記録しており、これらもイベント相関分析に利用できる。単独では脅威とならないイベントでも、複数のイベントを関連づけることで、脅威となるイベントを検知できる可能性がある。

ただし、イベント間の関係を示すための柔軟性に欠ける部分がある。ルールを基にした多くの実装では基本的にイベントの発生順序に着目している。OSSIMやArcSightなどのルールを基にした検知では、イベント間の関係を逐次的な発生順序と1つのイベントの繰り返しによって表している[55, 56]。そのため、複数のイベントのいずれかが発生したという状態、もしくは全て発生した状態などを表現するのが難しくなってしまう。“Modeling network intrusion detection alerts for correlation”[57]ではイベントをrequire, provideという二つの状況に分けて分析をしている。しかし、情報の取得と取得した情報に基づいた行動が組み合わせて起きるという前提であるため、複雑なルールは記述できない。また、アルゴリズムに基づく場合も、イベントの傾向が変化してしまうと柔軟に対応するのが難しい。

相関分析は情報の収集を他のイベント検知機構に依存しているため、詳細な調整が難しいという問題がある。そのため、本来の検知対象とは直接関係しない事象を利用して検知しなければならない場合があり、精度に影響を及ぼしてしまう可能性がある。

他にもSnort[19]では、あるトラフィックパターンの出現が繰り返された場合にセキュリティイベントの検知と見なすルール[58]が記述できる。これは基本的に同じトラフィックパターンの出現回数を監視する。これはマルウェアの感染活動の繰り返しを検知できる可能性がある。ただし、単一パターンの繰り返ししか表現できないため柔軟なルールを書くのは難しく、複雑な挙動を正確に検知するのは困難である。

4.4.3 マルウェアの活動モデルに基づいた検知手法

マルウェアの亜種を検知する手法として、通信傾向を統計的にモデル化し検知する手法が挙げられる。第3章でも述べた通りマルウェアの亜種には共通する動作があるため、マルウェアの系統ごとにシナリオを用意し、それに適合したものを検知する手法である。主に統計的解析手法と振る舞い型検知手法が挙げられる。

統計的解析手法はネットワークトラフィックやホストの活動から異常を検知し、セキュリティイベントを出力する。これはマルウェアなどマルウェアを検出するための手法として用いられる。統計的解析手法の利点は未知のマルウェアでも検知できる可能性を持つ点である。トラフィックパターンに基づいて検知をしているIDSでは、パターンが僅かに変化しただけでもFalse Negativeを起こしてしまう。統計的解析手法では統計的処理などによって得られる

値を閾値と比較するため、閾値を上回っている限りはマルウェアとして検出できる。

“A DNS-based Countermeasure Technology for Bot Worm-infected PC terminals in the Campus Network”[59] と “Botnet の命令サーバドメインネームを用いた Bot 感染検出方法”[60] では、DNS 問い合わせの異常を検知してマルウェアの検出を試みている。マルウェアは迷惑メール送信や C&C サーバへの接続などで、特徴的な DNS の問い合わせを実行する。特に迷惑メール送信時にはドメイン名の MX レコードを大量に問い合わせるため、特徴が顕著に現れる。これを利用して、一定の閾値を超えた場合にマルウェアによる活動だと判断する手法である。“An Algorithm for Anomaly-based Botnet Detection”[31] は TCP パケットの種類の割合を調べ、マルウェアらしい傾向を検知する。“Security Operation Center のための IDS ログ分析支援システム”[61] は比率分析や稀率分析を利用し、通常観測されにくいセキュリティイベントの発生や発生頻度の急増などを検知する手法である。これによって監視ネットワークの異常を検知する。

未知のマルウェアに対応できる可能性を持つ統計的解析手法だが、実際の運用は難しい。基本的に統計的解析手法では異常と正常の境界を定める必要があり、これは監視しているネットワークの環境や接続しているホストの性質に依存する場合が多い。さらに閾値はネットワーク環境やホストの性質が変化に敏感であり、適切な閾値は頻繁に変化する。適切な値に保つためにはチューニングの手間などが必要となるため運用負担が大きくなりやすい。運用負担を大きくするもう 1 つの理由として、検知内容と対応するネットワークインシデントの不明確さが挙げられる。異常が発生したというセキュリティイベントを検知したとしても、それが何によって引き起こされたのか、どのような影響を及ぼすセキュリティイベントなのか不明な場合が多い。ネットワークインシデントは被害内容や影響範囲が明らかにならないと分かなければ事後対応を開始できないため、セキュリティイベント調査のための負担が生じてしまう。

Network Behavior Analysis (NBA)[62, 32] は Lancope 社が製品化しているシステムであり、sFlow[63] を用いてホストの振る舞いを検知する。sFlow は 2 ホスト間の通信を 1 つのフローとして扱い、通信プロトコルの種類や転送データ量、開始時間と終了時間などを記録する。NBA は複数の sFlow の情報を利用し、ホストの振る舞いに着目している。sFlow はスイッチングハブやルータから出力されるが、サンプリングしながら利用するのが一般的であり、sFlow データグラムに収納される情報も多くはない。そのため、詳細に挙動を捉えて検知することはできないが、マルウェアの感染活動など多数のフローを発生させる挙動からセキュリティイベントを検知できるとされている。また、P2P ファイル交換ソフトウェアのフローの傾向が分かれば、それを検知できる可能性もある。

また、“BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation”[33] において解説されている BotHunter はマルウェアの活動の状態遷移をモデル化し、汎用的な検知を実現する手法である。BotHunter の状態遷移モデルでは、マルウェアの動作をマルウェア本体の転送、C&C サーバとの接続、悪意ある活動など、抽象化した状態を定義しており、

各状態に該当するイベントを検出することで状態遷移を発見する。この状態遷移モデルを利用した検知ではイベントの種類や発生順序が変化しても汎用的にマルウェアを検知できるという利点がある。ただし、検知のためにモデル化しているパターンが感染前から感染後への遷移に限られているため、外部から持ち込まれたホストがマルウェアに感染していた場合の検知は困難だと考えられる。また、内部ホストが提供しているサービスの脆弱性を利用した攻撃を想定しており、電子メールの添付ファイルによる感染や Web ブラウザの脆弱性を利用した感染には対応が難しいと見られる。さらに、第3章で示した通り、受動感染型マルウェアが発生させる個々の事象は C&C サーバとの接続や悪意ある活動が一般ユーザによる通信に類似しており、マルウェアの通信であると判断するのが困難である。そして、BotHunter の実装では迷惑メールの送信や、他ホストへの感染活動などを検知のよりどころとしているが、受動感染型マルウェアではこのような動作が少なく、状態遷移モデルが有効に機能しないと考えられる。さらに、攻撃やマルウェアの転送、マルウェアの管理といった状態毎にイベントを抽象化しているため、汎用性に優れる反面、個々のイベント毎に相関関係を定義できないという構造上の弱点がある。例えば、送信された GET リクエストのパラメータとダウンロードされたファイルの形式の関係や、あるサーバから指示されたドメイン名と接続先の HTTP サーバのドメイン名との関係などのように、個々のイベント間の具体的な相関関係を定義し、検知に利用できない。第3章で示したように、受動感染型マルウェアの通信を断片的に見てもマルウェアに関連した活動だと判断するのは難しく、能動感染型マルウェアとは異なる判断基準が必要となる。これは、検知に利用しているモデルを変更すれば良いわけではなく、検知機構に関する根本的な問題だと言える。

4.4.4 ホスト上でのマルウェアの挙動解析と検知

ハーダがツールを用いて新しいマルウェアを生成できるようになったため、マルウェアの亜種が急増したが、これに着目したコード解析や検知手法の研究が取り組まれるようになった。マルウェアの亜種はコードの圧縮や難読化のパターン、必要となる機能に応じたモジュールの選択によって新しいマルウェアが生成される。しかし、新しいマルウェアを構成するモジュールそのものを頻繁に更新するのはハーダにとっても負荷が高いと見られ、新しいマルウェアを生成する際にもすでに開発済みのコードを再利用、再構築していると考えられる。Cozzie らによる “Digging For Data Structures”[64] ではメモリ空間に発生するデータ配列のパターンからデータ構造体を検出し、類似するプログラムを発見する手法を論じている。この研究成果によって、同じ系統のマルウェア亜種間ではソースコードの類似性が認められている。

このようなマルウェアの性質を利用し、ホスト上でコードレベルのマルウェアの挙動を観察し、検知する手法が提案されている。Clemens らによる “Effective and Efficient Malware Detection at the End Host”[65] では、マルウェアが呼び出す API の状態遷移と API に対する入出力を用いてマルウェアの挙動をモデル化している。また、Sami らによる “Malware Detection

Based on Mining API Calls”[66]でも、Windows 実行形式ファイルのヘッダ部分を分析し、呼び出される API の組み合わせの抽出して機能ごとの分類し、出現頻度を調査することで新しく出現したマルウェアを検出できるシステムについて論じている。このように、ホスト上で監視する場合は多数のホストを同時に監視できないものの、挙動に関するより詳しい情報を得ることができる利点を持つ。

4.5 要求事項

ネットワークセキュリティ監視によってマルウェア対策を実施するに当たっての要求事項についてまとめる。

4.5.1 検知精度

ネットワークセキュリティ監視において最も重要となる課題は検知精度である。検知精度は True negative, True positive の割合と同義である。検知精度が低いとネットワークセキュリティ監視を運用している監視者が検知結果が実際の被害を示しているのかどうかを検証する作業が生じてしまう。具体的には記録していたネットワークトラフィックを利用した検知結果の真正性の調査や、検知結果が示すホストにログインしてマルウェア感染の有無を調査するなどである。このような作業は監視者の負担を増加させてしまう。

多くのマルウェアは亜種によって実行可能ファイル本体のパターンを変更するだけではなく、通信パターンなどをわずかに変更させる場合がある。これによって、単純なパターンマッチ型の IDS で詳細な検知ルールを設定している場合には False negative を起こしやすくなる。一方で、False negative を避けるために汎用化した検知ルールを用いている場合、悪意のない一般の通信をマルウェアの通信として検知してしまうおそれがある。そのため、マルウェア亜種の発生による変化に対応できる耐性を持つことで、検知精度が向上すると言える。

マルウェアを高精度に検知するためには、抽象的な通信モデルと具体的な通信の相関関係を同時に処理できるシステムが必要となる。BotHunter で提案されている通信モデルに基づいた状態遷移の検知手法は、多くの亜種に対して柔軟に検知できるという利点をもつ一方で、細かい通信の相関関係を表現できないという欠点がある。そのため、細やかな通信の相関関係をもつ受動攻撃型マルウェアにうまく対応出来ない。今後も大局的な通信の発生順序と繊細な通信の関係を巧みに扱える機構が必要である。

一方で、新しい通信モデルをもつマルウェアに対する検知結果は検知精度の評価対象には含めない。これは、将来的にどのような通信モデルをもったマルウェアが出現するかは未知であり、予測が不可能なためである。したがって、検知対象に含めるのはすでに明らかになっているマルウェアとする。

4.5.2 検知対象への適応性

マルウェアを検知するネットワークセキュリティ監視システムには、検知するマルウェアの通信モデルの変化に適応できるだけの柔軟性が求められる。第3章で述べたように、能動攻撃型マルウェアと受動攻撃型マルウェアではその通信モデルに大きな違いがある。第4.4.3節で挙げた関連研究の多くはマルウェアの通信モデルを定義し、それに基づいた検知手法を提案、実装していた。そのため、受動攻撃型マルウェアに対しては多くの手法が無効になったと考えられる。

新しい通信モデルのマルウェアを出現直後に検知するのは非常に困難だが、当該マルウェアを解析した上で通信モデルが明らかになった場合、その亜種に対応できるだけの適応性が必要となる。そのためには検知機能を簡潔化した上で、複雑な動作にも対応できるフレームワークを構築しなければならない。

4.5.3 追調査の容易さ

ネットワークセキュリティ監視を運用するに当たり、検知結果に対する追調査は避けられない。これはネットワークセキュリティ監視システムの検知精度が高かったとしても、検知結果がどのようなインシデントを示しているか、どのような事象から検知結果が導かれたかをネットワークセキュリティ監視を運用する監視者は把握しなければならない。

これは、統計的解析手法などのネットワークセキュリティ監視において対応が難しい点である。パターンマッチによって検知をおこなうIDSでは、検知結果がどのルールによって発生したかを追跡し、被害の状況などを把握できる。しかし、統計的解析ではどの事象から検知結果が導かれたかの因果関係を得るためには、解析を再現するなどして調査をしなければならない。これは監視者の作業負担となるだけでなく、監視者が統計的解析手法の詳細な理論などについて熟知している必要があり、より多くの知識や経験が求められる。そのため、検知結果から検知の原因となった事象を導き出せる構造が重要となる。

4.5.4 処理性能

ネットワークセキュリティ監視では複数のホストを一括して監視できるだけの処理性能が必要である。ネットワークセキュリティ監視はホストごとの監視や対策手法に比べ、複数のホストを一斉に監視する環境が多いため、処理負荷が大きくなるのは避けられない。したがって、監視処理をする際にホスト数やトラフィック量の増加に対して急速に処理性能が低下するシステムは望ましくない。もしくは、分散システムとして必要に応じて処理負荷を分散できるシステムが重要である。

4.6 本章のまとめ

本研究ではネットワークセキュリティ監視によるマルウェア対策に着眼する。しかし、マルウェア対策を考慮したネットワークセキュリティ監視の既存手法は、検知精度やマルウェアの通信モデル変化に対する適応性などの点から効果的な対策が難しい。そのため、検知精度、検知対象への適応性、追調査の容易さ、処理性能などの点から問題提起と必要要件を整理した。

第5章 ビヘイビア型ネットワークインシデント 検知

本論文ではマルウェアの対策手法として、ソフトウェアの挙動に着目したネットワークセキュリティ監視手法を提案する。これまで多くのネットワークセキュリティ監視手法ではネットワークトラフィックの断片的な特徴を観測していたため、監視対象のホストで動作しているソフトウェアの特定には限界があった。本手法では1つのソフトウェアが複数のチャネルを用いて通信する点に着目し、ソフトウェアの挙動を追跡する。これによってネットワークトラフィックからソフトウェアの複雑な動作を把握できるようになり、マルウェアに感染したホストの検知が容易になると期待される。

5.1 マルウェアの挙動に基づいたネットワークセキュリティ監視手法 の提案

本論文ではネットワークセキュリティ監視によってマルウェアを検知するために、マルウェアの挙動に着目した手法を考案する。近年のマルウェアは亜種の発生頻度が高くなっており、従来のシグネチャなどを用いたマルウェア検知手法は限界に来ている。第3章での解析では、マルウェアは亜種の数が多くても共通する挙動を持ったものが多いこと、マルウェアはネットワーク通信においても自身を隠蔽化していることが明らかになった。本論文ではこの2つの事実にもとづき、ソフトウェアの挙動を俯瞰的にとらえることができるネットワークセキュリティ監視手法によって問題の解決を図る。

特に本手法では、ある特定のホストに関連した複数の事象を組み合わせることでインシデントを検知するイベント相関分析の手法を発展させ、ミクロな相関関係とマクロな相関関係の2種類を柔軟に組み合わせられる手法を提案する。既存のイベント相関分析やマルウェアの通信モデルに応じて活動していた手法は、ミクロな相関関係とマクロな相関関係のいずれかに分類できるが、どちらか一方だけではマルウェアに十分対応するのは難しい。そのため、本手法ではこの2つを組み合わせたネットワークセキュリティ監視手法を実現する。

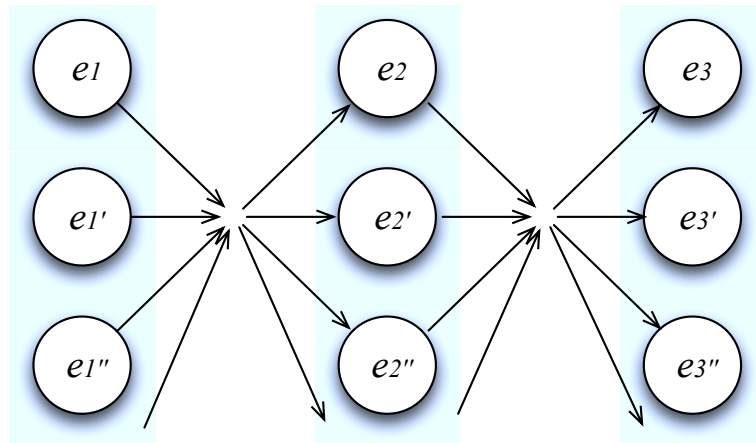


図 5.1: マクロな相関関係の例

5.1.1 マクロな相関関係

ソフトウェアの通信挙動を抽象化し、通信内容や発生順序の変化にも耐性がある相関関係をマクロな相関関係とする。これは、新たに出現したマルウェア亜種に対しての有効性を考慮した検知手法である。多くのボットの亜種は細部が異なるが、おおまかな通信内容や動作手順などに共通する部分が見られる。筆者による調査の他にも、Paul らの調査 [21] で示されているような C&C サーバからの命令形態や命令に関する処理は、新しい亜種でも大幅に変更されない傾向がある。また Guofei の研究 [33] や筆者らの調査でも、複数種類のボットに共通する動作がある点を指摘している。そのため、共通する動作をモデル化すれば新しいボットの亜種も既知のボット同様に検知ができると期待される。

図 5.1 にマクロな相関関係の例を示す。図中では 3 段階のイベントの発生段階を表している。 $e_1, e_1', e_1'' \dots$ が第 1 段階, $e_2, e_2', e_2'' \dots$ が第 2 段階, $e_3, e_3', e_3'' \dots$ が第 3 段階となっており、これらの 3 段階が満たされたことでマルウェアの活動とみなす。各段階では $e_n, e_n', e_n'' \dots$ のいずれの事象が検知されても、第 n 段階を満たしたと判断する。BotHunter でもこのような段階を攻撃、マルウェア転送、マルウェア管理などの活動に抽象化した検知機構を用いることによって、マルウェアが発生させる通信内容が変化した場合でも、高い検知精度を維持している。多くのイベント相関分析ではイベントの発生順序に着目しており、複数イベントが任意の順序で発生する条件や、イベントの発生回数などを条件として設定するのが困難であり、柔軟性に欠ける。しかし、このような相関関係を有効に利用することで、より制度の高い検知の実現が期待される。

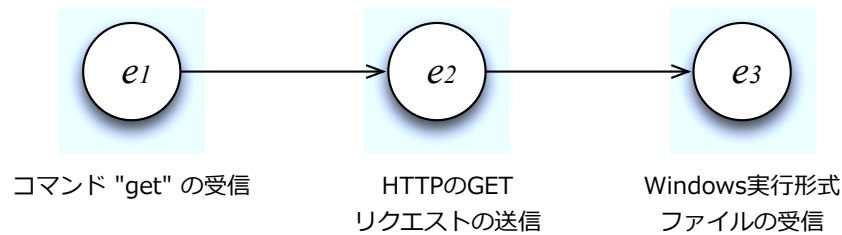


図 5.2: ミクロな相関関係の例

5.1.2 ミクロな相関関係

また、マルウェアの特徴を排した通信を検知するために、ソフトウェアの通信の厳密な関係を定義したものをミクロな相関関係と呼ぶ。第3.2節で示したように、受動攻撃型マルウェアの通信を断片的に見てもマルウェアに関連した活動だと判断するのは難しく、能動感染型マルウェアとは異なる判断基準が必要となる。このような通信を部分的に監視しても、マルウェアの通信なのか悪意のない一般ユーザによる通信なのかを判断するのは難しい。BotHunterは第5.1.1節で述べたとおり、攻撃やマルウェアの転送、マルウェアの管理といった状態毎にイベントを抽象化しているため、汎用性に優れる反面、個々のイベント毎に相関関係を定義できないという構造上の弱点がある。例えば、送信されたGETリクエストのパラメータとダウンロードされたファイルの形式の関係や、あるサーバから指示されたドメイン名と接続先のHTTPサーバのドメイン名との関係などのように、個々のイベント間の具体的な相関関係を定義し、検知に利用できない。

しかし、共通する通信内容の発生順序や通信内容に着目すると、それぞれ通信に相関関係があることが分かった。これらはマルウェア自身の行動アルゴリズムやC&Cサーバからの命令に基づいた行動であり、マルウェアの活動の基礎となる部分である。このようなシステムはボットネット構築に深く関連するため、短期間で著しく変化する可能性は低いと考えられる。また、マルウェアの開発者が相関関係の細部を変更するのは容易だが、厳密なシナリオではなく、マクロな相関関係と組み合わせて複数の分岐を許容するシナリオを記述できれば、検知の回避に対して耐性が高くなる。このように、個々のイベントの関係性を具体的かつ柔軟に捉えることで、汎用性の高いシナリオで精度の高い検知が実現できると見込まれる。そのため受動感染型マルウェアが発生させるような通信を高精度に検知するためには、マルウェアの活動の一部を厳密にシナリオ化した検知手法も必要であると考えられる。

図5.2にミクロな相関関係の例を示す。図中の例では e_1 , e_2 , e_3 が連続して発生した場合にマルウェアの活動とみなしている。これらの個々の事象は一般ユーザの通信にも頻繁に出現するため、各事象をIDSのシグネチャに設定して検知を試みた場合、False positiveが多発してしまう。しかし、マルウェアの活動形態が明らかであれば、どのようなコマンドに基づい

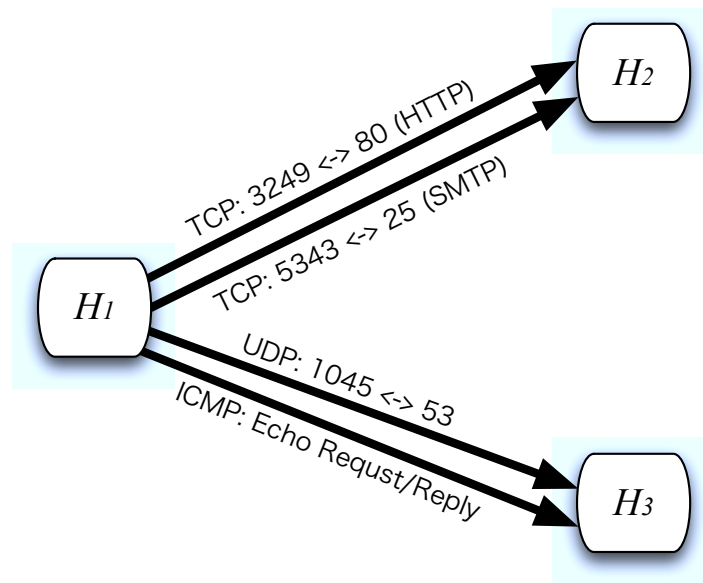


図 5.3: 通信におけるセッションの例

てどのような活動をするか、どのような要求に対してどのような応答があるか、という通信の関係性を利用できる。ただし、これらの事象はコマンドの受信、リクエストの送信、レスポンスの受信、などのように抽象化できるものではない。コマンドやリクエストの内容に応じて多少の活動の変化は起こりうるが、基本的には各事象の関係性を厳密に定義できなければ検知精度は期待できない。そのため、マクロな相関関係とは別にミクロな相関関係を処理できる検知手法が必要となる。

5.2 通信におけるソフトウェア挙動の抽象化

第5.1節で示したネットワークセキュリティ監視手法を実現するためには、多様な通信の相関関係を柔軟に検知できるシステムが必要となる。現状のIDSをはじめとする従来のネットワークセキュリティ監視手法では、このような用途は想定されていないため、本論文で提案する手法を実現するのは難しい。そのため、第5.1節で示したマクロな相関関係とミクロな相関関係の両者を実現するために、既存手法において扱うことができる通信の相関関係を整理し、提案手法で把握すべきソフトウェアの通信挙動を議論する。

5.2.1 セッションとイベントの定義

ソフトウェアの挙動を抽象化するに当たり、本論文におけるセッションとイベントの2つの用語を定義する。

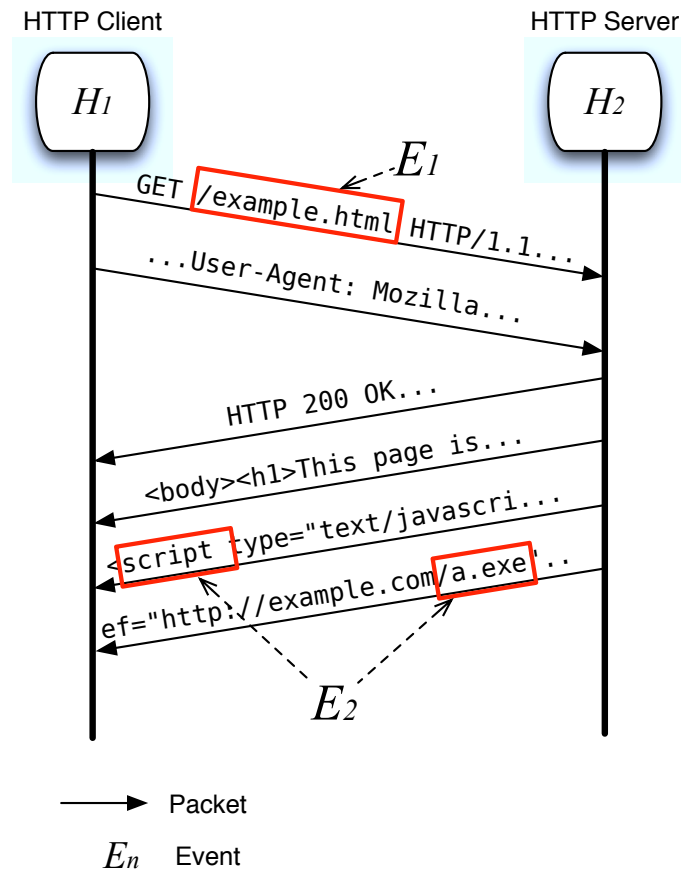


図 5.4: 通信におけるイベントの例

セッションとは IP ネットワークにおける 2 インターフェース間の通信を、目的ごとのチャネルで分割した単位である。実際には OSI 参照モデルにおける第 3 層(ネットワーク層)と第 4 層(トランスポート層)の識別子を用いて抽象化している通信の単位である。2 つの IP アドレス間で通信するとき、TCP を用いた通信では各 TCP セッション、UDP を用いた通信では送信元ポート番号と宛先ポート番号の組み合わせ、ICMP を用いた通信では要求と応答の組み合わせがそれぞれ別々のセッションとして扱う。これは TCP や UDP を使うアプリケーションが 1 つの入出力チャネルとしてソケットを扱うため、TCP セッションや UDP のポート番号の組み合わせ上で発生する通信は同期的になることが多いためである。実際には従来のネットワークセキュリティ監視手法でも、トランスポート層までを一つの通信の単位として扱う実装が多い。

図 5.3 では、セッションの例を示している。 $H_{1,2,3}$ はそれぞれ 1 つのネットワークインターフェースをもち、各インターフェースに 1 つずつ IP アドレスが割り当てられているとする。線にそった文にはトランスポート層のプロトコル名、送信元ポート番号、宛先ポート番号と第 7 層(アプリケーション層)のプロトコルが示されている。図中では H_1 と H_2 は 2 種類の TCP

セッションが存在し、2つのセッションで通信されている。上側のセッションのHTTPでは一つのTCPセッション中に複数回のHTTP要求とHTTP応答を送受信できるが、本論文ではこれらも1つのセッションとして扱う。一方、 H_1 と H_2 の間にはUDPとICMPの通信があり、これもそれぞれ2つのセッションとして扱う。

イベントとは、IPネットワーク上において特定の条件を満たす通信を指す。特定の条件とは送信元および宛先IPアドレス、IPバージョン、トランスポート層プロトコル、TCP/UDPの送信元および宛先ポート番号、ICMPの要求コードや種別コード、データグラムに含まれる文字列パターンなどの任意の組み合わせを指す。これは一般的なシグネチャ型IDSにおける1つのシグネチャに相当し、どのような条件が指定できるかはネットワークセキュリティ監視手法の実装に依存するが、本論文では各イベントはそれぞれ1つのセッションとの対応付けがされているものとする。

図5.4に通信におけるイベントの例を示す。図はHTTPのクライアントとサーバとの通信の一部を現しており、1つの矢印線が1つのパケットに相当する。図中の E_1 は**クライアントからサーバに送信されるパケット**と"/example.html"という文字列がTCPのデータセグメントに含まれることの2つ条件としたイベントとする。また、 E_2 は**TCPのデータセグメントに"<script"と"/a.exe"の両方が含まれること**を条件としたイベントである。この場合、2つのパケットにまたがって文字列が出現しているが、多くのネットワークセキュリティ監視手法の実装ではTCPストリームの再構成が適切に処理されており、このような条件でも1つのイベントとして扱える。

5.2.2 通信の相関関係を利用した既存手法の分類

IDSをはじめとするこれまでネットワークセキュリティ監視では、部分的に通信の相関関係を分析し、検知に役立てる実装を取り入れてきた。しかし、個々の手法は対応出来る相関関係の種類に限界がある。本論文では、既存の相関関係分析手法を5つに分類した。図5.5にそれぞれの手法と通信およびイベントの関係性を示した。

1. 同一セッション内でのイベント発生順序を監視する手法:

同一セッション内で発生する複数イベントの発生順序を監視することで、通信内容をより正確に把握したり、応答内容の違いから異常を検知する手法である。第5.2.1節で定義した通り、イベントの条件は複数のパケットを横断して設定できるが、文字列などの発生順序に着目することでプロトコルや通信内容の正確な把握が容易になる。例えば、Netscreenで採用されている*Stateful Signature*[46]やNetwork Flight Recorder[45]が挙げられる。これらは自由にイベントの出現順序を制御できるが、基本的にはクライアントからサーバ、もしくはサーバからクライアントなど、一方向の通信内容にしか対応していない。そのため、藤田は“侵入検知ポリシーの記述性向上によりログ出力量の低減を可

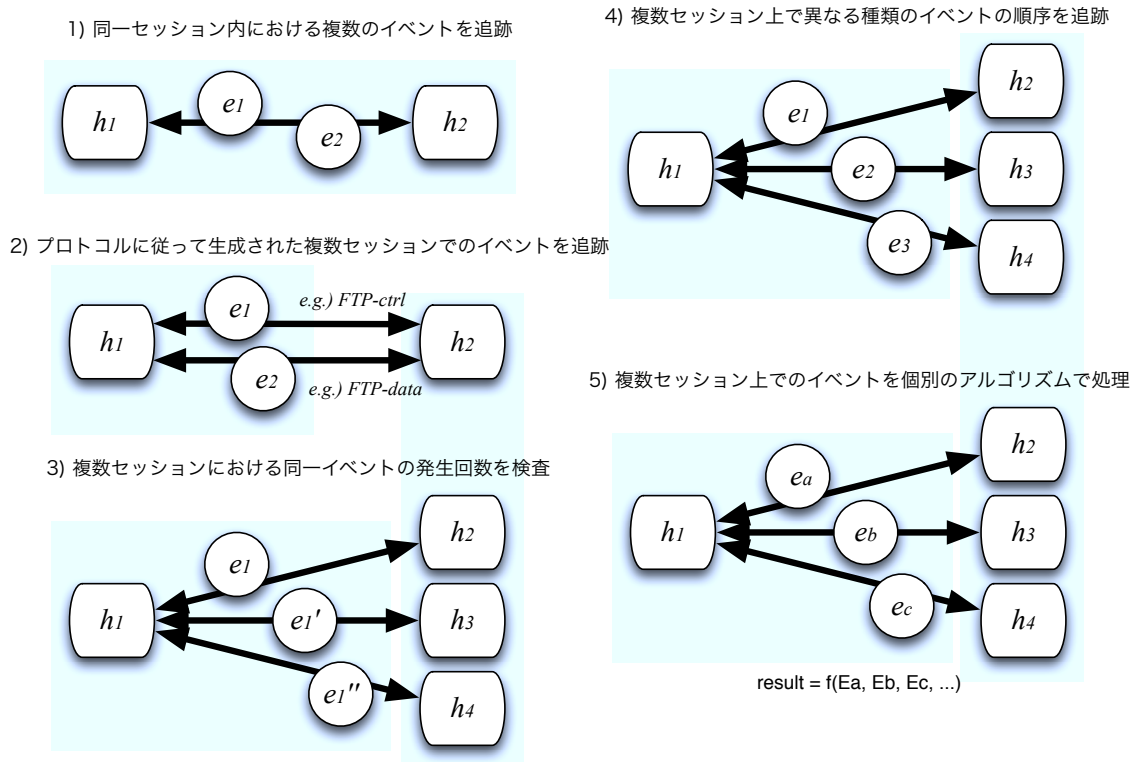


図 5.5: 既存の通信の相関関係を利用したネットワークセキュリティ監視手法

能とした不正アクセス処理システムの開発” [47] においてアプリケーションの双方向の通信をルール化したネットワークセキュリティ監視手法を開発している。

2. プロトコルに従った相関関係を扱う手法:

