修士論文　　2007年度 (平成19年度)

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

慶應義塾大学大学院 政策・メディア研究科
水谷 正慶

修士論文要旨 2007年度 (平成19年度)

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

論文要旨

ネットワーク上で発生する情報資産の侵害を検出するために，多くのネットワーク管理者は侵入検知システム (IDS) によってネットワークセキュリティに関する事象の検知を実施している．情報資産侵害が発生した場合の早期対応にネットワークセキュリティに関する事象の検知は不可欠だが，検知を回避するソフトウェアが多く出現し、ネットワーク管理者の負担が増加させている．特にボットは無害なソフトウェアに見せかけた通信やP2P型の暗号化通信を利用しているため，検知の精度が低下する，あるいは検知結果と脅威の関連づけが明確ではなくなっている，

本論文では通信の相関関係を利用することによって，脅威との関連づけが明確な検知を高い精度で実現した．既存のIDSではプロトコル毎のホスト間の通信（セッション）を1つずつ個別に検査していたため，検知に利用できる情報が限られていた．本論文では多くのソフトウェアが複数のセッションを発生させる点に着目し，既存のIDSによるパターンマッチに加えてセッション間の相関関係を利用した検知手法を提案した．セッションの相関関係を利用した検知とは1) セッションの出現規則の監視，2) あるセッションに含まれる情報を別のセッションの検知に利用する，という2点であり，これらを利用することでソフトウェアの振る舞いを検知ルールとして柔軟かつ明確に定義できる．提案手法を設計・実装し検知精度を評価した結果，既存のIDSによる手法ではボットによる活動の誤検知が多発したが，提案手法による実装では誤検知なく検知できた．本研究によって，これまで誤検知などの問題が起きやすかったネットワークセキュリティに関する事象を高精度に検知できるようになり，ネットワーク管理者の監視負担の軽減に貢献できた．

キーワード
1. インターネット，2. セキュリティ，3. ネットワークインシデント．4. 侵入検知，

慶應義塾大学大学院 政策・メディア研究科

水谷 正慶

Abstract of Master's Thesis

Academic Year 2007

Network Security Monitoring Method by using a Correlation of Communications

Network security monitoring is necessary for incident responses. Intrusion Detection System (IDS) is the major network security monitoring method, and it detects the variants of known threat activities (security events) to alert system administrators to potential system or network security threats and weaknesses. In order to evade the detections, authors of malicious softwares (malware, e.g. bot, spyware) frequently use the followed methods. One is disguising malware traffic as innocent traffic. One is encrypting the contents or the protocol headers of malware traffic. Many malware activities are therefore similar to those of non-malicious software, hence huge numbers of false positives are generated by the existing IDSs.

In this research, a high accurate and definitive intrusion detection method by a using correlations between sessions of an application is proposed. The meaning of "high accurate" is the method results in low false positives or negatives, and the meaning of "definitive" is the method is able to associate security events with threats. The method monitors host behaviors based on an order of generated sessions because each application has a decided order of session. The method also monitors a session based on the information within the former sessions, e.g. if the session is correctly routed to the previously-resolved host. Results of our evaluation shows that the implementation detected no false positives. Therefore the method can detect security events by malware with a high degree of accuracy and this research contributed to reducing costs of network security monitoring.

Keywords :
1. Internet, 2. Security, 3. Network Incident, 4. Intrusion Detection

Keio University , Graduate School of Media and Governance

Masayoshi Mizutani

目 次

第1章　序章 **1**
1.1　背景 1
1.2　本研究の目的 2
1.3　本研究により期待される成果 2
1.4　本論文の構成 3

第2章　ネットワークインシデントと検出手法 **4**
2.1　ネットワークインシデント 4
2.1.1　ネットワークインシデント検出 5
2.1.2　ネットワークインシデントの種類 6
2.1.3　ネットワークインシデントの対策 8
2.1.4　ネットワークインシデントの原因となる脅威の分類 9
2.2　セキュリティイベント 10
2.2.1　セキュリティイベントの種類 10
2.2.2　セキュリティイベントの活用 11
2.3　既存のセキュリティイベント検知手法 12
2.3.1　主要なセキュリティイベント検知手法 13
2.3.2　主要なセキュリティイベント検知手法の比較 14
2.3.3　セキュリティイベント検知における技術的な課題 15
2.3.4　セキュリティイベント検知における運用的な課題 16
2.4　まとめ 17

第3章　セキュリティイベント検知における現状の問題点 **18**
3.1　他のソフトウェアと類似した通信 18
3.2　通信の暗号化 20
3.3　まとめ 21

第4章　関連研究 **22**
4.1　Traffic Classification 22
4.2　Correlation Analysis 23
4.3　Anomaly Detection 24
4.4　Network Behavior Analysis 25
4.5　まとめ 25

第5章　要求事項 **26**
5.1 検知精度 26
5.2 検知結果の明確性 27
5.3 検知条件の柔軟性 27
5.4 検知処理の規模性 27
5.5 まとめ 28

第6章　複数セッション型ネットワーク監視手法の提案 **29**
6.1 セッションの相関関係 29
6.1.1 セッションの出現規則 30
6.1.2 セッション内に含まれる情報の相互利用 30
6.2 本手法が有効に機能する例 31
6.2.1 ボットの活動による例 31
6.2.2 P2P ファイル交換ソフトウェアの実行による例 32
6.3 提案手法によるセキュリティイベント検知の戦略 32
6.4 まとめ 33

第7章　予備実験 **34**
7.1 動作確認 34
7.2 誤検知発生率の調査 34
7.3 まとめ 36

第8章　設計 **37**
8.1 ROOK 設計概要 37
8.2 トラフィック監視型セキュリティイベント検知機構に対する要求 38
8.3 検知ルール構造 39
8.3.1 セッションルール構造 40
8.3.2 パラメータ構造 41
8.3.3 パラメータ構造例 42
8.3.4 ルールの表現力に関する考察 43
8.4 まとめ 45

第9章　実装 **46**
9.1 実装概要 47
9.2 Input コンポーネント 48
9.2.1 入力パケットデータのバッファリング 48
9.2.2 複数ネットワークインターフェースからの並列入力 49
9.2.3 処理負荷に関する考察 49
9.3 Decoding コンポーネント 50
9.3.1 モジュール構成 51
9.3.2 セッション管理機能 52

9.3.3 エンティティ管理機能 54
9.3.4 パラメータ管理 55
9.3.5 トラップ管理 55
9.3.6 フィルタ機能 56
9.3.7 多重フィルタ機能 56
9.3.8 Payload-based Protocol Detector 57
9.4 Rule コンポーネント 58
9.4.1 ルール書式 59
9.4.2 ルール記述例 60
9.4.3 セッションルールの分離性に関する考察 61
9.5 Detection コンポーネント 64
9.5.1 検知処理に関する実装 64
9.5.2 パラメータ操作 66
9.6 Output コンポーネント 67
9.6.1 検知結果の出力形式 68
9.6.2 出力される情報 68
9.7 本実装における処理負荷の考察 69
9.7.1 Decoding コンポーネントにおける計算量 69
9.7.2 Detection コンポーネントにおける計算量 70
9.7.3 処理負荷軽減についての考察 71
9.8 動作検証 72
9.8.1 フィルタを利用している条件の検証 72
9.8.2 パラメータを利用している条件の検証 74
9.9 まとめ 77

第10章 評価 **79**
10.1 検知精度の評価 79
10.1.1 評価条件 80
10.1.2 検知ルールの動作検証 82
10.1.3 評価結果 82
10.2 性能評価 85
10.2.1 性能評価のねらい 86
10.2.2 性能評価構成 86
10.2.3 性能評価条件 87
10.2.4 評価結果 89
10.3 関連研究との比較 90
10.4 まとめ 92

第11章 結論 **93**
11.1 本論文のまとめ 93
11.2 今後の課題と展望 94
11.2.1 ルール作成時のデバッグ 94
11.2.2 ルール記述方法の問題 94
11.2.3 ルール作成の自動化 95
11.2.4 検知処理実装の高速化 95
11.2.5 IPv4/IPv6混在環境やNAT利用環境への対応 95
11.2.6 未知のボットに対する応用 96

付 録A ルールの記述 **106**

付 録B 評価に利用したデータセットの詳細 **109**

付 録C 精度評価用ルール一覧 **112**

図 目 次

2.1 ネットワークインシデント検出概要 5

3.1 ボットと C&C サーバの IRC を用いた通信の例 19

6.1 ボットの活動におけるセッションの発生例 30
6.2 P2P ファイル交換ソフトウェアの実行におけるセッションの発生例 31

7.1 調査に利用したネットワークの構成 36

8.1 ROOK 設計概要 38
8.2 ROOK 実行想定環境例 38
8.3 ROOK セッションルール構造例 40
8.4 パラメータ構造概要 41
8.5 セッションの出現規則表現 (順接) 43
8.6 セッションの出現規則表現 (和) 43
8.7 セッションの出現規則表現 (積) 44
8.8 セッションの出現規則表現 (繰り返し) 44
8.9 セッションの出現規則表現 (括弧) 45

9.1 ROOK 実装概要図 47
9.2 Input コンポーネント概要 48
9.3 Decoding コンポーネント概要 50
9.4 セッション構造例 52
9.5 セッション管理構造例 53
9.6 セッション構造体のコード 54
9.7 エンティティ管理構造例 55
9.8 ルール基本構成概要 59
9.9 ルールの基本的な記述形式 61
9.10 ルール記述例 62
9.11 検知処理の擬似コード 64
9.12 パラメータ操作の概要 67
9.13 パラメータ機能検証用ルール R_p 76
9.14 URI に “game” を含む 172.16.113.105 からの HTTP GET リクエスト 77
9.15 R_p を適用した本実装による出力結果 77
9.16 パラメータ機能検証用ルール R_q 78

9.17 tcpdumpにより抽出したUDPの宛先ポート53に対するICMP Port Unreachableメッセージ 78
9.18 R_pを適用した本実装による出力結果 78

10.1 性能評価用構成 85
10.2 実験ホストのチューニング 85
10.3 評価用ルール R_a 87
10.4 評価用ルール R_b 87
10.5 評価用ルール R_c 88
10.6 データセット (N_4) を用いた性能評価実験結果 89
10.7 データセット N_5 を用いた性能評価実験結果 90

B.1 データセット N_4 のパケットサイズの分布 109
B.2 データセット N_5 のパケットサイズの分布 110
B.3 評価用データセット収集環境 (N_1) 110
B.4 評価用データセット収集環境 (N_2) 110
B.5 評価用データセット収集環境 (N_3) 111

表 目 次

2.1 インシデントの種類と情報資産への影響 7
2.2 ネットワークインシデントの種類と情報資産への影響 8
2.3 主要なセキュリティイベント検知手法一覧 13

7.1 実験用ルール 1(R_1): ボット検知用ルール 35
7.2 実験用ルール 2(R_2): Winnyp 検知用ルール 35
7.3 R_1 を用いたトラフィック監視結果の内訳 35
7.4 R_2 を用いたトラフィック監視の結果 35

9.1 ROOK で実装されているペイロードベースのプロトコル判定条件例 58
9.2 ROOK で実装されているペイロードベースのプロトコル判定条件 67
9.3 Decoding コンポーネントの計算量一覧 69
9.4 多重フィルタの計算量一覧 70
9.5 フィルタ機能検証項目の一覧 75

10.1 評価用データセット概要 79
10.2 ボット検知用ルールによって検知できたボットの種類一覧 83
10.3 精度評価結果 84
10.4 性能評価に使用したホストの構成 85
10.5 パラメータの保持数と参照回数 89
10.6 関連研究との比較一覧 91

A.1 各タグの属性一覧 107
A.2 ROOK で実装されているフィルタ一覧 108

第1章　序章

1.1　背景

インターネットはセキュリティの面において様々な脅威が発生し，社会的な問題となっている．2003年1月に発生したSlammerワーム[2]や2003年8月に発生したMS Blasterワーム[3]は，インターネットを通じて急速に世界中のホストに感染し，各所で業務に支障をきたすなどの被害をもたらした．独立行政法人 情報処理推進機構(IPA)の試算[4]によれば2003年1月から12月までの国内でのコンピュータウィルスによる被害総額は3025億円である．一方で，悪意を持った内部ネットワークの利用者（エンドユーザ）による機密情報の盗難や，内部システムへの侵入，情報の改ざんも多く発生している．また，悪意のない内部ネットワークの利用者による機密情報流出も多く発生している．Winny[5]やShareに代表されるPeer-to-Peer(P2P)ファイル交換ソフトウェアを介した情報流出事件も多数発生している[6]．インターネットに接続されたネットワークの利用は，高い利便性を得られるが様々な事件に遭遇する危険性も高くなってしまう．

本研究はネットワークや計算機に対する情報資産侵害（ネットワークインシデント）の対策手段として，検出に着目している．ネットワークインシデントによる被害を完全に防ぐのは事実上不可能である[1]．そのため，事件による被害を最小限にとどめるためには，事件の兆候や事件の発生に対して早急な対応が必要になる．しかし，被害を発生させようとする人物（攻撃者）は攻撃の成功率を上げるために，事件の発覚を防ぐ工夫をしている．事件の兆候や事件の発生に計算機（ホスト）の所有者やネットワークの管理に責任がある人物（ネットワーク管理者）が気づかなければ，事件の被害は拡大させてしまう．ホスト所有者が事件の兆候や発生を察知できればよいが，ホスト所有者のモラルが低い，あるいはホスト上での察知を困難にするRootkitなどを利用される場合があるため，ネットワーク側での対策が必要となる．そのため，ネットワーク管理者はネットワークの状況や通信を監視することで，事件の兆候や発生を検知し，早急に対応することで，被害を最小限にすることができる．全ての脅威を防げるだけの対策を導入・運用するためには，莫大なコストと利便性を犠牲にしなければならない．これに対して，防ぎやすい脅威と防ぎにくいが被害発生直後に対応すれば深刻な被害にならない脅威とを区別し，後者を検知と事後対応によって対策することで，効果的な対策が実現できる．

既存のネットワークインシデント検出技術は，新しい脅威に対する検知精度の悪さや，ネットワークインシデントとの対応が明確ではない検知結果を出力してしまうなどの問題を抱えている．検出に利用できる代表的な手法として侵入検知システム（Intrusion Detection System, IDS）が挙げられる．IDSは攻撃者による調査活動，ワームやコンピュータウィルスによる不正侵入の試み，特定アプリケーションによる通信などの検知に広く利用でき

る．これに対抗するため，攻撃者も今まで以上に検知を困難にするための工夫をするようになった．特に新しい脅威の一つであり，感染することで攻撃者の支配下に置かれてしまう**ボット**は，ホストの所有者やネットワーク管理者に活動を察知されないための様々な機能が実装されている．そのため，IDS で検知しようとしても誤検知やネットワークインシデントとの関係が明確ではない結果が出力されてしまう．

1.2 本研究の目的

本研究の目的は，ネットワーク上の**セキュリティイベント**を高精度かつ柔軟に検知する新しい手法の実現である．セキュリティイベントとは，ネットワーク上の脅威から引き起こされる事件に関連する可能性がある様々な事象を指す．本研究ではボットに代表される検知が困難な脅威に対応できる，新しいセキュリティイベント検知手法の確立を目指す．

既存の IDS では 1 つのパケットや 1 つのセッションの特徴に着目していた．あらかじめセキュリティイベントとして検知すべき通信の特徴を定義することで，精度が高くネットワークインシデントとの関係が明確なセキュリティイベントを検知してきた．しかし，ボットのような新しい脅威は IDS では検知が難しい通信を使用し，ネットワーク管理者による検出を妨害している．具体的な方法として**他のソフトウェアと類似した通信**と**通信の暗号化**の 2 つが挙げられる．通信の特徴点を条件としてセキュリティイベントを検知しようとする場合，暗号化された通信では特徴が全て隠れてしまう．他のソフトウェアと類似した通信を利用された場合，1 つのセッションのみで他のソフトウェアと区別するのは非常に困難になってしまう．

本論文では高精度かつ柔軟なセキュリティイベントの検知を実現するために，通信の特徴点に加えて**セッション間の相関関係**に着目している．セッション間の相関関係とは，セッションの出現規則とセッション間の情報交換の 2 つである．セッションの出現規則とは複数セッションが出現する順番やあるセッションの繰り返しを検知条件とすることである．セッションの出現規則を把握していれば，通信内容を見ることなく検知できるセキュリティイベントもある．セッション間の情報交換とは一方のセッションに含まれている情報を利用して，他方のセッションを検知する方法である．セッションの出現規則とセッション間の情報交換を条件として利用することで，1 つのセッションの通信の特徴に着目していた場合と比較し，多くの情報をセキュリティイベント検知に利用できる．既存の IDS では，1 つのセッションに含まれる特徴を断片的な手がかりとしていた．これに対し本手法では，1 つのホストの複数のセッションを俯瞰的に捉えることで，ソフトウェアの振る舞いを正確に把握し，より高い精度でセキュリティイベントを検知できると期待できる．

1.3 本研究により期待される成果

セキュリティイベントの誤検知や曖昧な検知結果はネットワーク管理者の運用負担を増加させ，ネットワークインシデント検出対策の適切な運用を困難にしてしまう．ネットワーク管理者が出力結果を見ただけで，それが誤検知もしくは曖昧な検知であると判断す

るのは難しい．そのため，誤検知もしくは曖昧な検知が多発する手法を使っている場合，ネットワークインシデント発生の有無を確認するために調査が必要となる．調査する方法として，該当するホストの直接調査，ネットワーク状態の調査，報告された検知結果に関連があると考えられる情報の収集など多岐にわたるが，どの方法もネットワーク管理者の負担が少なくない．誤検知や曖昧な検知が多発するとネットワーク管理者の負担が増大し，検知結果に対する信憑性も失われてしまう．このような状態では実際にネットワークインシデントが発生しても早急な対応がとれず，検知対策の有効性は低くなる．これが，誤検知や曖昧な検知結果による弊害である．

本論文で提案するセッション間の相関関係に着目したセキュリティイベント検知手法を確立することにより，検知に必要となるコストの抑制が期待できる．現状の IDS などでは運用コストが高く，適切にネットワークの監視・検知をするのが困難であった．本手法によって検知コストが低下することにより，ネットワーク管理者の負担を大幅に減らすことができる．また，高精度の検知手法を確立することによって，一般家庭や小規模組織などの専門家がいないネットワークでも検知が利用できるようになると考えられる．現状では誤検知などの多さから，誤検知かどうかを確認するための知識や経験，コストが必要となっていた．高精度な検知手法を利用することで，自動的な対応や危機確認ができるようになり，一般家庭や小規模組織でも多段的な対策をとることが可能になる．

1.4　本論文の構成

本論文は全 11 章で構成されている．第 2 章でネットワークインシデントを取り巻く現状について述べる．第 3 章では現状で検知が困難な暗号化された通信と他のソフトウェアに類似した通信によって発生するセキュリティイベントについて述べる．第 4 章でセキュリティイベント検知の関連研究を紹介する．第 5 章で検知が困難なセキュリティイベントに対応するために必要となる要求事項をまとめる．第 6 章で本論文がとる複数セッションを用いた検知手法について述べ，第 7 章ではその有効性を示すための予備実験について述べる．第 8 章で提案手法を実現するための設計を延べ，第 9 章では実装の詳細について述べる．第 10 章で本手法の有効性を示し，第 11 章で本論文の結論と今後の課題をまとめた．

第2章　ネットワークインシデントと検出手法

本論文ではネットワーク上で発生する情報資産の侵害である**ネットワークインシデント**に対抗する手段として，ネットワークインシデントの検出に着目している．ネットワークインシデントの対策は，事前に発生を防ぐものと，発生後の被害を極小化するものの2つに分類できる．ネットワークインシデントの検出は事後対策の開始を促す役割があり，ネットワークインシデント対策において重要な機能となっている．

さらに，本論文ではネットワークインシデント検出の手段として，**セキュリティイベント**を発見する**検知**に着目している．セキュリティイベントはネットワークトラフィックや各ホスト上における事象の中から，ネットワークインシデントを発見する手がかりとなる事象を抽出したものである．しかし既存のセキュリティイベント検知手法を用いた場合，ネットワークインシデント検出対策を運用するためにはネットワーク管理者の負担が大きくなってしまう問題がある．

本章では第3章で述べる問題点の背景として，ネットワークインシデントとセキュリティイベントについて述べる．具体的なネットワークインシデントの種類や対策の種類について述べ，検出がどのような役割を持つか論じる．さらに，検出に必要となるセキュリティイベントの種類やセキュリティイベント検知手法に関連する課題について述べる．

2.1　ネットワークインシデント

ネットワーク上で起きる情報資産の侵害事件を総じて**ネットワークインシデント**と呼ぶ．情報資産とはホスト上，あるいはネットワーク上に存在する情報や資源で，機密性，完全性，可用性を維持しなければならないものを指す．機密性，完全性，可用性は安全の基準として ISO17799[11] において定義されている．機密性とは，特定の情報へのアクセスが許可された者だけに限定されていることである．機密性の侵害の例として，不正な手段によってシステムの権限を取得される，ソフトウェアの不適切な設定により意図しない相手に情報が漏洩するなどが挙げられる．完全性とは，提供される情報やその処理過程の正確さが保証されていることである．完全性の侵害とは，ソフトウェアが改ざんされ正常に動作しなくなる，ファイルに記録されている情報が改ざんされる，などが挙げられる．可用性とは，システムがダウンすることなく継続的に稼働していることである．可用性の侵害とは，ホスト内のプロセスやネットワークに異常が発生し，サービスが提供できなくなる状態である．

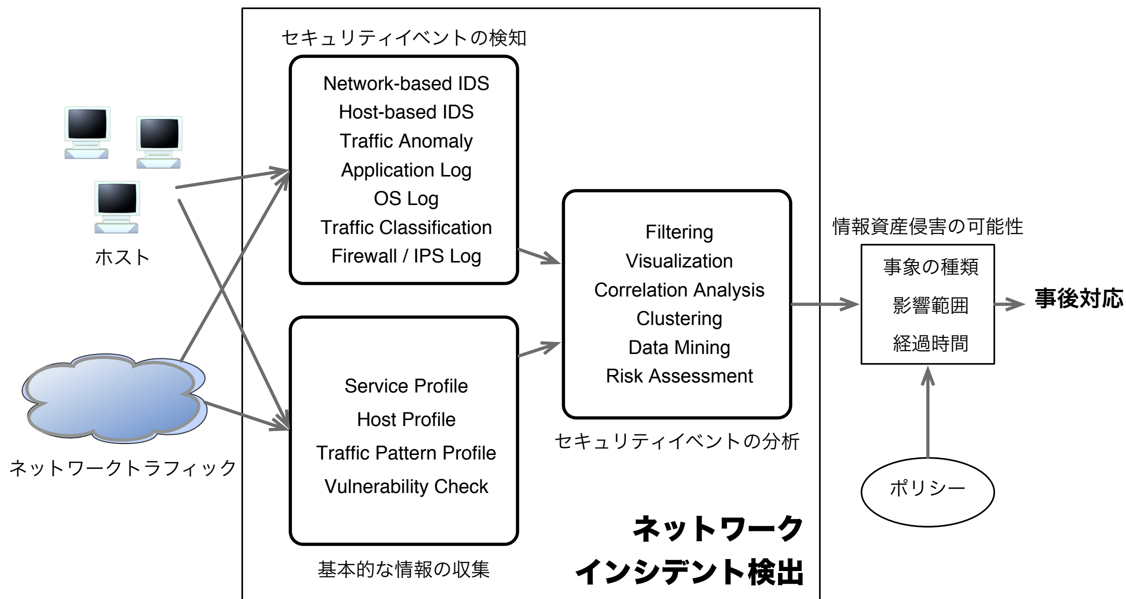

図 2.1: ネットワークインシデント検出概要

これらの脅威による被害を防ぐため，ネットワーク管理者は対策を講じる必要がある．対策を講じなければならない理由は主に以下の3点である．1つめは，エンドユーザの被害を最小限にとどめるためである．エンドユーザは利用しているホストについて管理責任を持つ場合，自らの責任で対策する必要がある．しかし，コンピュータやネットワークに関する知識や経験が十分ではないために，対策が誤っていたり，正しく機能していない場合がある．知識や経験が十分でも不注意や過失，あるいは想定外の脅威や未知の脅威によって被害を受ける可能性がある．そのため，多層的に対策をとる必要がある．2つめは，ネットワークで提供しているサービスを正常に継続させるためである．多くのネットワークではインターネットへの接続性をはじめとし，電子メールや Web サーバなどのサービスを提供している．多くの場合，これらの運用責任はネットワーク管理者が負っている．安定したサービス提供のためには，脅威による被害や妨害を最小限に食い止めなければならない．3つめは，ネットワークを所有する主体組織・団体の社会的信頼性を保護するためである．当該ネットワークにおいて，脅威が引き起こした被害が社会的に認識されると，組織や団体の情報管理体制に対する不信感が発生する．これは組織や団体の活動に支障をきたしてしまう．非営利団体の場合でも，情報管理体制への不信感から情報提供を拒否されるなどの不都合が起きると考えられる．そのため事件による被害を防ぐ対策や，発生後に緊急対応できる対策が必要となる．