プロトコルが規定しているセッションの相関関係を解釈することで、セッションの属性を把握する手法は多くの IDS 実装で導入されてきた。例えば File Transfer Protocol (FTP) や Simple Network Management Protocol (SNMP) ではプロトコルの処理手順に従って複数のセッションが発生する。FTP は制御用のセッションとデータ転送用のセッションが別々に通信するように規定されているため、制御用セッションの通信中に出現する命令を監視することで、新しく通信を開始したセッションが FTP のデータ転送であることがわかる。この相関関係は新しいセッションの属性を明らかにし、誤検知を軽減させるために用いられることが多いが、相関関係を処理できるのは実装されているプロトコルや使い方に限定されるのが難点である。

3. 複数セッション上でイベントの発生回数を数える手法:

あるイベントが一定時間中に何回発生したかを数えることで、同じ動作を繰り返すホストを検知できる手法である。2003 年に出現した Slammer ワーム [3] をはじめとする高速伝播型のマルウェアやスパムメールおよびトラックバックスパムを大量に送信するマ

ルウェアは、基本的に特定ポート番号への TCP-SYN パケット送信や、SMTP を用いたメール送信、特定の文字列が含まれる HTTP の POST リクエストなどを短時間で多く繰り返す。このようなイベントが一定時間内に何回発生したかを数える手法はマルウェア検知において有効な手段であり、Snort でも採用されている。ただし、あくまでもイベントの発生回数を数えるだけなので、同一ネットワーク内のサービスを探索するために SYN パケットを多数のホストへ送信するソフトウェアや、メール送信を繰り返すメールサーバなどに対して誤検知をおこす可能性が高い。

4. 複数セッション上でイベントの発生順序を監視する手法:

第 4.4.2 節で示したイベント相関分析では複数種類のイベントの発生順序を分析し、インシデントの全体像を把握を試みている。イベント相関分析が使える実装の多くは、イベントの発生順序を分岐のない状態遷移か木形式の状態遷移のどちらを扱える。これは、イベント相関分析が不正侵入のシナリオに基づいたインシデントの全体像把握を目的としており、個々のソフトウェアの動作などを監視するのが目的ではないためだと考えられる。

5. 規定された手順によってイベントの相関関係を監視する手法:

第 4.4.3 節で示した、目的に応じて通信の相関関係分析手法を定義している手法である。これらは目的に特化して個々の手法を検討、実装しているため、特定のインシデントの検知精度は高い。しかし、目的以外のインシデント検知には有用性が低く、目的に応じた追加の手法検討、実装が必要である。

5.2.3 通信の挙動を表す相関関係の抽象化

第 5.2.2 節の議論を元に、本論文では**イベント発生順序と異なるセッション間での通信内容比較**という 2 点から通信の挙動を抽象化した。本論文ではこの 2 つの抽象化を組み合わせることで多様な相関関係に対応する。

イベント発生規則

イベント間の相関関係を表すためには、従来の同一セッション内でのイベント発生順序の追跡やイベント相関分析などと同様に、イベントの発生規則を管理する必要がある。ただし、複数のイベントが逐次的に発生するのを監視するだけではマルウェアの複雑な挙動に対応するのが難しい。

図 5.6 では $e_{1,2,3,4,5}$ の 5 種類のイベントの組合せによってボットの活動を検知するモデルの例を示している。このモデルは C&C サーバとの通信とボットとしての活動の両方を発見した場合に、ボットを検知したとみなすモデルである。例えば e_1 と e_2 が共に発生した場合は IRC による C&C サーバとの通信を、 e_3 が発生した場合は HTTP による C&C サーバとの通信をそ

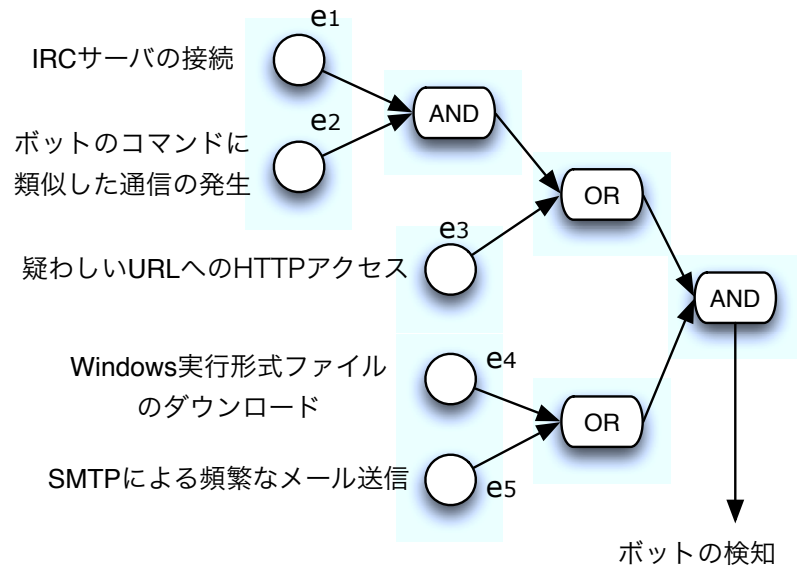


図 5.6: ボットの活動パターンのモデル例

それぞれ検知する。このどちらかを発見した場合、C&C サーバとの通信があったと判断する。さらに e_4 がボット更新用の Windows 実行形式のファイルダウンロード、 e_5 が SMTP による頻繁なメール送信の試みを示している。この2つのどちらかが検知された場合ボットに類する活動があったと判断する。このような活動モデルにもとづいた検知を試みる場合、多様なイベントの発生順序を処理できなければ効果的な検知は難しいと考えられる。

このようなイベントの発生規則に対応するため、本論文では**逐次**、**繰り返し**、**和条件**、**積条件**の4種類のイベント発生規則を定義する。図 5.7 に対応すべきイベントの発生規則をそれぞれ状態遷移図で表記している。独立したイベントを e_n と表記し、それぞれの発生規則を式(例: $e_a e_b$)で表現している。イベントと式は置換でき、複雑な式でも対応出来るようにする。

1. 逐次:

複数のイベントが順番に発生したことを検査する規則。イベント e_a と e_b があつたとき、 e_a 、 e_b の順番にイベントが発生するのを $e_a e_b$ と表記する。イベント相関分析の大部分は逐次型のイベント発生規則に基づいた検知手法となっている。

2. 繰り返し:

あるイベントが一定時間内に一定回数以上発生したかどうかを検査する規則。一つのイベントが N 回繰り返し発生するのを $e_a\{N\}$ と表記する。図中の (2) のように e_1 が 5 回発生したあとに e_2 が発生するのは $e_1\{5\}e_2$ と表記できる。

3. 和条件:

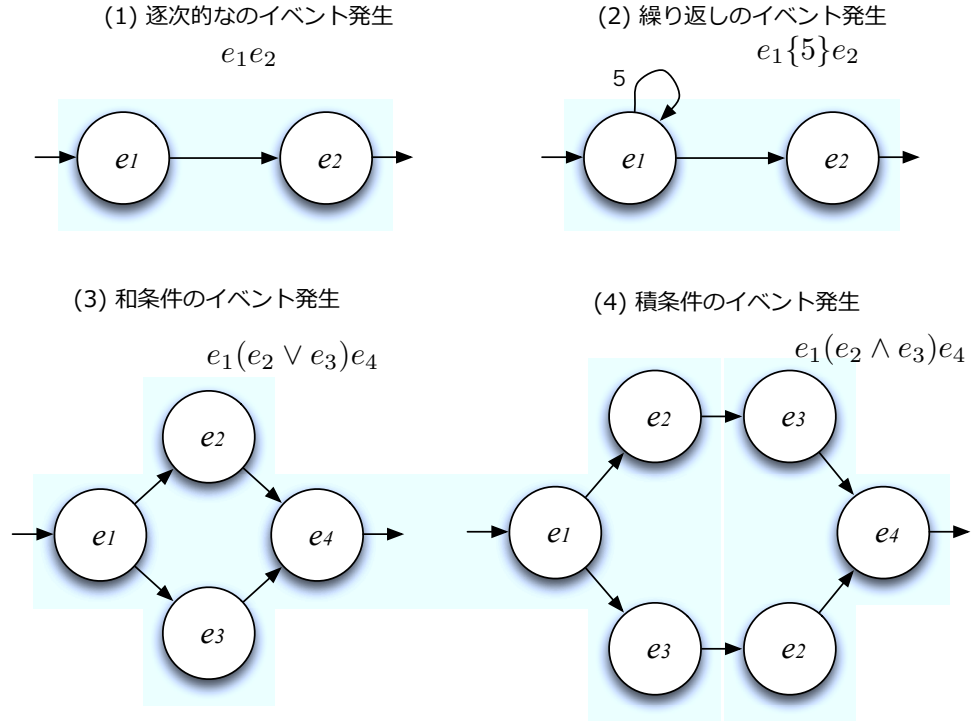


図 5.7: イベント発生順序の抽象化

複数存在するイベントのいずれかが発生したかを検査する規則. e_a と e_b のどちらかが発生したのを表記する場合は $e_a \vee e_b$ と表記する. 図中では $e_1(e_2 \vee e_3)e_4$ となっているため, 発生規則として $e_1e_2e_4$ か, $e_1e_3e_4$ のいずれかが成立すれば規則が成り立ったと判断する.

4. 積条件:

複数存在するイベントがすべて発生したかを検査する規則. e_a と e_b の両方が発生したのを表記する場合は $e_a \wedge e_b$ と表記する. 図中では $e_1(e_2 \wedge e_3)e_4$ となっているため, 発生規則として $e_1e_2e_3e_4$ か, $e_1e_3e_2e_4$ のどちらかが成立すれば規則が成り立ったと判断する.

本論文で提案する手法として重要なのが**和条件**と**積条件**である. イベント相関分析などの既存手法でも, 条件によって結果が分岐していく形式の和条件を実現している実装 [67] はある. しかし, 一度分岐したあと, 再び一つのイベント発生を条件とするような記述はできない. 例えば $(e_1 \vee e_2)(e_3 \vee e_4) \dots (e_{2n-1} \vee e_{2n})$ のような並列規則を実現するためには, 分岐後の状態が $2^n - 1$ 個必要となり, 空間計算量も $O(2^n)$ で増加する. ルールとして記述するのは現実的ではない. また, 積条件のイベントも同様に, $(e_1 \wedge e_2 \wedge \dots \wedge e_n)$ という規則を状態遷移によって満たすためには, $n!$ 個の状態数が必要となり, 空間計算量が $O(n!)$ となる. そのため, 本論

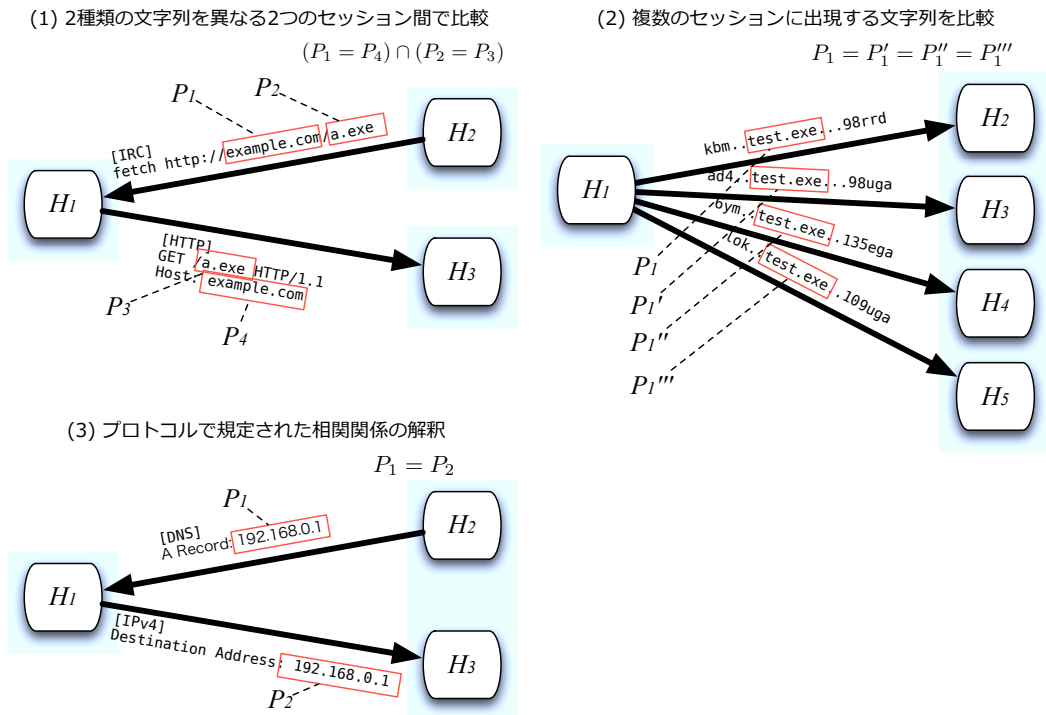


図 5.8: セッション間での通信内容比較例

文で提案する手法を実現するためにはこれらの発生規則を適切に扱える処理系が求められる。

異なるセッション間での通信内容比較

イベントの発生規則と同時に必要となるのが、異なるセッションに出現する通信内容を比較し、相関関係を検査する機能である。これは従来の手法におけるプロトコルに従った相関関係を扱う手法で部分的に実現されている機能を参考にしている。従来手法では FTP の命令を解釈して、新しいセッションの IP アドレスやポート番号を入手し、セッション間の相関関係を理解する。このような他のセッション上に出現したデータに基づいて新しいセッションを開始するソフトウェアは多く、マルウェアの活動でもこのような事例が見られる。そのため、異なるセッション上のデータ比較が抽象化されればソフトウェアの挙動を追跡する上で重要な機能となる。

図 5.8 はセッション間でのデータ比較の事例を示している。図中の (1) ではマルウェアの活動で見られる IRC セッションと HTTP セッションの相関関係をデータ比較によって表現している。 H_1 がマルウェアに感染したホスト、 H_2 が IRC サーバを起ち上げている C&C サーバ、 H_3 が新しいマルウェアの配布サーバである。 H_1 が IRC によって新しいマルウェアをダウンロードする命令を受信したのち、 HTTP によって新しいマルウェアを取得している。この際、 H_2 から送信される命令の URL のドメイン名 (図中 P_1) とファイルパス (図中 P_2) はその後に

発生する HTTP GET リクエストの Host パラメータ (図中 P_4) と要求ファイルパス (図中 P_3) と対応しており, $P_1 = P_4$ と $P_2 = P_3$ の両方が成立てば, H_1 から H_3 への HTTP セッションは H_2 から H_1 への命令が起因していると推測でき, 両セッションの相関関係が把握できる. マルウェアに感染したホストでもユーザが操作していれば HTTP のセッションが発生する可能性は高い. しかし, このような相関関係を見ることで, リスクの高い HTTP セッションの特定ができるようになる.

また, 図中の (2) では部分的に同一のデータパターンが発生しているのを比較している例である. これはマルウェアが感染を拡大させる際に, 連続して複数のホストに新しいマルウェアを送信しているセッションを表している. 図中では `test.exe` というファイル名がすべてのセッションに共通しているが, 近年亜種が大量に発生しているマルウェアではこのような細部の名前などは頻繁に変更される場合が多い. しかし, 送信しているデータの特定部分が共通したパターンを送信するということが分かっているならば, それを利用してセッション間の相関関係を検査できる. 最初に P_1 を保持し, その後発生したセッションの P'_1 , P''_1 , P'''_1 をそれぞれ P_1 と比較する. これによって, 一定回数パターンが一致した場合にマルウェアの感染とみなすようなネットワークセキュリティ監視が実現できる.

図中の (3) では規定された手順で相関関係のあるセッションが生成されたのを, 追跡している例を示している. これは従来手法などでも実装によっては追跡ができた. 図中の例では DNS での問い合わせ結果に含まれる A レコードの IP アドレスを P_1 として保持し, その後発生したセッションの宛先 IP アドレスを検査している. 図中では H_3 が P_1 に保持していた IP アドレスと一致したため, 2つのセッションの相関関係があると判断できる.

5.3 挙動のモデルを用いた挙動追跡型ネットワークセキュリティ監視システムの構想

第 5.2 節で示した, 複雑なソフトウェアの挙動を追跡するために, 本論文では挙動のモデルを示すテンプレートのルールと変数を用いたネットワークセキュリティ監視手法を提案する. 従来, 複数のセッションの相関関係を検知に利用するシステムが提案, 実装されてきたが, それらの多くはオートマトンなどの状態遷移を利用している. しかし, 複雑なマルウェアの挙動を追跡する際, 発生規則が和条件や積条件など, 複雑な規則になると状態数が指数関数的に増加してしまうため, 記憶領域の圧迫が懸念される. また, 異なるセッション間での通信内容比較を実現するためには, 単純な状態遷移は機能が不十分である. そのため, 状態遷移とは異なる方式でセッション間の通信内容比較を実現し, 柔軟なセッションの相関関係を表せるシステムが必要になる.

本論文ではホストの挙動を監視するにあたり, 挙動のテンプレートを用いたネットワークセキュリティ監視手法を提案する. 本手法ではあらかじめマルウェアに感染したホストがとりうる挙動の雛形をモデルとして作成する. このテンプレートには様々なイベントが複数含

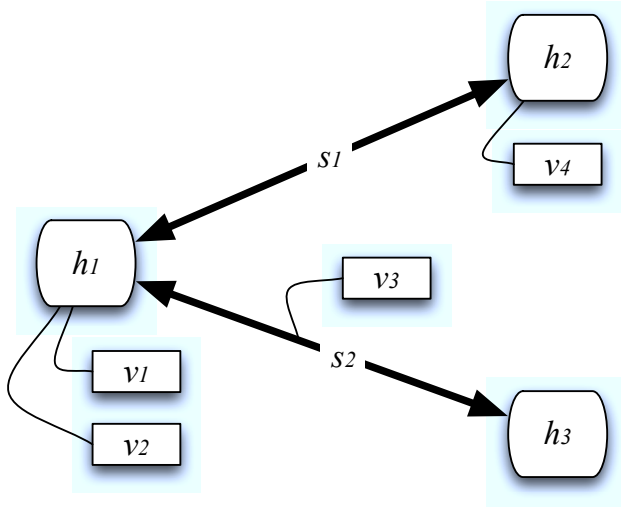


図 5.9: 挙動のテンプレート例

まれ、各イベントの相関関係が示されている。このイベントの相関関係に基づき、マルウェアの活動を検知する。

本手法では挙動のテンプレートにおけるイベントの相関関係を示すために、テンプレートに**変数**を用いている。まず、本論文ではセッションの相関関係をイベントの相関関係として解釈する。イベントは各セッションに対応付けられるため、イベントの相関関係とセッションの相関関係は同様のものとして扱える。**変数**は、イベントの定義と相関関係を示すテンプレートにおいて、任意の指定されたデータやセッション上に出現したデータを保持する記憶領域となり、真偽値、数値、文字列、IP アドレス、バイナリデータなどを必要に応じて記憶する。そして、変数はルールで指定された手順に従って生成され、ホスト (IP アドレス) やセッションと結びつく。ルール R 、イベントを e 、変数を v 、それぞれ R に含まれる個数を N_e 、 N_v 、セッションを s 、ホストを h とすると、式 5.1, 5.2, 5.3, 5.4 のように表現される。

$$R = \{\forall e_{n_s}, \forall v_{n_v} | 0 \leq n_v < N_v, 0 \leq n_e < N_e\} \quad (5.1)$$

$$\exists e_x \in s_n : \{0 \leq x < N_e\} \quad (5.2)$$

$$\exists v_x \in s_n : \{0 \leq x < N_v\} \quad (5.3)$$

$$\exists v_x \in h_n : \{0 \leq x < N_v\} \quad (5.4)$$

図 5.9 に変数 v_n とセッション s_n 、ホスト h_n からなるテンプレートの構成例を示す、図はホスト $h_{1,2}$ と s_2 にそれぞれ変数が割り当てられている様を表している。ホストやセッションはそれぞれ変数として割り当てられており、 $v_{1,2,3,4}$ でそれぞれ異なる値を保持している。そして、ホストやセッションの変数は特定のイベントが発生した際、ルールに従って割り当てが実施される。そのため、図中の h_3 , s_1 のように、変数を持たないホストやセッションも存在

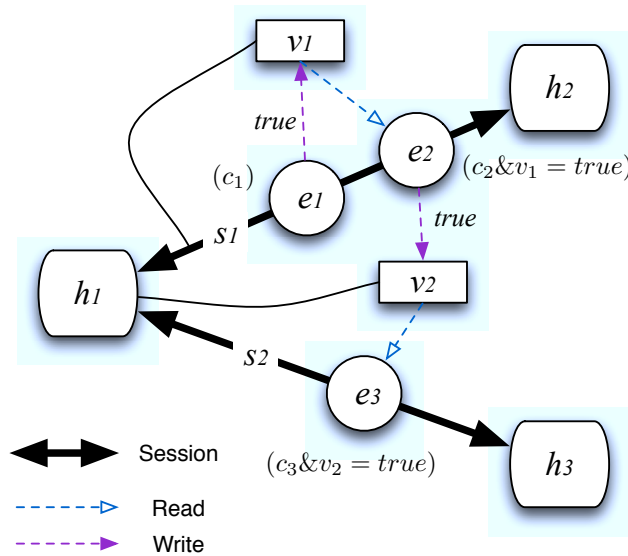


図 5.10: 挙動のテンプレートにおける変数操作の例

する。ただし、変数そのものはネットワークセキュリティ監視システム上で管理されているため、実際のホストやセッションに対して影響をおよぼすものではない。

変数には任意の数値や文字列を記録できるため、これを利用することによってイベントの発生規則やセッション間の通信内容比較が実現できる。まず、イベント発生の有無を変数によって管理すれば、イベントの発生規則を柔軟に捉えることができる。図 5.10 では変数を用いたイベント発生規則管理の例を示している。図中ではイベント発生規則 ($e_1 e_2 e_3$) の検知を試みている。 e_1 , e_2 は同一セッション上でのイベントであり、 e_3 は別セッションでのイベントである。 $e_{1,2,3}$ の通信内容に関する条件を $c_{1,2,3}$ とすると、まず c_1 を満たす通信が s_1 上で発見され、 e_1 が検知される。その際、 s_1 に関連付けられた v_1 を生成し、あらかじめルールで決められた手順に従って v_1 に true を代入する。そして、 e_2 が発生したかを検出する際には、通信内容が c_2 と合致するかを調べるのと同時に、 v_1 が true であることを確認する。この2つが同時に成立すれば、同一セッション内のイベント発生規則 $e_1 e_2$ が成立する。同様に e_2 が検知された際には、 h_1 に関連付けられた v_2 を生成し、そこに true を代入する。 e_3 は c_3 の検出と v_2 が true であるという両者が成り立てば、セッション間のイベント相関関係 ($e_2 e_3$) が成り立ち、($e_1 e_2 e_3$) の規則が満たされたことが分かる。このようにセッションとホストに関連付けられた変数の中継させることで、どのセッションで発生したかに依存せずにイベントの逐次発生規則を処理できる。

さらに繰り返し、和条件、積条件も同様に変数を用いて処理できる。特に第 5.2.3 節で示した状態数の爆発を避けて複雑な条件を表現できるようになる。図 5.11 に例を示す。図中の (1) の繰り返しでは、 e_1 が検知されたとき v_1 に 1 を加算している。よって、 e_2 発生条件は ($c_2 \& v_1 \geq 5$) なので、 e_1 が 5 回発生すれば条件が満たされる。(2) の和条件では 2 つのイベン

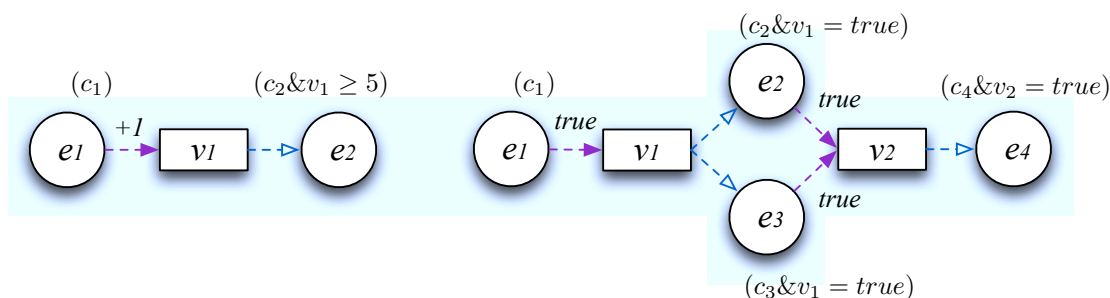
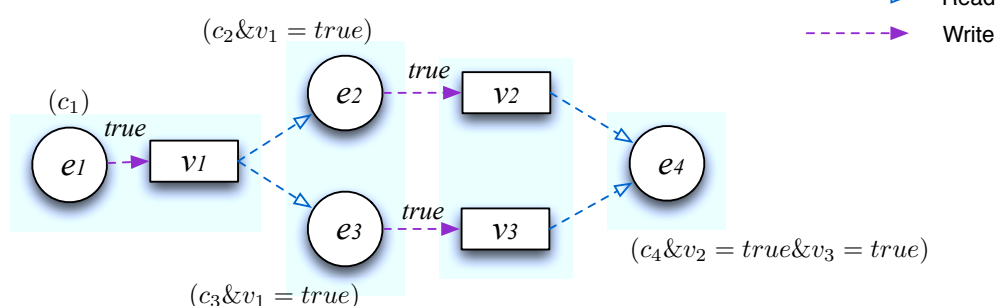
(1) 繰り返し条件 $e_1\{5\}e_2$ (2) 和条件 $e_1(e_2 \vee e_3)e_4$ (3) 積条件 $e_1(e_2 \wedge e_3)e_4$ 

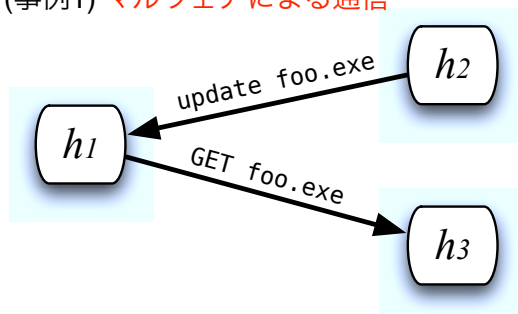
図 5.11: 変数を用いたイベント発生規則の処理例 2

ト $e_{2,3}$ のいずれかが変数 v_2 に書き込みをする. $e_{2,3}$ のどちらでも書き込みができれば, どちらか一方, もしくは両方が発生した後に c_4 が満たされたことを検査でき, $e_1(e_2 \vee e_3)e_4$ の成立を確認できる. そして, (3) でも e_4 が検査する変数を 2 つに増やすことで積条件の表現を実現している. $v_{2,3}$ の 2 つを検査することで, $e_{2,3}$ の両者が任意の順序で発生したことを確認できる. これらの例から分かるように, それぞれ変数を n 個増やした場合でも空間計算量は $O(n)$ にとどまり, 現実的な処理が実現される.

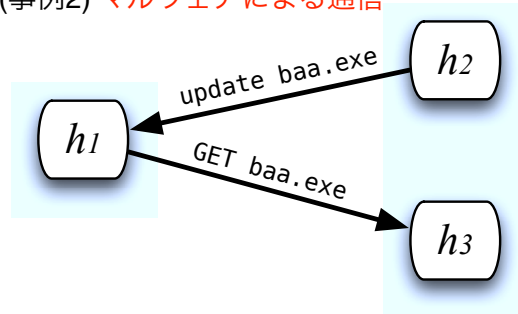
また, 任意の情報を保持できるため, 第 5.2.3 節で挙げた URL(文字列形式) や, 部分的なデータ(文字列, もしくはバイナリデータ形式)の比較, あるいは DNS の A レコードと宛先 IP アドレス(IP アドレス形式)の比較は構造上容易である. 他のセッションで発生したデータをホストの変数に保持しておき, 新しいセッションが発生した際にその変数を参照して比較すればよい.

このように挙動のテンプレートをイベントと変数によって表すことで, イベントの発生規則の処理や異なるセッションでの通信内容比較に基づいた相関関係を表現できるようになった.

(事例1) マルウェアによる通信



(事例2) マルウェアによる通信



(事例3) 通常の通信

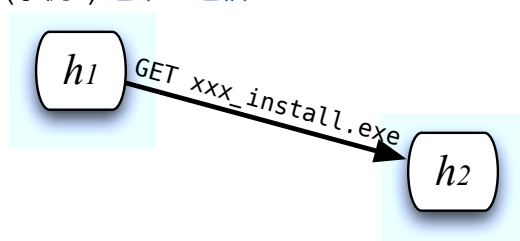


図 5.12: 本手法が有効なシナリオの例

5.4 効果的なシナリオ

本手法を用いることで、検知精度が向上するシナリオの例を示す。図 5.12 は悪意ある通信と正常な通信の例を、表 5.1 は従来のパターンマッチ手法と本手法で記述できる検知ルールと、検知ルールを適用した場合の検知結果を示している。

図 5.12 ではマルウェアによる通信の例 (1,2) と通常の通信の例 (3) を示している。それぞれ (1,2) はマルウェアに感染したホスト (h_1) が C&C サーバ (h_2) から命令を受信し、更新用マルウェアを配信サーバ (h_3) に取得要求を送信している様子を表している。図中の (3) は悪意のない通常のアプリケーションを取得するための要求をサーバ (h_2) に送信している。事例 (1) と事例 (2) の違いは取得するファイル名が異なる点のみである。表 5.1 では、従来の代表的なネットワークセキュリティ監視手法の IDS による検知ルール (シグネチャ) と、本手法で用いられる検知ルールを示している。従来の IDS(1) では要求したファイル名が "foo.exe" であるかどうかを検査し、(2) では要求ファイル名の末尾が ".exe" かどうかを検査している。以前のワームなどを検知する際にはファイル名などの特徴的な情報が変化しないものが多く、このような手法での検知が容易であった。しかし、近年のマルウェアではファイル名などの情報は頻繁に変化するため、このようなルールだけで正確に検知するのは困難である。一方、本手法は感染したホストが C&C サーバから命令と思われる通信を受信した場合にファイル名と思われる部分を変数に保持し、監視対象のホスト (h_1) から送信される要求が保持しているファイル名と一致するかを検査する。これによって疑わしい要求が C&C サーバからの命令に基

表 5.1: 図 5.12 のシナリオに基づいた本手法と既存手法の比較

	検知ルール		
	従来の IDS (1)	従来の IDS (2)	本手法
	recv: GET foo.exe	recv: GET *.exe	recv: (*.exe) & send: GET \1.exe
事例 (1)	True Positive	True Positive	True Positive
事例 (2)	False Negative	True Positive	True Positive
事例 (3)	True Negative	False Positive	True Negative

づいて送信されたと判断できるようになる。

これらのルールに従い、図 5.12 にある通信例の検知を試みた場合の結果を表 5.12 の下 3 段にまとめている。従来の IDS(1) のルールを用いた場合、検知対象となるファイル名が十分に特徴的であった場合、誤検知の可能性は低くなる。しかし、ファイル名が変更された場合に見逃しが発生しやすくなり、事例 (2) では発見すべき通信に対して False Negative を起こしている。しかし、従来の IDS(2) のルールに示されているような検知対象を広くしたルールでは誤検知が発生しやすくなる。事例 (1), (2) については検知できているが、(3) については False Positive を起こしている。これに対して本手法は事前に C&C サーバからの命令を発見し、命令内容の解釈に基づいて検知を実行する。そのため、実際に命令を受信した事例 (1), (2) については True Positive として正常に検知し、事例 3 については通常の活動と判断して検知しない。結果として検知精度を向上できる。

本手法の利点は精度を向上させつつ、マルウェアの亜種発生に対する耐性も改善できる点である。ルールによる検知では、精度と亜種に対する耐性はトレードオフの関係となる場合が多かった。本手法は状態遷移による相関関係だけではなく、セッションに出現したデータの比較もできることから、より柔軟にルールを記述できるようになった。

5.5 本章のまとめ

ネットワーク上で活動するマルウェアは、IDS に代表される従来のネットワークセキュリティ監視手法では対応が難しいため、本論文ではマルウェアの挙動に対応する新しい手法を提案する。一般的なソフトウェアと同様に、マルウェアも複数のチャネルで通信をしながら悪意ある活動を実行する。さらに、第 3 章の解析では異なるマルウェアでも共通する挙動や状態遷移があることが明らかになっているため、検知の精度と亜種への耐性の両者を向上させる手法を提案した。提案手法では TCP や UDP などの通信チャネルをセッションとして抽象化し、セッションの相関関係を把握することでマルウェアの挙動を把握を試みる。これま

でもセッションの相関関係を利用するネットワークセキュリティ監視手法はあったが、機能の面から十分ではなかった。本論文では挙動のモデルを示すテンプレートのルールと変数を導入し、ホストやセッションの状態を管理するシステムを提案した。これによって、今まではネットワークセキュリティ監視手法毎に扱えるセッションの相関関係が限られていたが、本手法ではそれらを包含している。マルウェアの解析結果を用いてルールを作成することで、高精度かつ亜種耐性の高いネットワークセキュリティ監視が期待される。

第6章 ネットワークセキュリティ監視システム: ROOK の設計と実装

本研究ではネットワークにおけるマルウェアの活動を検知するために、変数を用いて複数セッションの相関関係を把握し、高精度な検知を実現するネットワークセキュリティ監視システムとして **ROOK** を設計、実装した。名称は、塔などの高い建築物に視点をおくことで、全体を俯瞰できることに由来する。提案手法は従来手法と比べて検知精度や亜種耐性の向上が期待できる一方で、システムの複雑化や処理性能の低下などが起きる可能性がある。そのため、データ構造の単純化や効率的な処理を目指した設計と実装をおこなった。

6.1 システムの要求事項

提案に基づいたネットワークセキュリティ監視手法を設計、実装するあたり、必要となる要求事項を定義する。要求事項は処理性能、ルール記述の柔軟性、実装の拡張性の3点から議論する。

6.1.1 処理性能

本論文で提案するネットワークセキュリティ監視手法は、従来のIDSに変数という機能を拡張したものに相当する。そのため、従来のIDSと比較して処理が多くなり性能が低下する恐れがある。ネットワークセキュリティ監視システムの処理性能はトラフィックの性質、ルールの種別、パラメータのチューニングなど様々な要素が複雑に影響するため単純な比較は有意とは言えない。しかし、実運用ネットワークで利用するにあたり、ネットワークトラフィックは継続的に監視を続ける必要があるため、到着するネットワークトラフィックを実時間で処理できなければならない。また、並列化などによって処理性能が向上するとしても、運用上は導入できる機材の上限などがある。したがって、可能な限り処理に関する負荷を低下させなければならない。

また、ユーザやシステムに与える影響も考慮しなければならない。監視及び検知対策は他の手法と比較してユーザやシステムに与える影響が少ないが、最も問題になるのが検査の処理性能である。すべてのネットワークトラフィックを確実に検査させなければならない場合、ネットワークの中継地点にネットワークセキュリティ監視システムを導入し、検査の処理性

能に応じたネットワークトラフィックしか通過させないという設置方法(ブリッジ型)がある。このような構成にした場合、著しく処理性能が低ければ大量のパケットロスが発生してしまう。現在、大部分のIPネットワーク通信で利用されているTCPは、10%以上のパケットロスが発生すると急激に性能が低下してしまうため、ネットワーク全体に多大な影響を与えてしまう。したがって、実運用ネットワークでも運用可能な実装にしなければならない。

処理性能を考慮するにあたって、処理にかかる時間と記憶領域の使用効率の2点を考慮する。実運用ではこの2点が同時に満たされる必要があるため、ネットワークプロトコルの性質や変数の使われ方などを考慮し、適切なアルゴリズムなどを選択する必要がある。

6.1.2 ルール記述の柔軟性

本提案手法は事前にルールを設定し、それによってマルウェアを検知する手法である。このルールが柔軟に記述できれば、より多くのネットワークインシデントに対応できる。第3章の解析では亜種の発生頻度が急増し、問題になっている点を指摘したが、新しいモデルのマルウェアも順次出現している。現状での検知率が十分であっても、将来に出現するマルウェアには対応できなくなる可能性が高い。能動攻撃型マルウェアの特徴を利用していた検知手法では、受動攻撃型マルウェアへの対応が難しいのも一例である。このような問題に対応するためには、研究者や開発者だけではなく実際にネットワーク運用をしている運用者などでも検知規則を変更できるルール型が望ましいと言える。

本論文では多様なモデルが表現できることを拡張性と呼ぶ。BotHunterや統計的解析を元にした手法ではマルウェアのモデルを作成し、それに基づいて検知を試みている。しかし、既存の各検知手法は画一的なモデルのみで検知しているためモデルの追加・改善ができず、全く新しいマルウェアが発生した場合に対応が難しい。新しく発見されるマルウェアの大部分は既知のマルウェアの細部が変更された亜種だが、ごくまれに大幅に活動モデルが変更されたマルウェアも出現している。そのためモデルを追加、修正できる拡張性はマルウェア対策において重要であると言える。

6.2 設計

本手法は1つのソフトウェアとして実装する。本手法ではセッションやホストに対して変数を設定し、管理する必要がある。セッションやホストの管理においてはIPだけではなく、TCP、UDPなどのプロトコルを解釈し、各セッションや各ホストの状態も同時に管理しなければならない。ネットワークトラフィックの解釈と管理はIDSなどで基本的に実装されている機能であり、既存のIDSと組み合わせるなどした場合、1つのシステム上で冗長的に処理することになる。また、変数を用いた検知を実施する場合、検知エンジンを根本的な部分から変更する必要がある。従来のIDSでは静的な条件によって検知することを前提に検知エン

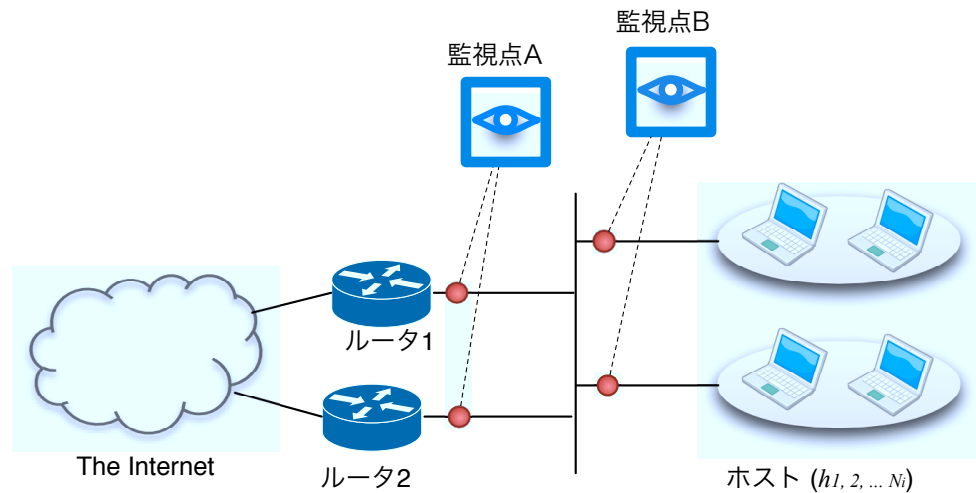


図 6.1: 本手法によるネットワーク監視点

ジンが設計されているが、本手法では動的な変数を積極的に使うことが前提となるため、効率的な検知エンジンの設計が異なってくる。そのため、本手法は独自のソフトウェアとして開発した。

6.2.1 想定するネットワークと観測地点

ネットワークセキュリティ監視システム本手法は IP ネットワークのトラフィックを監視し、マルウェアの活動を検知する。図 6.1 にネットワーク構成の例と監視点の例を示す。図は監視対象となる内部ホスト (h_n) が N_i 台接続しているネットワークである。この例ではインターネットに接続するために、2 つルータが配置されている。近年は冗長化されたネットワークが多く、外部ネットワークに接続する部分でもルータを分割して負荷分散や障害対策をしている場合が見られる。

本手法は監視対象となる管理ネットワーク内のホストの挙動に着目しているため、監視対象となるホストのトラフィックを可能な限り監視できる監視点が望ましい。マルウェアの活動を検知する際にもっとも重要となるのがインターネットを含む外部ネットワークとのトラフィックである。これはマルウェアに感染したホストが外部ネットワークにある C&C サーバやマルウェアの配布サーバなどと通信することによって、新しいマルウェアが導入される、機密情報が漏洩するなどの影響があるためである。そのため、必要最低限の監視をするために外部ネットワークに接続されているルータの前後を監視点 A としている。これは経路が冗長化されている場合はすべての経路のトラフィックを同時に監視しなければならないことを示している。これは、冗長化構成の場合、負荷分散や帯域の調整のために 1 つのルータを通過するトラフィックだけを見ているとあるホストに関連するトラフィックが部分的にしか把握で

きないことがあるためである。もう一方の監視点 B は各ネットワークセグメントや部屋などの物理的な単位で分けられたネットワークのトラフィック通過点をすべて監視している。各セグメントの境界を監視できれば同一管理ネットワーク内で悪意のある活動をしているマルウェアを発見しやすくなるという利点はあるが、簡素装置の設置やネットワークトラフィックの集約にコストがかかってしまうため、状況に応じて監視点を増やすというのが現実的な運用方法になると考えられる。

6.2.2 ルールの記述と運用

本手法は検知対象に詳しい人物が主要なルールを作成し、それを各ネットワークの管理者に配布する形態を想定している。マルウェアを検知するためのルールを記述するためにはマルウェアの動作解析ができる環境が必要となり、一般的なネットワーク管理者がその環境の容易と解析の手間をかけられるとは考えにくい。また、マルウェアを動作をルールとして記述するためには、マルウェアに関する背景などを十分に理解していなければならない。ただし、本手法では検知すべきマルウェアに関する知識をルールという形式に落としこむことができるため、専門家がルールを記述し配布すれば多くのネットワークで本手法の運用ができる。多くの IDS やセキュリティ対策ソフトでも、専門家が検知や防御に必要となるルールやシグネチャを作成し配布するという運用形態をとっているため、本手法も同様に実現可能であると考ええる。

また、ルールも記述内容を簡略化できるようにすることで、一般のネットワーク管理者も必要に応じてルールを記述できるようになる。ルールを配布する運用形態の問題点は、脅威が発生してからルールの作成、配布に時間がかかることだが、ネットワーク管理者が暫定的なルールを作成できれば、突発的に発生した危険性の高い脅威に対しても対応ができる。多くの商用製品ではユーザによるルールやシグネチャの追加や変更を許可していないが、許可している製品やオープンソースのソフトウェアなどでは任意のルール・シグネチャを追加でき、ネットワーク管理者が独自にルールを作成したり、有志でルールを作成して共有するなどの活動が見られる。よって、本手法もルールの記述方法をオープンにし、任意のルールを追加できるようにすることで、この問題を軽減できると考える。

6.2.3 変数の設計

本実装で用いる変数は第 5.3 節で示した要件を満たしつつ、柔軟にあつかえるよう設計する。表 6.1 に必要となる変数の種類と処理内容を示す。

本実装では主に数値型とデータ列型の 2 種類のデータ型が必要となる。変数の重要な機能の 1 つは状態遷移の管理である。第 5.3 節では逐次的に発生するイベントの状態遷移、繰り返し条件、和条件、積条件が必要要件としてあげられている。逐次的なイベント発生と和条件、

表 6.1: 変数に必要な機能一覧

値の型	操作	操作の種類
数値	定数の書き込み	書き込み
	ネットワークトラフィックの解析結果に基づく書き込み	書き込み
	定数の加算	読み込み+書き込み
	定数との比較 (一致・大小)	読み込み
	ネットワークトラフィックの解析結果との比較 (一致・大小)	読み込み
データ列	定数の書き込み	書き込み
	ネットワークトラフィックの解析結果に基づく書き込み	書き込み
	定数との比較 (一致)	読み込み
	ネットワークトラフィックの解析結果との比較 (一致)	読み込み

積条件はそれぞれ変数に対する値の有無だけなので真偽値を管理十分である。しかし、繰り返しについては発生回数を数えるという処理は真偽値型では難しい。したがって、数値型のデータ型が必要となる。真偽値は数値型が包含しているため表中からは省略している。また、本実装では異なるセッションで発生したデータを比較する機能が必要となる。第 5.3 節における事例のように、比較するデータは数値として扱えるものだけではなく、文字列など連続したデータを扱う場合がある。また、これらのデータはデータ長は書き込み時まで不明な場合がほとんどであると考えられる。そのため、データ列型を用意し、任意の長さの文字列やデータ列を比較できるようにする。

変数に対する比較、書き込みはあらかじめルールによって設定されている定数とネットワークトラフィックの解析結果の両者に対応させる。異なるセッション上のデータを比較する場合はネットワークトラフィックの解析結果を用いる必要がある。また数値は加算処理により変化するため、状況に応じてネットワークトラフィックとの比較結果が異なる可能性がある。また、データ列型の変数を v_{array} 、定数を D_{fixed} とした場合、 v_{array} に対して書き込まれるネットワークトラフィックの解析結果 D_{nt} との関係が $v_{array} = D_{nt}$ となり、 D_{fixed} と D_{nt} の比較になるため、イベントの発生条件が v_{array} と D_{fixed} との比較だけの場合は、無意味になる。しかし、イベントの発生条件が v_{array} と D_{fixed} との比較だけではなく、別の変数 v_{other} が積条件や和条件として関係していた場合、 v_{other} と v_{array} の変化はそれぞれ非同期的になるため、 D_{fixed} と D_{nt} の比較だけでは対応できない。そのため、数値型およびデータ列型はともに定数とネットワークトラフィックの解析結果の両方に対して比較と書き込みができなければならない。

また、変数は個別に生存時間が設定できる必要がある。変数の保持には記憶領域を消費するため、無限に保持することはできない。変数が記憶領域を圧迫した場合のために削除するための基準を設け、それにしたがって不要な変数を削除しなければならない。しかし、本手法で用いられるルールはマルウェアを検知するという共通の目的であるという前提があるため、変数ごとに優先度という基準を設けるのは難しい。そのため、本手法では変数に対する書き込み処理時に生存時間を指定するものとする。変数はマルウェアの活動に基づいて状態

$$\begin{aligned}
S &\rightarrow S \vee S \\
S &\rightarrow S \wedge S \\
S &\rightarrow SS \\
S &\rightarrow (S) \\
S &\rightarrow e_n \\
S &\rightarrow e_n\{N\}
\end{aligned}$$

* N は任意の数値

* e_n は任意のイベント

図 6.2: 本実装が記述できる基本的なイベント状態遷移の生成規則

を保持するため、マルウェアの活動に関するタイミングがわかれば、変数を保持すべき時間が分かる。

変数の生存時間がマルウェアの活動に基づく場合、ハーダーはマルウェアの通信する間隔を長くして本手法を回避できる、という議論がある。生存時間を設定したとしても記憶領域は一定であるため、いずれにしても記憶領域が圧迫される可能性はある。しかし、この問題についてはハーダーのリスクという側面から反論できる。通信間隔を長くする場合、マルウェアの活動全体の周期が長くなり、ハーダーの命令が失敗する可能性が高くなる。例えば、マルウェアに感染する際、攻撃対象のホストに対する攻撃コードの送信と、マルウェア本体のダウンロードは異なるセッションになる場合が多い。本手法はこの2つの相関関係を利用して感染を検知できるが、攻撃コードの送信とマルウェア本体のダウンロードの間隔が長くなるほど、状態を保持する変数が記憶領域を圧迫する。ただしこれはハーダーにとってもリスクであり、マルウェア本体のダウンロードが遅れれば、対象となるホストがネットワークから離脱したり、OS がシャットダウンされるおそれがある。マルウェア本体がダウンロードされて実行されなければ、最終的に攻撃は失敗してしまうため、ハーダーはみだりに通信の間隔を長くしにくいと予想される。これは、マルウェア感染後にボットネットとして命令を発行する際にも同様のことが言える。また、マルウェア解析によって最新のマルウェアの通信間隔を把握しながらルールを修正することも可能である。したがって、本手法は変数の生存時間を設定することによって、必要最低限の記憶領域使用で稼働できると考えられる。

6.2.4 変数処理に基づく挙動のテンプレートの制御とマルウェアの検知

本実装では挙動のテンプレートをルールとして記述し、変数によって監視対象となるホストの状態監視を制御している。図 6.2 では本実装で表現できるイベント発生順序の生成規則

を表している。図 5.11 でも本実装が変数を用いて状態遷移を制御する例を示したが、図 6.2 では各条件の処理を拡張できることを示している。マルウェアの活動を大局的に示している第 3.3 節の図 3.6 では各状態には複数のイベントが含まれており、このモデルは BotHunter においても採用されている。このような活動モデルでは、イベントの発生順序の相関関係は $(e_{1,1} \vee e_{1,2} \vee \dots e_{1,n_1})(e_{2,1} \vee e_{2,2} \vee \dots e_{2,n_2}) \dots (e_{m,1} \vee e_{m,2} \vee \dots e_{m,n_m})$ と表せる。本実装では 1 つのブロック内に任意の数のイベントを記すことができるため、このような活動モデルのマルウェアを検知するために十分な表現力があると言える。さらに変数を用いた制御によって、マルウェアの特徴をもつ活動が一定回数以上観測された場合にマルウェアの活動を検知したとみなすルールも可能である。例えば、悪意ある活動は迷惑メール送信や他のホストへの感染活動、マルウェア本体の拡散などのイベントが任意の順序で発生したものを補足し、マルウェアの活動とみなすルールである。これは、複数種類のイベント $e_{p,1}, e_{p,2}, \dots, e_{p,n}$ を検知する場合、各イベントが検知されるたびに変数 v_p を加算し、一定数を超えた際にアラートを発行する。このように、和条件と積条件を自由に利用できることで、マルウェアを検知するために必要なマクロな相関関係を表現できる。

さらに、明確な因果関係があるミクロな相関関係を利用し、より検知精度を向上させられる。マルウェアを検知する上でのミクロな相関関係とは、1) マルウェアが C&C サーバや他のボットなどと通信する際の通信プロトコル、2) 受信した命令に基づくマルウェアの活動、3) マルウェアの自律的な動作の 3 点に分類できる。最も単純な相関関係は 3) マルウェアの自律的な動作である。これは、他のホストへの感染活動や DoS 攻撃、C&C サーバの探索などが挙げられる。それぞれ、マルウェア固有の文字列やデータパターンを含んでいれば、それを条件とできる。あるいは、感染活動や DoS 攻撃は類似する通信を数え上げ、あらかじめ設定したしきい値を超えた場合に検知するという方式がある。この方式は、従来のネットワークセキュリティ監視手法でも適用できる実装があり、本実装も同様に実現できる。本実装がより優位となるのは 1) と 2) になる。1) のマルウェアの通信プロトコルに基づいた検知では、プロトコルの状態遷移を追跡するのが望ましい。マルウェアは標準化された IRC や HTTP などのプロトコルを用い、マルウェアであると特定されにくい内容で通信をするが、必ず何かしらの規約に基づいて通信していると考えられる。そのため、通信の断片に出現する個々の文字列やデータパターンが、悪意のない通信にも発生するような単調なものであったとしても、要求と応答の関係や状態遷移の追跡によって検知ができる。これはイベントの逐次的な発生を追跡するだけで実現できるものもあるが、通信の内容に応じて条件分岐が発生する場合もあるため、和条件や積条件を柔軟に扱える本実装なら十分に対応できる。そして、2) の受信した命令に基づくマルウェアの活動を検知するためには、本実装が通信内容の一部を変数に格納する機能を用い、ドメイン名や IP アドレス、URL のような識別子を保持、比較するのが有効である。発行された命令には引数を伴う場合が多く、引数によって挙動が変化する。しかし、引数を利用して相関関係のある通信を発見できれば、より確実にマルウェアの挙動を追跡でき、検知が容易になる。

表 6.2: ROOK 開発環境

OS	コンパイラ
MacOSX 10.6.3 (Darwin Kernel Version 10.3.0)	i686-apple-darwin10-g++ 4.2.1 (Apple Inc. build 5659)
Linux 2.6.31-20-server #58-Ubuntu x86_64 GNU/Linux	g++ (Ubuntu 4.4.1-4ubuntu9)

6.2.5 監視対象の分割

本手法は管理ネットワークに接続しているホストのマルウェア感染を検知することからホスト毎に状態を保持すればよい。監視対象を分割して処理負荷を軽減できる。第6.2.3節で示した通り、変数はルールによって生存時間が指定されるが、断続的に保持していても期待されない検知結果になってしまう恐れがある。また、通常のプロトコル解析でもパケットの到着順序が変わったり、並列で処理になると、正しくプロトコルを解釈できず、破棄すべきパケットを検査したり、検査すべきパケットを破棄してしまうなどの不都合が生じる可能性がある。そのため、本手法では実時間性を維持しながらネットワークトラフィックを解析するのが望ましい。また、イベントの発生が前後してしまうと検知の条件に影響をおよぼす可能性があるため、解析はネットワークトラフィックを逐次的に処理しなければならないという制約がある。

本手法はさらに負荷を軽減させるために、対象となるホストを分割した監視ができる。変数を用いた解析手法は“*Graph-based Intrusion Detection System*”[68]のようにネットワーク全体を把握し、ホスト間の通信を比較することもできる。ただし、第5章で示したように、本論文では管理ネットワーク内のホストの挙動を監視し、ホスト上で動作するマルウェアの検知を目的としている。そのため、監視対象となる内部ホストに関連する変数を継続的に管理できれば、検知結果は変わらない。実際にはある内部ホスト h_i と、 h_i が通信しているホスト $h_{e1}, h_{e2}, \dots, h_{N_e}$ 、およびこれらのホストが通信する際のセッションと変数を管理すればよい。したがって、監視対象となる管理ネットワーク内のホストが N_i だとすると、問題領域は最大で N_i まで分割でき、記憶領域や処理負荷も平均して $\frac{1}{N_i}$ になる。これによって、処理の負荷を大幅に軽減させられる。

6.3 実装

第6.2節の議論に基づき、変数によってホストの挙動を解析できるネットワークセキュリティ監視システムである **ROOK** を実装した。ROOK は C++ を用いて Unix OS 上で開発した。開

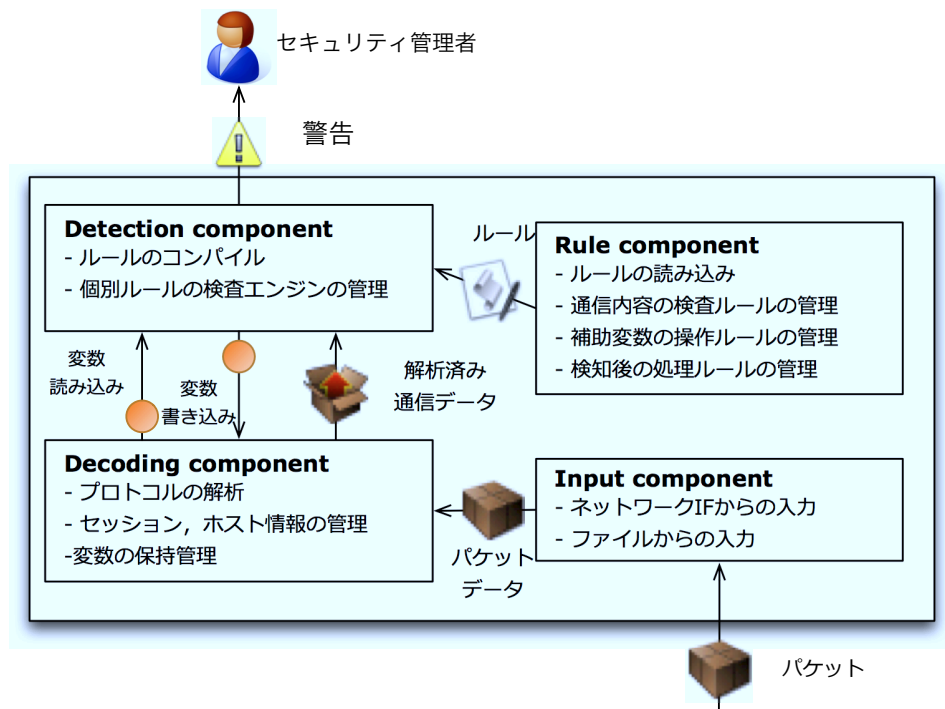


図 6.3: ROOK のコンポーネント概要

発環境を表 6.2 に示す。ライブラリとしてパケットキャプチャに libpcap[69] を、正規表現のパターンマッチに libpcre[70] を用いた。

図 6.3 では ROOK の必要となる機能の概要とを示している。ROOK は主に 4 つのコンポーネントから構成されている。ネットワークトラフィックを収集する Input コンポーネント、パケットデータを解析する解析コンポーネント、ルールから検査パターンや変数の操作を読み込むルールコンポーネント、検知のためのデータ最適化や検査処理を実施する検知コンポーネントである。本節では変数の操作と処理負荷に関わる解析コンポーネントと検知コンポーネント、そしてルール構成をあつかうルールコンポーネントを中心に述べる。

6.3.1 データ構造と資源管理

解析コンポーネント

解析コンポーネントにおけるデータ構造は、記憶領域の使用効率と処理負荷のバランスを考慮して構築している。図 6.4 は解析コンポーネントのクラス図であり、データ構造の概要を示している。ROOK は IPv4 と IPv6 に対してホストとセッションを管理している。ホストは 1 つの IP アドレス、セッションは 2 つの IP アドレスの組み合わせをキーとしてインスタンスを生成している。ホスト及びセッションはパトリシア木、あるいはハッシュテーブルのいずれ

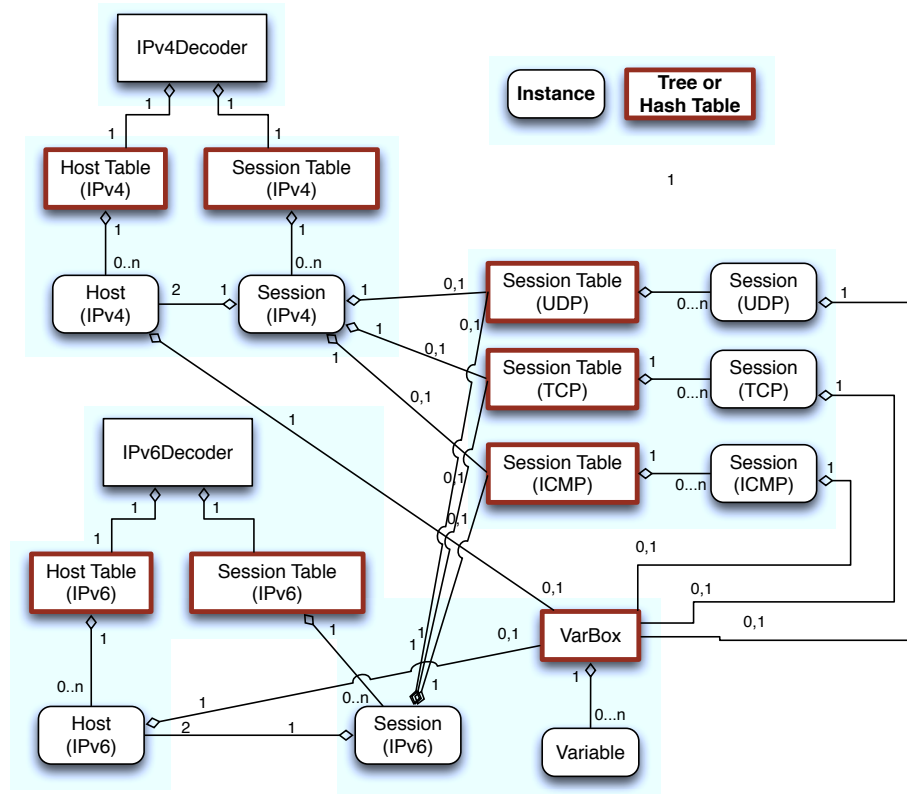


図 6.4: 解析コンポーネントのクラス図

かを起動時に指定でき、計算機のメモリ制約や監視範囲などに応じて変更できる。各 IP セッションのインスタンスは必要に応じて TCP, UDP, ICMP セッションの管理テーブルを持ち、これもパトリシア木かハッシュテーブルかを選択できる。一般的な監視環境では IP セッションが大量に発生し、IP セッション上の TCP, UDP, ICMP の各セッションは平均して 10 程度と予想される。ハッシュテーブルは十分な性能を出すためには広くテーブルを持つ必要があるため、空間計算量がテーブルのサイズに応じて線形に成長するが、検索にかかる時間計算量は定数になる。一方、パトリシア木はノード数 N に対して、空間計算量が $O(N)$ 、時間計算量が $O(\log N)$ になる。ROOK はある管理ネットワークに接続しているホストを監視するのが目的であり、多くの環境において内部ホストの IP アドレスは近接している。したがって、IP アドレスをキーにした場合はパトリシア木に偏りが発生する可能性が高い。よって、IPv4 と IPv6 のホスト、セッションはハッシュテーブル、UDP, TCP, ICMP セッションはパトリシア木を使うのが望ましい。

変数は解析コンポーネントで管理しており、IPv4 と IPv6 のホスト、および UDP, TCP, ICMP のセッションは変数用のテーブル (VarBox) を持つことができる。これは必要に応じて各インスタンスに付与される。変数のテーブルもメモリ使用効率と検索性能を考慮してパト

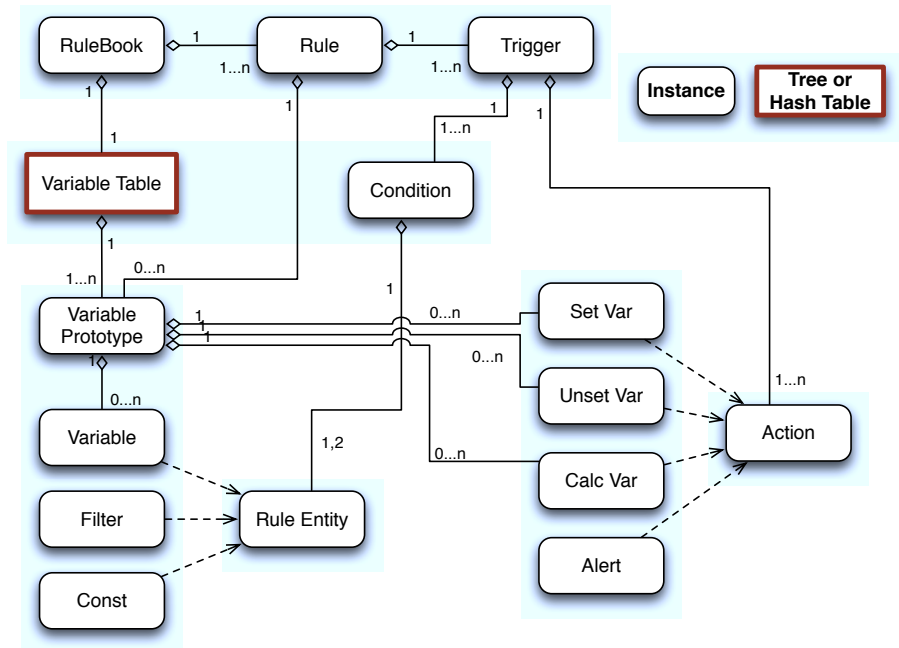


図 6.5: ルールコンポーネントのクラス図

リシア木で実装されており、32bit のキーによって変数が管理されている。よって、1つのインスタンスは最大 2^{32} 個の変数を扱える。

各インスタンスは生成時にタイマに登録され、資源管理される。変数の生存時間や破棄条件はルールによって定義されるため、ルールにしたがって生成、破棄が管理される。一方、セッションやホストのインスタンスについては一定間隔で変数を保持しているかを確認される。入力されたネットワークトラフィックに関連しているセッションやホストは、次回の変数保持の確認を遅らせ、変数を保持していた場合はセッションやホストを破棄せずに、再度タイマに登録される。

ルールコンポーネント

ルールに関する基本的なクラス図を図 6.5 に示す。図 6.5 はルールをファイルから読み込んだ時点でのデータ構造で、ネットワークトラフィックを用いた検査に対する最適化などはされていない。検査のための最適化は検知コンポーネント上でおこなう。

ルールは **RuleBook** インスタンスが統合して管理する。**RuleBook** インスタンスは変数テーブル (**Variable Table**) を持っており、全てのルールに出現する変数のプロトタイプ (**Variable Prototype** インスタンス) を管理し、一意な 32bit の ID を発行する。これをキーとして解析コンポーネント上の **VarBox** テーブルにアクセスし、変数の読み込み、書き込みを実施する。

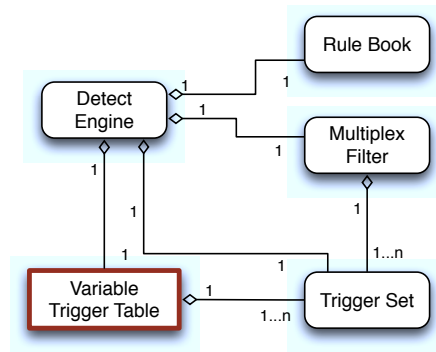


図 6.6: 検知コンポーネントのクラス図

ルールには 1 つ以上の条件式 (Trigger インスタンス) があり, ネットワークトラフィックや変数の比較条件 (Condition インスタンス) や条件式に合致した後の処理 (Action インスタンス) をまとめている. Trigger インスタンスは互いに独立しているため, 1 つのトランザクションと解釈できる. Trigger インスタンスがもつ Condition インスタンスは積条件, 和条件や条件の結合に対応している. よって, ルールの記述内容次第では複雑な条件式となり, 検索の最適化が難しくなる. そのため, 検査すべき Trigger の絞り込みは検知コンポーネントで実施するが, 最終的には Trigger インスタンスによって条件式の精査をする必要がある.