2.1.1 ネットワークインシデント検出

ネットワークインシデント検出とは，情報資産侵害の兆候と侵害の発生を発見することである．実際の運用において，ネットワークインシデントの確実な発見は難しい．これは

攻撃者がネットワークインシデントの発生を隠蔽するため，明確な痕跡が残りにくいためである．様々な情報を分析することで確実性は増すが，実際には情報資産の侵害とは無関係の事象を追跡してしまう可能性がある．あるいは少ない情報からネットワークインシデント発生の有無を判断しなければならない場合もある．ネットワークインシデント検出はネットワークインシデント発生後の対応を促すためのものなので，確実に発生した事が示されなくても，高い確率で起きる兆候を発見できれば有効な対策であると言える．

図 2.1 はネットワークインシデント検出のモデルを示している．ネットワークインシデントの検出はセキュリティイベントの検知，基本的な情報の分析，セキュリティイベントの分析の 3 段階で構成される．セキュリティイベントの検知はホスト上やネットワーク上の事象からネットワークインシデントに関連する可能性がある事象を抽出する．一方でホストやネットワークの基本的な情報を収集する．これは OS の種類や情報，アプリケーションのログ，ネットワーク内ホストの脆弱性情報，ネットワークのトポロジ情報などが挙げられる．検知されたセキュリティイベントと基本的な情報を必要に応じて利用し，セキュリティイベントを分析し，ネットワークインシデントの可能性として事象の種類，影響予想範囲，発生時間（経過時間）を出力する．

異なる組織において同様の事件が起きたとしても，ネットワークインシデントと判断するかは当該組織のポリシに依存する．本論文における組織とは各種企業や団体など，ある目的に応じて活動する集団を指している．また，ポリシとは保護すべき情報資産とその重要性の設定を指す．明文化されていない暗黙の情報資産・重要性設定もポリシとして扱う．ネットワークインシデントとは情報資産の侵害であるため，情報資産の設定が異なればネットワークインシデントの判断基準も異なる．そのため，本論文で議論するネットワークインシデント検出とはネットワークインシデントの可能性がある事象の発見である．ネットワークインシデントであるかの判断は組織毎に異なる．また，ネットワークインシデントのリスクや重要性についても，組織のポリシによって大きく異なる．ただし，多くの組織で共通した種類の情報資産や重要度設定があるため，ある程度の一般化ができる．これを基にして，ネットワークインシデントの可能性がある事象を判断し，検出する．

2.1.2 ネットワークインシデントの種類

“Mining in a Data-flow Environment: Experience in Network Intrusion Detection”[12] などで取り上げられている攻撃の分類方法を参考に，ネットワークインシデントを Remote to Local(R2L)，Denail of Service(DoS)，Probing，Insider の 4 種類に分類した．表 2.1 はネットワークインシデントの種類と，それによって引き起こされる情報資産の侵害を機密性，完全性，可用性の 3 点からそれぞれまとめた．

Remote to Local (R2L) とは，あるホストに対して権限を持たない外部ユーザが不正にホストを捜査する権限を取得するネットワークインシデントである．ネットワークの不正アクセス全般と言い換える事もできる．被害ホストが提供しているサービスの脆弱性を利用して不正に情報を取得する（機密性の侵害），サービスの内容を改ざんする（完全性の侵害），サービスを利用できないようにする（可用性の侵害）が挙げられる．これは人間による攻撃だけではなく，コンピュータウィルスやボットなどのマルウェアによる攻撃も

表 2.1: インシデントの種類と情報資産への影響

インシデントの種類	機密性	完全性	可用性	具体例
Remote to Local (R2L)	o	o	o	不正アクセス，マルウェアの感染
Denial of Service (DoS)			o	SYN Flood 攻撃，無意味なデータの大量送信
Probing	o			ディレクトリハーベスト攻撃
Insider	o	o	o	機密情報の書き換え，持ち出し．情報の流出

含む．また，攻撃者側ホストから接続を開始する能動的な攻撃だけではなく，被害ホストから Web サービスなどに接続し，マルウェアに感染する事例も R2L に分類する．R2L で得られる被害ホストの権限は様々だが，任意のコードが実行できる権限を取得された場合に機密性，完全性，可用性全てに影響を及ぼす可能性があるため，深刻なネットワークインシデントである．

Denail of Service (DoS) は計算機，あるいはネットワーク資源に対して過剰な要求を送信する，あるいは脆弱性を利用した攻撃を実行することで，サービスの可用性が低下，あるいは侵害されるネットワークインシデントである．一般的に 1 カ所からの攻撃を DoS，複数箇所からの攻撃を Distributed DoS (DDoS) と呼ぶが，本論文ではまとめて DoS として扱う．ネットワーク資源への DoS は主に帯域の占有と中継機の処理能力の占有である．どちらも過剰にパケットを送信することで，他のホストがネットワークを利用できなくなる．一方，計算機資源への DoS は様々である．代表的な例として TCP の接続確立要求である SYN パケットを大量に送信する SYN Flood 攻撃や，ICMP スタックの脆弱性を利用した Ping of Death 攻撃 [13] が挙げられる．他にも正規の通信を大量に発生させて処理能力を低下させ，可用性を侵害する．

Probing は調査行為により，秘匿すべき情報が攻撃者側に伝わってしまうネットワークインシデントである．代表的な例としては，メールサーバに大量のユーザ名を問い合わせることで，有効なメールアドレスを取得するディレクトリハーベスト攻撃である．有効なメールアドレスの一覧は迷惑メールの送信やマルウェアの送信に利用されるため本来は秘匿すべき情報だが，これが外部の攻撃者に漏洩してしまうことで機密性の侵害と言える．他にもサービスの設定不備により，秘匿すべき情報が外部から参照されてしまうなどが挙げられる．また，受動的に攻撃を受けるだけではなく，能動的に機密情報を送信してしまう事例としてフィッシングも Probing の一種となる．

Insider はエンドユーザが自らの権限を持って情報資産の機密性，完全性，可用性を侵害するネットワークインシデントである．既に持っている権限と脆弱性を利用した不正アクセスを両方利用したネットワークインシデントは，R2L との複合と見なせる．具体的には，エンドユーザが自らの権限を利用した機密情報の持ち出しによる機密性の侵害，書き換えによる完全性や可用性の侵害などが挙げられる．P2P ファイル交換ソフトウェア

表 2.2: ネットワークインシデントの種類と情報資産への影響

対策の種類	対策内容	具体例
事前対策	防御	ファイヤーウォール
	抑止	機密情報をネットワーク接続性が無い環境へ移す
	回避	規則，法的措置による対応の明確化
	予防	ペネトレーションテスト
	検出	IDS による監視
事後対策	反応	連続したパスワード試行に対するアクセス拒否
	軽減	サーバの停止，ネットワークの切断
	調査	ログの調査
	回復	サーバの再インストール

や誤操作による機密情報の流出も，これに含まれる．ただし，1 つのホスト内で完結している情報資産の侵害についてはネットワークインシデントに含まないものとする．

2.1.3 ネットワークインシデントの対策

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ことが大きな目的となる．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を支払うと同時に，多くの利便性を犠牲にしなければならない．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．表 2.2 に事前対策と事後対策の例をまとめる．

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，予防が挙げられる．防御とは攻撃者の活動を制限することで，攻撃の実行を困難にする対策である．抑止とは脅威そのものを取り除くことによって，ネットワークインシデントの発生を防ぐ対策である．回避とは攻撃者側の費用対効果を悪くすることで被害の発生を防ぐ対策である．予防とはネットワークインシデントの要因を取り除くことで，発生を防ぐ対策である．複数の手法を導入することによって，防げる攻撃の種類と確実性は増す．ただし，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多々ある．

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，回復が挙げられる．反応とは攻撃者に対して活動を制限するなどの対応である．軽減とは情報資産に対してネットワークインシデントによる被害を極小化するための対応である．調査とは残された痕跡を分析し，ネットワークインシデントの発生要因や経

過時間などを調べる対応である．回復とはネットワークインシデントによって停止，あるいは不具合を起こしているサービスの可用性を戻す対応である．

検出は事前対策と事後対策の中間に位置し，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出，対応できれば，実質的な損害を出さない事も可能である．その理由として，ネットワークインシデントの被害が徐々に進行する点と，情報資産の重要度が等価ではない点があげられる．ネットワークインシデントの発生後，時間とともに被害をうける可能性がある情報資産は増えるが，全ての被害が同時に発生するわけではない．侵攻にかかる時間はネットワーク規模や環境，各ホストの環境，ネットワークインシデントの発生防止対策（抑止，防御，回避，予防）の導入状況によって異なるが，全ての情報資産が同時に掌握される可能性は少ない．そのため，対応までの時間が短ければ大部分の損害の発生を抑制できる．また，保護すべき情報資産は全て等価ではない．重要度は様々であり，ネットワークインシデント発生直後に重要度の高い情報資産が危機にさらされるとは限らない．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．そのため，検出は最も重要な対策の 1 つであると言える．

2.1.4　ネットワークインシデントの原因となる脅威の分類

ネットワークインシデントの要因となる脅威について，活動の形態と攻撃対象の選択方法の 2 点から指標を設ける．ネットワークインシデント対策は様々な手法があるが，対策にかかるコストや対策が有効に機能する脅威はそれぞれ異なる．そのため対策の考案において，対策の対象とする脅威を限定することは重要となる．

一つは脅威の攻撃手法について，活動をあるパターンで示せる**定型的**と，多様的でパターンでは示しにくい，もしくは未知なパターンである**変則的**に分類する．コンピュータウィルスの感染活動は定型的の典型である．また，0-day 攻撃は変則型に分類される．SQL インジェクション [14] は攻撃に利用されるデータの内容は攻撃毎に異なるが，データの形式には共通点があるため，定型的な性質を含んだ変則的脅威と言える．

もう一つは**目標型**と**無差別型**である．無差別型はボットやコンピュータウィルスのように攻撃対象の特性を無視して活動する脅威である．一方，目標型はスピア型攻撃のように攻撃対象を選別する脅威である．この 2 つの違いは攻撃対象の特性を考慮した上で活動するか否かである．攻撃される対象は各々が特有の弱点を持つため，それを利用した攻撃は成功する可能性が高い．

一般的には定型的かつ無差別型の脅威は，攻撃側も対策側もコストが少なくなりやすい．攻撃側は特に工夫しなくとも，少数の攻撃プログラムを繰り返し実行するのみである．対策側も無差別に発生している攻撃であれば，他のネットワークで検出された事例をもとにできるため，検出が容易になる．一方で変則的かつ目標型の脅威は，攻撃側も対策側もコストが高くなりがちである．

2.2 セキュリティイベント

セキュリティイベントとはネットワークインシデントに関連する可能性がある事象の総称である．本論文ではネットワーク上で発生した事象を中心に扱っている．セキュリティイベントの粒度は任意である．そのため，非常に細かい事象から，広い範囲での事象を示すセキュリティイベントまで様々なものが存在する．複数のセキュリティイベントから一つのセキュリティイベントを導き出す場合もある．セキュリティイベントを表すための共通フォーマットとしてIntrusion Detection Message Exchange Format (IDMEF)[15] などが提唱されているが，一般に普及していない．そのため，セキュリティイベントの粒度や表現方法は出力するソフトウェアによって様々である．

2.2.1 セキュリティイベントの種類

不正侵入の試み　不正な手段によって権限を取得する試み．具体的には，ソフトウェアが持つ脆弱性を利用して不正に認証手順を省略し権限を取得しようとする試みや，識別情報（パスワードなど）を，不正に取得しようとする行為が挙げられる．ソフトウェアのバッファオーバーフロー脆弱性を利用した攻撃 [16] や，脆弱性を持つ Web サービスへの攻撃，自律的に他のホストの脆弱性を利用して感染を拡大させるタイプのコンピュータウィルスが分類される．また，パスワードの総当たり攻撃（ブルートフォースアタック）も挙げられる．

調査活動　内部ネットワークの構造や，各ホストに関する情報を収集する活動．外部から到達できる内部ネットワークのホストの IP アドレス，提供しているサービスの種類，使用している OS，使用しているソフトウェアの種類やバージョンなどを様々な手法によって取得する．特定のツールの利用による調査はトラフィックに特徴的な部分があるため，検知が可能である．しかし，正常な通信に見せかけて調査する手法もあるため，全ての調査行為を正確に検知するのは困難である．

プロトコル異常　Request For Comment (RFC)[17] などで定められた通信プロトコルから，逸脱したトラフィック．具体的には特定プロトコル中での使用禁止文字の発見や，送信データのフォーマット異常，通信手順の異常などが挙げられる．例えば Simple Mail Trannsfer Protocol (SMTP)[18] や Post Office Protocol3 (POP3)[19] などのテキストベースのプロトコルでは，送信するコマンドと引数が全て ASCII[20] の英数字および記号のみで構成されるものとして，RFC で標準化されている．そのため，SMTP や POP3 の通信中に英数字や記号以外のデータが含まれていた場合，何らかの異常が発生していると判断し，セキュリティイベントとして検知する．プロトコル異常は未知の攻撃などを検知できる可能性が高い．しかし，プロトコルを正確に実装していないアプリケーションも少なくないため，リスクの無いトラフィックもセキュリティイベントとして多数検知されてしまう傾向がある．したがって，プロトコル異常を示すセキュリティイベントの多くは誤検知を前提としており，専ら状態異常による検知やインシデント発覚後の調査に利用される．

不審なトラフィック　プロトコル異常ではないが，希にしか観測されないようなトラフィック．通常利用されない特殊なプロトコルや，異常なサイズのパケットなどがこれに相当する．このような種類のトラフィックは悪意のあるものばかりではない．例えば，エンドユーザが特殊なソフトウェアを使用していたり，データ転送中に偶然エクスプロイト攻撃に類似したトラフィックが流れる可能性もある．このセキュリティイベントもプロトコル異常のセキュリティイベントと同様，過剰検知を前提としており，主に状態異常による検知やインシデント発覚後の調査に利用される．

規約違反　規約によって管理ネットワーク内で利用を禁止されているソフトウェアによるトラフィック．特に企業ネットワークにおいては業務に関連しないアプリケーションの利用を禁止している場合が多い．例としてチャットソフトウェア，ネットワーク対戦型ゲーム，P2P ファイル交換ソフトウェアなどが挙げられる．また，企業などの組織では，業務とは関連しないサイトの閲覧を禁じている．このようなソフトウェアの利用や，サイト閲覧の際に発生するトラフィックも規約違反に含まれる．

悪意のあるソフトウェアの活動　エンドユーザが意図せずインストールされたプログラムによる不正な活動．具体的にはマルウェア（コンピュータウィルス，ボット，スパイウェア）の活動が検知される．また，電子メールなどを経由して感染するコンピュータウィルスもこれに分類される．内部の管理ネットワーク内から外部に向かうトラフィックを検知することで，既に内部ネットワークで活動している悪意のあるプログラムを発見できる.「禁止されている行為」でも特定ソフトウェアの利用を検知するが,「悪意のあるプログラムの活動」との違いは，エンドユーザが意図して利用しているか否かという点である．本論文における悪意あるソフトウェアとは，エンドユーザが意図して利用していない，あるいは一部で意図しない動作をしているものを指す．内部にいる悪意のあるソフトウェアの活動検知については “EXTRUSION DETECTION”[21] や “Outbound Intrusion Detection”[22] において述べられている．

サービス妨害攻撃　大容量のトラフィックや大量のリクエストを送信することで，ネットワークインフラストラクチャの障害を引き起こしたり，ソフトウェアを過負荷な状態にして，サービスの提供を阻害する攻撃．一般的に Denial of Service(DoS) 攻撃と呼ばれている．DoS の検知には DoS を目的として作成されたソフトウェアが送信するトラフィックの特徴や，通常より大きいサイズのパケットの発見によりセキュリティイベントとして検知する．

2.2.2　セキュリティイベントの活用

現状のネットワークインシデント検知はセキュリティイベントの**検知**と**分析**の 2 つの機能に分類できる．検知のみでネットワークインシデントを検知できる種類のセキュリティイベントや，複数のセキュリティイベントの分析により導出されるセキュリティイベント，

検知と分析を経ることで意味を持つセキュリティイベントなど，検知と分析の組み合わせは様々である．

検知はネットワークトラフィックの監視により，セキュリティイベントを発見する．セキュリティイベントを出力するシステムをセキュリティイベント検知機構，もしくはセキュリティデバイスと呼ぶ．セキュリティイベント検知機構は様々であり，IDSや侵入防止システム（Intrusion Prevention System, IPS），ファイヤーウォール，トラフィック流量監視装置，各通信装置が挙げられる．多くのセキュリティイベント検知機構は内部ネットワークと外部ネットワークの境界を監視する設計となっている．監視地点を限定した方が，設備の投資が少なくてすむ．ホスト上にあるサーバ型プログラムのログもセキュリティイベントとして利用できるが，ログを収集するためには当該ホストに専用の設定をしなければならない．また，個人用ホストではプライバシーに関わる情報が含まれる場合もある．そのため運用コストなどを考慮すると，主にサーバなどの一部のホストからのみ収集するのが一般的な運用である．

分析はセキュリティイベントの取捨選択や，複数のセキュリティイベントから新しいセキュリティイベントを出力する機能である．具体的な手法として相関分析や，セキュリティイベントの可視化，統計的分析，セキュリティイベントのリスク評価が挙げられる．最終的にセキュリティイベントがネットワークインシデントであるかどうかは，システムを利用しているネットワーク管理者によって判断される．

本研究ではセキュリティイベントの検知に着目している．セキュリティイベントからネットワークインシデントを検出するためには，有効なセキュリティイベントの収集が重要となる．ネットワークインシデントと関連する可能性が低いセキュリティイベントを大量に検知したとしても，分析によって重要なセキュリティイベントを抽出するのは難しい．そのため本研究は，有効なセキュリティイベントの検知を目的としている．

2.3 既存のセキュリティイベント検知手法

エンドユーザが利用するホストとは別に，内部ネットワークに設置するインシデントの事前対策，発見，事後対応を目的としたシステムをセキュリティイベント検知機構と呼ぶ．セキュリティイベント検知機構は，セキュリティイベントの検知を目的としてした実装以外に，付加的な機能としてセキュリティイベントを検知している実装がある．

初期のセキュリティイベント検知機構は1990年に発表されたネットワークから異常な性質のトラフィックを検出するアノーマリ型の検知システム[23]であった．その後，1994年に最初の商用IDSであるNetRanger[24]が登場し，1998年には最も有名なオープンソースIDSの1つであるSnort[25]が公開された．その後，様々なセキュリティイベント検知機構が提案，実装されてきた．

2.3.1 主要なセキュリティイベント検知手法

既存のセキュリティイベント検知手法から，主要な手法としてミスユース型 IDS，アノーマリ型 IDS，ファイヤウォール，IPS，ホスト型 IDS を挙げ，これらの比較を表 2.3 に示す．

表 2.3: 主要なセキュリティイベント検知手法一覧

	監視点	検知方法	自動反応
ミスユース型 IDS	ネットワーク	パターンマッチ	なし
アノーマリ型 IDS	ネットワーク	閾値	なし
IPS	ネットワーク	パターンマッチ	あり
ホスト型 IDS	ホスト	パターンマッチ	設定による

ミスユース型 IDS　ミスユース型 IDS とはネットワークのトラフィックを監視し，あらかじめ定められた特徴を持つトラフィックを発見する検知手法である．まず，セキュリティイベントであると考えられるトラフィックの特徴を定義したシグネチャを用意する．シグネチャには送信元および宛先 IP アドレス，送信元および宛先ポート番号，トラフィックに含まれる文字列のパターン，その他 IP ヘッダや TCP ヘッダの各項目などを，1 つもしくは複数指定できる．このシグネチャと監視しているトラフィックの特徴を比較し，一致すれば IDS はセキュリティイベントを発見したものとする．あらかじめ定められた特徴によってセキュリティイベントを検知するため，検知結果は概ね正確だが，未知のセキュリティイベントを検出しにくいという特性がある．本論文において IDS とはミスユース型 IDS を指す．

アノーマリ型 IDS　アノーマリ型 IDS とはネットワークのトラフィックを監視し，瞬間的なトラフィック流量や，宛先 IP アドレスの数，宛先ポートの分布状態のような，トラフィックの傾向にそれぞれ閾値を設定し，その閾値を超えるトラフィックが発生した場合にセキュリティイベントが発生したと見なす手法である．閾値を適切に設定できれば，管理ネットワーク上のトラフィック異常を発見できるため，未知の脅威によるセキュリティイベントも検知できる場合がある．ただし，閾値が適切でなければ誤検知が多発してしまう．したがって，ネットワーク管理者は当該ネットワークのトラフィック傾向を十分に理解し，それにもとづいて閾値を設定しなければならない．また検知されたセキュリティイベントが，具体的にどのような脅威によって引き起こされたのかが明確ではないという性質を持つ．

IPS　侵入防止システム (Intrusion Prevention System, IPS) とはネットワークを監視し，悪意のあるトラフィックを遮断するシステムである．遮断すべきトラフィックであるかの

判断方法はミスユース型 IDS と同様にトラフィックのパターンに着目している．基本的には悪意ある通信の遮断が目的だが，遮断時に残したログをセキュリティイベントとして扱える．遮断の方法としては，中継機として動作しながら必要に応じて通信を遮断するインライン型や，必要に応じて TCP のリセットパケットや ICMP の Port Unreachable メッセージを送信するパッシブ型が挙げられる．

ホスト型 IDS　各ホスト上で動作し，アプリケーションのログや OS のログなどから不審な点を発見するシステム．ログの出現順序に不審な点がある場合や，許可されていないファイルの変更・改ざんを発見し，セキュリティイベントとして検知する．ホスト上の情報を利用できるため，ハードウェアや OS が保持しているプロセスの情報などを容易かつ正確に取得できる．ただし，RootKit などを利用された場合，ユーザランドからの監視が意味を為さなくなる場合もある．

2.3.2 主要なセキュリティイベント検知手法の比較

既存の手法の中では，ミスユース型 IDS が比較的利用しやすいセキュリティイベント検知機構となっている．あらかじめ定義したセキュリティイベントの特徴を利用した決定的な検知手法であるため，定型的な脅威に対しては正確に決定的な検知がしやすい．ネットワークトラフィックを監視するため各ホストに対する運用負担も少なく，ネットワークインシデント検出に対するコストは比較的少ない．

アノーマリ型 IDS はネットワーク管理者の負担が大きく，さらにネットワークやコンピュータについての知識と経験が求められてしまう．閾値の設定に必要となる正常状態とはネットワークによって様々であり，ネットワークの管理者は自ら正常状態や閾値を判断しなければならない．これに対してミスユース型検知で用いるシグネチャは，他のネットワーク管理者が作成したシグネチャでも転用できるものが多い．なぜならば，コンピュータウィルスやプログラムを用いた攻撃によって生じるトラフィックの特徴は，ネットワークによって変化しにくいためである．

ホスト型 IDS は管理ネットワーク内のホスト数が増加すると，管理の負担が大きくなってしまう．ネットワーク管理者が対策を考える場合にはホスト型 IDS の運用は難しい．ホスト型 IDS を動作させているホストを全て管理下におき，適切に設定・管理しなければならないため，特定・不特定にかかわらず多数のホストで運用する場合には負担が増大する．そのため少数の重要なホストで運用する以外は，運用負担と効果のバランスが悪い．

IPS はミスユース型 IDS の機能を包含しているが，自動的にセキュリティイベントに対応するため攻撃者に検知した事実を伝えてしまうという短所がある．どのようなトラフィックに反応したかによって，利用している実装を攻撃者に推測されてしまう．実装によっては検知できないセキュリティイベントがあるため，攻撃者はこれを利用して検知を回避してしまう可能性がある．遮断した情報はネットワークインシデント検出の参考として利用できるが，回避される危険性があるため IPS だけでは不十分である．

以上のような理由から，本論文執筆時点ではセキュリティイベント検知手法としてはミスユース型 IDS が主流となっている．

2.3.3 セキュリティイベント検知における技術的な課題

多くのセキュリティイベント検知機構には幾つかの課題があり，ネットワークインシデント検出の弊害となっている．本節では最も一般的な False Positive, False Negative, 対策済み検知，対応するネットワークインシデントが不明確な検知の 4 つについて述べる．これらの課題を取り除くためには，いずれもネットワーク管理者がセキュリティイベント毎に調査を実施する必要があり，運用コストを引き上げる大きな原因となっている．

False Positive 本来は検知対象ではない事象を，検知対象であると誤認してセキュリティイベントを発生させてしまう誤検知．原因としては検知条件の曖昧さ，検知アルゴリズム上の課題の 2 つが挙げられる．ルールに基づいて検知する場合，条件を厳しくすれば検知の精度は上昇する．しかし，対象とするセキュリティイベントの詳細が不明である，あるいは一般的な特徴が無いなどの理由で，曖昧な条件を記述せざるを得ない場合がある．また検知アルゴリズムがなんらかの閾値を元にしている場合，ネットワーク環境に応じて適切に閾値を設定する必要がある．閾値の設定は環境や時期などにも影響されるため，僅かな変化で False Positive を多発してしまう．

False Negative 検知対象であるはずの事象を見逃してしまう誤検知．この課題も検知ルールと検知アルゴリズムによる課題である．検知ルールの課題は False Positive とは逆で，ルールの条件が厳しすぎて検知できない場合が挙げられる．False Positive を減らすために可能な限り厳しいルールで書くべきだが，厳しすぎるルールは検知対象の一般化が適切に行われず，見逃しを誘発してしまう．さらに，ルールに基づいた検知では，全く知らない種類のセキュリティイベントは検出できない可能性が高い．検知アルゴリズムの課題も同様に，False Positive を防ぐための措置が False Negative を引き起こす場合が多い．閾値の設定が厳しすぎて，検知対象となる事象が発生しても検知しなくなってしまう．また，IDS の回避攻撃 [26] によっても False Negative が引き起こされる可能性がある．

対策済み検知 ホストに不正侵入を試みる攻撃など，情報資産の侵害を起こす可能性として検知されているが，既にホスト側で必要な対策がとられており，ネットワークインシデントが発生しないセキュリティイベント．例えば，ある脆弱性を利用した攻撃をセキュリティイベントとして検知したとしても，対象ホスト側でその脆弱性に対策が施されていれば，攻撃は失敗する．しかしホスト側で影響がなかったとしても，ネットワーク管理者がこれを認識する方法がなければ，攻撃の結果について対象ホストを調査しなければならない．

対応するネットワークインシデントが不明確な検知 検知はされているが，それがどのような脅威によって引き起こされたのか，あるいは本当にネットワークインシデントに関連しているのかが明確ではないセキュリティイベント．例えば，普段は観測されない種類のプロトコルによる通信がセキュリティイベントとして発見された場合，ネットワークイン

シデントに関連して攻撃者が発生させた通信なのか，あるいは利用者がたまたま新しく利用したソフトウェアの通信だったのかを判断するのは難しい．

2.3.4 セキュリティイベント検知における運用的な課題

セキュリティイベント検知においては，運用上の課題も大きい．特に第 2.3.3 節で述べた課題によって引き起こされる調査の負担は大きく，ネットワークインシデント検出の有効性が著しく低下してしまう．第 2.1 節で示したとおり，ネットワークインシデント発生後の事後対策を速やかに実行するためには検知が不可欠であるため，検知の有効性が下がることで事後対策全体の有効性を下げてしまう可能性がある．

検知に必要となる負担 セキュリティイベントの検知には運用負担がかかり，特にホスト上で検知する際に負担が大きくなりやすい．ネットワーク管理者の立場から考えると，ネットワークに接続しているホストからセキュリティイベントを収集するのは負担が大きい．大学のキャンパスネットワークのように接続しているホストが管理下にないネットワークでは，全てのホストにセキュリティイベント検知機構を導入するのは難しい．また，セキュリティイベント検知機構を導入していたとしても，エンドユーザとネットワーク管理者間に信頼関係があるとは限らず，セキュリティイベントが提供されない可能性もある．一方，全てのホストが管理されているネットワークでもセキュリティイベントを収集するのは負担が大きい．セキュリティイベント検知機構とホスト毎に導入したソフトウェアとが競合してしまう課題や，セキュリティイベント検知機構が正常に動作しない課題が発生する可能性がある．また処理負荷などの理由から，エンドユーザが意図的にセキュリティイベント検知機構を無効化してしまう恐れがある．ホスト数が多くなるほど，このような状況に対応するための負担が大きくなってしまうという課題がある．

調査に必要となる負担 第 2.3.3 節で述べたとおり，誤検知や対策済み検知，明確ではない検知の発生数上昇に伴って運用コストが大きくなる傾向にある．全てのホストが管理下におかれていないネットワーク（例えば大学のキャンパスネットワークなど）では当該ホストの所有者に了承を得て調査する負担は非常に大きく，頻繁に実行できない．全てのホストが管理下にあったとしても，実際にログインするなどの調査負担は少なくない．時間経過と共にホストがネットワークを離脱する可能性が高くなってしまうなど，様々な面で課題があり，運用負担が大きくなってしまう．

知識・経験の必要性 検知・調査の負担とも密接に関係するが，セキュリティイベントの調査のおいては，ネットワーク構築や計算機についての知識・経験の他に，ネットワークインシデントに関する知識・経験の高さが必要となる．ネットワークインシデントに関する知識とは攻撃の手法や脅威の種類，攻撃者の行動習性などを指す．ネットワークインシデントに関する経験とはネットワークインシデント検出作業の対応経験や，インシデントレスポンスの実施経験，あるいは演習の経験などを指す．関連するネットワークインシデントが明確ではないセキュリティイベントが検知された場合，十分な知識と経験がなけれ

ば，それがどのような影響を及ぼしているかを推測するのは困難である．知識や経験を取得するためには時間をかけた学習や演習，実務経験が必要となる．同時にネットワーク構築や計算機についての知識・経験も持たなければならず，修得には大きな負担が必要となる．

2.4 まとめ

ネットワークインシデントの検出は，被害を極小化するために重要な対策である．確実にネットワークインシデントを防ぐのは事実上不可能であるため，被害を最小限に抑えるためにはネットワークインシデントの発生を前提として対策を用意しなければならない．ネットワークインシデント発生後の対策として反応，軽減，調査，回復などが挙げられるが，いずれもネットワークインシデント発生からの時間が短いほど，効果が高くなる．ネットワークインシデント発生後，早急に察知して対応の開始を促す検出は事後対応にとって重要な役割を持つ．

本研究はネットワークインシデント検出手法の中でも**セキュリティイベントの検知**に着目する．ネットワークインシデント検出手法はセキュリティイベントの検知と分析の 2 段階に分けられる．検知はホストやネットワークからセキュリティイベントを収集し，分析は複数のセキュリティイベントやホスト，ネットワークの情報からネットワークインシデントを検出する．そのため，高精度に注目すべきセキュリティイベントが検知されていなければその後の分析は困難になってしまう．

第3章　セキュリティイベント検知における現状の問題点

ネットワークトラフィックの監視によるセキュリティイベント検知手法が発達したことで，意図的に検知の回避を試みるソフトウェアが多く出現している．例としてマルウェアの活動が挙げられる．マルウェアは感染したホスト上で長時間活動を続けられるほど，攻撃者にとって有益である．長期間感染できれば機密情報が記憶装置に入出力される可能性が高くなる，あるいは別ホストへの感染活動が継続できるなどの利点がある．

検知を回避するソフトウェアは，従来のセキュリティイベント検知機構がトラフィックの特徴に着目している点を利用して回避を試みる．第2.3節で述べたように，セキュリティイベント検知機構はIDSやIPS，ファイヤーウォールなど決定性の高いものが多く普及している．これらのセキュリティイベント検知手法では，通信内容に含まれる特徴に着目している．具体的には各プロトコルのヘッダ情報やペイロード部分のパターンが挙げられる．この2つをトラフィックパターンと呼ぶ．セキュリティイベント検知機構ではトラフィックパターンを1つ，あるいは複数条件として指定し，トラフィックをセキュリティイベントであるか判断する．

このようなセキュリティイベント検知機構の普及に対し，検知を回避するための具体的な方法として**他のソフトウェアと類似した通信**と**通信の暗号化**の2つが挙げられる．この2つを効果的に利用された場合，セキュリティイベント検知機構の問題の1つである誤検知や曖昧な検知を多発してしまう可能性が高く，セキュリティイベント検知の有効性が低下してしまう．

3.1　他のソフトウェアと類似した通信

従来の検知手法は，各セッションに含まれるソフトウェア固有の特徴やプロトコルに着目している．そのため，本来の検知対象とは異なるプロトコルが利用された場合，固有の特徴やプロトコルが異なるため，正確な検知は難しくなる．特にボットの場合，自身のソフトウェアアップデートや悪意のある活動の指示を受信するために，HTTPやSMTPといった異なる目的のプロトコルを利用し，別のソフトウェアの通信に見せかける．既存手法では，これらの脅威が発生させるセキュリティイベントを検知しようとすると，他の無害な通信も検知してしまう可能性が高い．このような検知結果を検証するためには，大量のトラフィックの情報を保存しておかなければならない．さらに多くの人的負担が必要となるため，実際の運用においては効果が期待できない．

クライアント(ボット)による送信	サーバ(C&C)による送信
NICK FRE-799644257 USER vbdmfqbckm 0 0 :FRE-799644257	
	:ns1.xxx.us 001 FRE-799644257 :Cisco :ns1.xxx.us 005 FRE-799644257 :ns1.xxx.us 422 FRE-799644257 : :FRE-799644257 MODE FRE-799644257 :+i
USERHOST FRE-799644257	
	:ns1.xxx.us 302 FRE-799644257 :FRE-799644257= +vbdmfqbckm@xxx.xxx.xxx.xxx
MODE FRE-799644257 +x+i JOIN #dd dpass USERHOST FRE-799644257 MODE FRE-799644257 +x+i JOIN #dd dpass	
	:FRE-799644257!vbdmfqbckm@xxx.xxx.xxx.xxx JOIN :#dd :ns1.xxx.us 332 FRE-799644257 #dd :xvvv msass 150 0 0 -b -r -s :ns1.xxx.us 333 FRE-799644257 #dd SWp 1191207793 :ns1.xxx.us 353 FRE-799644257 @ #dd :FRE-799644257 @qqq :ns1.xxx.us 366 FRE-799644257 #dd :End of /NAMES list. :ns1.xxx.us 302 FRE-799644257 :FRE-799644257= +vbdmfqbckm@xxx.xxx.xxx.xxx
PING :ns1.xxx.us	
	PONG :ns1.xxx.us
PING :ns1.xxx.us	

図 3.1: ボットと C&C サーバの IRC を用いた通信の例

多くのボットは一般的なプロトコルを利用して，Command & Control(C&C) サーバと通信をしている．これは Honeynet Project による調査結果 [27] や Cyber-TA Research and Development Project による調査結果 [28]，“Malicious Bots Threaten Network Security” [29]，“A multifaceted approach to understanding the botnet phenomenon” [30] などの論文において示されている．特に多いのが IRC を利用した通信である．送受信される命令も，一般的な IRC によるチャットに出現しやすい文字列が多く，一般的な IRC 通信と区別するのが難しい．一方でボットは HTTP[31] を利用し，命令の送受信や自身のプログラムの更新を行っている．HTTP を利用した命令の送受信ではサーバ側で CGI を用意し，ボットがファイル取得要求を送信する．これに対して，C&C サーバは応答に命令内容や命令の実行に必要となるデータを送信する．

図 3.1 ではボットと C&C サーバが IRC を利用して通信している内容の一部を抽出したものである．動作しているボットは Kaspersky Free online virus scan[32] において Backdoor.Win32.Rbot.cqe と判定された検体である．図の左部がクライアント（ボット）から

の送信内容，右部がサーバ（C&C サーバ）からの送信内容となっている．ボットと C&C サーバの通信は IRC のプロトコルに則った通信であり，各送信内容に含まれる文字列も一般的な IRC で出現する可能性のあるものとなっている．点線で囲まれた部分は，他のホストに対して Microsoft Windows[33] が提供している NetBIOS のサービスが使用可能になっているかを調査するための命令である．これも各文字列は一般的な IRC の通信に出現する可能性があるため，セキュリティイベントの検知条件として利用するのが難しい．

ボットが引き起こすネットワークインシデントは第 2.1.2 節で示したように，R2L による機密性，完全性，可用性の侵害である．特にホストに保存されている機密情報の流出という点では機密性の侵害が深刻な問題となる．また，OS の重要なファイルを改ざんされる完全性侵害や感染活動トラフィックが帯域や計算能力を圧迫する可用性侵害が挙げられる．これは多くの組織に共通して深刻なネットワークインシデントとなる．

3.2 通信の暗号化

ネットワークの監視では，暗号化された通信からパケットやセッションに含まれる特徴を判断するのは極めて難しい．暗号化されていないレイヤから部分的に情報を得られるが，セキュリティイベント検知に利用するレイヤが暗号化されている場合がほとんどである．

暗号化された通信を利用する例として，P2P ファイル交換ソフトウェアを挙げる．暗号化した P2P ネットワーク網によってボットネットを形成する種類のボットも出現してきているが，P2P ファイル交換ソフトウェアも同様に隠蔽を目的とした実装となっているため，事例として有意であると判断した．交換しているファイルの秘匿性を保つため，通信内容を暗号化している実装が多く，検知の弊害となっている．BitTorrent[34] や eDonkey[35]，Gnutella[36] のような一部のソフトウェアでは，当該ソフトウェアの特徴が暗号化されていないため，従来の手法でも検知できる．しかし，Share や Winny[5] では特徴も含めて通信を暗号化している．トランスポート層のポート番号も容易に変更でき，隠蔽を目的とした設計がなされている．そのため，トラフィックパターンから当該ソフトの利用を判断するのは難しい．

解読済みのソフトウェアから隠蔽のための暗号化の事例を見る．ここでは日本国内で多く使用されている P2P ファイル交換ソフトウェアとして Winny[5] と Share を取り上げる．

Winny のプロトコルは製作者である金子氏が著書 [5] において公開している．Winny はコネクションを確立した後に，暗号キーを送受信する．パケット形式としては TCP のデータセグメントの先頭 2 バイトがダミーで，続く 4 バイトが RC4 の暗号キーとなっている．Winny 本体の公開直後は暗号化アルゴリズムを秘匿し，暗号キー送信の前に 2 バイトのダミーデータを混ぜ，検知を困難化している．

Share の暗号化方式は鵜飼氏が 2007 年 1 月に解読し公開している [37]．Share は 1024 ビット RSA 公開鍵暗号と RC6 を併用している．ファイルの送受信などには RSA 公開鍵を利用している．公開鍵を交換する前に幾つかの情報を交換しているが，これも暗号化されているため，通常のトラフィックパターンを条件に使う手法では検知が困難である．

これらの P2P ファイル交換ソフトウェアは第 2.1.2 節で示した Insider のネットワークインシデントを引き起こす可能性を持っている．Antinny[38] などのコンピュータウィルスは，P2P ファイル交換ソフトウェア経由で感染を広げ，感染ホスト内にある機密情報を P2P ファイル交換ネットワーク上に漏洩する．漏洩された機密情報を全て回収するのは事実上困難であり，多くの組織において大きな被害をもたらす可能性がある．

3.3 まとめ

ネットワーク型のセキュリティイベント検知機構が普及したのに対し，これを回避するソフトウェアが出現している．回避方法は主に 2 つで通信の暗号化と他のソフトウェアで用いられる通信プロトコルの使用が挙げられる．現状で，決定性が高いセキュリティイベント検知機構である IDS はトラフィックパターンを条件として検知しているが，この 2 手法を用いられた場合には検知が困難になってしまっている．これらのソフトウェアによるセキュリティイベントを検知するためには，ネットワークインシデントとの対応が不明確な条件か，もしくは精度の低い条件を利用しなければならない．この 2 つはネットワーク管理者の負担を増やすものであり，ネットワークインシデント検出の弊害となってしまう．

第4章　関連研究

本章では IDS などによる検知の困難化手法を導入しているソフトウェアに対して，検知を試みている関連研究を取り上げる．ネットワーク管理者が運用するセキュリティイベント検知機構に検知されないために，幾つかのソフトウェアが**他のソフトウェアと類似した通信の使用**と**通信の暗号化**を利用している事を，第3章で示した．この2つの手法をそれぞれ導入している P2P ファイル交換ソフトウェアとマルウェアを検出するための手法や研究について述べる．

P2P ファイル交換ソフトウェアやマルウェアを検出するための手法として，Traffic Classification，Correlation Analysis，Anomaly Detection，Network Behavior Analysis の4つを取り上げる．Traffic Classification は P2P ファイル交換ソフトウェア，Correlation Analysis，Anomaly Detection はボットを中心としたマルウェアを，Network Behavior Analysis は両者の検知を主な目的としている．それぞれに特徴と問題点について説明する．

4.1　Traffic Classification

Traffic Classification はプロトコル毎に異なるトラフィックの傾向を利用してプロトコルの種類を特定する手法である．“Internet Traffic Classification Using Bayesian Analysis Techniques”[39] はベイズ推論を利用し，パケット長の分布などから通信をしているプロトコルの種類を推測する．P-CUBE[40] でも P2P ファイル交換ソフトウェアのトラフィックの特定を目的としている．“BLINC: multilevel traffic classification in the dark”[41] は IP アドレスやポート番号によるフローの発生をパターン化し，通信のプロトコルを判定している．このような手法は基本的にトラフィックエンジニアリングの実現を目的として利用されている．そのため，通信の傾向が P2P ファイル交換ソフトウェアであると判明しても，通信をしている具体的なソフトウェアの特定には至らない．

これらをセキュリティイベント検知に利用する場合，検知結果の検証が必要となる．検知結果はおおよその傾向しか得られないため，具体的にどのようなソフトウェアが利用されているかを特定するためには，別途トラフィックの情報を記録し，分析する必要がある．また，膨大なトラフィックを扱うことを前提にしている場合が多く，精度の向上よりは少ない情報と少ない処理による判定を目指している．そのため検知対象となるソフトウェアの痕跡を発見するためには利用できるが，正確にソフトウェアを特定するためには追加調査が必要となり，ネットワーク管理者の負担が大きくなりやすい．

4.2 Correlation Analysis

従来の IDS やその他の機器から得られたセキュリティイベントを元に，発生順序によってセキュリティイベントを関連づけし，新しいセキュリティイベントを発見する手法を Correlation Analysis 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セキュリティイベントの発生順序を抽出する “SEC - a Lightweight Event Correlation Tool” [42] や，ネットワーク上のセキュリティイベントを関連づけする “NetSTAT: A Network-based Intrusion Detection Approach” [43] が挙げられる．アルゴリズムに基づく方法はセキュリティイベントをクラスタリングする “Clustering intrusion detection alarms to support root cause analysis” [44] が挙げられる．

これは OSSIM[45] や ArcSight[46] 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セキュリティイベントを発見する．Argus[47] 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 Correlation Analysis に利用できる．単独では脅威とならないセキュリティイベントでも，複数のセキュリティイベントを関連づけることで，脅威となるセキュリティイベントを発見できる可能性がある．

ただし，分析に利用するトラフィックやセキュリティイベントは全て保存しなければならない．頻繁に発生するネットワークトラフィックをもとにセキュリティイベントを検知するためには，その全ての情報を一時的に記憶する必要がある．特にネットワークの広帯域化にともなって，膨大な保存領域と高速な記憶処理が必要になる．

さらに，セキュリティ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セキュリティ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，セキュリティ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 [48, 49]．この場合，ある複数のセキュリティ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” [50] ではセキュリティイベントを 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をもとにした行動が組み合わせて起きるという前提であるため，やはり複雑なルールは記述できない．アルゴリズムに基づく場合もセキュリティイベントの傾向が変化した場合に，柔軟に対応するのが難しい．

他にも Snort[25] 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 [51] 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．これはボット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．

また，複数の特徴を関連づける手法として IDS の 1 つである Network Flight Recorder (NFR)[52] が挙げられる．NFR は 1 つのセッション内で発生する複数の特徴を，柔軟に記述する事ができる．ルール内で独自の変数を利用することが可能であり，これによって複

数の特徴の関係性を表現することができるため，複雑なセキュリティイベントを検知できる．しかし，複雑に表現できるのは1つのセッションに限られるため，ホストそのものの挙動を示すのは難しい．

4.3 Anomaly Detection

Anomaly Detection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．これはボット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．Anomaly Detectionの利点は未知のマルウェアでも発見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．Anomaly Detection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ボットとして検出できる．

“A DNS-based Countermeasure Technology for Bot Worm-infected PC terminals in the Campus Network”[53]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”[54]では，DNS問い合わせの異常を検知してボットの検出を試みている．ボット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ボットによる活動だと判断する手法である．“An Algorithm for Anomaly-based Botnet Detection”[55]はTCPパケットの種類の割合を調べ，ボットらしい傾向を発見する．“Security Operation CenterのためのIDSログ分析支援システム”[56]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発見する．

未知のボットに対応できる可能性を持つAnomaly Detectionだが，実際の運用は難しい．基本的にAnomaly Detection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がちである．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が不明な場合が多い．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ら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．

4.4 Network Behavior Analysis

Network Behavior Analysis (NBA)[57, 58] は Lancope 社が製品化しているシステムであり，sFlow[59] 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 つのフローとしてあつかい，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，ボット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っていれば，それを検知できる可能性もある．

4.5 まとめ

セキュリティイベント検知を困難にしている P2P ファイル交換ソフトウェアやマルウェアに対応する手法として，Traffic Classification，Correlation Analysis，Anomaly Detection，Network Behavior Analysis の 4 つについて述べた．しかし，実際の運用場面を考慮した場合，それぞれに課題があると言える．Traffic Classification と Anomaly Detection は精度や検知結果の明確さに問題があり，ネットワーク管理者の負担を増やしまう．Correlation Analysis，Network Behavior Analysis では検知対象の挙動を詳細に記述できない．さらに Correlation Analysis は規模性に問題がある．よって，P2P ファイル交換ソフトウェアやマルウェアのセキュリティイベントを少ない負担で確実に検知するためには，新しい手法を考案する必要がある．