Condition インスタンスは条件検査のための項 (Rule Entity) を 1 つあるいは 2 つもち, これは変数の読み込み (Variable), ネットワークトラフィックに含まれる情報 (Filter), 定数 (Const) のいずれかによって実装されている. これらの Rule Entity インスタンスは真偽値型, 数値型, データ列型, IP アドレス型のいずれかの型を持ち, Condition インスタンスは比較方法 (一致, 大小, 一部を含む) を持つ. Variable インスタンスは Variable Prototype インスタンスから変数操作に必要となる情報を取得する. 条件式に合致した場合は, Trigger インスタンスが保持している Action インスタンスにしたがって検知後の処理が実行される. 実装されているのは変数の変更 (Set Var), 変数の削除 (Unset Var), 変数の計算 (Calc Var), 警告の発行 (Alert) となっている.

検知コンポーネント

検知に関するクラス図を図 6.6 に示す. 検知エンジン (Detection Engine) が全体のルールを Rule Book インスタンスとして持ち, 検査すべき Trigger インスタンスを絞り込む検索関数 (Multiplex Filter インスタンス) を持つ. Multiplex Filter は検査すべき条件数 N に対して時間計算量が $O(N)$ 未満, あるいは定数になるようなアルゴリズムによって Trigger インスタンスを絞り込む. ROOK では Aho-Corasick[71] によるペイロードの探索や IP アドレスやポート番号をキーとした木探索などを実装している. また, 変数による条件式を含む Trigger イン

スタンスをまとめて、変数の ID をキーから Trigger インスタンスを検索するためのテーブル (Variable Trigger Table) を用意した。これは、検査する Trigger インスタンスを最小限にするために、検査中のホストやセッションが保持している変数に関連した Trigger インスタンスを抽出するためである。

図 6.7: BNF によるルール文法

```

<rule_set> ::= {<rule>}
<rule> ::= 'rule' '(' ' ' <rule_name> ' ' ')' <prototype_set> '{' <trigger_set> '}'
<prototype_set> ::= {<prototype>}
<prototype> ::= <type> <var_name> '(' <prototype_arg> ')' ' ';
<prototype_arg> ::= <proto_scope> | <proto_scope> ' ,' 'stack'
<type> ::= 'bool' | 'int' | 'raw' | 'addr'
<var_name> ::= <alpha> { <digit> | <var_name> | '_' | '-' }
<proto_scope> ::= 'host' | 'session'
<trigger_set> ::= <trigger> | <trigger> <trigger_set>
<trigger> ::= 'trigger' '(' <condition_set> ')' '{' <action_set> '}'
<condition_set> ::= <condition> | <condition> <condition_op> <condition_set>
<condition> ::= '(' <condition_set> ')' | <entity_set>
<condition_op> ::= '&&' | '||'
<entity_set> ::= <entity_uni> | <entity_side> <entity_op> <entity_side>
<entity_uni> ::= <filter> | <var>
<entity_side> ::= <filter> | <const> | <var>
<entity_op> ::= '==' | '<' | '>' | '<=' | '>=' | '!='
<filter> ::= <filter_name> '(' <filter_arg> ')'
<filter_arg> ::= '"' <string> '"' | <int>
<const> ::= '"' <string> '"' | <int>
<var> ::= <scope> ' ' <var_name>
<scope> ::= 'dst' | 'src' | 'session'
<action_set> ::= <action> | <action_set>
<action> ::= <action_name> '(' <action_arg_set> ')' ' ';
<action_arg_set> ::= <action_arg> | <action_arg> ' ,' <action_arg_set>
<action_arg> ::= '"' <string> '"' | <int> | <var_name>

<string> ::= {<alpha> | <digit> | <symbol>}
<int> ::= <digit> {<digit>}
<symbol> ::= '!' | '#' | '$' | '%' | '&' | '(' | ')' | '*' | '+' | ',' | '-' |
  '.' | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '@' | '[' | '\' |
  ']' | '^' | '_' | '`' | '{' | '|' | '}' | '~'

<alpha> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
  a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0|1|2|3|4|5|6|7|8|9

```

{ } は 0 回以上の繰り返し

6.3.2 ルール構成

図 6.5 のクラス図をもとにルール構文を作成した。ルールは C 言語の構文を参考に行っている。変数の宣言やスコープの文法、条件式の文法なども C 言語などのプログラミング言語に

表 6.3: filter に利用できる要素と引数

filter_name	説明	引数
ipv4.src_addr	IPv4 ヘッダの送信元アドレス	IP アドレスを指定. プレフィックス長も指定可能
ipv4.dst_addr	IPv4 ヘッダの宛先アドレス	IP アドレスを指定. プレフィックス長も指定可能
ipv6.src_addr	IPv6 ヘッダの送信元アドレス	IP アドレスを指定. プレフィックス長も指定可能
ipv6.dst_addr	IPv6 ヘッダの宛先アドレス	IP アドレスを指定. プレフィックス長も指定可能
tcp.src_port	TCP ヘッダの送信元ポート番号	10 進数で 0 以上, 2^{16} 未満を指定
tcp.dst_port	TCP ヘッダの宛先ポート番号	10 進数で 0 以上, 2^{16} 未満を指定
tcp.pcre	TCP セグメントの正規表現	Perl 互換の正規表現で指定
tcp.seg_id	TCP セグメントの出現番号	10 進数で 1 以上, 2^{32} 未満を指定
tcp.seg_size	TCP セグメントのサイズ	10 進数で 1 以上, 2^{32} 未満を指定
tcp.proto	TCP の上位プロトコルの指定	10 進数で 1 以上, 2^{32} 未満を指定
udp.src_port	UDP ヘッダの送信元ポート番号	10 進数で 0 以上, 2^{16} 未満を指定
udp.dst_port	UDP ヘッダの宛先ポート番号	10 進数で 0 以上, 2^{16} 未満を指定
udp.pcre	UDP セグメントの正規表現	Perl 互換の正規表現で指定
http.uri	HTTP リクエストの URI	Perl 互換の正規表現で指定
http.dl_data	HTTP のダウンロードデータ	Perl 互換の正規表現で指定

表 6.4: action に利用できる要素と引数

action_name	説明	第 1 引数	第 2 引数	第 3 引数
set	変数に値を代入する	代入先の変数名	代入する要素	生存時間 (秒)
inc	変数に 1 を加算する	加算する変数名	生存時間 (秒)	
dec	変数に 1 を減算する	減算する変数名	生存時間 (秒)	
unset	変数の値を消去する	削除する変数名	なし	なし
alert	警告を発行する	メッセージ	なし	なし

類似させており, 記述が容易になっている. ROOK で利用するルールは変数の宣言とガード型の条件式および条件成立時の動作が主な要素となっている. 図 6.7 ではバックス・ナウア記法 (Backus-Naur Form, BNF) によって ROOK で利用するルールの文法を示している, この文法中における `<fillter>` 句と `<action>` 句で利用可能なキーワードは表 6.3 と表 6.4 にキーワードの種類と説明をそれぞれ示している.

図 6.8 にルールのサンプルを示す. ルールは主に `rule` セクションと, `rule` セクション内の `trigger` セクションで構成される. 図中では 1 行目から 16 行目までが 1 つの `rule` セクションであり, 5 行目から 7 行目, 9 行目から 11 行目, 13 行目から 15 行目がそれぞれ個別の `trigger` セクションとなっている. `trigger` セクションの引数になっている部分が条件式となっている. 例えば図中の 5 行目では `tcp.pcre ("msass")` が条件式となっており, これは `tcp` のセグメントデータに `msass` という文字列パターンが発生した場合, 真となる.

```
1 rule ("RECIEVED_COMMAND_AND_START_SCAN") {
2     bool recv_cmd (host);
3     int p135 (host);
4
5     trigger (tcp.pcre ("MSASS")) {
6         set (src.recv_cmd, true, 600);
7     }
8
9     trigger (src.recv_cmd && tcp.dst_port (135)) {
10        inc (src.p135, 600);
11    }
12
13    trigger (src.p135 >= 100) {
14        alert ("DETECTED_SCAN");
15    }
16 }
```

図 6.8: ルール記述例

変数に関する記述

変数はルールごとに宣言して利用する。図中の2行目と3行目は変数のプロトタイプ宣言となっており、変数の型(図中では真偽値型 `bool` と数値型 `int`) と変数名、変数のターゲットを指定する。変数の型指定やターゲット指定は記述性の向上のため、実装に組み込んだ。変数に関する操作をする場合にはターゲットを表す接頭語として、検査中のネットワークトラフィックの送信元ホストが`src.`、宛先ホストが`dst.`、セッションが`session.`となる。例えばTCPにおけるサーバとクライアントでも、どちらが送信元ホスト、宛先ホストになるかは検査中のネットワークトラフィックに依存する。これはUDPやICMPでは明確にサーバとクライアントを定義できず、TCPセッションでも同時オープンが発生する可能性があるため、状況に応じて指定する必要がある。

変数を参照する場合は、`trigger` セクションの引数の条件として記述する。条件として記述する場合、ネットワークトラフィックの解析結果、別の変数、定数のいずれでも比較対象にできる。変数はそれぞれ型が決まっているため異なる型同士の比較はルール読み込み時にエラーとなる。また、何かしらの値が書き込まれていない変数を含む条件式は常に結果が偽になる。

変数の書き込みは `trigger` セクションの内部に記述する。図中では6行目、10行目の変数の書き込みになる。6行目の `set` 句は図 6.5 における `SetVar` インスタンスに対応しており、変数の型に従った任意の値を書き込む。10行目の `inc` 句は、数値型の変数に対して1加算する。それぞれの句の最後の数値は生存時間であり、秒数で表記されている。書き込みされる度に生存時間は更新される。同じ変数でも状況によって生存させるべき時間が変わるため、書き込み時に任意の値が設定できる。

```

1 rule ("MACRO-EXAMPLE")
2 {
3     bool phase1 (host);
4     bool phase2 (host);
5
6     trigger (tcp.dst_port (139) && tcp.pcre ( exploit_code1 )) {
7         set (dst.phase1, true, 120);
8     }
9     trigger (tcp.dst_port (445) && tcp.pcre ( exploit_code2 )) {
10        set (dst.phase1, true, 120);
11    }
12    trigger (tcp.src_port (80) && tcp.pcre ( js_exploit )) {
13        set (dst.phase1, true, 120);
14    }
15
16    trigger (dst.phase1 == true &&
17        tcp.dst_port (139) && tcp.pcre ( win_exe_pattern )) {
18        set (dst.phase2, true, 120);
19    }
20    trigger (dst.phase1 == true &&
21        tcp.src_port (80) && tcp.pcre ( win_exe_pattern )) {
22        set (dst.phase2, true, 120);
23    }
24
25    trigger (src.phase2 == true && tcp.seg_id (1) &&
26        tcp.pcre ("^(NICK|PASS)") && tcp.pcre ( irc_cmd )) {
27        alert (" DETECTED 'MALWARE' S' ACTIVITY ");
28    }
29    trigger (src.phase2 == true && tcp.seg_id (1) &&
30        tcp.pcre ("^GET") && tcp.pcre ( http_cmd_pattern )) {
31        alert (" DETECTED 'MALWARE' S' ACTIVITY ");
32    }
33 }

```

図 6.9: マクロな相関関係を表すルール記述例

ルール記述例

本実装では挙動のテンプレートをルールによって記述し、各ホストの状態や通信に出現したデータを変数によって制御することで、マクロな相関関係とミクロな相関関係を自由に記述できる。

図 6.9 では第 3.3 節や BotHunter[33] などと示されているような通信モデルにしたがって検知するマクロな相関関係のルール例を示している。この例では 3 段階の状態 (攻撃の受信, マルウェアのダウンロード, C&C サーバとの通信) があり、各状態では複数種類のイベントが発生する可能性がある。図中のルールでは攻撃の受信が 2 種類のイベント ($e_{1.1}, e_{1.2}, e_{1.3}$), マルウェアのダウンロードが 3 種類のイベント ($e_{2.1}, e_{2.2}$), C&C サーバとの通信が 2 種類のイ

```

1 rule ("MICRO-EXAMPLE-1")
2 {
3   raw url (host);
4   bool matchRequest (session);
5
6   trigger (tcp.pcre ("HTTP ://[ A-Za-z0-9\\.-]*S+ ") && tcp.proto (irc)) {
7     set (dst.url, tcp.pcre ("HTTP ://[ A-Za-z0-9\\.-]*S+ "), 120);
8   }
9
10  trigger (tcp.dst_port (80) && tcp.pcre ("GET_/(\S+)") == src.url) {
11    set (session.matchRequest, true, 60);
12  }
13
14  trigger (session.matchRequest && tcp.pcre (win_exe_pattern)) {
15    alert ("DETECTED_MALWARE_DOWNLOAD");
16  }
17 }
18
19 rule ("MICRO-EXAMPLE-2")
20 {
21   int count (host);
22
23   trigger (tcp.seg_id (1) && tcp.seg_size (4)) {
24     inc (src.count, 120);
25   }
26
27   trigger (src.count > 10) {
28     alert ("DETECTED_MALWARE_P2P_COMMUNICATION");
29   }
30 }

```

図 6.10: ミクロな相関関係を表すルール記述例

表 6.5: マクロな相関関係を表すルール記述例における各イベントの説明と対応する行

イベント	説明	対応行
<i>e</i> _{1.1}	NetBIOS (ポート 139) への脆弱性を利用した攻撃	6-8
<i>e</i> _{1.2}	NetBIOS (ポート 445) への脆弱性を利用した攻撃	9-11
<i>e</i> _{1.3}	Web ブラウザの JavaScript 脆弱性を利用した攻撃	12-14
<i>e</i> _{2.1}	攻撃元ホストからのマルウェア送信	16-19
<i>e</i> _{2.2}	HTTP によるマルウェアのダウンロード	20-23
<i>e</i> _{3.1}	IRC による通信の開始	25-28
<i>e</i> _{3.2}	HTTP によるコマンドの受信	29-32

イベント ($e_{3,1}, e_{3,2}$) となっており, 詳細を表 6.5 に示す. 図中では `phase1` と `phase2` という 2 つの変数を宣言し (3 行目~4 行目) 監視しているホストがどの状態にあるかを管理する. 図中では, $e_{1,1}, e_{1,2}, e_{1,3}$ では `phase1` に (7 行目, 10 行目, 13 行目), $e_{2,1}, e_{2,2}$ では `phase2` に (18 行目, 22 行目) それぞれ “true” を代入し, 次の状態を検査する条件式において各変数を検査することで, ホストの状態遷移を把握できる. この状態数や各状態におけるイベント数は任意に設定できるため, マルウェアの活動に関するイベントの種類が増えても対応できる.

一方, 図 6.10 ではミクロな相関関係に関するルールを 2 種類示している. 1 行目から 17 行目は IRC 中に出現した URL を変数に代入し, 当該 URL からの Windows 実行可能形式ファイルダウンロードを検知するルールである. 宣言した変数 `url` (3 行目) はデータ列を保存するための変数であり, 正規表現に一致した URL のファイルパスの部分代入している (7 行目). この変数は URL が送信されたホストに関連付けされて保持されており, 当該ホストが HTTP の GET リクエストを送信した際に, 保持している URL のファイルパスの部分と変数 `url` が一致するかを検査し (10 行目), 一致した場合はセッションの変数 `matchRequest` に “true” を代入して, 状態遷移を監視する. その後, 同一セッションで Windows 実行可能形式のファイルがダウンロードされた場合, 一連の動作がマルウェアによるものだと判断することができる. さらに図中の 19 行目から 30 行目は, 特定の packets が連続して送信して他の感染ホストを探すマルウェアを検知するルールを示している. 21 行目で宣言されている変数 `count` は整数を表しており, TCP セッションの確立後に送信される最初の packets のセグメントサイズが 4 バイトだった場合に, 変数 `count` をインクリメントする. これが一定時間内に 10 回繰り返された場合, アラートを発行する (27 行目から 28 行目).

本説で示した例は, 説明を簡素化するためにマクロな相関関係とミクロな相関関係を別個のルールとして記述したが, 実際のルールではこの 2 つを組み合わせる使用できる. 例えば, 図 6.10 でしめした 2 つのルールを図 6.9 におけるイベントとして扱うことができる. このように 2 つの相関関係を組み合わせることによって, より高精度なマルウェアの検出が実現できる.

6.3.3 検知エンジンの動作

ROOK はネットワークトラフィックを入力として処理を繰り返す. そのため, ネットワークトラフィックの到着回数に対して処理負荷が線形に増加するのは自明である. ただし, ネットワークトラフィック到着後の解析や検査処理については, 実装によって計算量や処理時間が大きく異なる. 特にネットワークトラフィックを検査し, 合致するルールを検索する処理はアルゴリズムや入力されるデータに依存するため, 処理内容を分析し本システムの負荷について言及する.

Algorithm 1 に検知エンジンの処理手順を擬似言語で示す. 検知エンジンはネットワークトラフィックの解析結果 R と, 現在保持されている変数の集合 V を入力として受け取る. 出力としては, 検知処理の終了後には検知した Trigger インスタンスの集合 $T_{detected}$ となる. 第 6.3.2

Algorithm 1: 検知エンジンの処理

Input: Decoding result (R), Set of Related Variables ($V_a = V_{src} + V_{dst} + V_{session}$)

Output: Set of detected triggers (T_d)

begin

```

1   $T_m \leftarrow \text{doMultiplexFilter}(R)$ 
2   $T_v \leftarrow \emptyset$ 
   for  $v \in V_a$  do
   |  $T_v \leftarrow T_v \cup \text{getRelatedTrigger}(v)$ 
3   $T_c \leftarrow T_m \cup T_v \cup T_{other}$ 
    $T_{detected} \leftarrow \emptyset$ 
4  while  $T_c \neq \emptyset$  do
   |  $V_{changed} \leftarrow \emptyset$ 
   | for  $t \in T_c$  do
   | | if  $\text{match}(t, R, V)$  then
   | | |  $V_{imp} = \text{getChangeVariable}(t)$ 
   | | |  $V_{changed} \leftarrow V_{changed} \cup V_{imp}$ 
   | | |  $\text{writeVariable}(V_{imp})$ 
   | | |  $T_{detected} \leftarrow T_{detected} + t$ 
   | |  $V_{changed} \leftarrow V_{changed} \cup V_{imp}$ 
   |  $T_c \leftarrow \emptyset$ 
10 for  $v \in V_{changed}$  do
   |  $T_v \leftarrow \text{getRelatedTrigger}(v)$ 
   |  $T_c \leftarrow T_c \cup T_v$ 

```

節で述べた通り，trigger セクションの独立性を保つために変数の参照と変更は一貫しておこなうが，警告の発行などは別途実行するためである。

- **検査すべき Trigger インスタンスの絞り込み:** まず，一括検索の関数を用いて Trigger インスタンスの集合 T_m を取得する (手順 1)。この処理はルール数や変数の数に依存せず，時間計算量は定数となる。その後，検査中ネットワークトラフィックに関係するホスト，セッションが保持している変数の集合 V_a に含まれる変数 v をキーとして，検査すべき Trigger インスタンスを抽出し， T_v に加える (手順 2)。そして，一括検索される Trigger インスタンスや変数をキーとして抽出される Trigger インスタンス以外の Trigger インスタンス T_{other} と， T_m ， T_v を合成する (手順 3)。以上の手順により，検査すべき Trigger インスタンスの集合 T_c が生成される。
- **Trigger インスタンスの検査:** T_c に含まれる Trigger インスタンス t と，ネットワークトラフィック解析結果 R および変数 V とを比較し，順次検査する (手順 5, 6)。条件に合致していれば，変更すべき変数の集合を V_{imp} に格納し， $V_{changed}$ に加える (手順 7)。そして，変数の書き換えを実行する (手順 8)。また，検知した Trigger インスタンスの集

```

1  trigger (tcp.dst_port (80)) {
2      set (src.a, true, 600)
3  }
4
5  trigger (src.a == true && src.b == true)
6      alert ("ALERT 1");
7  }
8
9  trigger (tcp.dst_port (80) && src.b == true) {
10     alert ("ALERT 2");
11 }

```

図 6.11: 変数の変更に伴うルールの例

合 $T_{detected}$ に t を加える (手順 9).

- **変更された変数から影響のある Trigger インスタンスの抽出:** T_c をすべて検査し終わった段階で, $V_{changed}$ に変更された変数が含まれていた場合, ふたたび当該変数を条件に含む Trigger インスタンスを抽出し T'_c を生成する (手順 10). T'_c が空になるまでこれを繰り返す.

これらの処理から, 検知エンジンの計算時間を求める. まず, 手順 1 の一括検索は検索の処理がルール数や変数の数に依存しない. したがって処理時間を定数項 a_1 とおく. 手順 2 は V_a に含まれる要素の数によって `getRelatedTrigger` 関数の処理回数が決まる. `getRelatedTrigger` 関数の処理時間は一定であるため $|V_a|$ とおく. 手順 3 も処理時間は一定であり, a_2 とおく. 手順 4 からの繰り返しは T_c が空になるまでなので, 手順 5 以降に含めて議論する. 手順 5 のすべての繰り返しは $\sum |T_c|$ 回である. ただし, 一致した Trigger インスタンスの数によって処理が変わるため, 便宜上一致した Trigger インスタンスの集合を $|T_{true}|$, 一致しなかった Trigger インスタンスの集合を T_{false} とおく. 繰り返し中の手順 6 の比較にかかる処理時間を各 Trigger インスタンスに含まれる Condition インスタンスの数になる. また, Condition インスタンスでは必要に応じて変数を参照する必要があるため, T_c に含まれ変数を参照するすべての RuleEntity インスタンスを n_{rv} とおき, 参照しないすべての RuleEntity インスタンスを n_{rc} とおくと $n_{rc} + n_{rv}$ となる. 比較に合致した場合, 手順 7 から手順 8 は変更する変数の数 $|V_{imp}|$ となる. また変数の書き込みは変数のテーブル (VarBox インスタンス) を操作する必要がある. 変数はこれはパトリシア木によって構成されているが, 検索キーは固定長なので定数として扱う. さらに, 手順 10 の繰り返し回数は $|V_{changed}|$ となり, $V_{changed}$ から得られた影響をうける Trigger インスタンスの集合を $T_{affected}$ とおく. これらの指定により, 計算時間は式 6.1, 6.2 から求められる.

$$a_1 + |V_a| + a_2 + f(T_c) \quad (6.1)$$

$$f(T_c) = n_{rc} + n_{rv} + |V_{changed}| + |V_{changed}| + f(T_{affected}) \quad (6.2)$$

この式に対して2つ仮定をおく。本手法では $V_{changed}$ によって繰り返しが継続するルールは想定しない。整合性をとるため $V_{changed}$ が空になるまで検査を繰り返す手順となっており、ルールの記述によっては無限に繰り返しが発生するが、そのようなルール記述は意味がない。図6.11に例を示す。1行目の Trigger インスタンスの条件が満たされた場合、変数 a は変更され、5行目からの Trigger インスタンスが検査のキューに入る。しかし、このようなルールは9行目からのコードのように書いた方が効率的であるのは自明である。したがって、無為に Algorithm 1 における手順4からの繰り返しによる処理負荷の増加は無視する。

この2つの仮定に基づいて式を整理する。定数項を除外し、繰り返しを Σ 関数で表記して $N_{rc} = \Sigma n_{rc}$, $N_{rv} = \Sigma n_{rv}$, $N_{vc} = \Sigma |V_{changed}|$ とおき、 $N_v = |V_a|$ とし、便宜上、計算量の形式で表すと式6.3となる。

$$O(N_a) + O(N_{rc}) + O(N_{rv}) + O(N_{vc}) \quad (6.3)$$

この式から、本システムの検知処理には検査対象のホスト、セッションに含まれる変数の保持数 ($|V_a|$)、定数とネットワークトラフィックを比較する条件の総数 N_{rc} 、変数の読み込み数 N_{rv} 、変数の書き込み数 N_{vc} に依存すると言い換えることができる。 N_{rc} の増加は本システムのみならず、一般的な IDS でも同様の課題がある。よって、本システム特有の処理の負荷に影響するのは $|V_a|$, N_{rc} , N_{vc} の3つとなる。これらが個別に増加した場合、いずれも処理負荷の増加は線形となり、急速に処理負荷が増加することはない。

6.4 本章のまとめ

本研究ではホストの挙動を俯瞰できるネットワークセキュリティ監視手法を提案し、これを設計、実装した。本実装はネットワークに流れている通信を on the fly で監視するため、パケットの到着間隔に対して実時間で処理できる性能が求められる。これに対応するため、従来の IDS などに取り入れられたてきたネットワークセキュリティ監視の高速化手法を取り入れるとともに、ホストの挙動をテンプレートとして表現したルールの制御の負荷を低減する実装とした。具体的にはテンプレートのルールの状態や各通信の相関関係を保持するために、ルール内に変数を導入した。この変数は通信の発生規則や通信に出現した情報を管理するのに適しており、時間計算量、空間計算量の両方を急激に増加させることなく、監視ができる。本実装は第5章で示した要件を満たすよう設計、実装されており、その成果は第7章に詳しく示す。

第7章 評価

本章ではマルウェア検知のために提案，実装した変数をもちいたネットワークセキュリティ監視手法を評価する．本手法は従来のネットワークセキュリティ監視手法と比べて検知精度とマルウェア亜種への耐性が向上したが，変数の導入により従来のIDSなどと比べて処理負荷は上がっている．処理負荷の向上がどのような影響を与えるかを検証し，現実的な運用に耐えられることを実証しなければならない．

そのため，本論文ではマルウェアの通信や実運用ネットワークのトラフィックを用いた精度評価，本実装の性能評価，および関連研究との比較といった3つの視点から評価をおこなう．検知精度の評価では能動攻撃型マルウェアと受動攻撃型マルウェアに関連するトラフィックを用意し，それぞれ用意したルールによる検知率 (True Positive 発生率) を検証した．また，実運用ネットワークのトラフィックを用いて，誤検知率 (False Positive 発生率) を検証した．性能評価については，変数の処理機構がどのような影響を及ぼすのかを計算量によって提示した．さらに，実運用ネットワークのトラフィックやネットワークセキュリティ監視手法の評価に用いられる公開トラフィックデータを利用して，実際に稼働させた場合に適切に動作することを示した．

7.1 検知精度評価

本手法の有効性を示すために，マルウェアに感染したホストが発生させたネットワークトラフィックを収集し，検知率を評価した．第3章の調査により，マルウェアの変化に伴って1) 亜種の大量発生によるトラフィックパターンの頻繁な変更，2) ネットワークセキュリティ監視を回避するための傾向変化という2点が明らかになっている．この2点に対向するために，本論文では変数を用いて複雑な検知を実現するネットワークセキュリティ監視を実装しており，この有効性を証明するために検知精度の評価を実施した．

本論文では，ルール作成用の通信データと評価用のデータの2種類を用意して，評価を実施した．現在，マルウェアによる感染活動は世界的に波及しているが，筆者による解析環境ですべてのマルウェアの活動を網羅できているとは言い難い．そのため，ルール作成用の通信データと評価用のデータをそれぞれ異なる方法で収集することで，本手法が多くのマルウェアに対して有効であることを示す．また，本手法が亜種への耐性があるルールを作成できることを示すために，ルール作成用のデータの収集と評価用のデータの収集は，期間をあけて