第5章　要求事項

第4章ではセキュリティイベント検知を回避するソフトウェアに対抗するための技術や研究について述べたが，既存の手法では検知において様々な問題があることが明らかになった．そのため，問題解決のために新しいセキュリティイベント検知手法を考案する必要がある．

本章ではセキュリティイベント検知手法を考案するにあたり，検討しなければならない要求事項について議論する．要求事項は検知精度，検知結果の明確性，検知条件の柔軟性，検知処理の規模性の4つを挙げ，達成すべき内容を述べる．

5.1　検知精度

セキュリティイベントを検知する際には，誤検知(False Positive と False Negative)を最小限にしなければならない．第2.3.3節で述べたとおり，誤検知が頻繁に発生するセキュリティイベント検知手法では検知結果に対して調査が必要になり，ネットワーク管理者の負担が大きくなってしまう．特に検知を困難にしているソフトウェアによるセキュリティイベントは，誤検知が多発しやすいことがセキュリティイベント検知機構運用の弊害になっている．本論文では要求事項の中でもとくに検知精度の向上を重要な目的として設定する．

検知精度は，ネットワーク管理者に対する負担を小さくするために必要となる．ネットワーク管理者の負担はある程度までは許容できるが，精度の低さは負担を増加させしまう．負担を許容できる程度は組織によって異なるが，ネットワーク管理者が通常業務とネットワークインシデント対応の両方を担っていると仮定した場合，一般的にネットワークインシデント対策・対応に多く時間をかけることはできない．特に定型的，あるいは無差別型の脅威を対策・対応する場合は可能な限り自動化し，ネットワーク管理者による対応を極力少なくする必要がある．そのため，誤検知発生率の目安としては多くても1日あたり1件，可能であれば数日に1件の割合が望ましいと考えられる．1日あたりのセキュリティイベント数はトラフィック流量，トラフィックの種類，IPアドレスの範囲，時期，組織的特徴によって様々であるが，ネットワーク管理者による運用形態を考慮して基準を設定した．

5.2 検知結果の明確性

検知されたセキュリティイベントは，どのような脅威がどの情報資産に対してどのような被害を与えているかを可能な限り明確にしなければならない．関連する脅威と情報資産，予想される被害が明確になっているセキュリティイベントを，明確性のあるセキュリティイベントと呼ぶ．第 2.3.3 節で明確ではない検知についての問題を述べた通り，明確性の低いセキュリティイベントは運用コストを増加させる．第 2.1.4 節で述べた変則的もしくは目標型の脅威によるネットワークインシデントを検知するためには不明確なセキュリティイベントも必要になるが，本論文では定型的あるいは目標型の脅威を対象としているため，セキュリティイベントとして検知できたとしても運用コストが高いと効果的ではない．

明確性があるか否かを判断する基準として，被害・影響の内容と，影響範囲の特定を挙げる．1 つ目はネットワークインシデントによる被害・影響の特定，あるいはソフトウェアの特定である．具体的な被害の内容が分かれば，自ずと組織に対する影響の度合いも判断できる．またソフトウェアの特定でも，影響を推測しやすい．例えばマルウェアの種類と被害内容は各アンチウィルスベンダにより公開されている．他のソフトウェアでも概ねの動作や影響は知ることができる．2 つ目は被害・影響が起きている範囲の特定である．具体的には影響を及ぼしていると考えられるホストの特定となる．この 2 つを満たすセキュリティイベントを明確性があると見なす．

5.3 検知条件の柔軟性

セキュリティイベント検知機構として，様々なセキュリティイベントを検知できる必要がある．セキュリティイベントとして検知すべき対象は日々増加している．特にボットは亜種が発生するまでの期間が短く，次々に新しい機能が追加されている．P2P ファイル交換ソフトウェアもソフトウェアのアップデートや機能拡張が発生する場合がある．そのため，検知対象をセキュリティイベントと判定するための条件は柔軟に設定できなければならない．

5.4 検知処理の規模性

セキュリティイベントは内部ネットワークで発生しているネットワークインシデントをネットワーク管理者が発見するために利用するため，バックボーンなどの超高速回線での利用は想定していない．インターネット白書 2007[60] によると，2007 年現在では一般家庭における世帯毎のブロードバンド普及率は 41.4%であり，ブロードバンド世帯の 28.2%が FTTH となっている．この普及率は共に増加しており，今後も回線の高速化が進むと予想される．一方，高速回線を導入しクラス C やクラス B 規模のネットワークを運用する教育機関も増加している．インターネット白書 2007 では調査に回答した企業の 31.6%が 100Mbps 以下，9.0%が 1Gbps 以下の回線を有している．計算機の発展により，監視装置

に搭載されている中央演算装置や記憶装置の高速・大容量化や低価格化が進んでいるが，検知に利用できる計算機資源には限りがある．そのため，内部ネットワークの保護を考えたセキュリティイベント検知機構でも，高速化や記憶領域の効率的利用を十分に考慮されなければならない．

具体的な規模性とは，対応可能なネットワーク規模（監視ホスト数，セッション数），1 秒あたりの処理容量 (Byte per Seconds, bps)，1 秒あたりのパケット処理数 (Packet per seconds, pps) の 3 つとなる．今後も高速回線が普及していくことを考慮すると最低でも実行速度が 100Mbps，パケットの平均長を 500 バイトとしたとき 25,000pps，クラス B のネットワークを想定した場合 65,000 台のホストを監視できなければならない．演算機能，1 次記憶容量，2 次記憶容量をそれぞれ圧迫せずに，処理性能を発揮できる必要がある．

5.5 まとめ

新しいセキュリティイベント検知手法を提案するにあたり，検知精度，検知結果の明確性，検知条件の柔軟性，検知処理の規模性の 4 つを考慮しなければならない．検知精度は誤検知の発生率の低さを表しており，誤検知は数日に 1 件以下が理想である．検知結果の明確性は検知結果に明確さが無い状態を表しており，セキュリティイベントから被害の種類と影響範囲を推測できる事が求められる．検知条件の柔軟性とは多様なセキュリティイベントに対応できる仕組みを持つ事である．検知処理の規模性はトラフィックの処理性能であり，具体的には bps, pps，セッション数が一定以上処理可能であり，その際に演算能力や記憶容量を圧迫しないことが条件となる．これらの要求を満たすセキュリティイベント検知手法を提案しなくてはならない．

第6章　複数セッション型ネットワーク監視手法の提案

本論文では，複数セッションの相関関係を利用した脅威の検知手法を提案する．セッションとはプロトコルと目的毎に分類された 2 ホスト間の通信を指す．例えば同じ 2 ホスト間の通信でも TCP，UDP，ICMP ではそれぞれセッションが異なる．さらに，同じ TCP でも送信元・宛先ポート番号によってセッションが異なる．

あるホストから発生する複数のセッションは互いに関連し合う可能性がある．特に，同一のソフトウェアが発生させている複数のセッションは何らかの形で関連が強い場合が多い．一般的にネットワークを利用する大多数のソフトウェアは複数のセッションを発生させる．これは，あるセッションで得られた情報をもとに別のセッションを発生させる場合や，あらかじめ定められた手順に従って複数のセッションを発生させる場合が挙げられる．このようにあるソフトウェアの複数セッションの関係を全体像として把握することで，当該ソフトウェアのネットワーク上での振る舞いを知ることができる．

本論文で提案する手法はソフトウェアのネットワークにおける振る舞いを，関係し合う複数セッションの発生（相関関係）として捉え，セキュリティイベントの検知に利用する．セッションの関係を見る際には既存の IDS と同様にトラフィックパターンにも着目する．これは相関関係のあるセッションを探索する際に，目的のセッションを確実に発見するためである．Network Behavior Analysis では sFlow を使っているため，目的としているセッションに着目しているかどうかを判断するのが難しい．例えば sFlow は TCP のポート番号を扱うため，これを手がかりにある程度のプロトコル判断は可能である．しかし，特にボットなどの場合は，あえて IANA で定められた標準ポート番号 [61] を使わない場合もある．正確な情報を得るためにはトラフィックのヘッダ情報とペイロード情報を効果的に活用する必要がある．そのため，提案手法は従来の IDS と同様のペイロードを含むトラフィックパターンを条件としたセキュリティイベント検知手法と，複数セッションの相関関係を条件としたセキュリティイベント検知手法を組み合わせたものになる．

6.1　セッションの相関関係

複数セッションを利用してセキュリティイベントを検知するために，**セッションの出現規則**と**セッション内に含まれる情報の相互利用**の 2 つを相関関係として定義する．

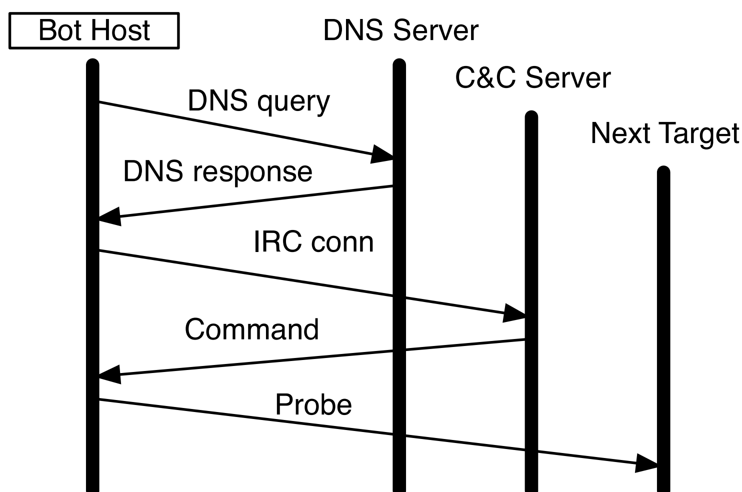

図 6.1: ボットの活動におけるセッションの発生例

6.1.1 セッションの出現規則

セッションの出現規則とは，あるソフトウェアが発生させるセッションの種類と繰り返しの出現順序をあらかじめ定義し，あるホストが発生させているセッションの規則が，定義と合致あるいは類似しているかを検査する方法である．マルウェアや P2P ファイル交換ソフトウェアも通信を発生させる手順はあらかじめ決まっており，これに従ってセッションを発生させる．これらのソフトウェアは解析によって動作手順を分析できる [62]．このようなセッションの出現順序は，検知対象によって大きく異なる．そのため単純な繰り返しや逐次的な出現順序だけではなく，様々なパターンを定義できなければならない．

6.1.2 セッション内に含まれる情報の相互利用

セッション内に含まれる情報の相互利用とは，一方のセッションに含まれる情報を他方のセッションの発見に利用する方法である．特に逐次的な順序で発生するセッションでは，一方のセッションによって得た情報を他方のセッションに利用しやすい．例として，DNS による名前解決が挙げられる．DNS は HTTP や SMTP の通信を開始する際に接続先の IP アドレスを問い合わせるプロトコルであり，そのセッション中には問い合わせ先のドメイン名や問い合わせ結果の IP アドレスが含まれる．一定時間内に DNS の問い合わせのセッションと問い合わせ結果に含まれる IP アドレスへのセッションが発生を確認できれば，二つのセッションは互いに関連していると推測できる．これによって，複数のセッションの関係性を明らかにし，セキュリティイベントの検知に利用できる．

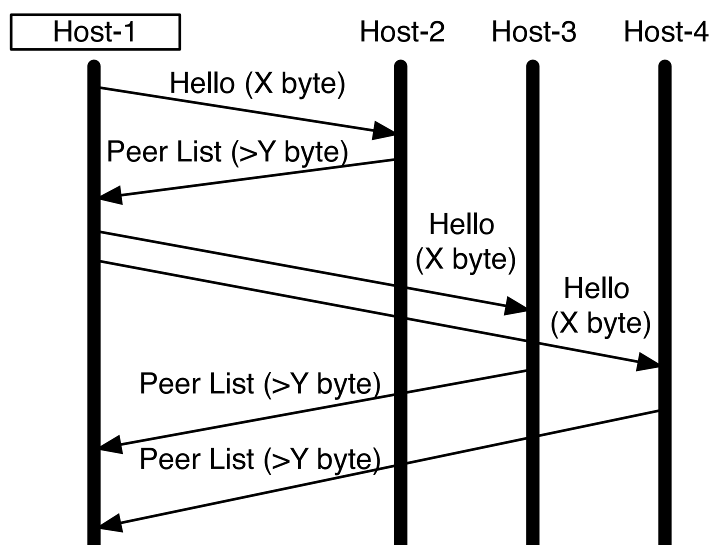

図 6.2: P2P ファイル交換ソフトウェアの実行におけるセッションの発生例

6.2 本手法が有効に機能する例

6.2.1 ボットの活動による例

具体例としてボットによる通信の一例を図 6.1 に示す．図中では，ボットに感染したホスト 1 が DNS によるホスト名の問い合わせ，IRC サーバへの接続，他のホストへの調査活動という順序で活動している．

この場合，各セッションはボットの活動と特定できるほどに特徴的とは言えない．例えば疑わしいドメイン名の DNS の問い合わせ [63] は SMTP などでも発生する可能性があり，誤検知が起こりやすい．また IRC で join するチャンネル名や命令も適宜変更されるため，チャンネル名や命令の文字列を特徴として検知するのは難しい．調査活動も急速なものについては検知できる可能性があるが，長時間にわたる調査活動は一般的な通信との区別が困難である．

一方，提案手法を用いる場合，これらのセッションの出現規則とセッションに含まれる情報に着目できる．疑わしいドメイン名の問い合わせ結果に含まれる IP アドレスへの IRC 接続を発見することで，IRC を利用するためにドメイン名を問い合わせたことが分かる．また，IRC でメッセージを受け取った直後に調査活動の可能性があるセッションを発見することで，命令の受信と活動の開始という順序にそった動作をしていると分かる．このように各セッションを個別に監視するのではなく，複数のセッションの関係性に着目しホストの振るまいを把握することで，ボットの検知が期待できる．

6.2.2 P2P ファイル交換ソフトウェアの実行による例

もう一つの例として，P2P ファイル交換ソフトウェアを取り上げて説明する．大部分の P2P ファイル交換ソフトウェアでは，ピアの探索やファイルの検索などで多くの情報を収集するため，類似するセッションを短時間で多数発生させる傾向がある [64].

これらのセッションが暗号化されていたとしても，ソフトウェア固有のプロトコルがあるため，わずかながら特徴点が見いだせる．P2P ファイル交換ソフトウェアの具体例として Winny の通信の一部を図 6.2 に示す．Winny は他ノードに接続した直後に，自ノードの回線速度や解放しているポート番号などの情報を他ノードへ送信する．そして，他ノードからノード一覧などの情報を受信する．一連の通信は暗号化されているため，セッション毎の特徴からソフトウェアを特定するのは困難である．しかし，類似した通信手順のセッションを短時間に多数発生させるという通常は見られない振る舞いから，特定のソフトウェアを発見することが可能だと考えられる．このようなセキュリティイベントを検知するためには，単一ではなく複数のセッションを監視する必要があり，本手法によって実現できると考えられる．

6.3 提案手法によるセキュリティイベント検知の戦略

本手法はセッション間の相関関係を利用することによって，セキュリティイベントの検知精度と検知結果の明確性向上を期待している．複数セッションの関係性を扱えるため，従来の検知手法と比較して多くの情報を検知条件として利用できる．そのため，セキュリティイベントを引き起こした脅威が特定しにくい不明確な検知結果ではなく，脅威やネットワークインシデントを特定しやすい明確な検知結果を出力できる．

ただし，検知結果の明確性が上がることによって，断片的な内容のセキュリティイベントを排除してしまう短所がある．変則型の脅威が発生させるセキュリティイベントは，その検知結果だけから原因を推測するのは難しい．変則型の脅威によるネットワークインシデントを検出するためには，脅威と完全に一致しない曖昧な検知条件を利用する，あるいは断片的な内容のセキュリティイベントとして検知する必要がある．本手法はネットワークインシデントとの対応が明確な検知結果を出力するかわりに，このような曖昧さや断片的な結果を排除してしまう．

これは，本手法が基本的に定型型の脅威によるセキュリティイベントの検知を目的とし，変則型の脅威によるセキュリティイベントの検知には別手法を利用する運用方法を想定しているためである．断片的な内容のセキュリティイベントは第 2.3.3 節で述べたように，検知に必要な運用負担が大きくなってしまう．しかし情報資産を守る場合，情報資産の価値によって対策にかけるべきコストは異なってくる．企業のように価値の高い情報資産や損失を防がなければならない情報資産を持つ組織の場合，それらを守るためにかけられるコストも高い場合が多い．さらに変則型の脅威による攻撃は攻撃者側もコストが低くないため，高い価値をもつ情報資産を狙う場合が多い．変則型の脅威への対策は，組織の持つ情報資産の価値や攻撃を受ける可能性を把握し，対策にかけられるコストとのトレードオフを考慮する必要がある．一方，本手法が対象としている定型型の脅威は無差別に攻

撃する場合が多いため，インターネットに接続しているホストはほぼ同様の危険性にさらされる．そのため，一般的なネットワーク運用の場合，本手法によって定型的な脅威によるネットワークインシデントを検出し，必要に応じて変則的な脅威への対策を導入する事を想定している．

6.4 まとめ

新しいセキュリティイベント検知手法として，本論文では複数セッションの相関関係を利用した検知手法を提案する．これは，従来の IDS が 1 つのパケットや 1 つのセッションに着目していたのに対し，それを拡張して複数セッションの関係性をセキュリティイベント検知の条件として扱えるようにしたものである．複数セッションの相関関係とは，複数セッションの出現順序を表した**セッションの出現規則**と，あるセッションを発見するために他のセッションの情報を使う**セッション内に含まれる情報の相互利用**の 2 つと定義する．この相関関係を利用しボットや P2P ファイル交換ソフトウェアの特性を表現できれば，これらを高精度に検知できると期待される．

第7章　予備実験

提案手法の有効性を検証するために，ボット活動の検知とP2Pファイル交換ソフトウェアの検知について実験した．本研究の目的とする検知は正確に対象を発見できなければならない．これを踏まえ，ボットとP2Pファイル交換ソフトウェアを検知するルールを用意し，それぞれが意図したセキュリティイベントとして検知するかを確認した．そして，実際の運用ネットワークのトラフィックを監視し，誤検知が発生しないかを調査した．

7.1　動作確認

RBot亜種の検知動作確認　ボットの検知に利用したルールを R_1 として表7.1に示す．RBotの亜種(Kaspersky free online virus scan[32]はBackdoor. Win32.Rbot.bmsと判定)の検体をWindows XPで感染，動作させた際の通信を監視したところ，R_1 を用いることでボットの活動として判定した．

Winnypの検知動作確認　現在，検知が難しいP2Pファイル交換ソフトウェアとしてWinnypのトラフィックの検知を検証した．Winnypはwinnyと異なりプロトコルが公開されておらず，検知手法が一般化していない．Winnypのトラフィックを調査したところ，1)接続確立後の通信手順がほぼ同一である2)複数のノードに接続し情報を送受信するの2点が確認できた．また，Winnypが発生させるセッションの大部分がTCPセッション開始直後の5パケットが同じサイズのセグメントを持つと分かった．以上の点をふまえ，設定したWinnyp検知用のルールを R_2 として表7.2に示す．R_2 を用いて3種類のWinnypトラフィックを監視し，正常に検知できることを確認した．

7.2　誤検知発生率の調査

調査は著者が所属する大学の研究室ネットワークにおいて境界点を監視した．期間は2007年4月30日の03:00から2007年5月8日02:00まで監視した．流通トラフィックは平均7.18Mbpsであり，接続しているホストは約500台である．境界点を通過したトラフィックをボット検知用のルールで分析し，通常のトラフィックによってボットが誤検知されないかを検証した．図7.1に調査を行ったネットワークの構成を示す．ネットワークは外部から内部へのトラフィックを制限しているファイアウォールが設置された一般セグメントおよびサービスセグメントと，ファイアウォールがない実験セグメントで構成されている．観測は，これらのセグメントからの通信が集約されるルータの手前で行われた．

表 7.1: 実験用ルール 1(R_1): ボット検知用ルール

S_1	DNS による Spyware, ボットで使用されるドメインの問い合わせを検知する．検知対象となるドメインは [63] を参考に，3322.org, cjb.net, maxonline.com, vendaregroup.com, seznam.cz, botstealer, dawnsoul のいずれかの文字列を含むドメイン名と設定した．
P_1	S_1 の発見後，当該問い合わせに対する応答の A レコードを，問い合わせ元ホスト H に保存する．パラメータの保持期間は 120 秒とした．
S_2	IRC サーバへの接続として，記録していた A レコード (P_1) への接続を検知する．

表 7.2: 実験用ルール 2(R_2): Winnyp 検知用ルール

S_1	TCP の接続確立後から，5 パケット連続して同じサイズのパケットが送受信されるセッションを検知する．
P_1	0 から開始し，S_1 が検知されるごとにクライアントとサーバの両ホストの P_1 が 1 ずつ増加する．保持期間は 180 秒とした．
S_2	P_1 が 5 以上になったことを検知する．

表 7.3: R_1 を用いたトラフィック監視結果の内訳

ドメイン名に含まれる文字列	S_1 検知数	S_2 検知数
3322.org	313	0
cjb.net	2,772	0
maxonline.com	102	0
vendaregroup.com	2,678	0
seznam.cz	0	0
botstealer	0	0
dawnsoul	0	0

表 7.4: R_2 を用いたトラフィック監視の結果

パラメータの状態	ホスト数
$P_1 = 1$ のホスト	2,542
$P_1 = 2$ のホスト	5
$P_1 = 3$ のホスト	2
$P_1 = 4$ のホスト	0
$P_1 = 5$ のホスト	0

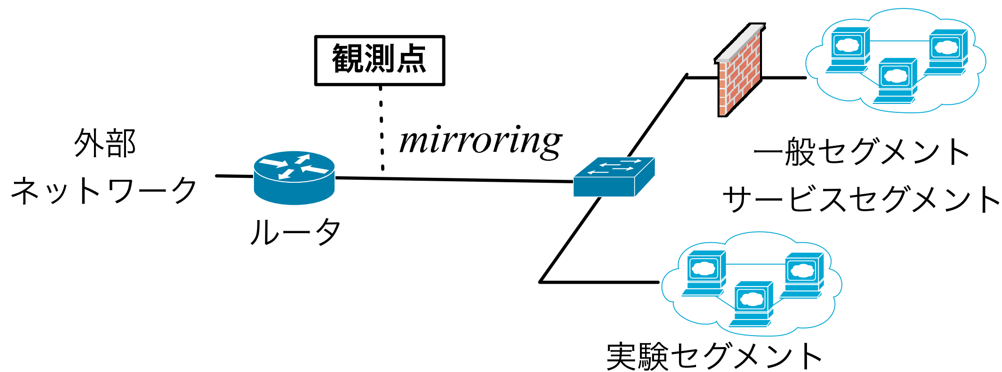

図 7.1: 調査に利用したネットワークの構成

R_1 の誤検知発生率の調査　表 7.3 に R_1 を用いた検知結果を示す．S_1 としてボットが頻繁に利用するドメイン名の問い合わせが合計 5,865 件検知されている．しかし，問い合わせ結果の IP アドレスに対して IRC の接続をしたホストは確認されず，ボットの活動は確認できなかった．既存のネットワーク監視では不審なドメイン名の問い合わせは検知するが，それらが悪意のある活動とは限らない．提案手法を用いることによって悪意のないドメイン名の問い合わせをセキュリティイベントから除外することができた．

R_2 の誤検知発生率の調査　本ルールによって S_1 を検知したホストに対して，P_1 がとった値を表 7.4 に示す．S_1 を 1 回だけ検知したホストは 2,542 台あったのに対し，2 回検知したホストは 5 台，3 回検知したホストは 2 台であり，4 回以上検知したホストは存在しなかった．

7.3　まとめ

第 7.2 節の結果は，提案手法の有効性を示している．各ルールを用いて，それぞれのソフトウェアによる通信を正常に発見することに成功した．また，同様のルールを用いて実ネットワーク上で誤検知が発生しないことを確認した．2 つのルールに見られた傾向として，共に S_1 の検知数は少なくないという点が挙げられる．これは，既存の検知手法で S_1 のみを検知ルールとして用いた場合，誤検知が多発する可能性を示唆している．よって，提案手法はトラフィックが暗号化された P2P ファイル交換ソフトウェアや他のソフトウェアと類似した通信を利用するボットが検知に有効である事が分かった．

第8章　設計

本研究では，これまで検知が難しかったセキュリティイベントを発見するために複数セッションの関係性に着目したセキュリティイベント検知機構として **ROOK** を設計，実装した．第8章では設計について述べ，第9章では実装の詳細について述べる．本章ではROOKの基本的構造と検知に用いるルールの構造を中心に，設計を述べる．

ROOKは複数セッションの関係性を柔軟にルールとして記述することを可能とする．第5.3節でも要求事項として述べたが，これまで複数セッションをルールによって扱うセキュリティイベント検知機構は複数セッションの関係性を柔軟的に扱うのが難しい．SIMの実装ではセキュリティイベントの出現順序に着目しており [48]，他のセキュリティイベントの情報はIPアドレスなどの一部の情報しか利用できない．また，Snort[25] では `trk_by` というルールがあり，同じトラフィックが同じ送信元ホストから何度発生したかを条件として記述できる．しかし，同じホストを送信元とした繰り返ししか表現できないため，柔軟性は著しく低い．これを解決するために，ROOKでは**パラメータ**機能を導入する．パラメータはプログラミング言語における変数と同様の機能を持ち，セッションの出現規則検知に関する制御や，あるトラフィックを発見するために別のセッションの情報の利用を実現している．詳しくは第8.3.2節で述べる．

8.1　ROOK設計概要

ROOKはトラフィックを監視，分析し，予め用意されたルールに基づいてセキュリティイベントを検知する実装である．図8.1では，ROOKの設計概要を示している．収集されたトラフィックはプロトコル解析され，ルールに記述されている条件と比較される．ROOKでは1つのルールの中に1つ以上のセッションルールがある．セッションルールはプロトコル毎に設定ができる．図中ではあるセキュリティイベントを検知するために4つのセッションルール（HTTP, IRC, TCP, IPv4）がある．これらのセッションルールは互いにパラメータを利用して出現規則の判断や情報の交換を行う．セキュリティイベントを検知した場合は，ルールに記述された出力（図中の例では警告の発生）を実行する．

ROOKはネットワークの境界，もしくはネットワークセグメントの接続点での動作を想定している．本手法は内部ネットワークに接続しているホストの挙動を監視する方法が有効である．一部のマルウェアでは内部ネットワークから内部ネットワークへの通信で完結するものや，内部ネットワークへの通信で独特の特徴を示すものもある．図8.2はROOKの実行環境例を示している．図中では一般的なエンドユーザが接続するユーザセグメントと，Web, メール，DNSなどのサービスをしているサービスセグメントが存在

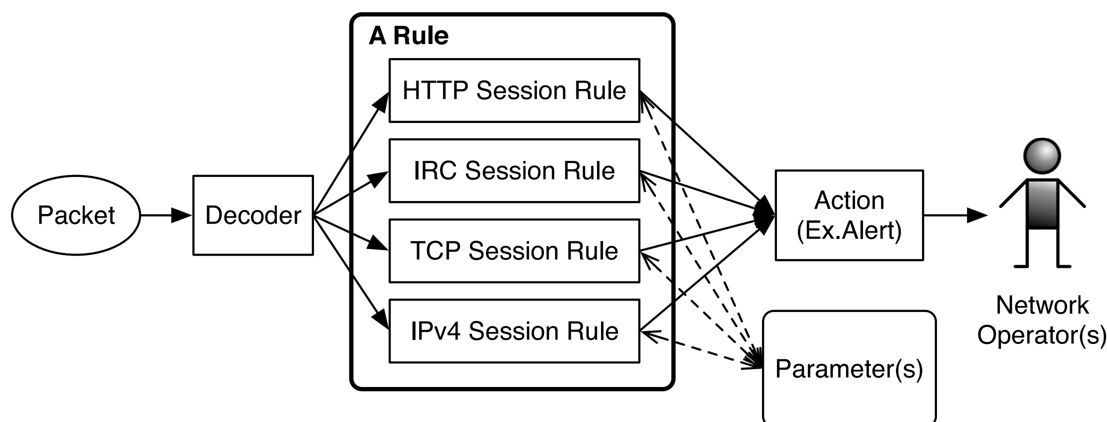

図 8.1: ROOK 設計概要

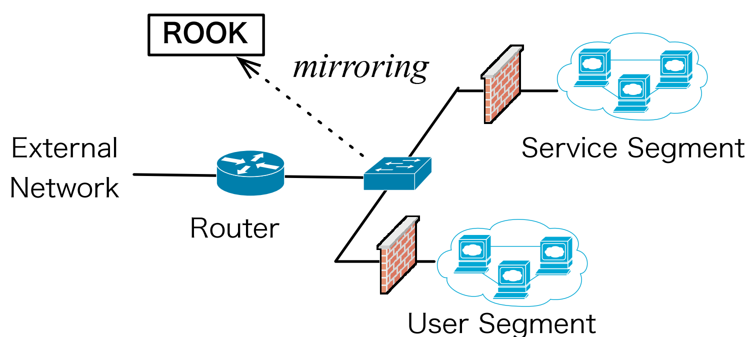

図 8.2: ROOK 実行想定環境例

し，ROOK はその 2 セグメントと外部ネットワークを中継するスイッチのトラフィックを監視している．この構成であれば，外部ネットワークと内部ネットワークのトラフィックと同時に，ユーザセグメントからサービスセグメントへのトラフィックも監視できる．

8.2 トラフィック監視型セキュリティイベント検知機構に対する要求

トラフィックの監視によってセキュリティイベントを収集する実装に対する要求事項をまとめる．

- **規模性**: 第 5.4 節で要求事項として述べたように，トラフィック監視型のセキュリティイベント検知機構では処理速度，記憶容量の効率的な利用の面から規模性が重要となる．具体的には一秒あたりの処理容量 (Bit per Second, bps) と一秒あたりの処理パケット数 (Packet per Second, pps) が十分に大きくなる必要がある．また，トラフィック監視型の場合，セッションの状態を保持する必要がある．特に IPv4 では

fragmentation の再構築，TCP では状態遷移や再送制御，シーケンス番号の処理が必要であり，セッションによる状態管理は必須である．そのため，現在監視しているセッションの状態を保持しつつ，検査中のトラフィックに該当するセッションを検索し，かつセッションの時間切れなどの処理を適切にしなければならない．そのため，セッション情報記憶時の容量の効率的な利用と，検索時の計算量低下も必要な要件となる．

- **入力データの制御**: 実ネットワークのトラフィックを監視する場合，パケットの到着間隔は正規分布ではなく，ばらつきがある [65]．また，各パケットの処理負荷もプロトコルやパケットのサイズ，記述されているルール，内容によって様々であり，一定ではない．特にセキュリティイベント検知機構の場合，検知時の出力処理によって負荷が高くなる場合もある．そのため，観測されるパケットの傾向変化によって急激に処理負荷が高くなり，トラフィックを取りこぼしてしまう恐れがある．これを防ぐため，入力されるパケットの傾向が変化してもパケットを保持しておける制御機構が必要となる．一般的に基本ソフトウェア (Operating System, OS) 上でトラフィックを取得する場合，パケットを一時的にバッファリングして取りこぼしを防ぐが，OS が適切にチューニングされていないと有効に機能しない．

- **プロトコル解析機能の独立化**: プロトコル解析はプロトコル毎にモジュール化し，それぞれの依存性を最小限にしなければならない．各プロトコルは受信したデータグラムを正規化してから解釈する場合が多い．IPv4 のパケットフラグメンテーションや TCP のデータストリーム再構築がこれにあたる．HTTP でも URL の encoding を分析してから解釈しなければ，アプリケーションによる解釈と異なってしまう可能性がある．これらがセキュリティイベント検知機構によって適切に正規化されない場合，回避する攻撃が可能になってしまう [26]．トラフィックデータの解釈機能や正規化機能を開発する場合，多くのプロトコルに対応しなければならず，開発の負担が大きい．そのため，各プロトコルの解釈機能の開発を考慮すると，プロトコル毎に分離させた分割統治的な開発が可能となる構造が必要となる．

8.3 検知ルール構造

ROOK のセキュリティイベント検知ルールは一つ以上のセッションルールから構成される．セッションルールとは，各プロトコル毎のセッションについて表現したルールである．1 つのパケットでも各層のプロトコル毎に異なるセッションルールで検査される．図 8.1 中のルールは 4 つのセッションルール（IPv4, TCP, IRC, HTTP）で構成されている．IPv6, TCP, HTTP で構成されるパケットを受信した場合，TCP と HTTP のセッションルールと比較される．

複数に分かれたセッションルールは複数セッションの非同期性に対応するためである．既存のルールベースのセキュリティイベント検知機構は単独のパケットやセッションに着目していた．単独のパケットやセッションに着目した場合，トラフィックの出現規則は

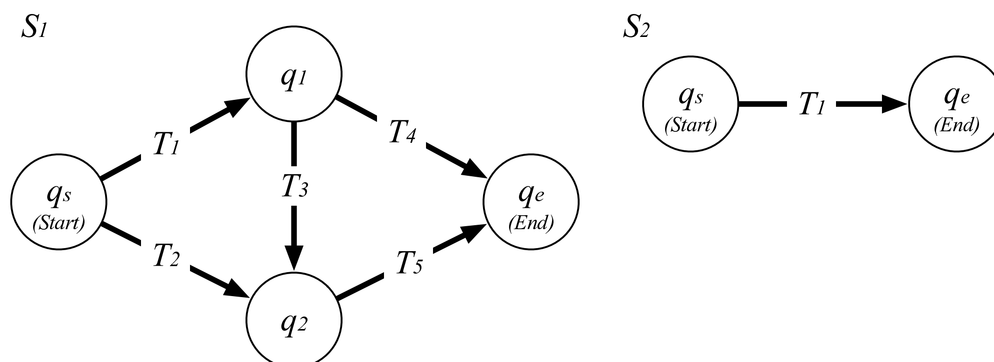

図 8.3: ROOK セッションルール構造例

セキュリティイベントに関連するトラフィックは逐次的に発生する場合が多い．例えば，HTTP[31] では基本的に要求と応答が対応しており，どの要求と応答が対応しているかが明確である．しかし複数セッションを扱う場合，各セッションは非同期に発生する可能性があり，出現規則を逐次的に扱うのが難しい．出現規則が分かっていたとしても，一方のセッションが終了してから他方のセッションが開始するとは限らない．そのため，ROOK のセッションルールは互いに非同期な構成になっている．

8.3.1 セッションルール構造

セッションルールは各プロトコルの一つのセッションを表現したルールである．セッション間のルールでは各セッションの非同期性を考慮する必要があるが，多くのプロトコルでは逐次的に通信が発生している．そのため，セッションルールでは各セッションの状態遷移を管理することで，より詳細な内容のセキュリティイベントを得ることができる．NFR[52] や Juniper IDP[66] では複数パケットの送信順序を条件として設定することで，検知精度を上昇させている．また，“Enhancing Byte-Level Network Intrusion Detection Signature with Context”[67]，Session-based IDS[68]，“侵入検知ポリシ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”[69] などの研究では，あるホストへの攻撃トラフィックと攻撃対象からの応答トラフィックを組み合わせることで，攻撃の成否を判定している．本手法では，1 セッション中における複数トラフィックの検査方法をより柔軟的に扱うために，セッションの状態を遷移させる機構を設計した．

この機構はセッション毎の状態の保持と，パラメータ操作のタイミング制御の 2 つの役割を果たしている．

状態は必ず 2 つ以上（開始と終了）の状態 (q_n) を持つ．状態数はルールに依存する．初期状態 q_s と終了状態 q_e は必ず一つずつになる．状態遷移の入力データは 1 つのパケットに相当する．実際には各プロトコル毎に解析された状態の情報を利用する．状態遷移の条件 (T_n) は 1 つの状態に対して 1 つ以上設定できる．入力データが設定されている遷移条件に一致しない場合は，何もしない．状態遷移が終了するのは 1) 状態が q_e に到達する，2)

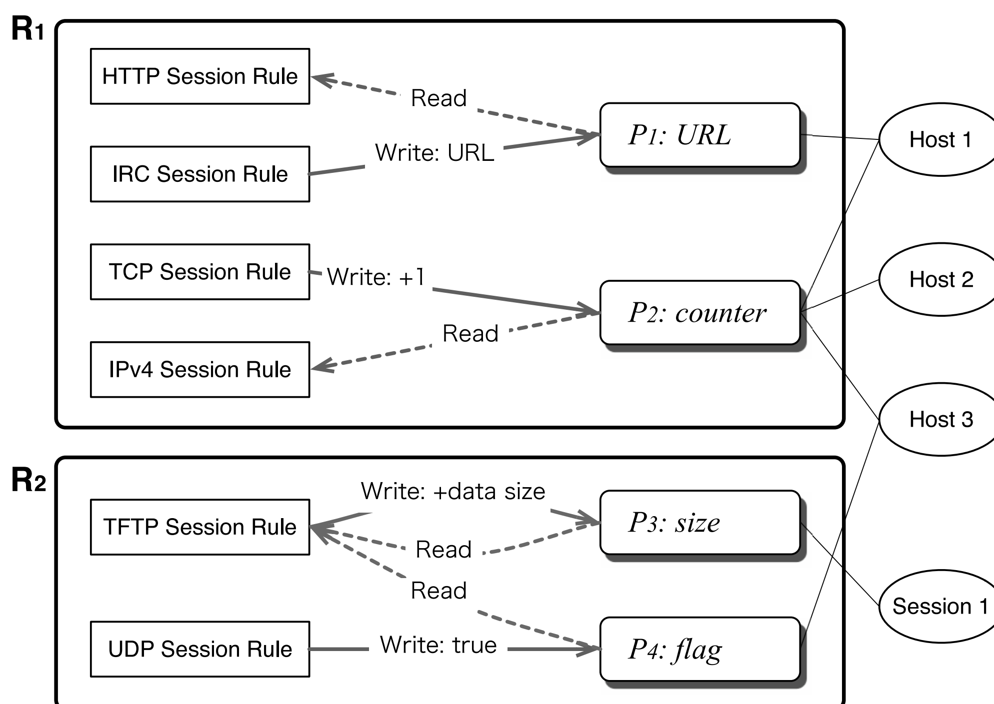

図 8.4: パラメータ構造概要

セッションが時間切れになる，3) セッションルールで指定したタイムアウトが発生する，のいずれかになる．

パラメータ操作はある状態遷移条件と関連づけされた操作と，状態に関連づけされた操作がある．状態遷移条件との関連づけは読み込みと書き込みの両方がある．読み込みは状態遷移条件を検査する際，パラメータに保存している情報を参照する操作である．書き込みはトラフィックに含まれている内容や予め定義された値をパラメータに保存する．

図 8.3 は，2 つのセッションルールの構造例 S_1 と S_2 を示している．S_1 では開始状態 q_s と終了状態 q_e 以外の状態として $q_{1,2}$ がある．遷移条件は $T_{1,2,3,4,5}$ が存在する．一つの状態に対して遷移条件は複数設定できる．初期状態 q_s において遷移条件 T_1 が満たされた場合，状態は q_1 に遷移する．make 状態 q_1 において遷移条件 T_2 を満たすトラフィックが発生しても，状態は変化しない．対して S_2 は状態が開始と終了のみである．これはセッションルール上は状態遷移が発生するが，実装では T_1 の検査と，q_e がもつパラメータ操作を実行するのみとなり，状態は保持しない．

8.3.2 パラメータ構造

非同期性を保ちながらセッションの出現規則監視やセッション間の情報交換を実現するため，各セッションルールはパラメータを使って同期や情報交換を行う．パラメータはプログラミングにおける変数と同等の機能を持つ．整数値や文字列，IP アドレスなどのデータ形式に対応し，あらかじめ定義されたデータやトラフィック中に出現したデータを書き

込み・読み込みができる．パラメータは以下のような性質を持つ．

- 書き込み，読み込みは単独のセッションのみで完結することも可能であり，複数のセッションルール間で共有することも可能である．複数のセッションルール間でパラメータを共有した場合，セッションの出現規則や情報の交換が実現される．
- パラメータはルール毎に独立した空間を利用しており，ルール毎にパラメータの個数が決められる．そのため，ルール間でパラメータに干渉することはできない．
- パラメータの保存はセッション，ホスト，全体の 3 種類の関連づけができる．関連づけの対象はルールによって異なる．セッションに関連づけた場合，書き込みと読み込みは当該セッション内で完結する．ホストに関連づけた場合，セッション間の相関関係を表すのに利用でき，本論文で提案する手法を実現するために最も重要な機能となる．複数のホストやセッションが存在している場合，パラメータはそれぞれ違う実体として保持される．同じルールの同じパラメータでも関連づけられたホストが違うと，異なる値を保持する可能性がある．全体に関連づけるのは，監視している内部ネットワーク全体で起きるセキュリティイベントを検知する場合である．未知のワームの急速的な拡散を検知する際に，Graph-based IDS[70] のようにネットワーク全体を捉えるようなルールを記述できる．
- セッションルールは 1 つ以上のパラメータを読み込み，状態遷移条件として利用できる．複数のパラメータを扱う場合，論理和としても論理積としても利用できる．
- パラメータはトラフィックと比較して検査するセッションルールだけではなく，パラメータ検査だけを目的としたチェッカーからも読み込まれる．詳しくは第 8.3.4 節で述べる．

8.3.3 パラメータ構造例

図 8.4 はパラメータ機能の例を示している．図中は 2 つのルール ($R_{1,2}$) のセッションルールによるパラメータ機能の利用を示している．パラメータ (P_n) とセッションルールを結ぶ線は，実戦が書き込み，点線が読み込みを表している．

R_1 は 4 つのセッションルール (IPv4, TCP, IRC, HTTP) と 3 つのパラメータ ($P_{1,2}$) が存在する．IRC セッションルールは IRC のメッセージ中に現れた URL を抽出し，P_1 に書き込んでいる．HTTP セッションルールは P_1 を読み込み，P_1 が書き込まれていたら HTTP のリクエストに含まれる URL と P_1 に書き込まれた URL とを比較できる．これが，**セッション間の情報交換**をセキュリティイベント検知の条件として利用する例となる．一方で，$P_{2,3}$ はセッションの出現規則を制御している．P_2 は特定の TCP セッションの出現回数を示す counter のような役割となっている．IPv4 セッションルールは特定の TCP セッションが発生した回数を読み込んで条件判断する．これによって，特定の TCP セッションが一定以上の回数で発生した後，特定の IPv4 セッションが発生することを検知できる．これが，**セッションの出現規則**をセキュリティイベント検知の条件として利用する例となる．

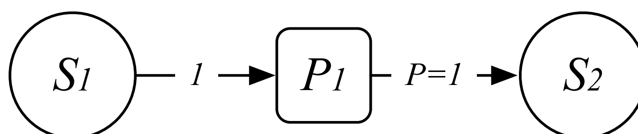

図 8.5: セッションの出現規則表現 (順接)

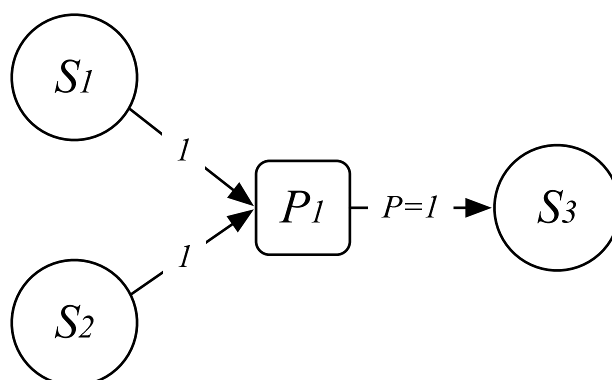

図 8.6: セッションの出現規則表現 (和)

R_2 は 2 つのセッションルール (TFTP, UDP) と 2 つのパラメータ ($P_{3,4}$) が存在する．P_3 は TFTP セッションルールのみが利用するパラメータである．TFTP セッションルールが送信されたデータのサイズを書き込んでいるが，読み込むのも TFTP セッションルールのみである．これはセッション間の関係性を示すものではないが，セッションルールの状態遷移機能と併用することで，より柔軟性の高いルールを記述できる．また，パラメータの読み込み，書き込みは 1 つのセッションルールから，任意の数のパラメータに対して可能である．R_2 の TFTP セッションルールは自分で書き込んだ P_3 と UDP セッションルールが書き込んだ P_4 の両方を読み込んでいる．

図中では P_1 がホスト 1，P_2 がホスト 1,2,3，P_3 がセッション 1，P_4 がホスト 3 と関連づけられている．ホスト 1 とホスト 2 にそれぞれ関連づけされた P_2 は 2 つのホストで実体が異なるため，異なる値を持つ．P_3 はセッションと関連づけされているため，他のセッションで利用することはできない．ホスト 3 はルールの違う複数のパラメータ $P_{2,4}$ を保持している．記述されたルールによって，各ホストやセッションは任意の数のパラメータを保持できる．

8.3.4 ルールの表現力に関する考察

本節ではルールによって記述できるセッションの出現規則の表現力について述べる．パラメータはセッションの出現規則の判定とセッション間の情報交換に，幅広く応用できる．特に出現規則の判定は逐次的な出現規則だけではなく，様々な状況に対して柔軟に対応できる．

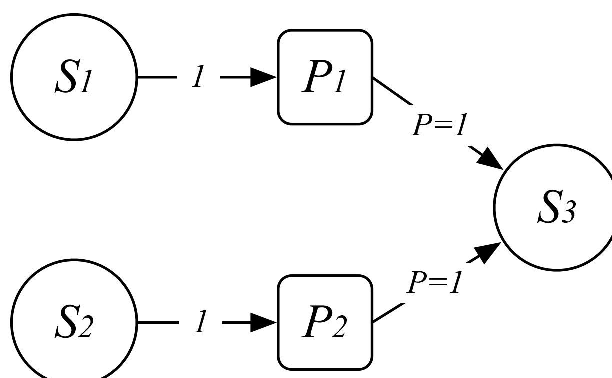

図 8.7: セッションの出現規則表現 (積)

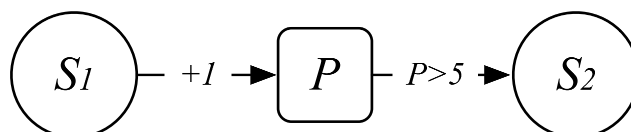

図 8.8: セッションの出現規則表現 (繰り返し)

セッションの出現規則の表現は順接，和，積，繰り返しの 4 つが基本となる．r と t がセッションとした場合，rt は r の後に t が発生した状態を指す (順接)．$r+t$ は r と t のどちらかが発生した状態を指す (和)．$r \times t$ は r と t が両方発生した状態を示す (積)．$r\{n\}t$ は r が n 回数発生した後に t が発生した状態を指す (繰り返し)．