実施している。これによって、本手法によって作成されたルールは出現した直後のマルウェアだけではなく、一定期間内に発生した新しいマルウェアにも有効であることを示す。

また、多様化しているマルウェアの感染経路に対応するため、本論文では自律的に通信を開始して攻撃先ホストへの接続を試みる能動攻撃型マルウェアと、ユーザ自身の行動が攻撃の引き金となってしまう受動攻撃型マルウェアの2種類に関連する通信を評価した。第3章で示したように、能動攻撃型マルウェアと受動感染型マルウェアでは、感染経路だけではなく感染後の活動にも大きな違いが見られる。そのため、本論文ではハニーポットによって収集されたマルウェアと悪意のあるウェブサイトの閲覧時に収集したマルウェアの両者に対する有効性を示すことで、多くのマルウェアに対応できる本手法の柔軟性を示す。

7.1.1 評価用通信データの準備

能動攻撃型マルウェアの通信データ収集

False Negative の発生率を確認するために、筆者らがハニーポットから収集したマルウェアを利用した。ハニーポットは2008年6月から同年10月にかけてNepenthes[16]を運用し、936件のマルウェアを収集している。Nepenthesは攻撃コードを受信した場合に、攻撃対象となったソフトウェアの脆弱性を模倣してプログラムを取得する。よって、ハニーポットによって収集されたプログラムは悪性であると思わせるため、これらを実行した際に発生した通信には検知すべきデータが含まれていると言える。本評価では悪性プログラムを自動的に実行して得られた通信データに対して、本手法が活動の見逃しが発生しないかを調査した。

ハニーポットによって収集したマルウェアを2008年11月1日から同月6日の間に実験環境で実行し、False Negative 評価用の通信データを取得した。実験環境は第3.1節と同様にVirtualBox 2.0[17]上で修正パッチを適用していないWindows XP ProfessionalをゲストOSとして動作させた。ゲストOSはNetwork Address Port Translation(NAPT)を用いて構成されたネットワークに接続しており、外部からの接続要求はできない。また、外部への被害防止のため外向きのTCP/UDPポート25, 135, 137, 139, 445, 1433, 1434をそれぞれ遮断し、転送速度を制限した。収集されたマルウェアは起動直後のゲストOS上で実行し、その後1時間かけて通信データを収集した。“*Improving the Efficiency of Dynamic Malware Analysis*”[72]によれば、マルウェアの活動は起動後100秒で主な特徴的な動作が観測されるという。そのため、評価に用いた通信データは十分にマルウェアとしての特徴を含んでいると言える。マルウェアはC&Cサーバからの命令に応じて悪意ある活動を実行したり、マルウェア自身のソフトウェアを更新するため、活動パターンが変化して検知が難しくなる。そのためC&Cサーバからの命令が機能していない場合はマルウェアとしての脅威は低減されるため、通信が発生しないプログラムや、調査活動以外の活動が観測されなかったプログラム、コントロールチャンネルから命令を受信しない、あるいは命令を無視するプログラムは除外し、マルウェアとしての活動が見られた299件の通信データを対象に検知性能を調査した。

受動攻撃型マルウェアの通信データ収集

False Negative 評価には、新たに通信データ D_q を用意した。 D_q は悪意のあるドメイン名のリストを提供している Web サイト [73, 74] の情報と Web 検索エンジンを利用し、Web ブラウザを対象とした攻撃コードが含まれると見られる Web ページを調査した。2009 年 4 月 17 日から同年 5 月 14 日の間に第 3.1 節で示した分析用環境において、脆弱性のある Internet Explorer によって Web ページを閲覧し、通信データを取得した。このうち Windows の実行形式ファイルが自動的にダウンロードされた通信データを D_q とする。Malware Domain List に掲載されていた悪意のある Web ページに含まれる受動感染型マルウェアの通信データ D_p と、他のサイトから確認した悪意のある Web ページに含まれる受動感染型マルウェアの通信データ D_q が共に検知できるかを評価し、検知手法の性能と一般性を示す。ただし、第 3.1 節で示した分析環境では ActiveX コントロール、Adobe Flash Player、PDF 閲覧プラグインを導入していなかったため、これらの脆弱性を利用した攻撃では Windows 実行形式ファイルがダウンロードされても、攻撃が失敗したと見られる。したがって、これらの脆弱性を利用した攻撃の通信データはあらかじめ除外しており、それぞれ D'_p , D'_q とする。

7.1.2 評価用ルールの準備

能動攻撃型マルウェアの検知ルール

本手法による能動攻撃型マルウェアの検知精度を評価するために、IRC によって命令を送受信するボットの主要な活動パターンを示すルールを作成した。これを表 7.1 に示す。ルール作成には Evan らの研究 [34] や Cyber-TA による調査 [75] において明らかにされている能動攻撃型マルウェアによる通信データを参考にした。

ルール R_1 , R_2 , R_3 はそれぞれイベント $e_{1,2,3,4,5}$, $e_{6,7,8,9}$, $e_{10,11,12}$ で構成されている。イベントの発生条件を満たすと指定された動作を実行し、最終的にボットの活動を検知した場合は管理者に対応を促すアラートが発行される。各イベントの動作では変数 v_n にあらかじめ指定した値や通信データの一部を代入し、発生順序の制御や通信データの保持を実現している。

- R_1 : IRC で受信した C&C サーバからの命令にもとづいて、TCP ポート 135, 137, 139, 445 のいずれかを調査する活動を検知するルールである。明らかにされており、 e_2 ではその一部を用いている。また、Base64 形式の命令とみられる文字列も e_3 で検知対象としている。IRC セッションに命令と予想される文字列が発見された後に各 TCP ポートへの接続要求が一定回数を上回った場合、ボットの活動と見なす。
- R_2 : IRC で受信したボット更新ファイルの取得命令にもとづいて、HTTP 経由で実行ファイルを取得するふるまいを検知するルールである。 e_7 において IRC 上で発見した URL を v_5 に保存し、 e_8 の HTTP 要求と v_5 の URL が一致するかを確認する。 e_9 は e_8 からの

表 7.1: 能動攻撃型マルウェアの検知用ルール概要

		イベントの発生条件	イベント発生時の動作
R_1	e_2	(TCP セッションにおける IRC) $\wedge$ $data("(root.mass ipscan adv.start $ $135 137 139 msass advscan lsass scanall")$	$v_2 \leftarrow true$ (120 秒)
	e_3	$(v_1 = true) \wedge data("[0-9A-F]\{64\}")$	$v_2 \leftarrow true$ (120 秒)
	e_4	$(v_2 = true) \wedge$ TCP ポート 135, 137, 139, 445 番への 接続要求	$v_3 \leftarrow v_3 + 1$ (60 秒)
	e_5	$v_3 > 32$	アラート
R_2	e_7	IRC セッション $\wedge$ URL 形式の文字列の出現	$v_5 \leftarrow$ URL 形式の文字列 (120 秒)
	e_8	$data(v_5)$	$v_6 \leftarrow true$ (60 秒)
	e_9	$(v_6 = true) \wedge format(WinExe)$	アラート
R_3	e_{10}	TCP ポート 135, 137, 139, 445, 3127 番への接続要求	$v_7 \leftarrow v_7 + 1$ (120 秒) $v_8 \leftarrow src_addr$ (120 秒)
	e_{11}	$(v_8 = src_addr) \wedge$ Windows 実行形式のファイルの送 信	$v_9 \leftarrow true$ (300 秒)
	e_{12}	$(v_7 > 32) \wedge (v_9 = true)$	アラート

src_addr : 送信元 IP アドレス

$data(R)$: HTTP 応答データに正規表現 R に一致するデータが含まれれば true

$format(WinExe)$: HTTP 応答データが Windows の実行形式ファイルであれば true

状態遷移であるため、 e_8 と同一セッションでの HTTP 応答に Microsoft Windows において実行可能な形式のファイルが含まれていた場合、これをボットの活動とみなす。実行形式の検知は [28] を参考にルールを作成した。

- R_3 : 同一ホストから調査活動と Windows 実行形式のファイル送信が同時期に発生していることを検知するルールである。TCP ポート 135, 137, 139, 445 を用いた調査活動を検知し、同時に送信元アドレスを v_8 に保存する。当該ホストから送信されるパケットに Windows 形式を検知した場合、ボットの感染活動およびマルウェアの送信を実施しているとみなす。

受動攻撃型マルウェアの検知ルール

受動攻撃型マルウェアによるネットワークトラフィックを評価するためのルールを用意した。Web 閲覧時に Java Script 処理をきっかけとして攻略される脆弱性を利用した攻撃を受け、マルウェアの転送や悪意ある活動が発生するというシナリオに基づいて定義されている。ア

表 7.2: 受動攻撃型マルウェアの検知用ルール概要

S_n	e_n	条件の詳細	一致後の動作
S_1	e_{13}	$data("<script function") = true$	$v_{10} \leftarrow true$ (10 秒)
	e_{14}	$data("(\\u[0-9a-fA-F]{4}){10} (\\x5cx[0-9a-fA-F]{2}){10} ([0-9]{2,3},[]*){10} [\\x21-\\x23\\x26-\\x7e]{64} [A-Fa-f0-9]{16})" = true$	$v_{11} \leftarrow true$ (10 秒)
	e_{15}	$v_{10} = true \cap v_{11} = true$	$v_{12} \leftarrow true$ (60 秒) $v_{15} \leftarrow s_addr$ (60 秒)
S_2	e_{16}	$v_{12} = true \cap url(".(jpg gif css txt)$") = true$	$v_{13} \leftarrow true$ (10 秒)
	e_{17}	$v_{13} = true \cap format(WinExe) = true$	A_1 の検知
S_3	e_{18}	$v_{12} = true \cap format(WinExe) = true$	$v_{14} \leftarrow v_{14} + 1$ (10 秒) $v_{16} \leftarrow true$ (60 秒)
S_4	e_{19}	$v_{14} > 0 \cap format(WinExe) = true$	$v_{14} \leftarrow v_{14} + 1$ (120 秒)
	e_{20}	$v_{14} \geq 5$	A_2 の検知
S_5	e_{21}	$v_{15} = s_addr \cap format(WinExe) = true$	A_3 の検知
S_6	e_{22}	$v_{16} = true \cap var_name("^(id ver smtp url [a-z])$" = true$	A_4 の検知

s_addr : HTTP サーバ側 IP アドレス

$data(R)$: HTTP 応答データに正規表現 R に一致するデータが含まれれば true

$url(R)$: HTTP 要求の URL に正規表現 R に一致するデータが含まれれば true

$var_name(R)$: HTTP の GET/POST 要求に正規表現 R に一致する変数名が含まれれば true

$format(WinExe)$: HTTP 応答データが Windows の実行形式ファイルであれば true

v_n (変数の生存時間)

ラートは1つの通信データに対して1つ以上発生する可能性があり、1つでも発見されればマルウェアによる活動があったと見なす。アラート検知用のルールの作成や各事象間の待機時間設定については、通信データ D_p を参考として作成した。

受動攻撃型マルウェアの検知ルールは能動攻撃型マルウェアのそれとは異なり、1つのルールが分岐する形式になっている。これは受動攻撃型マルウェアによる攻撃に共通項が見られ、1つのルールとして集約できたため、このように表現した。受動攻撃型マルウェア検知のルールを A_2 とし、出力されるアラートを $A_{1,2,3,4}$ の4種類に分類した。各々のアラートのシナリオについて説明する。

- A_1 : Web ブラウザの Java Script 処理をきっかけとして攻略される脆弱性を利用した

攻撃(これを攻撃 w とする)が発生し、画像やテキストデータを示す拡張子のファイルが Windows 実行形式ファイルだった場合にマルウェアの攻撃、転送とみなすアラート。".exe"の拡張子のファイルをダウンロードした際にネットワーク監視によって検知されるのを防ぐため、一部のマルウェアはマルウェア本体を画像やテキストデータに見せかける。

- A_2 : 攻撃 w が発生した後、短時間に 5 回以上 Windows 実行形式のファイルがダウンロードされるアラート。マルウェアは耐性を高めて長期間ホスト上で活動するために数種類のマルウェアを転送し、感染させる [12]。そのため、攻撃後に短時間で数種類の Windows 実行形式ファイルのダウンロードがあった場合にマルウェアの活動とみなせる。CCCDataset 2008 の各通信データにおいて実行形式ファイルが平均 4.78 回(小数点第 3 位以下切捨)ダウンロードされていたため、短時間での転送を 5 回以上と設定した。
- A_3 : 攻撃 w が発生した後、短時間内に攻撃を受けたサーバから Windows 実行形式のファイルがダウンロードされるアラート。マルウェアの攻撃と転送には同じサーバが使用される場合が多く、同じ通信相手に対して疑わしい事象が断続的に発生したことを検知する。
- A_4 : 攻撃 w が発生した後、Windows 実行形式のファイルが最低 1 回ダウンロードされ、受動感染型マルウェアの HTTP 要求に多く発生する変数名が含まれた場合にマルウェアの活動とみなすアラート。個々の事象は悪意のない通信でも発生する可能性があるが、一定時間内に連続して発生したことを検知する。

$A_{1,2,3,4}$ を検知するために作成した ROOK のルールの説明を表 7.2 に示す。ROOK は異なる TCP セッションの相関関係を表現するため、ルールは TCP セッション (S_n) と各 TCP セッションに属する検査条件 (e_n) を示している。同じ TCP セッションに属する検査条件は、同一の TCP セッションにおいて連続して発生した場合に条件を満たしたと判断する。各検査条件の発生順序や検査に関する情報の保持はパラメータ (v_n) によって制御している。パラメータはホスト毎に数値や IP アドレス、真偽の値を保持する記憶領域であり、各検査条件では各プロトコルヘッダの値や固定値とパラメータを比較することで、状態遷移の発生などを検出する。また、図 7.1 では各検査条件がどのパラメータによって相関関係を表しているのかを示している。

e_{13} は HTTP の応答データに Java Script が含まれるかを調査する条件である。一方、 e_{14} は Internet Explorer の Java Script 処理をきっかけとして攻略される脆弱性を利用した攻撃コードが含まれる場合、出現する可能性の高い文字列を表している。 $e_{13,14}$ が共に満たされることで、 e_{15} の条件が成立し、悪意のある Java Script を受信した可能性を示す v_{12} に *true* が代入される。 $e_{16,17}$ は e_{15} の成立後にテキストデータや画像データの拡張子を含む要求パスを送信しているにもかかわらず、Windows 実行形式のファイルがダウンロードされた場合に A_1 の検知と見な

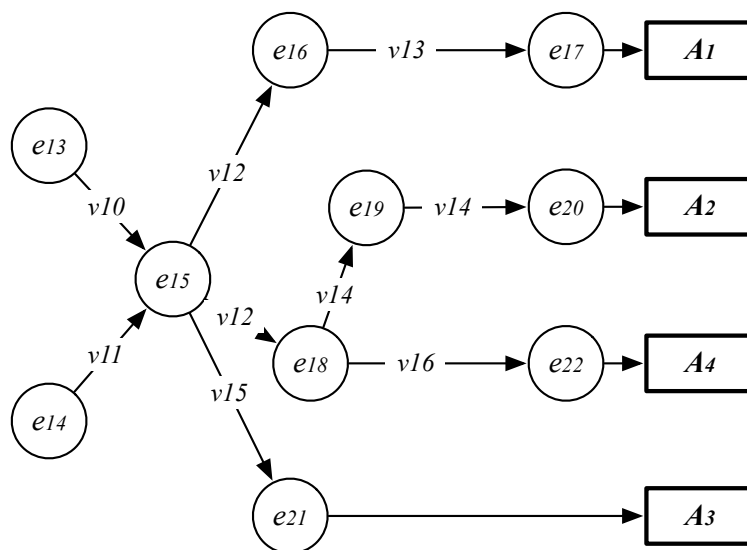


図 7.1: 検知ルールにおける検査条件とパラメータの関係

す. また, $e_{18,19}$ は e_{15} 後に Windows 実行形式のファイルがダウンロードされた回数をカウントしている. e_{20} で 5 回以上のダウンロードが確認された場合は, A_2 の検知と見なす. さらに e_{15} が成立した際に, 悪意のある Java Script が送信されたと考えられるサーバの IP アドレスを v_{15} に保存している. e_{21} では Windows 実行形式のファイルが v_{15} に保存された IP アドレスからダウンロードされたのかを確認し, 一致すれば A_3 の検知と見なす. そして, e_{22} では 1 つ以上の Windows 実行形式ファイルがダウンロードされた後に, マルウェアが頻繁に利用すると見られる単語が HTTP 要求の変数名に含まれているかを確認する. これが成立すれば A_4 の検知と見なす.

各ルールにおける変数保持時間の設定

本実装ではマルウェアの活動を追跡するために変数によって状態を保持するが, 変数を長時間保持することで記憶領域を圧迫する可能性があるため, 保持時間の設定は慎重に検討しなければならない. 本評価における各ルールの保持時間設定は, ルール作成時に参考とした通信データの他に第 3.3 節での調査結果を利用した.

第 3.3 節においてモデル化したマルウェアの状態遷移は大きく分けて初期化フェイズ (図 3.6 における $S_{1,2,3,4}$) と活動フェイズ (図 3.6 における S_5) とに分類できる. 表 3.7 において示した各状態開始から遷移までの所要時間によれば, $S_{1,2,3,4}$ における状態遷移の平均所要時間は 9.743 秒 ($S_1 \rightarrow S_2 \rightarrow S_4$) から 19.138 秒 ($S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$) となっている. 評価で利用したルールは, R_4 が初期化フェイズ, $R_{1,2,3}$ が活動フェイズに位置づけられる. 第 3.3 節での調査結果でも $S_4 \rightarrow S_5$ の状態遷移にかかった所要時間は平均で 137.091 秒, 最大で 489.683 秒と

表 7.3: 比較に用いた Snort のルール

ルールファイル名	能動攻撃型マルウェアの検査	受動攻撃型マルウェアの検査
backdoor.rules	O	O
emerging-malware.rules	O	O
emerging-virus.rules	O	O
emerging-web.rules		O
emerging-web_client.rules		O
emerging-web_server.rules		O
emerging-web_specific_apps.rules		O
emerging-web_sql_injection.rules		O
emerging.rules		O
spyware-put.rules	O	O
virus.rules	O	O
web-activex.rules		O
web-attacks.rules		O
web-cgi.rules		O
web-client.rules		O
web-coldfusion.rules		O
web-frontpage.rules		O
web-iis.rules		O
web-misc.rules		O
web-php.rules		O
合計ルール数	3,387 件	9,012 件

なっており、初期化フェイズ中での状態遷移に比べると長時間待たなければならない。これは、C&C サーバが任意のタイミングで命令を発行するために、初期化フェイズが終了してから活動フェイズに入るまでの時間が不定であるためだと考えられる。しかし、評価で利用したルールは初期化フェイズと実行フェイズが完全に分離しており、この間の状態を変数として保持する必要はない。実装の仕様上、長時間にわたって変数を保持するルールも記述可能だが、 $R_{1,2,3,4}$ では変数の保持時間が一定時間内になっていると言える。

比較用ルールの準備

比較対象の Snort は公式サイト [19] で配布されている Sourcefire VRT Certified Rules と Emergin Threats[20] で配布されているルールを利用した。検査に利用したルールファイルの一覧を表 7.3 に示す。能動攻撃型マルウェアとの比較には 2008 年 11 月 27 日版のルールセットを用意し、3,387 件のルールを利用した。受動攻撃型マルウェアとの比較には 2010 年 5 月 14 日版のルールセットを用意し、から 9,012 件のルールを利用した。

表 7.4: 能動攻撃型マルウェアの通信データを用いた False Negative 評価結果

	Snort で検知成功	Snort で検知失敗	合計
本手法で検知成功	219	75	294 (98.33%)
本手法で検知失敗	2	3	5 (1.67%)
合計	221 (73.91%)	78 (26.09%)	299

* 割合は小数点第3位以下を四捨五入

7.1.3 能動攻撃型マルウェアの検知精度評価

マルウェアの通信データを用いた False Negative 発生率調査

表 7.4 は False Negative の評価結果を示している。それぞれ本手法による検知の成功、失敗と Snort による検知の成功、失敗の内訳を示している。本手法は検知すべき 299 件の通信データに対して、全体の 98.33%である 294 件を検知している。しかし、Snort は全体の 73.91%である 221 件しか検知できていない。さらに、Snort で検知できなかった 78 件中、75 件は本手法で検知している。これは Snort と比較して本手法の見逃しの発生率が大幅に少ないことを示している。一方で本手法が見逃してしまった 5 件のうち、2 件は Snort で検知している。さらに、どちらの手法でも検知できなかった通信データは 3 件であった。

False Negative 評価において本手法が検知できなかった 5 件に対し、追調査を実施した。検知できなかった通信データについて調査したところ、この 5 件は主に HTTP 通信をベースにした活動モデルであり、IRC によるコントロールチャネルは持たなかった。本論文の評価で用いた $R_{1,2}$ は IRC によるコントロールチャネルを想定したモデルであり、HTTP による通信を基盤としたボットのモデルは含まれていない。しかし、この 5 件の活動パターンを分類したところ 3 件には共通性が見られたため、モデル化が可能である。また、残りの 2 件もそれぞれモデル化に利用できると考えられる活動パターンの特徴がいくつか見られた。よって、HTTP 通信による検知用モデルを作成することで、これら 5 件の検知が可能になると考えられる。

7.1.4 受動攻撃型マルウェアの検知精度評価

第 7.1.2 節で示したルールが汎用的に受動感染型マルウェアを検知できることを示すため、False Negative と False Positive の発生率をそれぞれ評価した。False Negative の評価にはマルウェアに感染した際に発生した通信のデータを用い、見逃しが発生しないかを確認した。False Positive の評価には筆者らが利用している実運用ネットワークの通信を用い、悪意のない通信をマルウェアの活動として誤検知しないか確認した。

表 7.5: 検知された受動攻撃型マルウェアの通信データ数内訳

	D'_p		D'_q	
検知成功	250 (100.00%)	$A_1 = 33$ $A_2 = 126$ $A_3 = 224$ $A_4 = 147$	135 (99.26%)	$A_1 = 26$ $A_2 = 61$ $A_3 = 103$ $A_4 = 51$
検知失敗	0 (0.00%)		1 (0.74%)	
検知対象合計	250		136	

* 割合は小数点第3位以下を四捨五入

表 7.6: 受動攻撃型マルウェアの通信データを用いた False Negative 評価結果

	Snort で検知成功	Snort で検知失敗	合計
本手法で検知成功	93	42	135 (99.26%)
本手法で検知失敗	1	0	1 (0.74%)
合計	94 (69.11%)	42 (30.89%)	136

* 割合は小数点第3位以下を四捨五入

False Negative 発生率の評価

表 7.5 は ROOK を用いて受動感染型マルウェアの通信データに対する検知結果の内訳を示している。表では各通信データにおける検知成功数と検知失敗数を示しており、検知成功においては各アラートの検知数も示している。アラートは1つの通信データで複数回発生する可能性があるため、アラートの合計値と検知数は異なる。検知用ルールは D'_p を参考として作成したため D'_p の通信データは 100%検知しており、検知用ルールが正常に機能していることが分かる。一方、 D'_q は全体の 99%以上にあたる 134 件が検知に成功している。

また、検知できなかった D'_q の 1 件を調査したところ、攻撃コードの配布元サーバと Windows 実行形式ファイルの配布元が異なったため、検知に失敗していた。攻撃コードの配布元とは異なるサーバから Windows 実行形式ファイルをダウンロードする行為の検知を試みると、複雑な Ajax など複雑な Java Script を使用しているサイトの閲覧とユーザの意志によるソフトウェアのダウンロードを関連づけてしまう可能性がある。ただし、これはパラメータの有効時間を短くすることで誤検知を軽減できるとみられ、今後一般的な通信データを参考に調整できると考えられる。

さらに、本手法と Snort の検知結果の比較を表 7.6 に示す。本手法が 135 件中、134 件検知できたのに対し、Snort では 94 件のみが検知され、41 件は検知に失敗した。受動攻撃型マル

表 7.7: 誤検知率評価用の通信データ一覧

	収集開始日時	期間	パケット数	データ長	説明	内部ホスト数
D_1	2008-11-04 12:00	14 時間	134.45M	69.26GB	研究ネットワーク	約 500 台
D_2	2008-09-10-14:00	26 時間	1139.16M	48.06GB	カンファレンスのネットワーク	約 250 台
D_3	2008-11-05 13:00	5 時間	743.10M	513.97GB	バックボーンネットワーク	約 100,000 台
D_4	2009-01-25 18:00	3 時間		58.8GB	研究ネットワーク	約 500 台

ウェアは Web に関連した通信が多いため、Snort には能動攻撃型マルウェアを検査した際のルールセットの他に Web 関連のルールセットを追加した。さらに、検査に利用したルールセットは 2010 年 5 月のものであり、通信データの取得から 1 年あまりが経過しているため、当該マルウェアを検査するためのルールセットは十分に用意できる期間があったと言える。しかし、結果として True positive の割合は 69.11% となり、能動攻撃型マルウェアと比べてさらに減少した。これは、第 3.2 節で示したように受動攻撃型マルウェアはより狡猾化しており、既存の IDS では検知が難しくなっていることを示している。一方、本手法では 99.26% を検知しており、高い検知精度が実現できていると言える。

7.2 誤検知率評価

7.2.1 評価環境

False Positive の発生率を評価には 4 種類の実運用ネットワークの通信データを用いた。図 7.2 はボットの通信を発生させている第 7.1.3 節の通信データとは異なり、通信データに含まれるボット通信の件数を正確に把握するのは困難である。そのため、実運用ネットワークの通信データは False Positive の発生率調査のみに利用し、検知数は評価の対象としない。

表 7.7 は各通信データの概要を示しており、図 7.2 は各通信データの収集環境を示している。 D_1 は筆者が定常的に運用している研究ネットワークの通信である。ファイアウォールによって外部からの接続が遮断されているセグメントと、実験用に通信を制限していないセグメントが存在している。 D_2 は 4 日間開催されたカンファレンスのネットワークである。特にファイアウォールなどによる通信制限はない。 D_3 はバックボーンネットワークであり、複数組織が接続しているネットワークである。各組織の運用ポリシーは様々であり、観測点は各組織内ルータの上位となっている。 D_4 は D_1 と同様のネットワークから取得した通信データだが D_1 とは取得時期が異なる。また各ネットワークにおいて Web サーバ、メールサーバ、DNS サーバなどのサービスホストは各セグメント内部に設置されている。

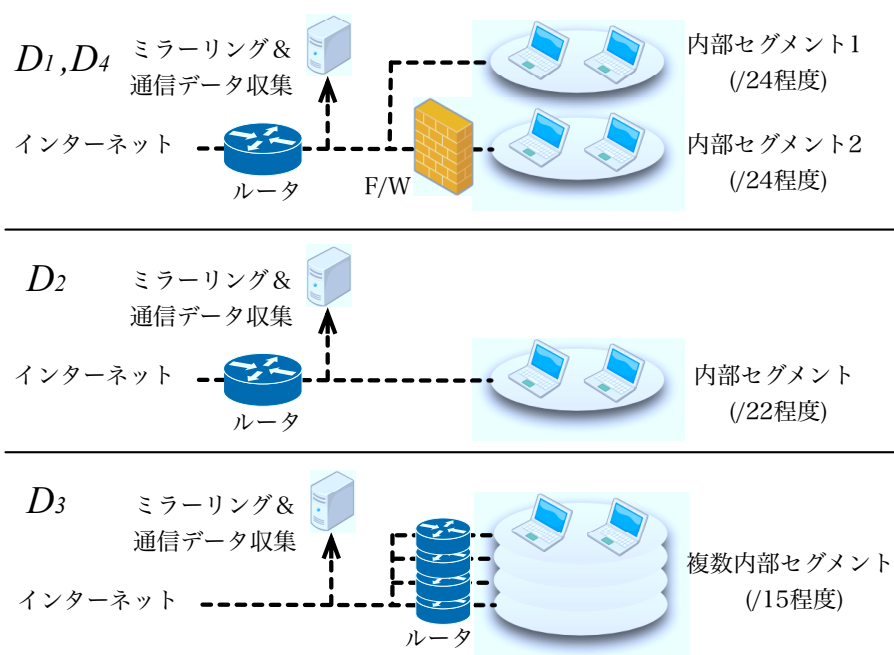


図 7.2: False Positive 評価用通信データ取得環境概要

$D_{1,4}$ の収集環境は研究用ネットワークであり、約 250 台が実験用ホスト、約 250 台が個人利用のホストとなっている。個人利用のホストが接続するネットワークでは一般的利用形態である Web 閲覧、E メール送受信、メッセージサービスが多用されている。一方、実験用ホストでは新しいプロトコルや独自の Web クライアント実装、自律的な Web 情報収集ツールなどが動作している。そのため、一般的な企業・家庭ネットワークと比較して多様性があるため、False Positive が発生しやすい環境となっており、False Positive 発生率の調査に適している。

検知には第 7.1 節の調査同様、本手法による $R_{1,2,3,4}$ および Snort を利用した。ただし、検知結果が実際のボットの活動を検知したのか False Positive なのかを判断するために、Snort で利用したルールは第 7.1.3 節で検知されたものののみを利用した。また、収集および検査時期が異なったため、 $R_{1,2,3}$ と R_4 は異なるデータセットを用いて懸賞を実施した。

表 7.8 に False Positive 発生率調査の結果を示す。表は各データセットに対し、本手法がボットの活動を検知したルールと Snort が検知したルールの各検知数を示している。ただし、パケットを検知対象とする Snort と複数のセッションを検知対象とする本手法とでは検知数の数え方が異なるため、括弧内に一意な攻撃元 IP アドレスの数を示している。本手法が検知した R_3 は、ボットによる調査活動および攻略済みホストに対する Windows 実行形式ファイルの送信を検知するルールである。 D_1 で発生した 2,450 件と D_3 で発生した 942 件を調査したところ、 D_1 では 17 個、 D_3 では 7 個の外部の IP アドレスから筆者らが運用するハニーポットに

表 7.8: False Positive 評価結果

手法	ルール	D_1	D_2	D_3	D_4
本手法	R_1	0	0	0	-
	R_2	0	0	0	-
	R_3	17 (2450)	0	7 (942)	-
	R_4	-	-	-	0
Snort	BACKDOOR DoomJuice/mydoom.a backdoor upload/execute attempt	3 (7)	0	1 (1)	-
ボットの活動が確認された IP アドレス数		20	0	8	0

※ 全ての通信データにおいて検知数が0だった Snort のルール，および明示的にボットの活動を示しておらず誤検知の可能性が高いルールは表から除外している

※ 数値が通信データ中でボットとして検知された IP アドレス数，括弧内の数字が各種法での検知を示す

対して攻撃している通信であった。これはハニーポットのログからも同時間帯に攻撃があり、Windows の実行形式ファイルがハニーポットホストに送信されていたことを確認している。よって、これらの検知結果は False Positive ではなくボットの活動であると判断できる。一方、Snort が検知している “BACKDOOR DoomJuice/mydoom.a backdoor upload/execute attempt” は既にボットに感染しているホストに対して Windows 実行形式のファイルを送信する際のプロトコルを検知している。 D_1 で発生している7件と D_3 で発生している1件も本手法の検知と同様にハニーポットへの攻撃であることを確認した。これらの結果から、両手法は共に第7.1.3節でボットを検知したルールでは False Positive が発生していないことが明らかになった。また、同様に D_4 において R_4 を検証した結果、False Positive が発生しなかったことを確認した。

本手法と Snort の検知結果の内容を比較したところ、Snort と本手法が別のイベントを検知していることが明らかになった。Snort で検知していたのは外部から内部への攻撃で、調査活動をせずに1ホストに対する攻撃を検知していた。筆者らが運用しているハニーポットは検体収集を目的とした Nepenthes であり、攻撃受信後はボットの活動を一部のみ模倣する。そのため、本手法では感染したホストとして検知しなかった。一方、本手法で検知したイベントに対して追調査を実施したところ、第7.1.3節とは異なる Snort のシグネチャによってボットの活動を検知した。

7.3 性能評価

本論文では実装の性能評価を変数に関連する処理に着目して実施した。ネットワークセキュリティ監視の実装では、パケット処理の効率化に関する研究は多くあるため、これらを参考にすればパターンマッチ検索の高速化や記憶容量の節約などが実現できる。本手法は従来の

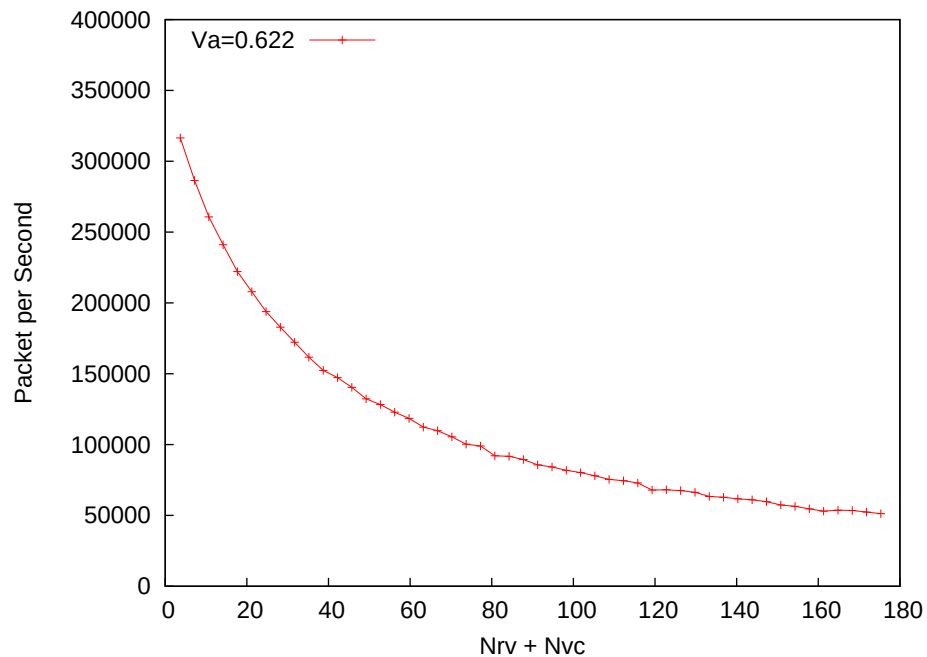


図 7.3: MAWI トラフィックデータを用いた処理性能の測定

ネットワークセキュリティ監視手法に変数という概念を導入し実装しているため、変数の書き込み、読み込み、検査などを実行する際に、どのように処理負荷が変化するかについて言及し、実運用において本実装が利用できることを示す。

トラフィックを利用し、本実装の性能評価を実施した。評価には MAWI Working Group Traffic Archive[26] で公開されている、観測点 F の 2010 年 5 月 11 日の通信データを用いた。本データは 899.26 秒間監視されたデータであり、2923.94MB、41,356,011 のパケットが記録されている。出現した IPv4 アドレスは 508,869 個、IPv6 アドレスは 1,130 個である。解析に用いた環境は CPU が Intel(R) Core(TM) i7 CPU 860@2.80GHz、メモリが 8GB である。通信データはソリッドステートドライブ上に保存し、ディスク I/O による読み込み速度の低下を防いでいる。OS は Linux 2.6、コンパイラには g++ 4.4.1 を用いた。

評価では意図的に負荷をかけるルールを作成し、どのように性能が変化するかを測定した。ルールは 50 種類用意し、それぞれ変数の保持数が一定で、読み込み回数と書き込み回数を変化させている。書き込み回数や読み込み回数、変数の保持数はルールの記述方法やトラフィックの性質によって大きく変化する。同一のルールを用いた場合でも書き込みが発生する条件に一致するネットワークトラフィックが到着する確率によって、書き込み回数が増減し、保持している変数が変化し、条件として読み込まれる変数が変化する。そのため、本システムの定常的な性能を示すのは極めて困難であるが、本評価では書き込み回数の変化が処理性能にどのような影響を与えるかを示すことを目的として、測定を実施している。

評価結果の指標とするために、第7.1節で用いた $R_{1,2,3}$ を使って、実運用ネットワークのトラフィックを観測した際に、発生する変数の平均保持数と読み込み、書き込みの平均発生回数を測定した。MAWIから提供されている通信データはプライバシー配慮のため、パケットの取得サイズが96バイトに制限されており、パケットのペイロード部分も変換されている。よって、評価用ルールが正常に動作しないと見られるため、別途観測用のデータを用意した。観測は第7.2節における D_1 と同じ観測点で2010年5月12日に実施した。その結果、パケットが到着した際に解析対象となるホストやセッションが保持していた変数の平均保持数は0.0331(小数点第4位以下切り捨て)であった。また、平均読み込み数と書き込み数は1.6631(小数点第4位以下切り捨て)となった。今後、新しいマルウェアの通信モデルに対応していくためには、より多くのルールが必要になる。しかし、第7.1節ではSnortが3,387件のルールを用いていたのに対し、本手法は3件のルールしか適用していなかった。このことから、ルールが急速に増えていくことは予想しにくい。平均書き込み、読み込み数が約100倍の160回を目安として考えることができる。

図7.3に測定結果を示す。X軸が1パケットあたりに発生した変数の書き込みおよび読み込み数の平均、Y軸が1秒間に処理したパケット数となっている。評価データでは変数の保持数は平均して0.622になった(小数点第4位以下切り捨て)第6.3.3での分析通り、書き込み、読み込み回数の増加は線形的に処理負荷を増加させていることがわかる。そして、図中では変数の書き込みおよび読み込みの回数が160回を超えても、50,000pps以上の性能を示している。変数の保持数も平均して0.622であり、基準とした測定結果の約20倍となっている。

MAWIで提供されている通信データを参考にとすると、平均パケットデータ長が約700バイトなので50,000ppsはおおよそ280Mbpsに相当すると推定される。インターネット白書2007[76]では1Gbps以上の回線を持つ企業は全体の9.7%であり、その他の100Mbps以下の回線をもつ企業において監視する場合には、十分な性能を発揮していると言える。また、1Gbpsの回線を持ちその帯域をすべて使い切っている企業でも約5台に分割して処理すれば十分に処理できると言える。以上のことから、本システムは実運用ネットワークにおいても、利用可能な性能をもつと言える。

7.4 関連研究との比較

本研究と関連研究を検知精度、通信モデルの変化に対する適応性、追調査の容易さ、処理性能の4点から比較する。比較対象はイベント相関分析、統計的解析手法の他に、BotHunterおよび既存のIDSの4点とする。

既存のIDSは他手法に比べると検知精度が低い。第4.5節で述べた通り、検知精度は亜種に対する耐性も含まれており、単純なパターンマッチによって検知をしている既存のIDSは通信パターンの僅かな変化に対応出来ない。ただし、シグネチャを変更することで通信モデルの変化にも一部は適応できる。また、どのようなネットワークトラフィックによって検知が

表 7.9: 関連研究との比較

	検知精度	適応性	追調査が容易	処理性能
本手法	○	○	○	△
既存の IDS	×	△	△	○
統計的解析手法	○	×	×	-
イベント相関分析	△	△	○	△
BotHunter	○	△	○	△

おこなわれたかを記録しているため、検知結果の原因を調べる比較的容易だが、さらに状況を詳しく知るためには前後のトラフィック調査するなどの手間がかかるため、追調査も容易であるとは言い難い。処理性能については多くの研究で議論されており、それらのアルゴリズムを用いることで性能の向上は容易である。

統計的解析手法は、種類によって大幅に処理内容が異なるため、処理性能については言及しない。検知精度については特定の通信モデルに対してのみ高い検知精度を発揮する。しかし、ほとんどが特定の通信モデルを基準に考案された手法であるため、通信モデルの変更に対応するのは非常に難しい。さらに、複雑な演算をして得られた結果は、どのようなネットワークトラフィックが原因となったかを追跡するのは多くの手間を必要とする。このような理由から、総じて実運用が難しい手法であると言える。

多くのイベント相関分析はイベントの発生順序を分析して、インシデントを検出する手法である。そのため、複数の通信にまたがる断片的な手がかりを集めて、検知精度を向上させることは可能であり、既存の IDS よりは精度が高くなる手法である。ただし、イベント相関分析もマルウェアの微細な変更に対する耐性が低く、イベントの発生順序が変わったり、部分的に通信内容が変化すると見逃しが発生してしまうため、検知精度が十分とは言えない。ただし、検知に用いる相関分析のルールは自由に変更出来る場合が多いため、新しいマルウェアの通信モデルにもある程度は対応できる。また、イベント相関分析は検知に複数の IDS の検知結果を利用し、検知ルールも明確に示されている場合が多いので、どのような過程で検知に至ったのかを知ることが容易である。しかし、IDS の検知結果を用いるため、既存の IDS の処理に加えてイベント相関を分析する処理が発生し、既存の IDS に比べると負荷が高くなると言える。

BotHunter はイベント相関分析を応用した手法であり、高い検知精度と追調査の容易さを実現している。BotHunter はマルウェアの活動に関する複数の状態と各状態に関連する IDS の検知結果のセットを定義し、状態遷移を満たすような IDS の検知結果が発生した場合に、マルウェアを検知したとみなす。これは、マルウェアの亜種の通信パターンや活動パターンのわずかな変化に対応するための手法であり、能動攻撃型マルウェアに対しては強い耐性をもつ。また、この手法も検知結果に対してどのような状態遷移が発生したかを示すことができ、追

調査が容易であると言える。しかし、通信モデルが変化したボットに対応する適応性が十分ではないと考えられる。BotHunter はホストの挙動を俯瞰することによってマルウェアの活動を発見することについては優れているが、細かい通信の相関関係を分析するのには適していない。これは本論文で示した受動攻撃型マルウェアへの対応には致命的な欠点であると言える。受動攻撃型マルウェアは、個々の通信を調べただけでマルウェアの通信であることを見ぬくのは難しい。そのため、個々の通信のつながりを細かく分析する必要があるが、BotHunter はそのような動作に対応していない。そのため、新しい通信モデルをもつマルウェアへの対応は十分であるとは言い難い。また、処理性能についてはイベント相関分析と同様で、既存の IDS として Snort をイベントの収集エンジンとして用いているため、既存の IDS 以上の負荷がかかることは自明である。

本手法は第 7.1 節、第 7.2 節で示したように検知精度と適応性は十分であると言える。検知においては能動攻撃型、受動攻撃型の両方で 98% 以上の検知に成功しており、新しい通信モデルのマルウェアにも適応できている。また、イベント相関分析や BotHunter と同様に検知の過程を示すことができるため、追調査も容易であると言える。ただし、第 7.3 節で実運用ネットワークでも動作することを示したが、既存の IDS と比べると処理の負荷が大きくなるのは避けられないという点のみが問題になる。

7.5 考察

7.5.1 能動攻撃型マルウェアの検知

第 7.1 節の評価において、本手法は従来の IDS の代表的な実装である Snort と比較して False Negative 発生率が低く、False Positive の発生もないことから、従来の手法と比較してボットの検知精度が高いと考えられる。能動攻撃型マルウェアの評価に用いた配布されている Snort のシグネチャは Sourcefire VRT Certified Rules が 1 週間に約 1 回、Emerging Threats が毎日更新されており、評価時点での網羅性において最新の状態にあったと言える。また、ボットの収集期間が 2008 年 6 月から同年 10 月であり、通信データの収集が同年 11 月 1 日から同月 6 日までの間であるのに対し、本評価で使用したルールは 2008 年 11 月 27 日版となっている。よって、少なくとも 20 日以上が経過した時点でのルールを使用しており、実際のネットワーク監視運用と同様の条件であると言える。このことから、既存の IDS では構造上の限界があることがわかる。そして、本手法は亜種に対して強い耐性を持つことも明らかになった。

さらに、抽象的かつ拡張可能なルールを用いて検知できる点も有用性が高いと言える。本論文での評価において、本手法は 3 種類のルールを使用したのに対し、Snort は 3,387 件のルールを使用している。記述方法の違いや検知対象の違いがあるため単純なルール数の比較は難しいが、IDS やウィルス対策ソフトはシグネチャの肥大化が問題となっている。本手法では限られたルールで複数種類のボットを検知できるため、頻繁なルール更新の必要性が低いと

いう点で優れている。ただし、本手法は作成されたモデルによっては具体的な攻撃や活動の内容を把握しにくいいため、IDS と併用することでネットワークフォレンジックなどの調査が容易になると考えられる。

本実装は活動パターンを XML によって記述しているため、モデルの追加、更新、削除の負担が少なく、拡張性が高い。本評価で用いたモデルは同系統のボット亜種には有効だが、全く新しい通信方式を採用したボットには対応できない。しかし、モデルの作成や拡張が容易になっているため、モデルが明らかになれば早急な対応が可能だと言える。

7.5.2 受動攻撃型マルウェアの検知

受動感染型マルウェアの通信データを対象とした検知結果と実運用ネットワークの通信データを用いた誤検知率の評価から、受動感染型マルウェアの通信モデルを利用したマルウェア検知は有効性が高い。受動感染型マルウェアの検知では4種類の検知結果を設定するだけで、99%以上のマルウェア通信データを検知できた。受動攻撃型マルウェアの検知でもルール作成に利用した通信データセット D'_p と検知精度評価に用いた通信データセット D'_q は取得の間に5ヶ月あまりの期間がある。一方、比較に用いた Snort のルールは2010年5月のものであり、データ収集から1年以上経過しているものを利用している。このことから、シグネチャ型のネットワーク侵入検知やホスト上で動作するウィルス対策製品のシグネチャは、構造上の限界がある事が明らかになった。本手法もマルウェア作成者が作成、配布する新しいマルウェアの動向を見て遷移条件の調整が必要になる。しかし、少ないルールで多くのマルウェアに対応できる手法は、今後のマルウェア対策において重要な意味を持つ。

また、受動攻撃型マルウェアを用いた精度評価では、ルールを作成した D'_p では出現していなかった系列のマルウェアが D'_q では出現している。 D'_q で新たに出現したマルウェアの一覧を表7.10に示す。これはルール作成時に観測したマルウェアと評価データに出現したマルウェアとの系統の距離があることを示している。これは、新しいマルウェアの作成時にも過去のマルウェアのモジュールなどを再利用していることなどから、新しいマルウェアにも対応できていると考えられる。

本論文で示した通信の状態遷移モデルによるマルウェア検知手法では、各状態を判断するための手法が重要な要素となる。表7.2で示した検知用のルールでは攻撃コードを含む JavaScript のパターンを示しているが、攻撃コードは様々な種類があるため、表7.2で示したルールで全ての受動感染型の攻撃を網羅しているとは言い難い。一方で、能動感染型マルウェアを検知する BotHunter では NetBIOS や DCE/RPC への攻撃を汎用的に検知するために、Statistical Payload Anomaly Detection Engine という攻撃コード検知エンジンを実装している。新たに出現する受動感染型に対応するためにも、攻撃コードを含んだ Web ページを汎用的に検知する手法だけではなく、マルウェアの転送、管理、活動を検知するための手法を提案、実現していく必要がある。

表 7.10: D'_p には出現せず D'_q において新たに出現したマルウェア一覧

Backdoor.Win32.Bredolab.l
Backdoor.Win32.PcClient.kmi
Backdoor.Win32.Small.*
Backdoor.Win32.Zdoogu.*
Packed.Win32.Krap.*
Packed.Win32.Tdss.*
Trojan-Downloader.Win32.Geral.*
Trojan-Downloader.Win32.Murlo.*
Trojan-Dropper.Win32.BHO.*
Trojan-Dropper.Win32.HDrop.b
Trojan-GameThief.Win32.Nilage.eym
Trojan-PSW.Win32.QQPass.ghy
Trojan-PSW.Win32.WebMoner.ev
Trojan-Spy.Win32.Amber.au
Trojan.Win32.Agent2.*
Trojan.Win32.AntiAV.*
Trojan.Win32.Patcher.ef
Trojan.Win32.Rabbit.ac
Worm.Win32.Bezopi.ni

本論文で調査した受動感染型マルウェアは、Internet Explorer の脆弱性を利用したもののみとなっている。受動感染型マルウェアは Web ブラウザの脆弱性を利用した感染だけではなく、画像や動画の再生ライブラリや再生ソフトウェアの脆弱性を利用したものや、メールによって送付されるマルウェア、ソーシャルエンジニアリングの手法を用いてユーザ自身に実行させるマルウェアなどが挙げられる。Cooke らの活動 [34] や各ウィルス対策ソフトベンダでは筆者ら以上に多くの受動感染型マルウェアを分析していると考えられる。このような機関と協力し、今後はさらに多様な受動感染型マルウェアの分析を継続し、受動感染型の種別による活動形態の違いや通信の傾向などを明らかにしていく必要がある。

7.5.3 評価に用いたマルウェアの網羅性

検知精度評価に用いた通信データに関連するマルウェアから、評価に用いたルールの網羅性について議論する。付録 A では D_c を観測する際に最初に感染させたマルウェアと、 D'_p ,

表 7.11: 2008 年カスペルスキー社のセキュリティ対策ソフトによって検知されたマルウェアの種別 ([1] より抜粋)

マルウェアの系統	検知数 (*1)	評価用データに出現
Agent	32,919	O
AutoRun	49,753	O
Black	19,586	O
Krap	27,575	O
Obfuscated	15,795	O
Sality	29,804	O
VB	36,067	O
Alman	16,799	X
Brontok	13,142	X

*1 検知数はオブジェクトが検知されたコンピュータの数

D'_q の HTTP 通信中に出現したマルウェアを Kaspersky Internet Security 2010 で検査した結果を掲載している。定義データベースは 2010 年 6 月 2 日 20 時 04 分の最新版を適用した。また表 7.11 では、Kaspersky Internet Security が各ホスト上で検知したマルウェアを報告した結果を Kaspersky 社の報告書 [1] から抜粋している。これによると 2008 年に検出されたマルウェアは 6,394,359 件であり、そのうち検知数の多かったマルウェアの上位 20 種類から表を作成している。カスペルスキー社によると同社のセキュリティ対策ソフトは世界 60ヶ国で使用されている。マルウェアの活動は地域によって異なるという報告もあるが、同ソフトウェアは多くの国で利用されているため、偏りが少ないと考えられる。また、他の文献では攻撃数の観測によるマルウェアの活動報告も見られるが、攻撃数はマルウェアごとの攻撃回数や攻撃対象選択のアルゴリズムによって大きく変化するため、個々のマルウェアの増減を推計するのは利用できるが、全体的な分布を調べるには適していない。しかし、カスペルスキー社の報告では感染したマルウェアを対象としているため、実際のマルウェアの勢力分布に近い結果になっていると考えられる。

本論文における評価用通信データに出現したマルウェアは、カスペルスキー社のセキュリティ対策ソフトが最も多く検出したマルウェアで、第 2.5 節で述べた本研究が対象とするマルウェアのうち 80.098% (小数点第 4 位以下切り捨て) に相当している。表 7.11 では評価用通信データに該当する系統のマルウェアが出現したかを示している。この結果、評価用データに出現しなかった系統のマルウェアは Alman と Brontok の 2 種であり、全体の検知数合計から評価用データに出現したマルウェア系統の検知数合計の割合を見たところ、80.098%と

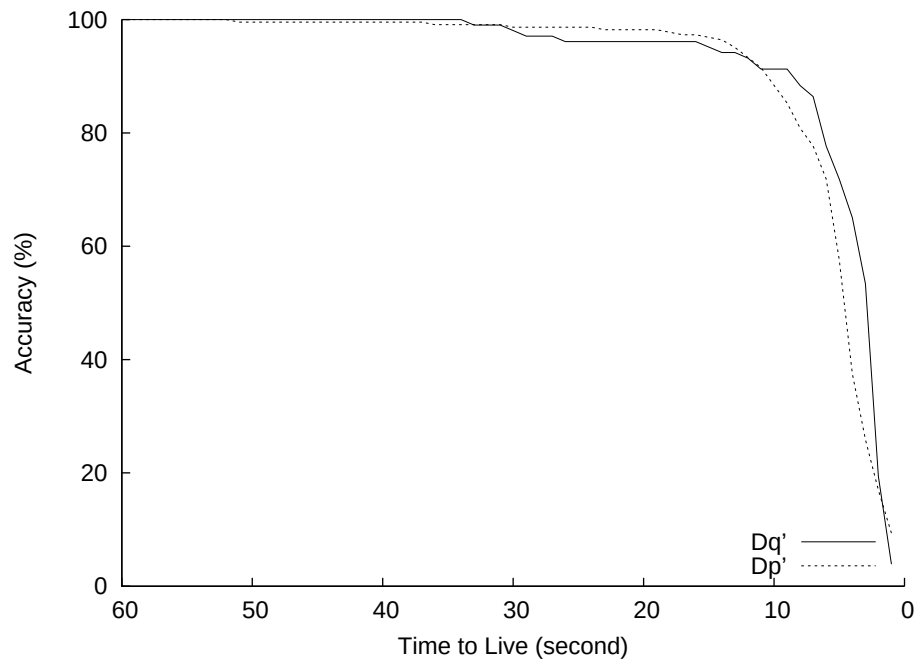


図 7.4: v_{15} パラメータ持続時間設定の変更に伴う検知率の変化

なった。このことから、本論文において使用した評価用通信データでは一定の網羅率があると言える。

7.5.4 変数の保持時間設定の変更に伴う検知精度の変化

本論文での有効性の検証では合計 386 種類の受動感染型マルウェアによる通信データと、1 つの実運用ネットワークの通信データのみを対象としている。受動感染型マルウェアの通信データは報告されているマルウェアの全数に対して非常に少なく、実運用ネットワークの通信データもネットワーク毎の傾向を網羅しているとは言い難い。さらに、各ルールの設定値の妥当性を検証するためにもより多くの通信データを検証する必要がある。図 7.4 は v_{15} のパラメータの持続時間を変化させた場合において、 D_p' 、 D_q' に対する A_3 アラート発生率の変化を示したものである。X 軸が秒単位の v_{15} の持続時間を、Y 軸が持続時間 60 秒における検知数を 100% とした場合の検知数割合を表している。両者とも、持続時間が 40 秒以上の場合にはほぼ 100% を検知しているが、実際には持続時間 52 秒の時点で D_p' に含まれる 1 つの通信データの検知に失敗している。このようにサーバ側あるいはクライアント側の処理状況やネットワークの回線状況などによって遅延が発生する可能性があるため、持続時間は長めに設定するのが望ましい。しかし持続時間を長時間に設定してしまうと、動的な IP アドレス設定などによって異なるホストが同一の IP アドレスを使うような状況が発生し、False Positive の原因となる。今回の False Positive 評価の通信データでは v_{15} の持続時間を無制限にしても

False Positive は発生しなかったが、実際の運用環境では記憶領域の圧迫も懸念される。よって、今後はより多様な通信データに対して評価を実施するとともに、ルールに関する設定値の調整を進める必要がある。

7.5.5 運用コストの低減

本論文で提案した手法によって、ネットワークインシデント検知に関する運用のコストが低減したと言える。ネットワークインシデント検知におけるコストは、検知をする際に運用者が担う作業量と検知精度に依存する。

監視システムによって検知をする場合でも、運用する人物がなんらかの作業を必要とする場合には負荷が高まってしまう場合がある。具体的な負担の例としては、異常検知型監視システムを用いた場合のチューニングや追調査が挙げられる。多くの異常検知システムではネットワークやホスト毎に特性があるため、ネットワーク毎に設定値などを調整しなければならない。このような設定をするためには、ネットワークや計算機科学に関するかなり詳細な知識とともに、運用ネットワークの細かい傾向を把握している必要があり、通常のネットワーク管理者には負担が大きい。さらにこのようなシステムの場合、異常検知型システムが発行したアラートが本当に攻撃か否かの判定をするのは容易ではない。異常検知のアルゴリズムなどを把握し、ネットワークトラフィックの再現などが必要になるため、総じて運用コストが高くなってしまう。

また、出力されるイベントの有効性も運用コストに大きく影響する。顕著な例としては検知精度の低下が挙げられる。False positive が増えてしまった場合、どのアラートにリスクがあるのかを判断する必要がある。この作業は日々の運用において発生する負担となる。一方、False negative が増加した場合はネットワークインシデントを見逃して被害を拡大させる可能性があるため、複数の監視システムによって運用するなどの対策が必要だが、これについても同様に運用負担を上げる要因となってしまう。さらに、記述力の問題から攻撃の成否がわからないイベントを検知するシステムも運用負担を押し上げる原因となる。従来の IDS では攻撃を正しく検知できたとしても、その成否を判断するための機能が十分ではないイベントを出力する場合がある。これらは明示的にリスクがなかったとしても、インシデントを検知するために必要となるイベントではあるが、運用者にそのまま負担がかかってしまうため、イベントを無視してインシデントを見逃すなどの問題へとつながる。

本論文で提案したネットワークセキュリティ監視システムではこれらの課題を解決しており、従来の IDS などと比べて運用者の負担軽減が期待される。本システムはマルウェアの挙動を示すルールに基づいた検知手法であり、ネットワークごとのルールのチューニングは必要にならないと考えている。従来のシグネチャ型 IDS ではルールベースであるにもかかわらず、ルールのチューニングをしなければならない場面が見られたが、これはルールとして表現できるイベントが限られていたため、リスクにばらつきがあるイベントをアラートとして出

力して管理者に判断を促すような運用形態だったためである。本手法はルール表現の幅が広く、マルウェアの活動を的確に判断できるため、チューニングなどに関する負担は少ない。またマルウェアの検知精度についても、False positive, False negative が共に従来の IDS 以上となっており、運用者の負担を増加させることはない。ルールについても運用者本人が記述するのではなく、専門家や研究者が記述したルールを受け取って利用するモデルのため、ルールの自動更新機能を付与することで運用負担は最小限に抑えられる。よって、本システムの導入に関するコストは従来のネットワークセキュリティ監視システムと同様ではあるが、継続的な運用に関するコストは従来のシステムより少なくなると見込まれる。

7.6 本章のまとめ

本研究ではマルウェア対策のために新たに考案したネットワークセキュリティ監視手法を、検知精度、マルウェアの通信モデル変化への適応性、追調査の容易さ、処理性能の4点から評価した。

検知精度は能動攻撃型マルウェアと受動攻撃型マルウェアの通信データをそれぞれ用意し、それぞれの検知精度を評価した。ルール作成と検知精度評価にはそれぞれ異なる情報源を利用し、検知性能の一般性を重視した。能動攻撃型マルウェアは98.3%、受動攻撃型マルウェアは99.26%のマルウェアを検知できた。これにより、本手法はマルウェア亜種への耐性が高い手法であるとわかった。さらに、通信モデルの異なる能動攻撃型マルウェアと受動攻撃型マルウェアの双方に対応しており、マルウェアの通信モデル変化への適応性も高いと考えられる。さらに、評価に用いたルールを実運用ネットワークのトラフィックの監視に適用し、False positive が発生しないことを確認した。監視中に28件のマルウェアの活動を検知したが、これらはハニーポットによる通信であった。この結果から、本手法はFalse positive も少なく、正常に動作していることが確認できる。

処理性能は一般に公開されている通信データを利用し、ルールの変化が処理にどのような影響をあたえるかを示した。本手法はルールの書き方や到着するネットワークトラフィックの傾向によって性能への影響が変化するため、恒久的な評価結果を示すのは難しい。しかし、評価用データを実運用ネットワークで動作させた際に発生する、変数の処理数や保持数を参考に、処理回数が増加しても急速に性能が低下しないことを示した。そして、広帯域な回線を持つネットワークでも現実的な構成で処理が可能であることを示した。

また、本手法を関連研究と比較した。特に検知精度だけではなく、追調査の容易さとマルウェアの通信モデル変化に対応している点から、他手法と比べても優位性があり、実運用ネットワークにおいて容易に運用ができると考える。

以上の評価結果から、本手法なマルウェアを監視する際のネットワークセキュリティ監視がもつ課題を解決しており、新しい有効なマルウェア対策手法になったと言える。

第8章 結論

8.1 本論文のまとめ

本論文では被害の拡大を続けるマルウェアに対して、新しいネットワークセキュリティ監視手法を提案し、実装した。

不正な活動をするマルウェアは計算機が登場した黎明期より存在していたが、コンピュータネットワークや家庭用コンピュータが急速に普及したことにより、犯罪に利用されるようになった。多くのユーザがネットワーク上で提供されているサービスを使い始めたことから、ユーザがあつかう情報資産や計算機・ネットワーク資源の価値が高まった。悪意ある人物たちは、マルウェアに感染させた不正に操作できるホストを集め、ボットネットを形成した。ボットネットは迷惑メールの送信、サービス妨害攻撃、情報資産の盗難等、様々な社会的被害を与える一方で、悪意ある人物たちに経済的利益を与える非合法市場を形成した。これによって、マルウェアによる攻撃や活動の効率化、巧妙化が加速した。

悪意ある人物たちはボットネットを用いた利益取得を安定させるために、既存のマルウェア対策手法を回避、無効化するように戦略を変更している。マルウェア対策として最も普及している方法は各ホストで動作するウィルス対策ソフトだが、この対策ソフトは個々のウィルスについての識別情報を保有し、識別情報に一致したファイルをマルウェアとみなす。そのため、False Positive を起こしにくいという利点はあるものの、新しいマルウェアに対応しきれないという問題点がある。悪意ある人物たちはマルウェアの生成を自動化し、マルウェアが持つ機能や難読化のパターンをわずかに変更した亜種を高速に生成している。これは1つの種類のマルウェアでも一日あたり300種類以上の亜種が発生しており、ウィルス対策ソフトでは対応しきれないのが現状である。また、ボットネットを拡大させるための攻撃手法も変化している。これまでは感染元となるホストから攻撃コードを送るための通信を開始していたため、ファイヤウォールを用いることによって攻撃コードの送信を防ぐことができた。しかし、Webサイトに悪意のあるコードが埋め込まれるという事例が多発し、Webブラウザやブラウザのプラグインなどの脆弱性を利用した攻撃も増加している。このような通信はファイヤウォールで防ぐのが難しく、他にも効果的な対策はない。この他の対策でもマルウェアの急速な亜種の発生や、攻撃手法の変化への対応が十分ではなく、マルウェアによる被害はますます深刻なものになりつつある。

本論文ではマルウェアの対策を考案するに当たり、マルウェアの収集と解析を実施した。マルウェアの対策をするためには、マルウェアの詳細な活動記録やマルウェア本体の情報が必