図 8.5，8.6，8.7，8.8 はそれぞれセッション出現規則の順接，和，積，繰り返しの表現における，セッションとパラメータの関係について示している．図 8.5 は順接として S_1S_2 を示している．順接を表現するためにはパラメータを 1 つ用意し (P_1)，S_1 が発生した際に P_1 に 1 を書き込む．S_2 はあらかじめ用意されたプロトコルやトラフィックに含まれる内容の条件と P_1 に 1 が書き込まれているかどうかを検査し，双方が合致していた場合に S_1S_2 が成立したと判断する．図 8.6 は和として $(S_1+S_2)S_3$ を示している．S_1，もしくは S_2 が発生した場合に，P_1 へ 1 を書き込む．S_3 はこれを検査し，1 が書き込まれていた場合に $(S_1+S_2)S_3$ が発生したと判断する．図 8.7 は積として $(S_1 \times S_2)S_3$ を示している．S_1 が発生した場合には P_1，S_2 が発生した場合には P_2 へそれぞれ 1 を書き込む．S_3 は $P_{1,2}$ を検査し，両方に 1 が書き込まれていた場合に $(S_1 \times S_2)S_3$ が発生したと判断する．図 8.8 は繰り返しとして $S_1\{5\}S_2$ を示している．S_1 が発生した場合，P_1 に 1 が加算される．S_2 は P_1 を検査し，5 以上になった場合に $S_1\{5\}S_2$ が発生したと判断する．

セッションルールとパラメータの組み合わせにより様々な出現規則が表現できるが，より複雑な表現をするためにはチェッカー (C_n) を利用する．チェッカーは式における括弧の機能を提供している．図 8.9 ではチェッカーの機能を利用しており，式としては $((S_2 \times S_3)+S_1)S_4$ を表している．図では S_2 が発生した場合に P_2 に，S_3 が発生した場合に P_3 に，それぞれ 1 を書き込む．C_1 はチェッカーを表している．チェッカーは依存するパラメータが書き込

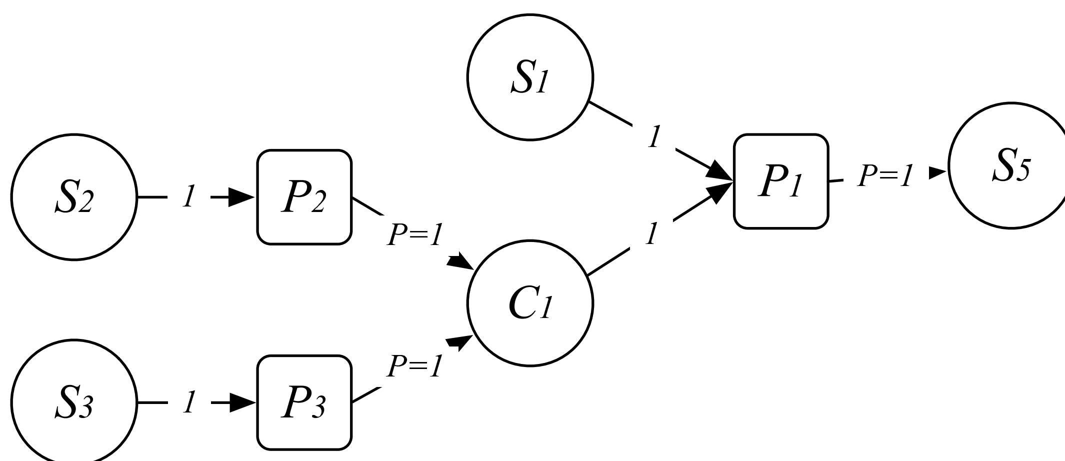

図 8.9: セッションの出現規則表現 (括弧)

まれた場合にパラメータの値を検査し，条件を満たしていれば規定された動作（主にパラメータ書き込み）を実行する．図中では P_3 と P_2 の両方の値が 1 になった場合に C_1 が P_1 へ 1 を書き込む．S_1 が発生した場合も P_1 に 1 を書き込む．S_4 は P_1 を検査し，1 になった場合に $((S_2 \times S_3) + S_1)S_4$ が発生したと判断する．チェッカーを用いることで柔軟に式を設定できるようになる．

本手法で設計したルールにおけるセッション出現規則は，出現要素の順接，和，積，繰り返し（閉包）を表現可能であり，これは正規言語に相当する表現力を持つ．

8.4 まとめ

本章では複数セッションの関連性に着目したセキュリティイベント検知機構 ROOK の設計について述べた．ROOK はトラフィックを監視・分析し，セキュリティイベントを発見するという点において，既存の IDS に類似している．しかし，非同期的に発生する複数のセッションをルールとして扱う必要があるため，パラメータという概念をルールに導入した．これによって，セッションの出現規則やセッション間の情報交換を柔軟に実現することが可能となった．さらに，セッションの出現規則の表現については逐次的な出現規則だけではなく，指定した複数セッションのいずれかが発生した状態 (和) や，指定した複数セッションが全て発生した状態 (積)，セッションの発生回数に応じた状態 (繰り返し) もそれぞれ定義できるため，非常に柔軟なルールが記述できる．

第9章　実装

本章ではROOKの実装構造について述べる．ROOKはトラフィックを監視・解析し，セキュリティイベントを検知する実装である．実装はC言語を利用しており．開発環境はDebian GNU/Linux Kernel 2.6.18(gcc 4.1.2 20061115)，MacOSX 10.4.x-10.5.x (i686-apple-darwin9-gcc-4.0.1)の2つである．ライブラリとしてlibpcap[71] (バージョン0.9.5, rev. 1)，libxml[72] (バージョン2.6.30)，libpcre[73] (バージョン7.4)を利用した．

ROOKは上述した3つのライブラリをのぞいて，全て独自に実装している．トラフィック監視によるセキュリティイベント検知機構を実装するにあたり，既存のオープンソースの実装(Snort[25]やPrelude[74])を拡張する方法もあったが，2つの理由により独自実装にした．1つめは本手法が必要とするセッションやホスト管理機構が，現状の実装を利用した場合に不都合が生じると考えられるためである．本手法はセッションやホストにパラメータを関連づけるため，プロトコルに依存しないセッションとホストの抽象化が必要となる．しかし，現状で公開されている実装の多くは幾つかのプロトコルに限定してセッション管理をしているため，新たに抽象化したセッションやホストの管理機構を追加実装するのは冗長になってしまい，処理効率も低下すると予想される．2つめはルールの記述における問題である．先述したとおり，IDSに代表されるルール記述型のセキュリティイベント検知機構では，1つのパケットや1つのセッションに着目した検知機構となっている．第8.3.1節で述べたとおり，本手法を実現するためには複数のセッションルールを一つのルールとして扱うなど，既存のルールとは構造的な違いが必要となる．この2つの理由により，ROOKはほぼ全ての部分を独自に実装した．

ROOKの実装におけるコンセプトは規模性と拡張性である．規模性は処理性能の向上を指す．第8.2節で述べたとおり，トラフィック監視によるセキュリティイベント検知機構では複数のホストを同時に監視できる反面，大量のトラフィックを処理しなければならない．特に本実装は，一般的なIDSと比較しても機能が拡張されており，負荷は大きくなると予想される．よって，随所において計算量増加の抑制やメモリ使用量の効率化が必要となる．拡張性は機能追加における労力の少なさを指す．第8.2節ではプロトコル解析機能の独立化について述べた．プロトコル毎に異なる処理をするとしても，セッションやホスト管理，あるいは解析結果の取得などは抽象化されていないと，Detectionコンポーネントで結果を利用するのが難しい．そのため，可能な限り柔軟性の高い抽象化によって細部の自由度を保ちつつ，他の部分と連携できるような構造が必要となる．

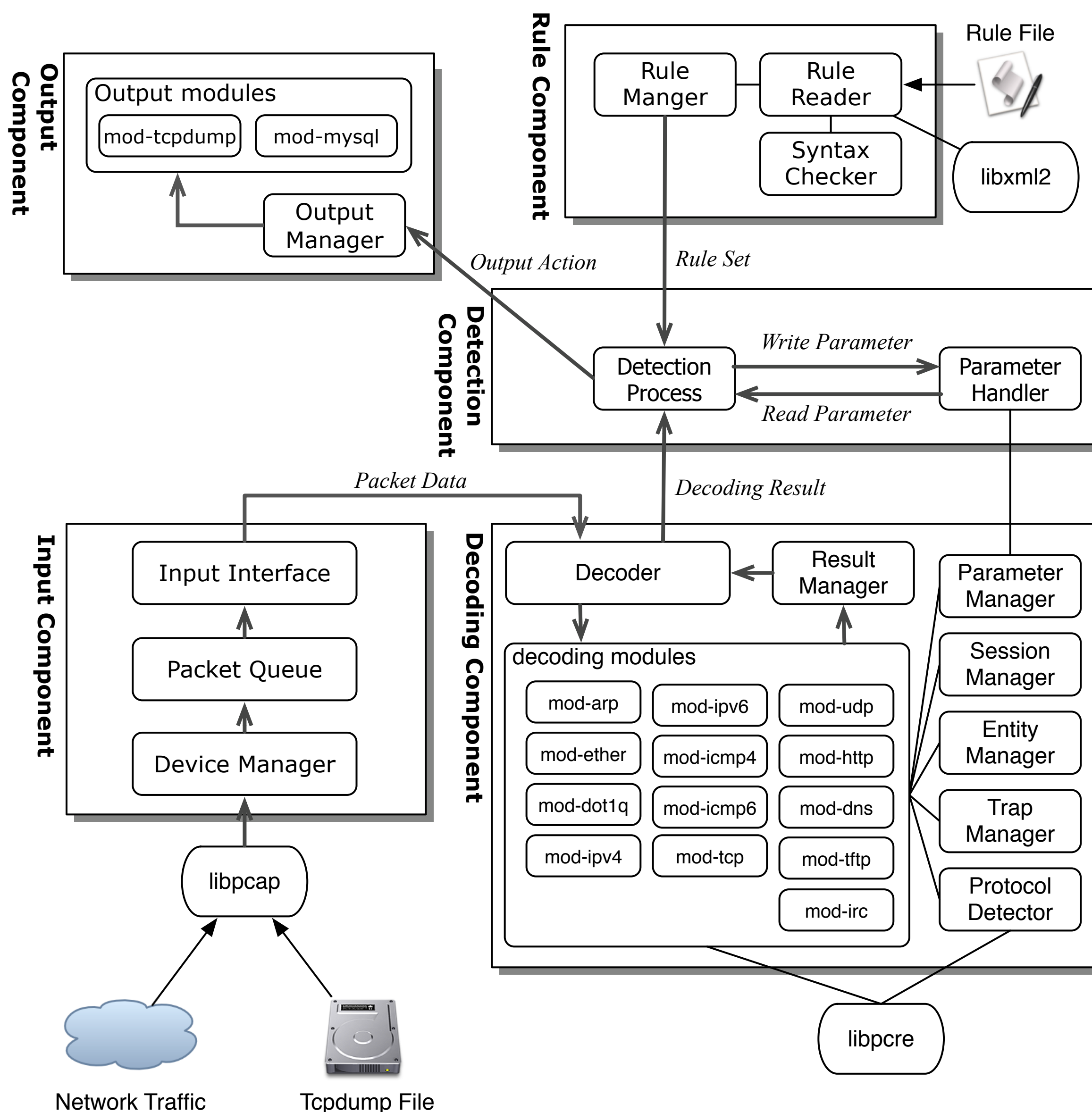

図 9.1: ROOK 実装概要図

9.1 実装概要

ROOK は入力部 (Input コンポーネント)，解析部 (Decoding コンポーネント)，ルール読み込み部 (Rule コンポーネント)，検知部 (Detection コンポーネント)，出力部 (Output コンポーネント) の 5 つから構成されている．これらは分割統治の構想を元に，コンポーネント毎の依存性を最小限にするように考慮されている．図 9.1 において実装の全体像を示している．

Input コンポーネントはトラフィックからパケット単位でデータを読み込む．独自にキューイングの機能を有しており，突発的な入力が発生しても入力データを消失させずに処理で

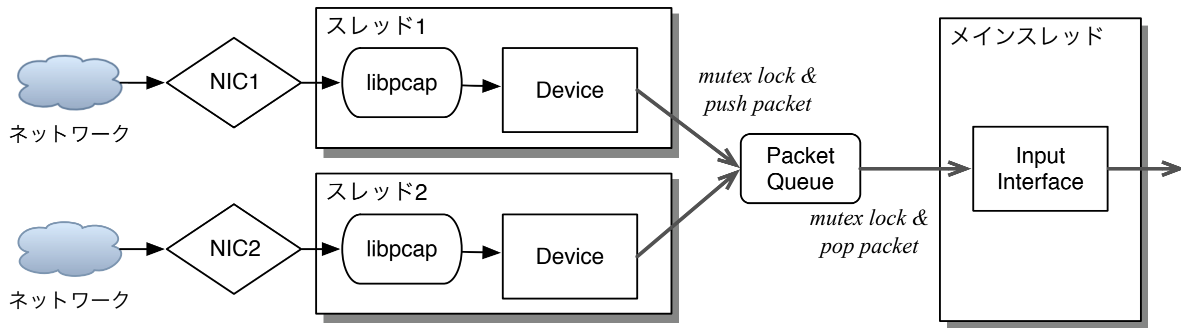

図 9.2: Input コンポーネント概要

きる．詳細は第 9.2 節で述べる．

Decoding コンポーネントはパケットを解析し各プロトコルに適応した処理やセッションの管理を行う，検知処理をするためには，パケットに含まれるプロトコルの種類やプロトコル毎のデータの正規化，セッション状態の管理が必要となる．詳細は第 9.3 節で述べる．

Rule コンポーネントは XML[75] で記述された検知用ルールファイルを読み込む．ルールセットは構造の正当性チェックを経て，Detection コンポーネントに渡される．詳細は第 9.4 節で述べる．

Detection コンポーネントは Input，Decoding，Rule コンポーネントから得られた情報を元に，セキュリティイベントを検知する．詳細は第 9.5 節で述べる．

Output コンポーネントは Detection コンポーネントで検知されたセキュリティイベントを様々な方法で出力する．出力方法や出力条件などはルールに記述された内容による．詳細は第 9.6 節で述べる．

9.2 Input コンポーネント

Input コンポーネントはネットワークインターフェースの監視によるパケットデータ入力と，ディスクに保存されたパケットデータ入力の両方に対応している．これは libpcap[71] を使うことで，ネットワークインターフェースの監視と tcpdump[76] が出力するファイル形式から読み込むことが可能になっている．

9.2.1 入力パケットデータのバッファリング

パケットデータを取得する際，取得処理は別スレッドで実行される．これはパケットデータの読み込み処理と解析・検知処理を分離させるためである．第 8.2 節で述べたとおりパケットデータの入力はばらつく場合が多いため，解析・検知処理の負荷が一時的に高まるとパケットを取りこぼしてしまう可能性がある．そのため，パケットデータの入力と解析・分離処理は異なるスレッドで実行し，入力されたパケットデータは Queue に一度

保存される．現在の実装では初期設定で 20MB までパケットデータを蓄積できる．Input Interface は別のコンポーネントから呼び出され，Queue に蓄積されたパケットデータを FIFO で排出する．

9.2.2 複数ネットワークインターフェースからの並列入力

パケットデータの取得では複数の入力に対応しており，並列入力と逐次入力に分けられる．複数のネットワークインタフェースから入力する場合は並列入力になり，ファイルからの読み込みの場合は逐次処理となる．

ネットワークを監視する場合，構成上の問題から一つのインターフェースで全てのトラフィックを監視できない可能性がある．これは非対称経路や，回線のメディアタイプの違い，冗長化構成が要因となる事が多い．このような状況の場合，セキュリティイベント検知機構のプロセスをネットワークインターフェース毎に複数実行する方法がある．しかし，非対称経路の場合は一方向のパケットしか監視できず，TCP の状態遷移管理やシーケンス番号処理に問題が生じる．また，プロセスを分けることで設定の共有などの負担が生じてしまう可能性がある．

そのため，プロセス側で複数ネットワークインターフェースからの入力機能を実現した．本実装が利用している libpcap では複数ネットワークインターフェースからのパケットデータ入力に対応していないので，本実装が独自に対応している．図 9.2 は Input コンポーネントが複数のネットワークインターフェースからパケットデータを読み込む様を表している．Input Interface と別コンポーネントによる解析・検知処理はメインスレッドで実行される．一方，ネットワークからのパケットデータ読み込み処理はネットワークインターフェース毎にスレッドを作成し，libpcap により読み込んでいる．図中ではスレッド 1 とスレッド 2 がそれぞれパケットデータを読み込んでいる．これらのスレッドはパケットデータの読み込み後，Packet Queue をロックしてパケットデータを書き込んで (push して) いる．Input Interface は別コンポーネントから要求が来た際に，Packet Queue をロックしてパケットデータを取り出して (pop して) いる．

9.2.3 処理負荷に関する考察

L をパケットの入力長とすると，Input コンポーネントでの計算量は $O(L)$ となる．一度パケットデータをバッファリングするためメモリに全パケットデータをコピーする必要があり，この処理は入力されるパケット長に比例する．Input コンポーネントではパケットデータのコピー以外には繰り返し処理が発生しないため，パケット長 L に影響される．実際の監視では Ethernet のトラフィックを取得する場合が多く，Ethernet の MTU は 1500 バイトであるため，ほとんどの L は 1514 バイト以下である．そのため，1 つのパケットデータ入力に対する処理負荷は無視できる程度である．

性能低下が発生する可能性として，多数のネットワークインターフェースから並列入力した場合が想定される．別スレッドで実行しているため，Packet Queue の書き込み・読み

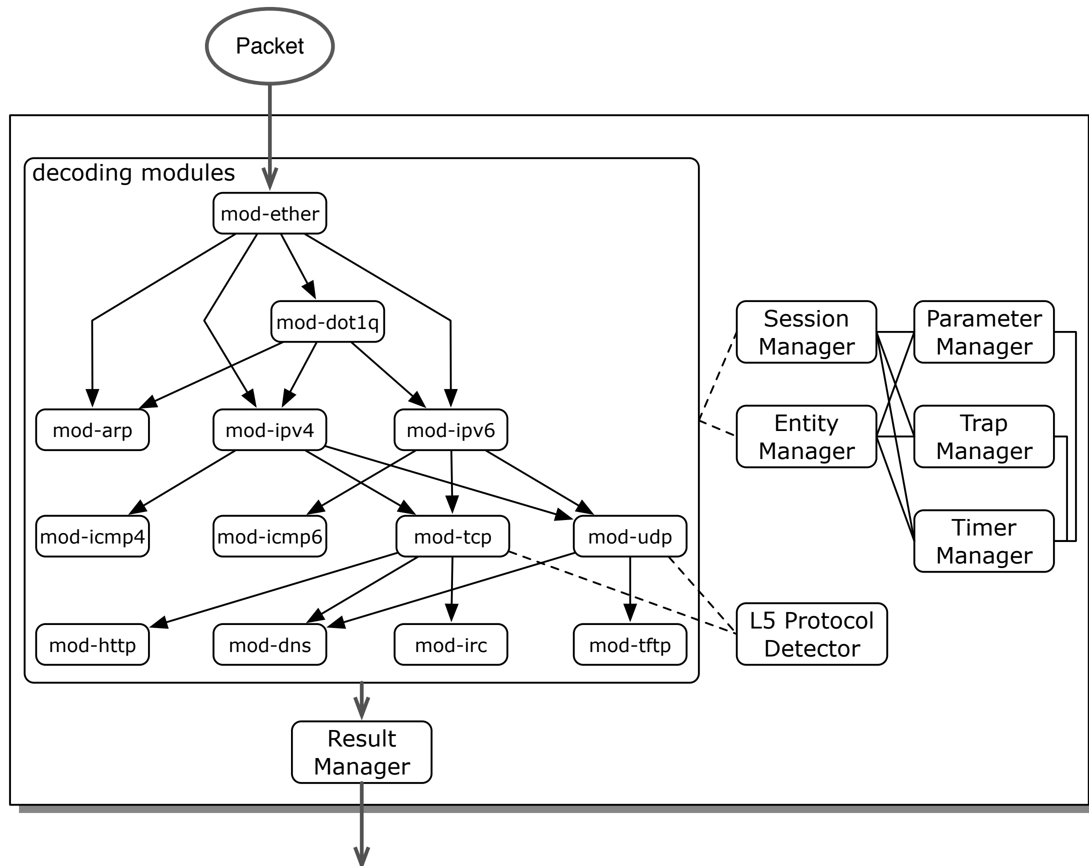

図 9.3: Decoding コンポーネント概要

込み時にはこれをロックする必要があり，ロック待ちによって別コンポーネントがInput Interface からパケットデータを読み込めない，あるいは読み込みに長時間かかる可能性がある．ただし，一般的な計算機に装着できるネットワークインターフェースはたかだか4，5 個程度であるため，特殊な状況で使用しない限り，深刻な性能低下は起きないと考えられる．

9.3 Decoding コンポーネント

Decoding コンポーネントでは Input コンポーネントから得られたパケットデータを解析し，プロトコルの認識やセッションの管理を担当するコンポーネントである．

ROOK ではパケットデータの解析処理とセキュリティイベントの検知処理とを完全に分離している．セキュリティイベント検知機構において解析処理と検知処理が混在してしまうと，機能を分割するのが困難になり適切に分割統治が実現されない．そのため，ROOK は Decoding コンポーネントにおいて解析処理を全て終えてから，その結果を参照しながら Detection コンポーネントが検知処理を実行する．

ROOK の Decoding コンポーネントの特徴の一つはプロトコルの解析処理をプロトコル毎に分離させ，モジュール化している点である．これは Bro IDS[77] でも採用されている実装方式である．プロトコル毎に分離したモジュールとして扱うことによってそれぞれの依存性が最小限になり，全体の開発負荷を下げられる．また，モジュール化は拡張性も向上させる．一般的なトラフィック監視型セキュリティイベント検知機構では，部分的にしかプロトコルを解析しない．例えば Snort[25] は基本的にトランスポート層までしかプロトコルを解釈しない．一部検知に必要な部分を解釈する場合はあるが，構造的に柔軟に処理を追加するのが難しくなっている．libnids[78] もトランスポート層までの解釈しかしない．よって，本実装ではプロトコル解析処理のモジュール化により分割統治と拡張性を実現している．

図 9.3 は Decoding コンポーネントの構成を示している．Decoding コンポーネントはパケットデータを入力し，解析結果を出力する．解析処理はモジュール毎に実行され，モジュール間の遷移も各モジュール内に条件を持つ．各モジュールは必要に応じてセッション管理とホスト（エンティティ）管理をしている．これらは Parameter Manager，Trap Manager を利用するのに必要となる．また，特に TCP と UDP のモジュールでは OSI 参照モデルにおけるレイヤ 5（プレゼンテーション層）のプロトコルを判定するためのサブモジュール（L5 Protocol Detector）を利用して，次の解析モジュールを決定している．各モジュールでの解析結果は Result Manager によって統括され，出力される．これをもとにして，別コンポーネントは解析結果を参照している．

9.3.1 モジュール構成

ROOK では各プロトコルの解析機能をモジュール化して，プロトコル毎に機能を分割している．現在実装されているプロトコルは Ethernet, ARP, 802.1q, IPv4, IPv6, ICMPv4, ICMPv6, UDP, TCP, HTTP, IRC, TFTP, DNS である．順次，FTP, SMTP, POP3, NetBIOS, SMB, IMAP, SSL, MSNP, AIM, YMSG, Postgres, eDonkey, BitTorrent, Gnutella に対応していく予定である．各モジュールはそれぞれ独立して実装されているが，他のプロトコルとの依存関係や類似した機能の再開発防止，解析結果の参照などで共通した解析機能を幾つか有している．

- **解析関数**: 当該モジュールが担当するプロトコルが出現した場合，プロトコル解析するためにパケットデータとパケット長が渡されるコールバック関数を登録する機能．開発状況や解析が不要な場合には，解析関数を登録しないことで呼び出されなくなる．

- **解析結果管理**: 解析関数によってパケットデータを解析した後，解析結果を共通の構造体に格納する機能．他のコンポーネントが解析結果を参照する際に使用する結果一覧が管理されている．

- **セッション管理機能**: 2 ホスト間の通信をプロトコル毎に抽象化したセッションを管理する．セッションを特定するキーはプロトコル毎に定義する．詳細は第 9.3.2 節に

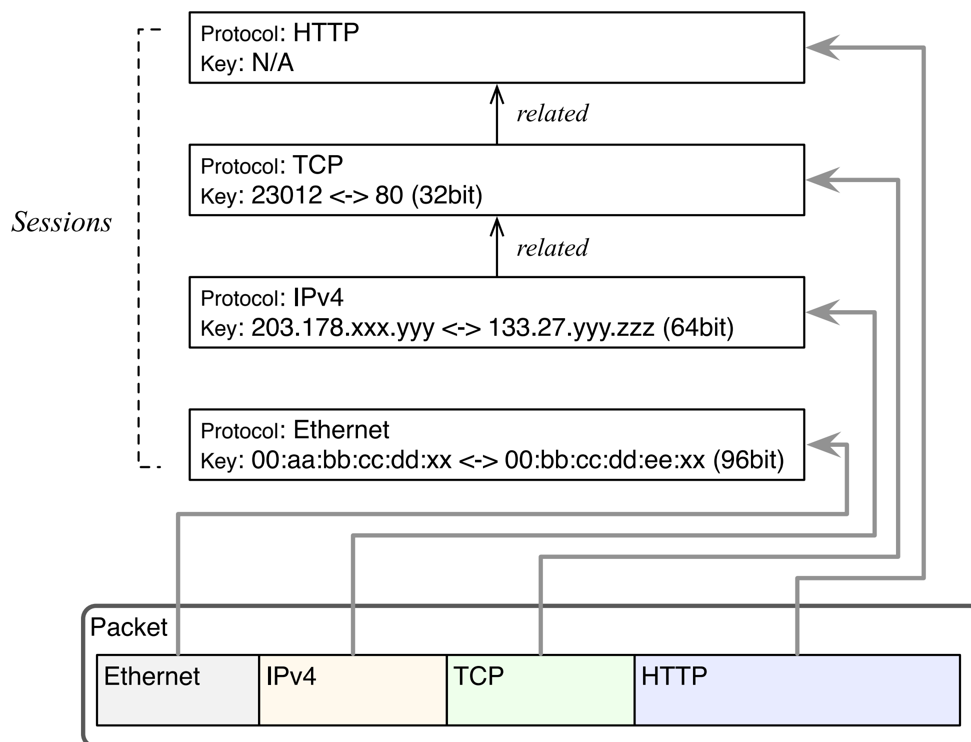

図 9.4: セッション構造例

て述べる．

- **エンティティ管理機能**: トラフィック中に出現したホストを管理する機能．ホストをエンティティとし MAC アドレス，IPv4 アドレス，IPv6 アドレスをそれぞれエンティティの識別子としている．詳細は第 9.3.3 節にて述べる．
- **フィルタ機能**: 解析結果をもとに，プロトコル固有の情報を外部コンポーネントから参照するための機能．
- **設定関数**: プロトコル毎の動作で，ネットワーク環境に準じた設定や，ネットワーク運用方針によって決定する設定を外部コンポーネントから変更する機能．

9.3.2 セッション管理機能

ROOK は各プロトコルのセッションの情報と状態を管理している．本実装における**セッション**とは，2 ホスト間の通信を各プロトコルに分割したものである．図 9.4 にパケットとセッションの関係を示す．本実装では同じパケットに含まれていても，違うプロトコルは異なるセッションとして見なす．図中では Ethernet, IPv4, TCP, HTTP で構成されるパケットがあり，それぞれ 4 種類のセッションとして扱われる．各セッションは独立して扱われるが，IPv4 セッションと TCP セッション，TCP セッションと HTTP セッション

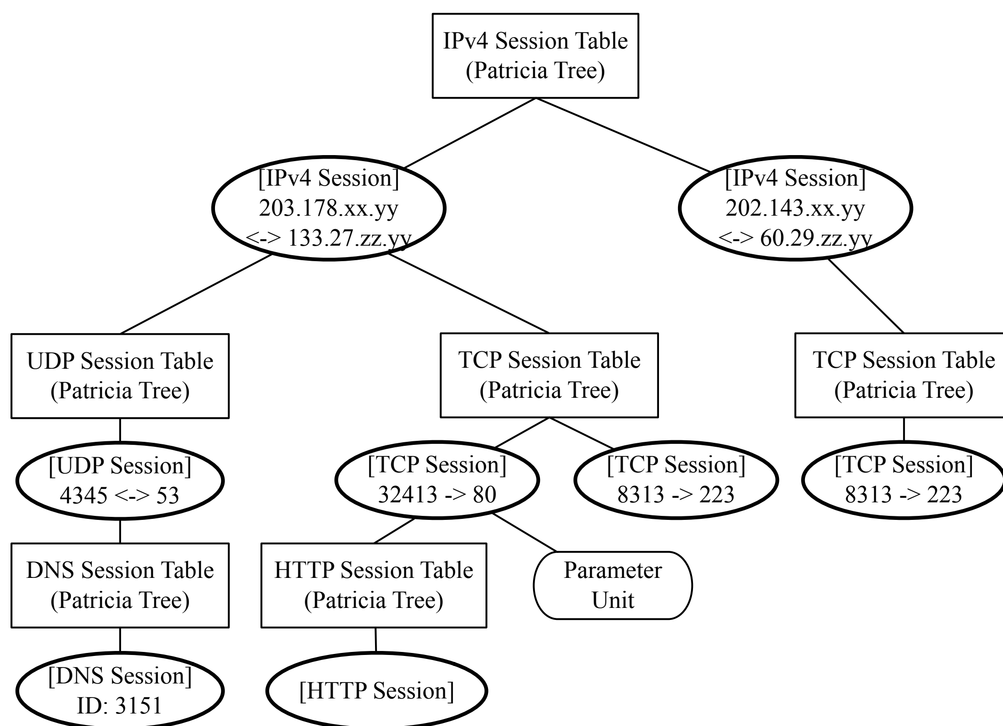

図 9.5: セッション管理構造例

は互いに関係がある．ある TCP セッションは特定の IPv4 セッション上に存在し，ある HTTP セッションは特定の TCP セッション上に存在するためである．しかし冗長構成などにより，ある IPv4 セッションが必ずしも同じ Ethernet セッション上に出現するとは限らない．そのため下位プロトコルのセッションと関係があるかは，プロトコル毎に判断する．

図 9.5 は IPv4 を基底としたセッションの管理構造例を示している．全てのセッションはパトリシアツリー [79] に格納されており，ツリーは必ずプロトコル毎に分かれている．図中では四角形がセッション格納用のパトリシアツリー（セッションテーブル）を表しており，丸形はツリー内のノード（セッション）を表している．ツリーは 1 つのプロトコルに 1 つの場合もあれば，1 つのプロトコルで複数もつ場合もある．基本的に複数ツリーが存在するのは，下位プロトコルのセッションと結合しているためである．例えば，図中では TCP のセッションテーブルが 2 つあるが，それぞれ下位プロトコルである IPv4 のセッションが $203.178.xx.yy \leftrightarrow 133.27.zz.yy$ と $202.143.xx.yy \leftrightarrow 60.29.zz.yy$ とで異なっている．異なる IPv4 セッション上の TCP は同じキーだとしても異なるセッションなので，同一のプロトコルでも複数のツリーによってセッションを格納している．

各プロトコルのセッションは同じ構造体によって管理されている．図 9.6 は共通のセッション構造体のコードを示している．セッション構造体はパトリシアツリーのノードなので，ツリーのキーやキー長，ノード種別，子ノードへのリンクなどが上部に当たる．一方，下部はセッション特有のデータを表している．`void * data` はプロトコル毎のデータを保持するためのポインタである．プロトコル毎のデータの例としては TCP の状態管理やストリーム

```
struct _decode_session
{
  u_int8_t * key;
  int keylen;
  u_int8_t type;
  u_int8_t nodeKeyType;
  u_int8_t treeKeyType;
  u_int8_t dir;
  Session * tree;
  Session * parent;
  Session * link[2];

  void * data;
  void (*destroyData) (void *);
  ParamUnit * punit;
  Trap * trap;
};
```

図 9.6: セッション構造体のコード

の順序制御，IPv4 のフラグメント再構築などが挙げられる．これらは別途メモリを割り当てられ，ポインタのみセッション構造体で保持する．またセッションが破棄される際，プロトコル毎のデータを破棄するためのコールバック関数も `void (*destroyData) (void *)` に登録される．`ParamUnit * punit` はセッション毎にパラメータを保持するための**パラメータユニット**を，必要に応じて格納する．パラメータユニットについては第 9.3.4 節で詳細に述べる．`Trap * trap` は当該セッションを含むパケットデータが入力された場合に，指定した関数を呼び出す**トラップ**を必要に応じて格納する．トラップについてはは第 9.3.5 節で詳細に述べる．

1 セッションあたりの基本的なデータ量はおよそ $(L_k + L_p + (44 \times 2))$ バイトになる．L_k はキー長，L_p は各プロトコル毎に保持するデータの長さ，44 バイトはノード構造体 1 つあたりのバイト量であり，パトリシアツリーの分岐点で 1 つ消費するため 2 倍になっている．例えば IPv6 モジュールの場合，1 つあたりのキー長は 128 ビット (16 バイト)，プロトコル固有の情報は無しなので，1 セッション毎に約 104 バイト消費する．

9.3.3 エンティティ管理機能

ROOK はセッションの他に，入力データ中に発見したホスト（エンティティ）の情報を保持しており，IPv4，IPv6，Ethernet で対応している．各エンティティはプロトコル固有の識別子で管理されている．IPv4 は IPv4 アドレス (32 ビット)，IPv6 は IPv6 アドレス (128 ビット)，Ethernet は MAC アドレス (48 ビット) である．プロトコル毎のパトリシアツリーを持ち，各エンティティを格納する．各エンティティには時間切れが設定されており，それまでに関連するパケットデータが出現しなければ破棄される．

図 9.7 ではエンティティの管理構造の例を示している．エンティティもセッションと同

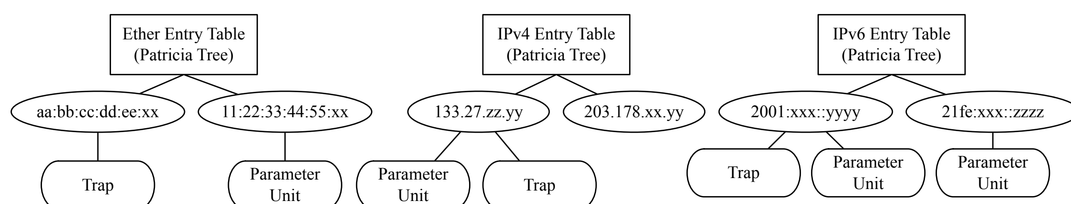

図 9.7: エンティティ管理構造例

様にパラメータユニットとトラップを必要に応じて利用できる．これによって，あるエンティティが出現した場合に特定のルールの検査をする，あるいはエンティティ毎にパラメータを設定することが可能となる．

9.3.4　パラメータ管理

Decoding コンポーネントでは ROOK で実現すべきパラメータ機能を全て実現していない．第 9.4 節で詳細に述べるが，第 8.3 節で述べたパラメータ機能を実現させるためには，いくつかの機能が必要となる．また，パラメータはセッションやエンティティへの関連づけがある一方で，ルールや検知処理とも関連しなければならず，コンポーネント間をまたがった実装となって分割統治の原則が崩れてしまう．そのため，Decoding コンポーネントで実装する機能，Detection コンポーネントで実装する機能を明確に分離させた．

Decoding コンポーネントで実装されているのは，基本的にセッション，エンティティへの関連づけとデータの書き込み，読み込みの管理である．他にはパラメータの ID 管理，パラメータ生存時間の管理が挙げられる．Decoding コンポーネントにおけるデータの扱いは，書き込み・読み込みの対象が値か参照かの違いのみである．値の場合は新たにメモリを確保し，データをコピーして保存する．一方，参照の場合はデータのポインタのみを保持する．

パラメータを保持するための構造体がパラメータユニットになる．パラメータユニットは 1 つにつき，1 つの名前空間を有している．第 9.3.2 節や第 9.3.3 節で述べたように，全てのセッションとエンティティにはパラメータユニットを生成できる．これはセッションやエンティティ毎に異なるパラメータユニットであり，セッションやエンティティはパラメータの名前空間を共有しない．名前空間はパトリシアツリーで構築されている．

9.3.5　トラップ管理

入力されたパケットデータに特定のセッションやエンティティが含まれていた場合，予め指定したコールバック関数を呼び出す機能である．パラメータユニット同様，外部コンポーネントに提供されている機能で，外部コンポーネントから呼び出さなければ動作しない．また，必要に応じてセッションやエンティティに生成される点もセッションと同様である．

トラップ機能はセッション毎の状態遷移を管理するのに利用されている．詳しくは第 9.5 節で述べる．

9.3.6 フィルタ機能

ROOK ではパケットやトラフィックの性質をモジュール毎に示すために，フィルタ機能を実装している．パケットやトラフィックの特徴はプロトコル毎に解釈しなければならないが，独立した実装になっていると外部コンポーネントからの参照が難しい．そのため，フィルタ機能を各モジュール共通の枠組みとして提供することで，外部コンポーネントからの解析結果参照が容易になる．

フィルタはフィルタ名と引数（任意）で呼び出し，返値を得られる．例えば，パケットの IPv4 送信元アドレスを得たい場合は，フィルタ名 `ipv4.src_addr` と引数無しで呼び出す．この場合，パケットデータに IPv4 が含まれていれば当該パケットの IPv4 送信元 IP アドレスを，IPv4 が含まれていなければ null(失敗) を返す．特定のアドレスレンジの IPv4 アドレスを抽出したい場合には，引数として `192.168.0.0/24` などの値を渡す．`192.168.0.0/24` の引数を渡したとすると，192.168.0.0 から 192.168.0.255 までの IPv4 送信元アドレスがパケットデータに含まれていれば IPv4 送信元アドレスを，それ以外の場合は null を返す．付録 A の表 A.2 に，ROOK で現在実装されているフィルタの一覧を示している．

フィルタ機能には状態遷移条件としてパケットやトラフィックの特徴を調べる機能と，パラメータ書き込み時にパケットやトラフィックの特徴を抽出する機能の 2 つからなる．詳しくは第 9.5.1 節と第 9.5.2 節で詳細に述べる．

9.3.7 多重フィルタ機能

ルール型のセキュリティイベント検知機構では，しばしばルールの肥大化が問題となる．検知対象の増加に伴って検知用ルールを増加させる必要があるが，計算量がルール数に比例してしまうと大量のルールに対応できなくなってしまう．例えば 2007 年 12 月 20 日現在，Snort[25] では 16,235 個のルールが存在している．有志で Snort のルールを作成している Bleeding edge Threat[80] が提供しているルールも 10,283 個となっており，全てのルールを線形に走査すると深刻な処理性能低下を起こしてしまう．

これを解決するために，ROOK では複数の条件を一度に検査できるマルチフィルタ機能を用意している．フィルタ機能は 1 つのパケットデータ入力に対して 1 つの出力だが，マルチフィルタ機能は 1 つのパケットデータ入力に対して複数の出力を可能としている．これは複数の条件に対して一括検査を可能とするフレームワークである．例えば TCP モジュールにおいて宛先ポート番号を条件とするルールが多い場合は，宛先ポート番号をキーとしたパトリシアツリーやハッシュ検索にすることで，計算量を大幅に減らせる．

検査手段は各モジュールでの実装に依存するため，一括検査の手法もモジュールによって異なる．現在は IPv4，TCP，DNS，HTTP において実装されている．IPv4 と TCP は

それぞれ，宛先 IPv4 アドレスと TCP の宛先ポート番号をキーとしたパトリシアツリーによって探索している．それぞれの条件数を N とした場合，IPv4 では平均の計算量が $O(\log N)$，最悪の計算量が $O(32)$，TCP では平均の計算量が $O(\log N)$，最悪の計算量が $O(16)$ となり，条件数に対して線形に増加しない．また，DNS，HTTP では Aho-Corasick アルゴリズム [81] を用いて出現文字列の検査をしている．DNS はクエリもしくは応答に含まれるドメイン名，HTTP はリクエストに含まれる URL と応答に含まれるコンテンツの内容を一括して検査できる．検査するデータ長を L とすると検査する計算量は $O(L)$ であり，これも条件数に対して線形に増加しない．

9.3.8 Payload-based Protocol Detector

ROOK では，ペイロードに着目した OSI 参照モデルにおける第 5 層（プレゼンテーション層）のプロトコル検出機能を実装している．TCP や UDP の上位レイヤーであるプレゼンテーション層は，IANA において TCP/UDP のポート番号が割り当てられている [61]．しかし通信をする二者間の合意があれば，どのようなプロトコルでも任意のポートでの通信が可能である．特にマルウェアなどの不正行為を目的としたプログラムは意図的に使用するポート番号を変更し，セキュリティイベントとして検知されるのを回避する傾向がある [28]．そのためポート番号だけに依存しないプロトコル判定機能が必要となる．

ROOK ではプレゼンテーション層のプロトコルの検出に，ペイロードのパターンを利用している．TCP のセグメント部分や UDP のデータグラム部分に着目し，正規表現やヘッダの情報を元にプロトコルを検出している．ペイロードのパターンを元にプロトコル判定する方法は [41] や [82] などで述べられている．[41] では幾つかの一般的なプロトコルに対して，ペイロードのパターンに着目するのが有効であると実ネットワークのトラフィックを用いて示している．[82] では，いくつかのメジャーな P2P ファイル交換ソフトウェアに対してもプロトコル検出手法の有効性を示している．他にもペイロードの機械学習によりプロトコルを検出する手法 [83] が挙げられるが，本実装では即時的かつ決定的にプロトコルを判定する必要があるため，ペイロードのパターンに着目する手法を採用した．

表 9.1 は ROOK に現在実装しているプロトコル判定条件の例である．条件として最初のパケットの宛先とペイロードのパターンを示している．宛先の列は S がサーバ，C がクライアント，空欄が任意の宛先を示している．正規表現の書式は Perl[84] 標準のものに準拠している．多くのテキストベースのプロトコルは，正規表現と宛先で判定ができる．DNS や SSL，TFTP などの一部のプロトコルに関しては，プロトコルヘッダに含まれているデータ長のフィールドと実際のデータ長を比較することで，当該プロトコルに準拠したパケットデータかどうかを判断できる．本実装では図 9.3 に示した通り TCP と UDP がこの機能を利用しており，この機能の結果によって次に遷移する解析プロトコルのモジュールを決定している．

表 9.1: ROOK で実装されているペイロードベースのプロトコル判定条件例

<table>
<tr><th>プロトコル</th><th>宛先</th><th>ペイロードのパターン</th></tr>
<tr><td>HTTP</td><td>S</td><td>正規表現: "^(get|post|head|search|connect|propfind|options|report|delete|notify|lock)[ ]+\S+"</td></tr>
<tr><td>SSH</td><td></td><td>正規表現: "^ssh\-\d\..*\x0a$"</td></tr>
<tr><td>IRC</td><td>S</td><td>正規表現: "^(pass|nick|user)[ ]+\S+.*[\x0d\x0a]+"</td></tr>
<tr><td rowspan="2">SMTP</td><td>C</td><td>正規表現: "^\d{3} .*\x0d\x0a$" かつ送信元ポート 25 番</td></tr>
<tr><td>S</td><td>正規表現: "(HELO|EHLO)[ ]+\S" かつ宛先ポート 25 番</td></tr>
<tr><td>FTP</td><td>C</td><td>正規表現: "^\d{3} .*\x0d\x0a$" かつ送信元ポート 22 番</td></tr>
<tr><td>POP3</td><td>C</td><td>正規表現: "^+OK "</td></tr>
<tr><td>eDonkey</td><td></td><td>正規表現: "^\xe3[\x4d\x19\x73][\x01\x00]\x00\x00\x01"</td></tr>
<tr><td>MSN MSGR</td><td>S</td><td>正規表現: "^(USR|ANS) \d+ \S+@[0-9A-Za-z\.-]+ [0-9]+\.[0-9]+"
もしくは "^VER [0-9]+ MSNP[0-9]+ "
もしくは "^CVR [0-9]+ [0-9a-zA-Z]+ \S+ "</td></tr>
<tr><td>IMAP</td><td>C</td><td>正規表現: "* OK \[CAPABILITY IMAP"</td></tr>
<tr><td>Samba</td><td></td><td>正規表現: "^\x00...\xffSMB\x72\x00\x00\x00\x00"</td></tr>
<tr><td>CVS</td><td>S</td><td>正規表現: "^BEGIN AUTH REQUEST"</td></tr>
<tr><td>AIM</td><td></td><td>正規表現: "^\x2a\x01.*\x00\x00\x00\x01"</td></tr>
<tr><td>YMSG</td><td></td><td>正規表現: "^YMSG\x00"</td></tr>
<tr><td>Gnutella</td><td></td><td>正規表現: "^GNUTELLA CONNECT/[\d\.]+\x0d\x0a$"</td></tr>
<tr><td>BitTorrent</td><td></td><td>正規表現: "^\x13BitTorrent protocol"</td></tr>
<tr><td>NetBIOS</td><td></td><td>1 バイト目が 0x81 か 0x82，かつ 3 バイト目と 4 バイト目を 16 ビット整数と見なし 4 を足したときにパケット長と等しい</td></tr>
</table>

注 1: 宛先の S はサーバ，C はクライアントを示している．空欄は任意の宛先．
注 2: 正規表現では大文字，小文字の区別をしていない．

9.4 Rule コンポーネント

第 8.3 節での議論をもとに，複数セッションの相関関係を記述するルールを実装した．ROOK のルールは XML[75] で記述されている．

ルールはセッションルール，トリガ，シグネチャ，条件の要素によって構成されている．まず，ルールは 1 つ以上のセッションルールに分解できる．セッションルールとはセッション毎に検知やパラメータ操作をするためのルールとなる．さらに，セッションルールは 1 つ以上のトリガと状態によって構成されている．トリガが複数の場合は状態の遷移条件として動作し，1 つの場合は単純な検査条件として動作する．トリガは 1 つ以上のシグネチャによって構成される．シグネチャは具体的な条件の集合で，1 つのトリガに複数のシグネチャがある場合は，並列条件として機能する．シグネチャを構成する条件はトラフィックのパターンやパラメータの状態が指定されている．複数の条件は直列条件として機能する．

図 9.8 にルール基本構造の概念を示している．図中のルールは 2 つのセッションルールを持っている．セッションルール 1 は 2 つのトリガと 1 つの状態によって構成されている．