要になる。しかし、これらの情報は悪用可能なものが多く危険性が高いため、一般には公開されていない。そのため、本研究では独自に収集と解析を実施する環境を構築し、解析結果を得た。これまで、マルウェアの収集や解析は大規模なテストベッドや専用のネットワークで実施されていたが、本研究では小規模ながら必要に応じて拡張できる解析環境を構築し、新しい感染経路によるマルウェアの収集にも取り組んだ。従来の環境では感染元ホストから通信を開始する種類のマルウェアのみ収集していたが、本解析環境では Web サイトの閲覧を契機に感染するマルウェアの収集と解析も実施した。これらの収集されたマルウェアを解析した結果、両者に大きな違いがあることが明らかになった。前者の能動攻撃型マルウェアは、感染後に他のホストに感染を拡大させるため、大量の攻撃用通信を開始させる。しかし、後者の受動攻撃型マルウェアは感染後も大量の通信を発生させることなく、その振る舞いは一般のユーザのそれに近い。これまで、マルウェアの対策には能動攻撃型マルウェアが大量に攻撃用の通信を開始する点に着目されてきたため、受動攻撃型マルウェアの対策に転用するのが難しいと言える。よって、調査結果からは攻撃経路の変化だけではなく、根本的な対策の見直しが必要であることが明らかになった。

本論文では多くのマルウェア感染ホストを効率的に検知、駆除するために、マルウェアの活動を検知するネットワークセキュリティ監視に着目した。マルウェア対策におけるネットワークセキュリティ監視では、検知が他のセキュリティ対策手法と比べてユーザへの干渉が比較的少なく、利便性の低下が少ない。また、感染対象となるホストから独立して監視ができるため、ホストがマルウェアに感染した場合でも監視に必要となる情報の完全性に影響が少なくなる。さらに、一斉に多数のホストを監視できる利点がある。ネットワークトラフィックの監視によってマルウェアの感染ホストを一斉に監視し、検知、駆除することによって、ボットネットを構成するマルウェア感染ホストを減少させることができる。これによって、マルウェアを用いた犯罪の減少へ貢献することができる。

本論文では、マルウェア対策にネットワークセキュリティ監視を用いる際、検知精度、適応性、追調査の容易さ、処理性能の4点が課題となることをまとめた。1点目はマルウェアの通信を的確に検知するための、精度に関する問題である。ネットワークセキュリティ監視ではホスト上でマルウェアの活動を監視するのに比べ、検知に利用できる情報が少ないという欠点があり、検知精度が低下しやすい。検知精度が低下することによって、マルウェアの活動を見逃してしまう False negative や、マルウェアとは関係しない通信を誤検知してしまう False positive などの問題がある。これらが多発するとネットワークセキュリティ監視の負担が増えて、効果が低下するという問題がおきる。2点目は異なる通信モデルを持つマルウェアが出現した場合、同じ手法で対応できるか、という問題である。マルウェアは時々刻々と新しい亜種が発生するだけでなく、C&C サーバや他の感染ホストとの通信プロトコルやアーキテクチャを変更したものが新しく出現している。ネットワークセキュリティ監視による既存のマルウェア対策でも、特定の通信モデルに依存した手法を用いているものが多く、これらは新しい通信モデルの出現によって著しく効果が下がってしまうという問題がある。3点

目は検知結果がどのような活動を検知したのかが明確に分かることと、どのような経緯で検知したのかを追調査できる容易さである。検知結果が出力された場合、対象のホストを調査、回復しなければならないが、その際にマルウェアのどのような活動が検知されたかが示されることで、調査、回復の事後対応の手がかりとなる。従来手法において統計的にマルウェアの活動を判定する場合などは、何が契機となってマルウェアと判断したかを見極めるのが難しいという問題がある。4点目の課題は、ネットワーク全体を一括して監視できるだけの処理性能の高さである。ホスト上での監視と比べると扱う情報の種類は少ないが、多くのホストを一括して監視する場合には、監視するホスト数や通信料に比例して処理の負担が増加する。そのため、実運用環境で用いるためには、一定以上の処理性能と分散可能な処理システムが必要になる。

本論文では効果的なマルウェア対策を実現するために、各ホストの通信を俯瞰しマルウェアの挙動を捉えることによって、変化するマルウェアの形態と頻繁な亜種の発生に対応するネットワークセキュリティ監視手法を提案、実装した。既存対策を回避のためハードはマルウェアの細部を変更した亜種を頻繁に生成、配布するが、マルウェアの挙動は共通する部分が多い点に着目した。ハードは新しいマルウェアを生成する際、既存のプログラムコードを再利用する傾向がある。そのため新しく生成されたマルウェアでも、過去のマルウェアと共通する動作が多く見られ、これを利用することで新しく生成されたマルウェアも高精度に検知できると見込まれる。ただし、新しく生成されたマルウェアは部分的な通信内容や、通信の手順などが変化している可能性がある。よって、本手法ではマルウェアの全体的な挙動を監視し、細部の変更や通信の発生順序の変化を吸収できる監視システムを構築することによって、亜種に対する耐性を向上させることに成功した。

本研究では、マルウェアによる通信を俯瞰することによって、通信内容や活動の順序、通信の宛先などが変化したとしても検知結果に影響されないマクロな相関関係と、独自に定義されている通信プロトコルの追跡などができるミクロな相関関係の両者を利用した手法を提案している。従来の研究ではマクロな相関関係とミクロな相関関係を別々に利用していたが、本研究ではこの2つを組み合わせることに着目し、より柔軟な検知手法を実現している。これは、マルウェアの挙動モデルをテンプレートとして示すルールを用意し、それに基づいてホストの挙動を追跡するネットワークセキュリティ監視システムとして実装している。本実装ではマルウェアによる通信を監視する上で重要になると考えられる通信の相関関係を、既存研究や既存の実装を参考にして取り上げ、それらを包含できるルール記法を実現した。

本手法によって、既存のネットワークセキュリティ監視手法がマルウェアを検知する際の問題点が克服された。まず、マクロな相関関係とミクロな相関関係を自由に組み合わせられることで、検知精度の向上が見込まれる。従来のネットワークセキュリティ監視による問題の要因として個々の通信内容に着目した検知エンジンを使用しているという点があった。経済的利益を得るために作成されたマルウェアはより狡猾化しており、通信の一部を切りだしてマルウェアによる通信であるか否かを判断するのが難しくなっている。一方でマルウェア

の活動には様々な種類があり、活動の順序が組み変わったり、活動の内容が入れ替わるなどの変化についても対応できなければならない。本手法では、ミクロな相関関係を用いることで通信の流れを把握し、マルウェアの通信における要求や応答などを解釈する。これは、マルウェアも一定のプロトコルに基づいて他のホストと通信していることが明らかになっており、このようなプロトコルの体系は急速に変化しにくいという特徴がある。これに着目することによって、マルウェアによる通信を検知しやすくなる。さらにマクロな相関関係を用いることで、マルウェアの活動内容の順序や内容の入れ替わりに対応することができる。マルウェアの活動はいくつかに分類ができるものの、発生順序についてはC&Cサーバからの命令などによって変化する。これはマルウェアの各活動をイベントとしてあらかじめ定義し、その発生順序や発生における関係性を自由に記述できることによって、順序の変更などに容易に対応できるようになる。そして本手法では、この2つの相関関係を分離せずに1つのルールとして記述できるようになっている。そのため、より柔軟性の高いルールが記述可能である。このような点から、新しい通信モデルを持ったマルウェアが出現した場合でも、本手法に寄って通信モデルを定義し、検知に対応できる可能性が高い。本手法で表現できる通信モデルは、従来のルール型ネットワークセキュリティ監視手法で表現できるものより多様化されている。これは一般的な通信プロトコルのモデル化にも利用できる構造となっており、マルウェアが新しい通信プロトコルを利用している場合でも、通信に相関関係がある場合は対応できると見込まれる。また本手法では、ルールによって記述された通信のモデルに基づいてネットワークを監視し、検知に関連したネットワークトラフィックを記録し、後から参照することができる。本手法で記述するルールは、複雑な統計処理などによるものではなく、それぞれがマルウェアの挙動を示すものとなっている。よって、全てのイベントには意味があり、これをネットワークを監視している運用者が容易に解釈できるという利点がある。実際のネットワークセキュリティ監視の運用においては、発行された警告が誤検知ではないかを検証するだけでなく、被害の波及などを検証する場合にも、どのような過程によってその警告が発行されたかを把握できることが重要となる。統計的解析を用いたネットワークセキュリティ監視手法でも警告発行に至る経緯を追跡する必要がある。過去のログなどから再計算する過程は管理者にとって大きな負担となる。そのため、検知に必要なルールを説明可能な本手法は、実運用においても優位性があると言える。さらに、本手法では処理の負担を考慮した構成となっており、ネットワークの上流地点で監視ができるように設計、実装されている。従来、通信の相関関係を利用した検知手法では、オートマトンによる状態遷移が利用されてきたが、これは状態数の増加や遷移構造の複雑化によって状態数が爆発するなどの問題がある。そのため、本手法ではマルウェアの挙動を示すルールに変数を代入することに寄って、状態数の爆発を防ぎ、かつ一定の処理性能をたもった実装を実現できた。

本論文では提案した手法を実装し、検知精度と処理性能、そして関連研究の比較の観点から評価を実施した。検知精度の評価では検知の見逃しである False negative と、誤って検知してしまう False positive の発生率を評価した。False negative の評価では、ネットワークサービ

スに直接攻撃することで感染する能動攻撃型マルウェアと Web サイトの閲覧を契機として感染する受動攻撃型マルウェアの両者を対象にそれぞれ検知ルールを作成し、収集した能動攻撃型マルウェアと受動攻撃型マルウェアの通信データを用いて評価を実施した。能動攻撃型マルウェア用のルール作成には異なる研究機関の調査結果を利用し、受動攻撃型マルウェア用のルール作成には評価用の通信データとは異なる情報源から得られた通信データを利用した。これによって検知精度を評価したところ、False negative は全体の 2% 未満であった。この結果は本手法の検知精度が十分に高いことを示すと同時に、異なる通信モデルのマルウェアに対応できることを示した。同様の環境で既存のネットワークセキュリティ監視手法である IDS の Snort に 3,000 種類以上のルールを適用し、各マルウェアの通信データの検知率を調査したところ、能動攻撃型マルウェアの通信データでは 26% 以上、受動攻撃型マルウェアの通信データでは 30% 以上の False negative が発生した。本手法と Snort ではルールの記述体系が異なるためルール数の差を論じるのは難しいが、本手法で用いたルールは 4 種類であり、記述量も Snort に適用したルールにくらべ少ない。また、実運用ネットワークの通信データを用いて False positive の発生率を調査したところ、誤検知は発生しなかった。このことから、本手法は従来手法に比べて検知精度が実現できたと言える。さらに本手法の実装を解析し、性能に影響が出る要素を調べた上で、性能評価を実施した。これによって、多くのユーザがいる環境でも実運用ネットワークを処理できることを示した。そして関連研究との比較により、検知結果の追調査が容易である点を示した。これらの評価によって、本手法が実用可能であり、マルウェアを対象としたネットワークトラフィック監視手法としての優位性があることを示した。

8.2 今後の課題

本節では、今後マルウェアの対策に取り組んでいく中で、着目すべき点や考慮すべき点について述べる。これは本研究の発展だけではなく、より広い範囲でのマルウェア対策に関して言及する。

8.2.1 本研究の発展

より多くのマルウェアを用いた検証

本論文での提案手法では 2 種類の通信モデルをもつマルウェアをそれぞれ 98% 以上の精度で検知している。しかし、現在はマルウェアの発生頻度が高く、感染経路やマルウェアの種類も多様化している。そのため、本手法の有効性は今後も継続的に検証を続けなければならない。

検知精度の検証をする際には、亜種への対応と新しい通信モデルへの対応の 2 つの観点が必要となる。通信モデルから作成したルールがどの程度の亜種耐性を持つかを検証しつつ、通

信モデルに沿わない亜種が発生した場合にはモデルの拡張およびルールの変更が必要となる。さらに、新しい通信モデルが出現した場合、本手法の適応性を検証する必要もある。

マルウェア通信モデル作成の自動化

本手法ではマルウェア感染の解析結果に基づき、手動で作成したルールを使用している。マルウェア作成者は既存の通信モデルを持つ亜種を作成するだけではなく、新しい通信モデルのマルウェアも開発している。そのため、新しく収集したマルウェアが現行のルールで検知ができるかどうかだけではなく、新しく出現したマルウェア通信モデルを自動的に分析し、ルール化するアプローチも必要となる。新しいモデルの出現から対応にかかる時間を短縮することで、持続的にマルウェアに対応していけると考えられる。

IP デュアルスタック環境への対応

本手法はホストの識別を IP アドレスに依存しているため、IPv4 と IPv6 のデュアルスタック環境になった場合に、不都合が生じる可能性がある。デュアルスタック環境では IPv4 と IPv6 のどちらで通信が発生するかは状況によって変化する。そのため、2つのイベントを見つける必要がある場合、一方が IPv4 を用いた通信で他方が IPv6 を用いた通信上で発生すると、2つのイベントの相関関係を示すのが困難になってしまう。結果として、検知精度に大きな影響を及ぼすと見られる。さらに、IPv6 では1つのインターフェースに複数のアドレスが設定される運用が一般的であり、これもホストの識別に影響を与えると見られる。

そのため、何らかの方法で IPv4 アドレスと IPv6 アドレスを関連付ける手段が求められる。監視対象が同一ネットワーク内のホストの場合は MAC アドレスを用いて特定する手法が考えられるが、ネットワークの上流で監視していた場合は OSI7 層モデルのデータリンク層に関する情報は利用できない。そのため、通信の相関関係などに着目し、同一のホストを同定するための仕組みが必要とされる。

マルウェア検知に特化した通信解析モジュール

マルウェアに関連した通信を発見するために、マルウェアに検知に特化した通信解析モジュールを組み込む必要がある。例えば近年はあるアルゴリズムに基づいてドメイン名を決定する機能がマルウェアに組み込まれている [15]。これは通信の相関関係を利用した検知手法でも一部は対応が可能であるが、専用の解析ができるモジュールを本手法に組み込むことによって、より高い精度での検知が期待できる。

8.2.2 マルウェアからの防御

これまで、ホスト上におけるマルウェアからの防御はセキュリティ対策ソフトの利用が一般的であった。セキュリティ対策ソフトは False positive の発生率を極力下げるため、マルウェアのシグネチャを用いた検知を優先してきたが、亜種の大量発生にともない、その有効性は低下している。多くのセキュリティ対策ソフトは新たにヒューリスティック検知、ビヘイビア検知などの手法を取り入れているが、False positive を下げるために十分な防御率、検知率が発揮できていない。そのためホスト上での対策についても、新しい手法の提案と実用化が急務となっている。

新しい防御方法の一例として仮想環境を用いた監視・防御機構の構築が考えられる。計算機性能の向上やソフトウェアの品質改善により、1つの筐体で複数のOSを動作させる仮想化環境が普及しており、これを利用したマルウェアの防御機構が構築できる可能性があると考えられる。OSの仮想化環境では利便性を高めるため、記憶領域やOSの状態復元機能や、仮想計算機の複製機能などがある。さらに、仮想化環境では仮想計算機とは独立した制御機構（ハイパーバイザ）があるため、記憶領域やネットワークへの入出力をある程度まで制御できる。これを利用して、マルウェアの検知、防御をするためのシステムが構築できると考えられる。

8.2.3 ボットネットの追跡

ボットネットを外部から観察したり、ボットネットを乗っ取ることによってボットネットの全容を調査した報告 [44, 15] はこれまでに多く存在したが、今後はこのような調査がますます重要になると考えられる。2006年以前の主なボットネットは、IRCを用いた比較的単純な構成の分散システムとして動作していた。しかし、近年ボットネットの構造は急速に複雑化しており、ボットネットの種類ごとにも大きな違いが見られるようになってきている。

ボットネットの全容を正確に把握するためには、静的解析、動的解析からボットネットの制御構造を明らかにし、接近していかなければならない。これはボットネットの制御構造における脆弱性を利用する必要がある。多くの負担や多彩な技術力が求められる。しかし、このようにして得られたボットネットに関する情報は、その活動形態や感染の拡大状況、ハーダの目的やハーダ自身の情報などを得られる可能性があり、非常に価値が高い。このようなアプローチは普遍的な手法がないため、より一層の負担が見込まれるが、マルウェア対策をする人物として必要な取り組みであると言える。

8.2.4 マルウェアの解析

静的解析と動的解析の連携

マルウェアを解析するための手法は静的解析と動的解析に分類できるが、この2つを効果的に組み合わせた解析手法の確立が求められる。一般的に動的解析は得られる情報が少ない代わりに、解析にかかる時間が少ない。そして、静的解析は解析にかかる時間が長い、得られる情報は多い。これらの互いの利点を活かした連携手法を考案することによって、マルウェアの解析と対策検討にかかる時間が大幅に減少することが期待される。

具体例としては、動的解析で得られた入出力情報を静的解析にフィードバックし、静的解析によって次に解析すべきポイントを特定する、という作業を繰り返す方法が考えらる。動的解析で特に効果的に得られるのはファイルやネットワークに関する入出力の情報である。動的解析からはどこからどのようなデータを取得し、どのようなデータを発信するかを知ることができる。ただし、動的解析からデータが得られた段階ではそれらがどのような意味を持つかがわからない場合が多いため、データの形式や発信したデータに対する応答などにもとづいて推測するしかない。しかし一方で、これらの入出力とマルウェアの動作の相関関係を調べることにより、マルウェアの処理のきっかけとなっている入出力を発見できる可能性がある。これを利用して、静的解析により分析すべきポイントを抽出し、その入出力のあとにある処理の選択肢や、より情報を収集するために必要となる条件を得られると見られるこのような処理を繰り返すことによって、動的解析より詳細な情報を静的解析より素早く入手できると期待される。

より広範囲なマルウェアの収集と解析

本研究ではマルウェアの収集、解析環境を用意し対策に必要な情報を集めたが、今度はより広範囲にマルウェアの収集と解析をおこなっていく必要がある。マルウェアは数が多いだけでなく、企業や大学、行政機関などのネットワーク種別や国や地域に応じて異なるマルウェアの攻撃をうけるという報告もあり、本環境だけでは対象範囲が限定的であると言える。

マルウェアの狡猾化にともない、感染経路もさらに多様化している。本研究では Microsoft Windows のネットワークサービスに直接攻撃コードを送ってくるマルウェアと Internet Explorer による Web 閲覧に関連した脆弱性を利用するマルウェアを対象としたが、他にもサードパーティのプラグインを利用した攻撃や、メールによって攻撃コードを含むファイルを送信する標的型攻撃がある。これらの感染経路も、今後さらに多様化すると見られる。そのため、今後はより多くの感染経路や多様な環境に対応した収集環境が必要になるが、これは収集、解析環境の構築にかかる負担が大きくなると予測される。本論文でも言及した通り、現状ではマルウェアに関する研究に必要な情報は、当該研究者が自身で収集しなければならない。し

たがって、より少ない負担で多様な環境や感染経路に対応する収集環境を構築できる手法やシステムが求められる。

マルウェアに関連する情報の共有システム

マルウェアの研究において障害となるのは、マルウェアに関する情報の多くが公開されておらず、情報の入手に関する敷居が高いことである。多くの研究論文や記事などによってマルウェアに関する情報が一般に公開されているが、詳しい動作に関する情報やマルウェアの検体を収集するのが難しい。これは、一般に向けて動作に関する情報やマルウェアを公開することにより、1) マルウェア作成者に自分たちの対策状況を知らせてしまう、2) 一般ユーザがあやまってマルウェアに感染する可能性がある、3) 一般ユーザがマルウェア検体などを悪用する可能性がある、などの危険性が考えられる。さらに、マルウェアに関する最新の情報は研究データとしての価値が高く、他の企業や研究者に対する公開は、価値を低減させる可能性もある。

これに対応するために、マルウェア関連情報に特化した情報共有システムの構築が必要であると考えられる。これは通常のアクセス制御とは異なり、多数の主体が存在する中で情報の種類や内容に応じたリスク評価機能や、リスク評価に基づいたアクセス制御、時間経過によるアクセスの許可機能、研究機関や研究者などの契約や信頼関係に基づいたアクセス制御が実装され、容易にデータを共有できるシステムが必要となる。このようなシステムの構築により、マルウェアの研究に対する敷居が低減し、より活発な研究が望まれる。

研究の成果を比較するためのマルウェアに関する情報収集

マルウェアの研究において提案された対策が、実際に感染を拡大させているマルウェアに対してどの程度の影響力があるかを、客観的に示す必要がある。本研究では第7.5.3節において、2008年にカスペルスキー社が検知した上位20種のマルウェアに基づいて網羅性を議論したが、より公平かつ広範囲に渡るデータを用いた議論が今後必要になると考えられる。これは本研究に限った問題ではなく、マルウェアに関する研究全般に対して言える。また、マルウェアの研究においては評価用のデータを公開しない、あるいはできない場合が多く、共通するデータを用いて互いの研究を比較検討ができず、追実験や反証が困難であるという問題もある。これは、情報処理学会のマルウェア対策人材育成ワークショップなどにおいて一部を解消するための試みが見られるが、まだ十分な状態であるとは言いがたい。

そのため、各研究のマルウェアに対する影響力を評価するためのフレームワークの構築が、今後のマルウェア研究に必要になると考える。例えば、マルウェアの収集処理をするためのシステムをパッケージ化して様々なネットワークに配布し、グローバルな情報を集約化するシステムの構築などが考えられる。また、各組織で個別に収集した情報を共有する際には、第8.2.4節で示したようなシステムも必要になると考えられる。

8.3 今後の展望

本研究によって、ネットワークセキュリティ監視によるマルウェアの検知精度が向上した。マルウェアは今後も消滅することはないと考えられるが、本手法が確立したことによってマルウェアによる脅威が軽減されたと考える。本手法は新しいマルウェアがでたとしてもそのパターンを解析することによって、検知ルールを追加することができる。そのため、本手法は持続的にマルウェアに対応し、長期的に利用できるシステムであると考えられる。

セキュリティにおける攻撃者と防御者の戦いはイタチごっこであり、それはマルウェアについても同様のことが言えるが、本手法をはじめとする対策技術の研究、開発を進めることは無意味ではない。これらの対策は、マルウェアを用いた犯罪で利益を得ている人物にも新しいマルウェアを開発や配布といったコストを強いることとなり、被害の縮小や防止につながると思われる。今後もこのようなマルウェアと悪意ある人物に対する取り組みを継続することで、多くのユーザの利便性と安全を脅かす悪意ある人物を追い詰め、情報技術やインターネットの成長に貢献できると確信する。

謝辞

本論文は私が大学入学時から取り組んでいた研究活動のまとめであり、ここまでのどり着くまでに本当に多くの方からご助力いただきました。今まで私を指導してくれた方々、支えてくれた方々、応援してくれた方々全てに、この場を借りて感謝の意をお伝えしたいと思います。

慶應義塾大学環境情報学部教授 村井純博士に感謝します。氏の一言に含まれる思いや考えはなかなか一度では理解できず、後からその真意に気づくことも多くありましたが、それでも独特の視点と鋭い切り口からの助言は研究を進めるにあたり大きな助けとなりました。また、研究活動に取り組む過程で多くの貴重な経験を積ませてもらい、研究を進める上で必要となる様々な視野を勉強させてもらいました。ありがとうございました。

慶應義塾大学環境情報学部教授 中村修博士に感謝します。氏にも多くのアドバイスを頂きましたが、特に研究としてまとめる上で大切なことを多く指導していただきました。私の研究の中で、論文の優位性を客観的に議論する際のポイントを指摘していただき、論文をまとめる上での大きな助けとなりました。ありがとうございました。

奈良先端大学大学院教授 山口英博士に感謝します。山口博士からは時として厳しい言葉をいただき、研究にどのように取り組むべきか大いに悩んだのですが、今となってはその言葉があったからこそ、このような形で博士論文をまとめられたのだと思います。ご指摘も情報セキュリティの専門家として本質を見ぬいた非常に鋭いものでした。そのおかげで内容を掘り下げるとともに、研究の可能性を多少なりとも広げられたかと感じております。ありがとうございました。

慶應義塾大学環境情報学部教授 武田圭史博士に感謝します。氏は私が博士課程で研究をする上で、様々な指導をして頂きました。特に情報セキュリティを専門とする方が身近にいて頂けることは大変心強く、博士過程の進め方や博士論文の執筆について度々相談に乗って頂きました。特に研究については専門である侵入検知の観点からの具体的なご意見と同時により広い視点からの議論をしていただき、研究の細かい部分から研究の位置づけなどを考える上で大変参考になりました。また研究の戦略的な進め方についても色々のご指導頂き、遠回りをせずに進むことができたと考えております。ありがとうございました。

また、本論文の執筆にご助言を頂きました、慶應義塾大学環境情報学部教授 萩野達也博士、同学部准教授 楠本博之博士、同学部准教授 植原啓介博士、同学部准教授 三次仁博士、同学部専任講師 重近範行博士、慶應義塾大学大学院政策・メディア研究科特別研究教授 Kilnam Chon 博士、同研究科特別研究准教授 朝枝仁博士、同研究科特別研究講師 吉藤英明博士、同研

究科特別研究講師 Achmad Husni Thamrin 博士，同研究科特別研究講師 羽田久一博士，同研究科特別研究助教 稲葉達也博士，同研究科特別研究助教 渡部陽仁博士，慶應義塾大学大学院メディアデザイン研究科教授 大川恵子博士に感謝します。同大学環境情報学部准教授 Rodney D. Van Meter 博士は研究に関する指導だけではなく，私が苦手とする英語に関してもご助力下さいました。また，同大学大学院政策・メディア研究科特別研究講師 齊藤賢爾博士はルールの記述方法に関する議論をして頂きました。ありがとうございました。

そして，慶應義塾大学大学院政策・メディア研究科特別講師 南政樹氏，北陸先端大学院大学 小原泰弘博士，慶應義塾大学大学院政策・メディア研究科 白畑真氏に感謝をします。南氏，小原博士，白畑氏には学部入学当時から研究についてご指導いただきました。私が大学入学直後に研究室へ所属した頃は情報セキュリティを専門に取り組むグループはなかったのですが，南氏，小原博士，白畑氏に指導してもらい，本論文の研究テーマに取り組むきっかけを作っていただきました。現在は活動の場も変わってしまい，直接指導していただく機会は少なくなりましたが，私が学部，修士課程であったときの氏等の指導がなければこの博士論文はありませんでした。ありがとうございました。

また，慶應義塾大学環境情報学部徳田・村井・楠本・中村・高汐・重近・バンミーター・植原・三次・中澤・武田合同研究プロジェクトのメンバーである鈴木茂哉氏，中根雅文氏，村上陽子博士，佐藤雅明博士，片岡広太郎博士，田崎創氏，岡田耕司氏，工藤紀篤氏，久松剛氏，三島和宏氏，松園和久氏，堀場勝広氏，Achmad Basuki 氏，松谷健史氏，空閑洋平氏，本多倫夫氏，江村桂吾氏，六田佳祐氏，黒宮佑介氏，波多野敏明氏，永山翔太氏，峯木巖氏，中村遼氏，村上滋希氏，小澤みゆき女史に感謝します。そして，同じ研究グループである Internet Security Center の上原雄貴氏，重松邦彦氏，福岡英哲氏，中島明日香女史，梅田昂翔氏，Doan Viet Tung 氏，山本智典氏，吉原大道氏，碓井利宣氏，中井研氏，相見眞男氏，藤原龍氏，吉原洋樹氏，関根冬輝氏，鴻野弘明氏，大矢崇央氏，湯本愛未女史，三ツ木あかね女史，椎野裕太氏，Pham Van Hung 氏，町田裕隆氏，由井卓哉氏に感謝します。また，研究室 OB の土本康生博士，今泉英明博士，三屋光史朗博士，川喜田佑介博士，杉本信太博士，海崎良氏，佐川昭宏氏，遠峰隆史氏，小椋康平氏，中村友一氏，金井瑛氏，奥村佑介氏，尾崎隆亮氏，佐藤龍氏，佐藤泰介氏にも感謝いたします。先輩と後輩の支えがあればこそこの博士論文でした。

村井研究室秘書の渡辺康恵女史，比嘉宏美女史，宇佐美由美子女史，青木りか女史，渋谷雪絵女史に感謝します。出張などの手続きに始まり，博士論文に関する予定調整や書類関係の作業などで大変お世話になりました。ありがとうございました。

米ブリザード・エンターテインメント社に感謝します。御社が発売するゲームはいずれも秀逸なものであり，博論執筆や博論に向けた研究に取り組む中で時に心を癒し，時に勇気づけてくれました。

最後に，大学入学から8年間に渡る研究生活で，私を支え続けてくれた父と母に感謝します。

参考文献

- [1] Eugene Aseev, Alexander Gostev. カスペルスキー セキュリティ情報：2008 年統計, Feb 2009. <http://www.viruslistjp.com/viruses/analysis/?pubid=204792052>.
- [2] Nikkei Business Publications, Inc. ヤフーがクラッカの攻撃を受けて3時間機能停止、イーベイ、バイ・ドット・コムなども相次いで被害, 2000. Feb.
- [3] CERT/CC. CERT Advisory CA-2003-04 :MS-SQL Server worm, Jan 2003.
- [4] Nikkei Business Publications, Inc. Telecom-isac japan が「ボットネット」の実験結果を報告, 2006. Apr.
- [5] 金井 瑛, 水谷 正慶, 武田 圭史, 村井 純. マルウェアの転送ログを利用したボットの活動分析ト活動. 情報処理学会シンポジウム論文集, 第8巻, pp. 75–80, Oct 2008. マルウェア対策研究人材育成ワークショップ 2008.
- [6] Microsoft TechNet. マルウェアの定義：FAQ, 2009. <http://technet.microsoft.com/ja-jp/library/dd362857.aspx>.
- [7] CERT Incident Note IN-2001-09 “Code Red II: Another worm Exploiting Buffer Overflow In IIS Indexing Service DLL”. http://www.cert.org/incident_notes/IN-2001-09.html, Aug 2001.
- [8] CERT/CC. CERT Advisory CA-2003-20 W32/Blaster worm, Aug 2003.
- [9] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The ghost in the browser analysis of web-based malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pp. 4–4, Berkeley, CA, USA, 2007. USENIX Association.
- [10] Nathan Friess and John Aycok. Black market botnets. MIT Spam Conference, 2008.
- [11] 総務省・経済産業省. ボット対策プロジェクト サイバークリーンセンター, 2006. <https://www.ccc.go.jp/>.

- [12] 竹森敬祐, 磯原隆将, 三宅優, 西垣正勝. ボットネットおよびボットコードセットの耐性解析. 情報処理学会シンポジウム論文集, 第 8 巻, pp. 49–54, Oct 2008. マルウェア対策研究人材育成ワークショップ 2008.
- [13] Cyber Clean Center. 2008 年 08 月度 サイバークリーンセンター活動実績, Aug 2008. <https://www.ccc.go.jp/report/200808/0808monthly.html>.
- [14] 金井瑛. 通信の類似性に着目したネットワークインシデント検知手法. Master's thesis, 慶應義塾大学大学院, Mar. 2009.
- [15] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. Technical report, Apr 2009.
- [16] Nepenthes - finest collection -. <http://nepenthes.mwcollect.org/>.
- [17] Sun Microsystems, Inc. VirtualBox. <http://www.virtualbox.org/>.
- [18] 水谷 正慶, 武田 圭史, 村井 純. 通信の状態遷移に着目したボット活動の調査. 情報処理学会シンポジウム論文集, 第 8 巻, pp. 25–30, Oct 2008. マルウェア対策研究人材育成ワークショップ 2008.
- [19] Martin Roesch. Snort, 12 1998. <http://www.snort.org>.
- [20] Matt Jonkman. Emerging Threats, 2008. <http://www.emergingthreats.net/>.
- [21] Barford Paul, Yegneswaran Vinod. An Inside Look at Botnets. *Special Workshop on Malware Detection, Advances in Information Security: Springer Verlag*, 2006.
- [22] 松木隆宏, 村上純一. 悪性プログラムの耐解析技術を逆用した活動抑制手法の提案. コンピュータセキュリティシンポジウム 2008. IPSJ CSEC 研究会, Oct 2006.
- [23] OpenVPN Technologies, Inc. OpenVPN Open Source Project. <http://www.openvpn.net/index.php/open-source.html>.
- [24] Junichi Murakami. dumnet.
- [25] Bart De Schuymer. ebttables. <http://ebtables.sourceforge.net/>.
- [26] the MAWI Working Group of the WIDE Project. MAWI Working Group Traffic Archive, 2000. <http://tracer.csl.sony.co.jp/mawi/>.
- [27] MDL. Malware Domain List, 2007. <http://www.malwaredomainlist.com/>.

- [28] Eelco Visser. PC Executable Format. Program-Transformation.Org: The Program Transformation Wiki, Apr 2000. <http://www.program-transformation.org/Transform/PcExeFormat>.
- [29] Thomas Karagiannis. Application-specific payload bit strings, Nov 2004. <http://www.cs.ucr.edu/~tkarag/papers/strings.txt>.
- [30] Symantec. W32.Mydoom.A@mm, Jan 2004. http://www.symantec.com/security_response/writeup.jsp?docid=2004-012612-5422-99.
- [31] Suresh Singh James R. Binkley. An Algorithm for Anomaly-based Botnet Detection. *SRUTI '06*, pp. 43–48.
- [32] Lancope. The Role of Network Behavior Analysis & Response Systems in the Enterprise, 2006. <http://www.lancope.com/resource/>.
- [33] Guofei Gu and Phillip Porras and Vinod Yegneswaran and Martin Fong and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *16th USENIX Security Symposium*, pp. 167–182, 2007.
- [34] Evan Cooke, Michael Bailey, Farnam Jahanian, and Richard Mortier. The dark oracle: Perspective-aware unused and unreachable address discovery. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pp. 8–8. USENIX Association, May 2006.
- [35] Jonathan B. Postel. RFC 821: Simple Mail Transfer Protocol. <http://www.ietf.org/rfc/rfc821.txt>, Aug 1982.
- [36] Internet Assigned Numbers Authority. Port numbers, 2007. <http://www.iana.org/assignments/port-numbers>.
- [37] Cyber Clean Center. 2008 年 11 月度 サイバークリーンセンター活動実績, Nov 2008. <https://www.ccc.go.jp/report/200811/0811monthly.html>.
- [38] G. Vigna and R. A. Kemmerer. NetSTAT: A Network-Based Intrusion Detection Approach. In *ACSAC '98: Proceedings of the 14th Annual Computer Security Applications Conference*, p. 25. IEEE Computer Society, 1998.
- [39] Shadowserver Foundation. <http://www.shadowserver.org>.
- [40] Grisoft. AVG Free Edition. <http://free.grisoft.com/>.
- [41] Team Cymru Research NFP. Malware Hash Registry. <http://www.team-cymru.org/>.

- [42] L. T. Herberlein, G. V. Dias, Karl N. Levitt, Biswanath Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *IEEE Symposium on Security and Privacy*, pp. 296–305, 1990.
- [43] Martin Roesch and Stanford Telecommunications. Snort - lightweight intrusion detection for networks. *USENIX LISA '99 Conference*, pp. 229–238. USENIX Association, 1999.
- [44] Paul Bacher, Thorsten Holz, Markus Kotter, Georg Wicherski. Know your enemy: Tracking botnets, May 2005. <http://www.honeynet.org/papers/bots/>.
- [45] NFR Security, Inc. Network Flight Recorder. <http://www.nfr.net>.
- [46] Juniper Networks Inc. "stateful signature, context-based packet signature (netscreen idp)". http://www.juniper.net/products/intrusion/detection.html#Stateful_Sign_Det.
- [47] 藤田直行. 侵入検知ポリシーの記述性向上によりログ出力量の低減を可能とした不正アクセス処理システムの開発. 電子情報通信学会論文誌, Vol. J88-D-1, pp. 391–400, 2005.
- [48] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *IN PROC. 10TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY*, pp. 262–271. ACM, 2003.
- [49] Risto Vaarandi. SEC - a Lightweight Event Correlation Tool. *Proceedings of the 2002 IEEE Workshop on IP Operations and Management*, pp. 111–115, 2002.
- [50] S.T. Eckmann and G. Vigna and R.A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, Vol. 10, No. 1/2, pp. 71–104, 2002.
- [51] Klaus Julisch and Zurich Research Laboratory, Rschlikon, Switzerland. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, Vol. 6, No. 4, pp. 443–471, Nov 2003.
- [52] OSSIM Open Source Security Information Management. <http://www.ossim.net/>.
- [53] ArcSight. <http://www.arcsight.com/>.
- [54] LLC QoSient. Argus, 2000. <http://qosient.com/argus/>.
- [55] ArcSight. Using Advanced Event Correlation to Improve Enterprise Security, Compliance and Business Posture. White Paper, Apr 2007.
- [56] Dominique Karg. *OSSIM Correlation engine explained. Sample scenario: NETBIOS DCERPC ISystemActivator*. OSSIM, Apr 2004. http://www.ossim.net/docs/correlation_engine_explained_rpc_dcom_example.pdf.

- [57] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, Vol. 10, No. 1, pp. 1–31, 2007.
- [58] Sourcefire. *Snort Users Manual 2.8.0*, 2007. http://www.snort.org/docs/snort_htmanuals/htmanual_2.4/node22.html.
- [59] Musashi Y., Ludena R., Dennis A., Nagatomi H., Matsuba R., Sugitani K. A DNS-based Countermeasure Technology for Bot Worm-infected PC terminals in the Campus Network. *Journal for Academic Computing and Networking*, Vol. 10, No. 1, pp. 39–46, 2006.
- [60] 秀誠朝長, 英彦田中. Botnet の命令サーバドメインネームを用いた bot 感染検出方法. 情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2006, No. 129, pp. 13–18, 20061208.
- [61] 竹森敬祐, 三宅優, 中尾康二, 菅谷史昭, 笹瀬巖. Security Operation Center のための IDS ログ分析支援システム. 電子情報通信学会論文誌, Vol. J87-A, No. 6, pp. 816–825, Jul 2004.
- [62] John Jerrim. Benefits of Flow Analysis Using sFlow: Network Visibility, Security and Integrity, 2006. <http://www.lancope.com/resource/>.
- [63] P. Phaal, S. Panchen, N. McKee. InMon Corporation’s sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, Sep 2001. <http://www.ietf.org/rfc/rfc3176.txt>.
- [64] Anthony Cozzie, Frank Stratton, Hui Xue, and Samuel T. King. Digging for data structures. In *OSDI*, pp. 255–266, 2008.
- [65] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang. Effective and efficient malware detection at the end host. In *USENIX’09, 18th Usenix Security Symposium*, Jun 2009.
- [66] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamzeh. Malware detection based on mining api calls. In *SAC*, pp. 1020–1025, 2010.
- [67] Paco X. Nathan, William W. Hurley, Mike W. Erwin, Jamie L. Pugh. OpenSIMS: A Proposed Framework for Collaborative Risk Metrics. <http://www.opensims.org/>.
- [68] S. Staniford-chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagl, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids - a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, pp. 361–370, 1996.

- [69] LIBPCAP. <http://www.tcpdump.org/>.
- [70] Philip Hazel. PCRE - Perl Compatible Regular Expressions. sep 1997. <http://www.pcre.org/>.
- [71] A.V. Aho and M.J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, Vol. 18, pp. 333–340, 1975.
- [72] Ulrich Bayer, Engin Kirda, and Christopher Kruegel. Improving the efficiency of dynamic malware analysis. In *25th Symposium On Applied Computing (SAC), Track on Information Security Research and Applications*, 2010.
- [73] Ilion. FFXI, 2009. <http://ilion.blog.shinobi.jp>.
- [74] Ur I.T. Mate Group. hpHosts, 2009. <http://hosts-file.net/>.
- [75] Cyber-TA Research and Development Project. SRI Honeynet and BotHunter Malware Analysis Automatic Summary Analysis Table, 2005.
- [76] 財団法人インターネット協会（編）．インターネット白書 2009. インプレス R&D, Jun 2009.

付 録 A 評価に用いた通信データに関連するマルウェアの分類

検知精度評価に用いた通信データに関連するマルウェアの検出名を表 A.1 に示す。能動攻撃型マルウェアの評価に用いた通信データ D_s では起動時に利用したマルウェア、受動攻撃型マルウェアのルール作成と評価に用いた D_p , D_q では通信データの HTTP 上に出現したマルウェアをそれぞれ Kaspersky Internet Security 2010 で検査した結果を示す。定義データベースは 2010 年 6 月 2 日 20 時 04 分の最新版を適用した。ただし、Kaspersky Internet Security 2010 は 1 つの検体に対して複数種類の検知結果を出力する場合があるため、表 A.1 の合計は出現した検体数の合計ではない。また、表 A.1 では出現したマルウェアに対してすべて検査をしているため、第 2.5 節において本研究では対象としないと述べた種類のマルウェアも含まれている。

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
Backdoor.Win32.Agent.*	0	53	10
Backdoor.Win32.Bredolab.l	0	0	1
Backdoor.Win32.Ceckno.cyy	0	4	0
Backdoor.Win32.Dreamy.u	0	1	0
Backdoor.Win32.Frauder.*	0	4	0
Backdoor.Win32.Gobot.*	39	0	0
Backdoor.Win32.Hupigon.*	0	4	10
Backdoor.Win32.IRCBot.*	7	2	0
Backdoor.Win32.Kbot.s	0	1	1
Backdoor.Win32.Nepoe.*	5	0	0
Backdoor.Win32.PcClient.kmi	0	0	2
Backdoor.Win32.PoeBot.c	1	0	0
Backdoor.Win32.Rbot.*	22	0	0
Backdoor.Win32.SdBot.*	1	1	0
Backdoor.Win32.Sinowal.*	0	5	1

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
Backdoor.Win32.Small.*	0	0	1
Backdoor.Win32.TDSS.ajm	0	1	0
Backdoor.Win32.UltimateDefender.*	0	10	0
Backdoor.Win32.VanBot.*	12	0	0
Backdoor.Win32.Wootbot.gen	200	0	0
Backdoor.Win32.Zdoogu.*	0	0	7
Email-Worm.Win32.Bagle.aav	0	1	0
Email-Worm.Win32.Joleee.*	0	1	6
Email-Worm.Win32.Zhelatin.*	0	9	0
HEUR:Backdoor.Win32.Generic	0	2	0
HEUR:Trojan-Downloader.Win32.Generic	0	4	0
HEUR:Trojan.Win32.Generic	0	13	3
HEUR:Trojan.Win32.Invader	1	0	0
HEUR:Worm.Win32.Generic	1	5	0
IRC-Worm.Win32.Small.*	0	2	0
Net-Worm.Win32.Aspxor.*	0	10	0
Net-Worm.Win32.Bobic.k	5	0	0
Net-Worm.Win32.Bozori.a	2	0	0
Net-Worm.Win32.Doomjuice.*	2	0	0
Net-Worm.Win32.Kolabc.*	28	0	0
Net-Worm.Win32.Koobface.*	0	1	5
Net-Worm.Win32.Padobot.*	62	0	0
Net-Worm.Win32.Sasser.a	5	0	0
P2P-Worm.Win32.Socks.es	0	2	0
Packed.Win32.Black.d	4	0	0
Packed.Win32.Katusha.a	0	1	0
Packed.Win32.Krap.*	0	0	3
Packed.Win32.PePatch.lc	11	0	0
Packed.Win32.PolyCrypt.m	0	1	0
Packed.Win32.Tdss.*	0	0	3
Packed.Win32.Tibs.*	0	8	0
Rootkit.Win32.Agent.*	0	3	0

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
Rootkit.Win32.Clbd.fm	0	1	0
Rootkit.Win32.Qandr.fg	0	6	0
Rootkit.Win32.TDSS.drp	0	1	0
Trojan-Banker.Win32.Banker.rqk	0	1	0
Trojan-Clicker.Win32.Delf.apc	0	1	0
Trojan-Clicker.Win32.VB.*	0	1	0
Trojan-Downloader.Win32.ACVE.*	0	6	0
Trojan-Downloader.Win32.Agent.*	0	67	28
Trojan-Downloader.Win32.Axload.m	0	1	0
Trojan-Downloader.Win32.BHO.*	0	6	0
Trojan-Downloader.Win32.Banload.aaor	1	0	0
Trojan-Downloader.Win32.Banload.vdl	0	1	0
Trojan-Downloader.Win32.Cntr.*	0	6	0
Trojan-Downloader.Win32.CodecPack.ilu	0	1	0
Trojan-Downloader.Win32.Delf.*	0	6	0
Trojan-Downloader.Win32.Exchanger.pv	0	1	0
Trojan-Downloader.Win32.FraudLoad.*	0	5	1
Trojan-Downloader.Win32.Geral.*	0	0	21
Trojan-Downloader.Win32.Hmir.*	0	6	0
Trojan-Downloader.Win32.Injecter.agq	0	2	0
Trojan-Downloader.Win32.Murlo.*	0	0	2
Trojan-Downloader.Win32.Mutant.*	0	17	0
Trojan-Downloader.Win32.Small.*	0	63	22
Trojan-Downloader.Win32.Tiny.*	0	19	0
Trojan-Downloader.Win32.Tuvir.*	0	4	0
Trojan-Downloader.Win32.VB.*	0	1	1
Trojan-Downloader.Win32.Winlagon.*	0	8	0
Trojan-Downloader.Win32.Zlob.*	0	8	0
Trojan-Dropper.Win32.Agent.*	2	24	36
Trojan-Dropper.Win32.BHO.*	0	0	3
Trojan-Dropper.Win32.DroVedro.*	0	3	0
Trojan-Dropper.Win32.HDrop.b	0	0	10

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
Trojan-Dropper.Win32.Killav.i	0	2	0
Trojan-Dropper.Win32.LJoiner.*	0	4	0
Trojan-Dropper.Win32.Mudrop.*	0	2	1
Trojan-Dropper.Win32.Small.*	0	17	0
Trojan-Dropper.Win32.Sramler.e	2	0	0
Trojan-Dropper.Win32.Wlord.gen	0	1	0
Trojan-GameThief.Win32.Magania.*	0	7	103
Trojan-GameThief.Win32.Nilage.eym	0	0	3
Trojan-GameThief.Win32.OnLineGames.*	1	340	47
Trojan-GameThief.Win32.WOW.*	0	7	0
Trojan-Mailfinder.Win32.Agent.*	0	1	0
Trojan-PSW.Win32.LdPinch.*	1	3	158
Trojan-PSW.Win32.OnLineGames.aniz	0	1	0
Trojan-PSW.Win32.QQPass.ghy	0	0	8
Trojan-PSW.Win32.WebMoner.ev	0	0	1
Trojan-Proxy.Win32.Agent.*	0	3	1
Trojan-Proxy.Win32.Slaper.n	4	0	0
Trojan-Proxy.Win32.Small.*	0	1	0
Trojan-Spy.Win32.Agent.*	0	9	9
Trojan-Spy.Win32.Amber.au	0	0	1
Trojan-Spy.Win32.Amber.mc	0	1	0
Trojan-Spy.Win32.Delf.dsy	0	1	0
Trojan-Spy.Win32.FtpSend.b	0	1	0
Trojan-Spy.Win32.Goldun.*	0	2	0
Trojan-Spy.Win32.Graball.*	0	2	0
Trojan-Spy.Win32.Zbot.*	0	11	10
Trojan.BAT.StartPage.ay	0	1	0
Trojan.Win32.Agent.*	1	108	121
Trojan.Win32.Agent2.*	0	0	3
Trojan.Win32.AntiAV.*	0	0	17
Trojan.Win32.Buzus.*	0	12	1
Trojan.Win32.Crypt.*	0	19	0

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
Trojan.Win32.Delf.dox	0	3	0
Trojan.Win32.Delf.fde	0	1	0
Trojan.Win32.FraudPack.*	0	2	0
Trojan.Win32.Genome.*	0	9	3
Trojan.Win32.Inject.*	3	5	3
Trojan.Win32.KillAV.amr	0	4	0
Trojan.Win32.Monder.mhm	0	1	0
Trojan.Win32.Nosok.*	0	4	0
Trojan.Win32.Obfuscated.gx	0	2	0
Trojan.Win32.Pakes.*	1	25	3
Trojan.Win32.Patcher.ef	0	0	1
Trojan.Win32.Pincav.gmv	0	1	0
Trojan.Win32.Qhost.*	1	14	0
Trojan.Win32.Rabbit.ac	0	0	1
Trojan.Win32.Shutdowner.asq	0	2	0
Trojan.Win32.Small.*	0	3	2
Trojan.Win32.Swisyn.*	0	1	0
Trojan.Win32.VB.*	0	1	0
Trojan.Win32.Vaklik.*	0	6	0
Trojan.Win32.Vapsup.*	0	38	0
Trojan.Win32.Vidal.g	0	1	0
UDS: DangerousObject.Multi.Generic	0	12	0
Virus.Win32.Sality.*	2	0	0
Virus.Win32.Tenga.a	2	0	0
Virus.Win32.Virut.*	173	4	0
Worm.Win32.AutoRun.*	0	13	5
Worm.Win32.Bezopi.ni	0	0	2
Worm.Win32.FTPShare.a	0	5	0
Worm.Win32.Feebs.or	0	2	0
not-a-virus:FraudTool.Win32.Agent.*	0	8	0
not-a-virus:FraudTool.Win32.Antivirus2008pro.al	0	1	0
not-a-virus:FraudTool.Win32.MSAntivirus.*	0	3	0

表 A.1: 各評価用データセットにおけるマルウェアの出現数

マルウェア検出名	D_s	D_p	D_q
not-a-virus:FraudTool.Win32.UltimateAntivirus.*	0	5	0
not-a-virus:FraudTool.Win32.WinAntiVirus.*	0	6	0
not-a-virus:FraudTool.Win32.XPSecurityCenter.*	0	3	0

付 録 B 精度評価で利用したルール

```

1 rule ("R1")
2 {
3     bool scan (host);
4     addr counter (host, stack);
5
6     trigger (tcp.proto ("IRC") &&
7         tcp.pcre ("(MSASS|ROOT\\.MASS|IPSCAN|ADVSCAN|ADV\\.START|LSASS|SCANALL|ASC\\.ASN|\\x2ESA\\x20\\x2DS|[A-ZA-
8         Z0-9\\+\\/]{30}|135|139|445)"))
9     {
10         set (dst.scan, true, 120);
11     }
12
13     trigger (src.scan == true &&
14         (tcp.dst_port ("135") || tcp.dst_port ("139") || tcp.dst_port ("445")) &&
15         tcp.flags ("S/SARF"))
16     {
17         set (src.counter, ipv4.dst_addr (), 60);
18     }
19
20     trigger (src.counter > 16)
21     {
22         alert ("BOT\\ACTIVITY :\\ORDER->PROBE");
23         unset (src.counter);
24     }
25 }
26
27 rule ("R2")
28 {
29     raw url_path (host);
30     bool matched (host);
31
32     trigger (tcp.proto ("IRC") && tcp.pcre ("HTTP :/[\\S+]"))
33     {
34         set (dst.url_path, tcp.pcre ("HTTP :/[A-ZA-Z0-9\\-:]+(\\S+)"), 120);
35     }
36
37     trigger (tcp.seg_id (0) && tcp.pcre ("GET.*/(\\S+)") == url_path)
38     {
39         set (src.matched, true, 120);
40     }
41 }

```

```

42   trigger ( src . matched == true &&
43       ( tcp . pcre ( "MZ.*THIS PROGRAM CANNOT BE RUN IN DOS MODE" ) ||
44         tcp . pcre ( "MZ.*THIS PROGRAM MUST BE RUN UNDER WIN32" ) ||
45         tcp . pcre ( "MZ.*KERNEL32.DLL" ) ||
46         tcp . pcre ( "MZ.{4,32}PKLITE" ) ||
47         tcp . pcre ( "MZ.{2,16}\x00\x00PE" ) ) )
48   {
49       alert ( " DETECTED " );
50   }
51 }
52
53
54 rule ( "R3" )
55 {
56     addr syn_dst ( host , stack );
57     bool flooder ( host );
58     bool downloader ( host );
59
60     trigger ( tcp . flags ( "S/SARF" ) &&
61         ( tcp . dst_port ( 135 ) || tcp . dst_port ( 137 ) || tcp . dst_port ( 139 ) ||
62         tcp . dst_port ( 445 ) || tcp . dst_port ( 3127 ) ) )
63     {
64         set ( src . syn_dst , ipv4 . dst_addr ( ), 120 );
65     }
66
67     trigger ( src . syn_dst > 32 )
68     {
69         set ( src . flooder , true , 300 );
70     }
71
72     trigger ( ( tcp . pcre ( "MZ.*THIS PROGRAM CANNOT BE RUN IN DOS MODE" ) ||
73         tcp . pcre ( "MZ.*THIS PROGRAM MUST BE RUN UNDER WIN32" ) ||
74         tcp . pcre ( "MZ.*KERNEL32.DLL" ) ||
75         tcp . pcre ( "MZ.{4,32}PKLITE" ) ||
76         tcp . pcre ( "MZ.{2,16}\x00\x00PE" ) ) )
77     {
78         set ( dst . downloader , true , 300 );
79     }
80
81     trigger ( ( src . flooder && src . downloader ) ||
82         ( dst . flooder && dst . downloader ) )
83     {
84         alert ( " DETECTED " );
85     }
86 }
87
88
89 rule ( "R4" )
90 {
91     bool javascript ( session );

```

```

92  bool exp_code ( session );
93  bool recv_script ( host );
94  int  recv_exe ( host );
95  int  exe_count ( host );
96  addr  script_server ( host , stack );
97  bool non_exe_url ( session );
98
99  trigger ( http . dl_data ( "\ x3CSCRIPT | FUNCTION _" ))
100  {
101      set ( session . javascript , true, 10);
102  }
103
104  trigger ( http . dl_data ( "(%u[0-9A-F]{4}) {10}[(\ x5cx[0-9A-F]{2}) {10}([0-9]{2,3},[ _]*) {10}[( A-Fa-f0
      -9]{16}][\ x21-\x23\x26-\x7E]{64} ") )
105  {
106      set ( session . exp_code , true, 10);
107  }
108
109  trigger ( session . javascript && session . exp_code )
110  {
111      set ( dst . recv_script , true, 60);
112      set ( dst . script_server , ipv4 . src_addr ( ), 60);
113  }
114
115  trigger ( src . recv_script && tcp . proto ( " HTTP " ) && http . uri ( " \.( JPG | CSS | GIF | TXT )" ))
116  {
117      set ( session . non_exe_url , true, 10);
118  }
119
120  trigger ( dst . recv_script &&
121      ( http . dl_data ( "MZ.*THIS_PROGRAM_CANNOT_BE_RUN_IN_DOS_MODE" ) ||
122      http . dl_data ( "MZ.*THIS_PROGRAM_MUST_BE_RUN_UNDER_Win32" ) ||
123      http . dl_data ( "MZ.*KERNEL32.DLL" ) ||
124      http . dl_data ( "MZ.{4,32}PKLITE" ) ||
125      http . dl_data ( "MZ.{2,16}\x00\x00PE" )))
126  {
127      unset ( dst . recv_script );
128      set ( dst . recv_exe , true, 60);
129      inc ( dst . exe_count , 120);
130  }
131
132  trigger ( dst . recv_count > 0 &&
133      ( http . dl_data ( "MZ.*THIS_PROGRAM_CANNOT_BE_RUN_IN_DOS_MODE" ) ||
134      http . dl_data ( "MZ.*THIS_PROGRAM_MUST_BE_RUN_UNDER_Win32" ) ||
135      http . dl_data ( "MZ.*KERNEL32.DLL" ) ||
136      http . dl_data ( "MZ.{4,32}PKLITE" ) ||
137      http . dl_data ( "MZ.{2,16}\x00\x00PE" )))
138  {
139      inc ( dst . exe_count , 120);
140  }

```

```

141
142 trigger ( dst . exe_count () >= 5)
143 {
144     unset ( dst . exe_count );
145     alert ( "A2: MANY EXECUTABLE FILE ");
146 }
147
148 trigger ( dst . script_server == ipv4 . src_addr () &&
149     ( http . dl_data ( "MZ.*THIS PROGRAM CANNOT BE RUN IN DOS MODE" ) ||
150     http . dl_data ( "MZ.*THIS PROGRAM MUST BE RUN UNDER WIN32" ) ||
151     http . dl_data ( "MZ.*KERNEL32.DLL" ) ||
152     http . dl_data ( "MZ.{4,32}PKLITE" ) ||
153     http . dl_data ( "MZ.{2,16}\x00\x00PE" )))
154 {
155     unset ( dst . script_server );
156     alert ( "A3: EXPLOIT AND DOWNLOAD MALWARE FROM SAME SERVER ");
157 }
158
159 trigger ( dst . recv_exe && dst . non_exe_url )
160 {
161     unset ( dst . recv_exe );
162     unset ( dst . non_exe_url );
163     alert ( "A1: FAKE EXECUTABLE FILE ");
164 }
165
166 trigger ( src . recv_exe &&
167     ( http . uri ( "[\x26?] ID =[\x26?][ A-Z ]=[\x26?] VER =[\x26?] SMTP =[\x26?] URL=" ) ||
168     tcp . pcre ( "NAME=\x22ID \x22 | NAME=\x22[ A-Z ]\x22 | NAME=\x22VER \x22" )))
169 {
170     unset ( src . recv_exe );
171     alert ( "A4: DETECT MALWARE ACTIVITIES ");
172 }
173 }

```

付 録 C 本研究に関する業績リスト

C.1 査読有り論文誌

1. **Session-based IDS の設計と実装**, 水谷正慶, 白畑真, 南政樹, 村井純, 電子情報通信学会 次世代ネットワークソフトウェア論文特集, Vol.J88-B, pp 551-562, Mar. 2005
2. **Web 感染型悪性プログラムの分析と検知手法の提案**, 水谷正慶, 武田圭史, 村井純, 電子情報通信学会 セキュアでサステナブルなインターネットアーキテクチャ論文特集 Vol.J92-B, No.10, pp. 1631-1642, 2009 Oct.
3. **通信の共通性を利用した悪性プログラム検知手法の実装と評価**, 水谷正慶, 武田圭史, 村井純, 情報処理学会論文誌 社会を活性化するコンピュータセキュリティ技術特集, Vol 50, No.9, pp. 2137-2146, 2009 Sep.

C.2 国内発表

1. **IDS のログ視覚化システムの構築**, 水谷 正慶, 白畑 真, 土本 康生, 村井 純, IPSJ DSM Symposium 2003. Vol. 6. pp 32-36., Jan. 2003
2. **セッション追跡によるプロトコルアノーマリ型防御機構の提案と実装**, 水谷 正慶, 白畑 真, 南 政樹, 村井 純, IPSJ DPS Workshop 2004. DPSWS2006 pp 227-232, Dec. 2004
3. **OS 情報にもとづいたNIDS のイベントリスク評価システムの設計と実装**, 水谷 正慶, 白畑 真, 南 政樹, 村井 純, IEICE IN Workshop 2006. Vol.105, No.628(20060223) pp. 459-464, Mar. 2006
4. **ネットワーク不正侵入検知のためのイベント分析手法の検討**, 水谷 正慶, 白畑 真, 南 政樹, 村井 純, IPSJ DICOMO 2007, 2007 Jul
5. **複数セッションの相関関係を利用したセキュリティイベント検知手法の提案**, 水谷 正慶, 白畑 真, 南 政樹, 村井 純, IPSJ CSEC Computer Security Symposium 2007, Vol. 10, pp. 595-600, 2007 Nov
6. **通信の状態遷移に着目したボット活動の調査**, 水谷 正慶, 武田 圭史, 村井 純, MWS 2008, Vol. 8, pp. 25-30, 2008 Oct

7. 悪性プログラムの調査を目的とした通信データ解析ソフトウェア, 水谷 正慶, 武田 圭史, 村井 純, The 71st National Convention of IPSJ, Vol. 71, pp. 3-383, Mar. 2009
8. 多様な感染経路に対応したマルウェア収集・解析環境の構築, 水谷 正慶, 武田 圭史, 村井 純, MWS 2009, 2009 Oct
9. 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, 金井 瑛, 水谷 正慶, 佐川 昭宏, 白畑 真, 南 政樹, 村井 純, IPSJ DPS 13th Workshop 2005. Vol. 19, pp.173-177, Dec. 2005
10. UPnP-IGD を応用した IPS バイパス手法の提案と実装, 中島 智広, 水谷 正慶, 佐川 昭宏, 白畑 真, 南 政樹, 村井 純, IPSJ DPS 13th Workshop 2005. Vol. 19, pp. 377-381, Dec. 2005
11. セキュリティインシデント対応のためのユーザ特定支援システムの実装, 上原 雄貴, 水谷 正慶, 武田 圭史, 村井 純, The 71st National Convention of IPSJ, Vol. 71, pp.3-415, Mar. 2009
12. エンタテイメント性を考慮した視覚化による意図せぬ通信の検知, 朝永 愛子, 水谷 正慶, 武田 圭史, 村井 純, The 71st National Convention of IPSJ, Vol. 71, pp.3-377, Mar. 2009
13. パケットヘッダ情報を用いた仮想ユーザ識別システムの提案, 上原 雄貴, 水谷 正慶, 武田 圭史, 村井 純, Computer Security Symposium (CSS) 2009, 2009 Oct
14. メタデータ作成によるマルウェア通信解析手法の提案, 榊 辰哉, 水谷 正慶, 武田 圭史, 村井 純, MWS2009, 2009 Oct
15. IPv6 導入に伴うセキュリティリスクに関する分析, 武田 圭史, 水谷 正慶, 中井 研, Computer Security Symposium (CSS) 2009, 2009 Oct

C.3 国外発表

1. “ROOK: Multi-Session based Network Security Event Detector”, Masayoshi Mizutani, Shin Shirahata, Masaki Minami, Jun Murai, Applications and the Internet, IEEE/IPSJ International Symposium, pp48-54, 2008 Jul
2. “ROOK: Behavior Rule based Intrusion Detection”, Masayoshi Mizutani, Keiji Takeda, Jun Murai, Asia Future Internet 2009 summer meeting, 2009 Aug.
3. “Analysis of Web-based Malware”, Masayoshi Mizutani, Internet2 2009 Fall meeting, 2009 Oct
4. “Behavior Rule based Intrusion Detection”, Masayoshi Mizutani, Keiji Takeda, Jun Murai, CoNEXT 2009 Student Workshop, pp57-58, 2009 Dec

C.4 その他

- *The Design and Implementation of Session Based IDS*, Masayoshi Mizutani, Shin Shirahata, Masaki Minami, Jun Murai, IEICE Transactions on Communications (Japanese Edition), IEICE, Vol.89, No.3, pp.46-58, Nov. 2005

付録C.1の1を電子情報通信学会の依頼に基づいて翻訳，掲載した論文