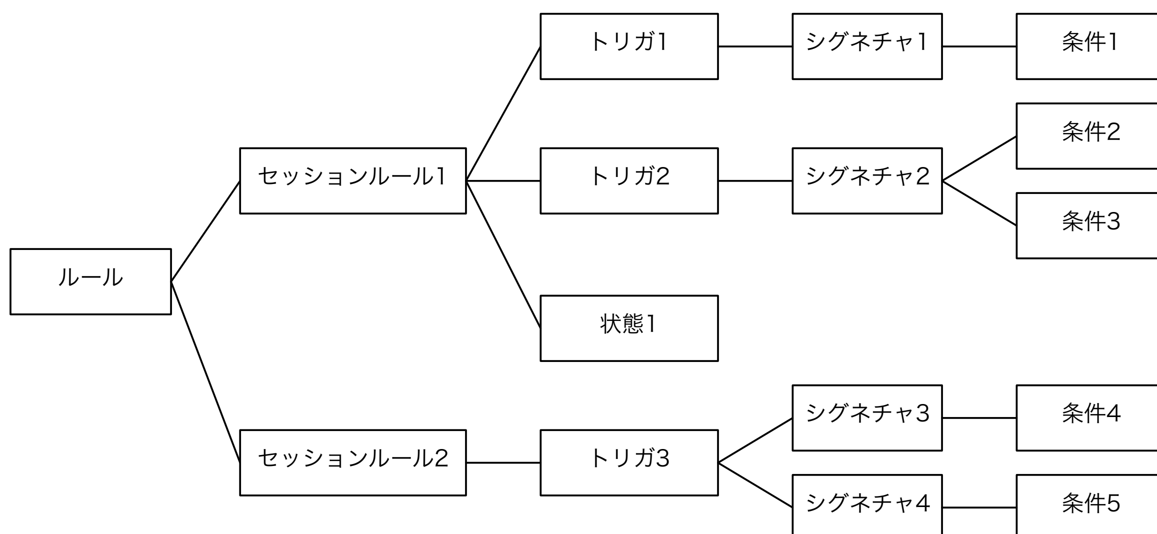

図 9.8: ルール基本構成概要

状態は記述者が任意に定義できる．トリガ 1 とトリガ 2 はそれぞれ 1 つずつシグネチャを持つ．シグネチャ2 は 2 つ条件を持つため，条件 2 と条件 3 に一致することでトリガ 2 が動作する．一方，セッションルール 2 は 1 つしかトリガを持たないが，トリガは 2 つのシグネチャで構成されている．この場合はシグネチャ3 に含まれる条件 4 とシグネチャ4 に含まれる条件 5 のどちらが一致してもトリガ 3 は動作する．

9.4.1 ルール書式

図 9.9 は基本的なルールの記述構造について示している．`<rule>`タグは 1 つのセキュリティイベントを表しており，タグ内部は`<param>`タグによるパラメータの宣言部，`<session>`タグによるセッションルールの定義部，`<confirm>`タグによるチェッカーの定義部の 3 つに分かれる．これらのタグの出現規則は任意である．

`<session>`タグ内部は`<stat>`タグと`<trigger>`タグで構成される．この二つがセッションルールの状態と状態遷移条件を表している．また，`<session>`タグは必ず一つのプロトコルを設定する．これは第 9.3.1 節で示したモジュール名のいずれか，もしくは表 9.1 で示したプロトコル名のいずれかを指定する．

`<stat>`タグは状態を表している．それぞれルール内で一意の名前を指定する必要がある．ただし，*start* と *end* は必ず状態遷移の開始と終了を意味している．*end* 状態の上書きは可能となっている．`<stat>`タグ内ではパラメータを書き込む`<set>`，パラメータを消去する`<unset>`，検知内容などを出力する`<act>`がそれぞれ記述可能である．

`<trigger>`タグは状態遷移条件を表している．遷移条件そのものは`<sig>`タグによるトラフィック比較を基準とした条件と，`<sig>`タグ内に`<eval>`タグを付与したパラメータとの比較による条件がある．両者は併用して条件にできる．状態遷移条件を満たした場合に実行される内容として，`<stat>`タグと同様に`<set>`，`<unset>`，`<act>`を記述できる．

<trigger>タグ内の<sig>タグはそれぞれ並列条件（論理和）として扱われる．<sig>タグには属性として複数の条件を指定でき，これらは直列条件（論理積）として扱われる．図 9.9 の<sig>タグでは，条件 1 と条件 2，条件 3 と条件 4 がそれぞれ論理積の関係になる．<sig>タグ内に包含されている<eval>タグはパラメータの値とトラフィックデータ，もしくは記述された静的なデータとの比較に用いられる．各タグに記述できる属性は付録 A の A.1 で示している．

9.4.2 ルール記述例

図 9.10 は本実装のルール記述例を示している．これはボットの活動によるセキュリティイベントを検知するためのルールとなっており，ボットが特定のドメイン名を DNS で問い合わせた後，IRC で C&C サーバに接続しにいく様を表している．図中は 1 つのルール (R_1) を示しており，ルール中には 2 つのセッションルール ($S_{1,2}$) と 1 つのパラメータ (P_1) がある．

P_1 はボットが接続する IRC サーバの IP アドレスを保持するためのパラメータである．`type="addr"`はパラメータの型が IP アドレスである事を示し，`mode="stack"`は上書き時に，値を積み重ねられる事を示している．

S_1 は DNS でボットが利用するドメイン名の問い合わせと，その応答を表している．S_1 には開始状態 q_s と終了状態 q_e の他に q_1(`res_wait`) がある．T_1 は $q_s \rightarrow q_1$ の状態遷移条件である．<trigger>タグの `from` 属性は初期値で`"start"`となっており，q_s を示している．<sig>では `dns.query="botcc.example.com:A"`でドメイン名 botcc.example.com の A レコード問い合わせの出現を表している．T_1 に一致するパケットデータが現れた場合，当該セッションの状態が q_1 に遷移する．q_1 では<act>タグでメッセージが指定されており，遷移したタイミング (T_1 が一致した場合) で`"waiting"`と出力される．T_2 は $q_1 \rightarrow q_e$ であるため，セッションの状態が q_1 をの時だけ T_2 が検査される．T_2 内の<sig>では `dns.answer_query="botcc.example.com:A"`として，応答に含まれるクエリに botcc.example.com の A レコードがあった場合に条件が一致する．条件が一致した場合，<set>タグで応答に含まれる A レコード (`filter="dns.answer_res_data" key=":A"`) を P_1 に書き込まれる．`timeout="120"`と設定しているため，120 秒後に破棄される．P_1 は上書きが stack モードに設定されているため，破棄されるまでに 120 秒間で再度 P_1 に書き込みが発生した場合は，以前に書き込まれた値とは別に，新たに 120 秒間保持される値が付加される．

S_2 は IRC 接続を発見するためのセッションルールである．T_3 では<sig>タグの `tcp.dir = "to_server"`でサーバへの接続を示し，かつ<eval>タグでパケットの宛先 IPv4 アドレスと P_1 に保存された IPv4 アドレスを比較している．P_1 に二つ以上の値が書き込まれていた場合，パケットの宛先 IPv4 アドレスと p_1 の全ての値を比較する．T_3 の条件が満たされた場合は，<act>タグに設定されているメッセージが出力され，<unset>タグによって P_1 の内容が破棄される．

```
<rule name="ルール名">
 <!-- パラメータ宣言部 -->
 <param name="パラメータ名" type="型" mode="記憶形式" />

 <!-- セッションルール 1 -->
 <session proto="対象プロトコル">

  <!-- トリガ（状態遷移条件）-->
  <trigger from="遷移元状態" to="遷移先状態">
   <sig 条件 1, 条件 2, ... /> <!-- 並列条件 1 -->
   <sig 条件 3, 条件 4, ... /> <!-- 並列条件 2 -->
   <sig>
    <eval パラメータ比較に関する情報 /><!-- パラメータと定義済みデータ or トラフィックとの
比較 -->
   <sig>

   <!-- トリガが発動した場合のパラメータ保存 -->
   <set パラメータ保存に関する情報 />

   <!-- トリガが発動した場合のパラメータ削除 -->
   <unset パラメータ削除に関する情報 />

   <!-- トリガが発動した場合の出力 -->
   <act type="出力形式" arg="必要情報" />

  </trigger>

  <stat name="状態名">
    <!-- パラメータ操作や出力に関する条件 -->
  </stat>

 </session>

 <!-- チェッカー -->
 <confirm>

  <sig> <!-- confirm ではトラフィックデータは条件に使えない -->
   <eval パラメータ比較に関する情報> <!-- 定義済みデータのみと比較可能 -->
  </sig>

  <!-- パラメータ操作に関する条件 -->
 </confirm>

</rule>
```

図 9.9: ルールの基本的な記述形式

9.4.3 セッションルールの分離性に関する考察

1 つのルールに含まれる複数のセッションルールは，基本的にそれぞれが独立した構造となっている．そのためルールの実装上，セッションルールの分離性が保たれることで検

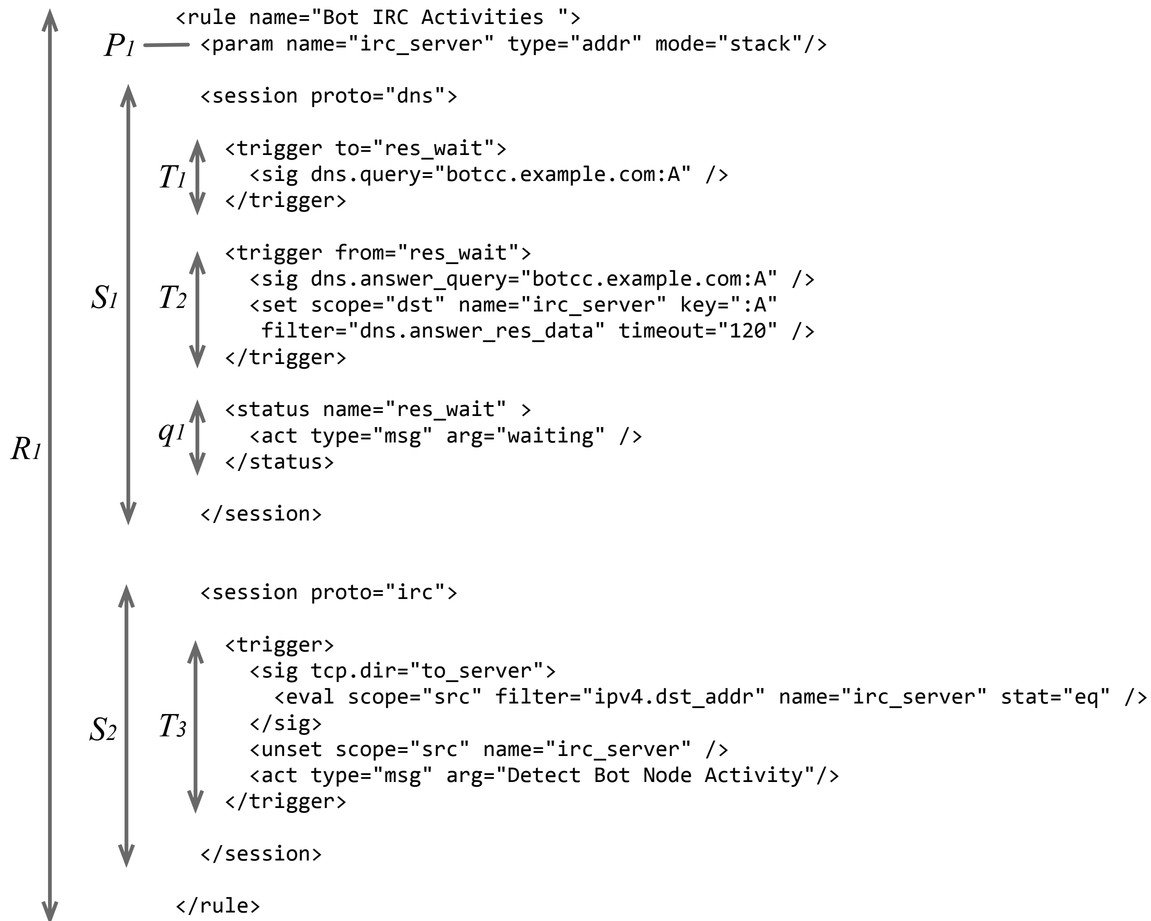

図 9.10: ルール記述例

知処理実装を並列処理化できる．

第 8.3.2 節で述べたとおり，各セッションルールはパラメータ機能を用いることによって依存関係が成り立っている．第 8.3.4 節で示したセッションの出現規則の制御や，セッションに含まれるデータの交換はパラメータ機能の利用によるものである．ただし，セッションルールがそれぞれ独立して動作したとすると，パラメータは書き込みと読み込みによって制御されているため，予期せぬ上書きが発生する可能性がある．そのため，ルール記述者に対してどのようなロック制御が起きているかを明確化する必要がある．

これは一般的なデータベースなどのトランザクション処理でも，同様の概念で実装されている．パラメータへの書き込みと読み込みは，オブジェクトに対する入出力として捉えることができる．[85] によると，以下の 3 種類の不正な依存関係が排除されていれば分離生を実現し正常に並列処理が実行されていると言える．それぞれ 2 つの処理 $T_{1,2}$ が実行されると仮定して，不正な関係が発生する様を説明している．

- 更新の消失: 別の処理によって書き込まれた値を無視して，値を上書きしてしまう状態である．T_1 が値を読み込んだ後，T_2 が異なる値を書き込む．その後，T_1 が読

み込んだ値を元に書き込みをすると，T_2 による書き込みが無視されてしまう．

- ダーティリード: 連続した書き込みの途中に別の処理が値を読み込むと，その値が矛盾している可能性がある．T_2 が書き込んだ後に T_1 が値を読み込み，再度 T_2 が書き込みをする状態である．T_1 が読み込んだのは T_2 の処理途中であるとすると，T_1 の値が正常ではない可能性がある．
- 非再現リード: 連続して値を読み込む場合，読み込んだ値が一貫していない状態である．T_1 が値を読み込んだ後に T_2 が書き込むと，T_1 が再度読み込んだ値が前回の値を異なってしまう．

本実装のセッションルールにおける状態遷移条件 (トリガ，`<triger>`タグに相当) を 1 つのトランザクション処理と見なす場合，一般的なトランザクション処理と異なる点が 2 つある．1 つめは，読み込みと書き込みの処理構成が全てのトリガで同じである．パラメータに対する操作は`<eval>`，`<set>`，`<unset>`の 3 つである．`<eval>`タグが読み込み処理に，`<set>`，`<unset>`が書き込み処理に相当する．トリガでは`<eval>`タグで遷移条件を検査し，遷移条件が満たされた場合に`<set>`，`<unset>`が適用される．そのため，常に読み込み → 書き込みの順序で処理が完結する．また，読み込みは 1 つのトリガで複数発生する可能性があるが，書き込みは 1 つのパラメータに対して必ず 1 度しか発生しない．これは書き込みが一度に実行されるので，複数回の書き込みが意味をなさないため，実装としてサポートしない．2 つめは，書き込み処理の取り消し (ロールバック) が発生しない点が挙げられる．1 つのトリガの書き込み処理はまとめて実行されるため，書き込み処理の実行中に書き込みの中止は発生しないという前提を設ける．

相違点を踏まえることで，ダーティリードは発生しない事がわかる．ダーティリードは 1 つのパラメータに対して，1 つの処理中に複数回書き込まれた場合に発生する．そのため，書き込み処理が 1 回しか発生しない本実装ではダーティリードにはならない．

更新の消失と非再現リードを防止するためにはパラメータのロックを用いる．一般的な 2 相ロックおよび，共有ロックと排他ロックを利用する．これらのロックはルール中で指定しなくても自動的に共有ロック，排他ロックが実行されるものとする．また，ロックの順序を規定することによって，デッドロックの発生を防ぐ．2 相ロック構造を使用しているため，ロックに失敗した場合は再度ロックを試みる．セッションルールは独立性を保つため検査順序は任意となっている．そのため，ルール数が少ない場合はセッションルールによるロックが同時に発生する確率が高いが，ルール数が多くなることによってロックが競合する確率は低くなる．また DNS と HTTP のように，同じトラフィックデータに含まれる可能性が極めて低いプロトコル同士であれば，ロックが競合する可能性もほとんどないと言える．

これらの機能を実装することによって，本実装で示したルールでも各セッションルール毎の分離性を保つことが可能になると考えられる．ただし現状ではセッションルール数が多くないため，並列処理によって機能が低下する恐れがあり，検知処理の並列処理機構は実現されていない．これは今後の課題として第 11.2.4 節でも述べる．

```
 1|detect (PacketData P, RuleSet R) {
 2|  for (i = 0; i < |P|; i++) {

 3|    if (NULL != (M  = getMultiplexFilter (R, P[i])))
 4|      Sm = runMultiplexFilter (P, M);

 5|    Ss = getSignleRuleSet (R, P[i]);
 6|    S  = merge (Sm, Ss);

 7|    foreach (S as v) {
 8|      flag = true;
 9|      foreach (v as c) {
10|        if (match (c, P) == false) {
11|          flag = false;
12|          break;
13|         }
14|      }
15|      if (flag == true)
16|        action (r, P);
17|     }

18|   }
19| return ;
20|}
```

図 9.11: 検知処理の擬似コード

9.5 Detection コンポーネント

Detection コンポーネントは Input コンポーネント，Decoding コンポーネント，Rule コンポーネントをもとに構成される実質的な検知機構である．図9.1 の通り，Input コンポーネントによってトラフィックデータを読み込み，Deecoding コンポーネントによってプロトコル解析，セッションを管理する．そして Rule コンポーネントによって読み込まれたルールを検知用にコンパイルし，Decoding コンポーネントから得られたデータと比較する．ルールの条件が満たされた場合，ルールで指定された情報の出力が実行される．

9.5.1 検知処理に関する実装

実際の検知処理は多重フィルタと単一フィルタの組み合わせによって実装されている．多重フィルタは計算量を大幅に減らせる反面，限られた項目しか一括検索の対象にできない．例として TCP を挙げると，Snort などでは TCP では宛先ポート番号が条件に含まれるルールが多ので，宛先ポート番号を多重フィルタで検査すれば処理負荷を大幅に下げられる．しかし，必ずしも全ての TCP ルールに宛先ポート番号が含まれているわけではないため，多重フィルタが適用できないルールもある．検査できる項目を 1 つしかもたない．そのため，本実装は多重フィルタと単一フィルタを組み合わせて検知している．

基本的な検知処理の流れを擬似コードによって図 9.11 に示している．図 9.11 は C 言語などの形式に類似させた擬似コードで記述しており，左部分が行番号となる．この行番号を用いて図を説明する．図で示した検知処理は 1 度のパケット到着に対して 1 回実行される流れである．

検知関数 `detect()` の引数 `PacketData P` は，Decoding コンポーネントによって得られたデータ，`RuleSet R` は Rule コンポーネントで読み込んだ全てのルールになる．`P[i]` は i 番目のレイヤを表している．ただし，必ずしも OSI 参照モデルのレイヤ構造に従うわけではなく，Input コンポーネントで取得したパケットデータの先頭レイヤや，ネットワークの構成などに依存する．例えば Ethernet，802.1q，IPv4，TCP を含むパケットの場合，`P[0]` が Ethernet，`P[1]` が 802.1q，`P[2]` が IPv4，`P[3]` が TCP の解析結果となる．3 行目から 17 行目までの繰り返しで，パケットに含まれる各層のプロトコルを検査している．

3 行目ではプロトコルに該当する多重フィルタのデータセット `M` を取得し，4 行目では構造体 `M` を利用して多重フィルタを実行している．`getMultiplexFilter()` は全てのルールセット `R` から，`P[i]` のプロトコルに対応した多重フィルタのデータセットを取得する．現在，多重フィルタは IPv4 の宛先 IP アドレス，TCP の宛先ポート番号がそれぞれパトリシアツリーによって実装されている．Ethernet，IPv6，UDP，ICMPv4，DNS，HTTP，TFTP には多重フィルタが実装されていないため，`getMultiplexFilter()` の結果は `NULL` になる．もしくは IPv4，TCP の各セッションルールに多重フィルタへ登録できる項目が無かった場合も，それぞれの `getMultiplexFilter()` 出力結果が `NULL` になる．`getMultiplexFilter()` の結果が非 `NULL` の場合，`M` と `P[i]` を用いて `runMultiplexFilter()` によって多重フィルタが実行される．多重フィルタはルール数の増加に対して処理負荷が線形に増加しないようになっているため，少ない負荷で一括して検査すべきルールを絞り込める．実行した結果はシグネチャの集合として `Sm` が出力される．

5 行目では単一フィルタのみで構成されるシグネチャの集合 `Ss` を取得し，6 行目で `Rs` と `Rm` を結合している．`Rs` は多重フィルタを利用できない条件で構成されているルールセットである．実際には`<sig>`タグに記述された内容の集合となっている．`<sig>`タグには複数のフィルタによる条件やパラメータを参照する条件がある．これらの条件の中で多重フィルタで処理できるものは一括して 4 行目で処理されるが，多重フィルタで利用できない条件しか持たないルールは個々に条件を検査している．一方で，多重フィルタによって一括検知されたルールも検査しなければならない条件を残している場合がある．例えば IPv4 では，送信元 IP アドレスと宛先 IP アドレスが条件となっていた場合，宛先 IP アドレスは多重フィルタで一括処理されたとしても，送信元 IP アドレスは別途確認しなければならない．そのため，6 行目では単独の条件で構成されるシグネチャの集合 `Ss` と多重フィルタで一致したが検査すべき条件が残っているシグネチャの集合 `Sm` をマージして `S` としている．

7 行目から 17 行目まではシグネチャの集合 `S` の条件を全て検査している．7 行目ではシグネチャの集合から個別のシグネチャ `v` を取り出しており，9 行目ではシグネチャ `v` から個別の条件 `c` を取り出している．`c` はトラフィックデータと条件の照合とパラメータの値の照合の 2 種類となるが，共に 10 行目の `match ()` によって検査されている．8 行目で `flag` に $true$ を代入し，一致しない条件が発見された場合には $false$ を代入する．15 行目では

`flag`の値を検査して，全ての条件に一致したかを確認し，一致していた場合は16行目で`action ()`を実行する．`action ()`はパラメータの操作や，結果の出力となる．

9.5.2 パラメータ操作

Detectionコンポーネントでのパラメータ操作は，パラメータのプロトタイプ管理，ルールにおける`<set>`，`<unset>`，`<eval>`との関連づけが挙げられる．

全てのパラメータはプロトタイプを元にして生成される．このプロトタイプはルールの`<param>`タグにおける記述に相当する．パラメータの型，上書きモード，名前が定義されている．また，ルールの読み込み時に全てのパラメータプロトタイプに一意の32ビットのID（Param ID）が割りあてられる．第9.3.4節で説明したとおり，パラメータユニットはパトリシアツリーで構成されており，Param IDはキーとして利用される．

第9.4節で述べたように，パラメータの書き込みや読み込みはトリガに関連づけられている．各トリガがSetやEvalの命令を持ち，各命令がそれぞれ必要とするパラメータのプロトタイプを参照している．書き込み（Set）が発生した際には，既に当該パラメータが対象とするパラメータユニットに存在しているかを確認し，存在しなければプロトタイプからパラメータを生成する．存在した場合は上書きモードを確認し，上書きが許可されていれば書き込む．上書きモードがstackの場合はデータ本体の変わりにパトリシアツリーを構成し，ポインタをパラメータとして書き込む．パラメータが破棄されるまでのタイムアウト値は各Set命令が保持しており，それぞれが異なるタイムアウト値を設定できる．パラメータを書き込む対象（セッション，送信元ホスト，宛先ホスト，グローバルのいずれか）もSet命令において定義されている．

図9.12ではDecodingコンポーネントで提供している機能も含めた，パラメータのデータ構造と管理構造の例を示している．図中ではトリガT_aがSet命令を2つ，トリガT_cがEval命令を2つ有しており，それぞれ2種類のプロトタイプを参照している．Param ID 1のパラメータ(P_1)は上部のパラメータユニットに，Param ID 2のパラメータ(P_2)は上部と下部の両方のパラメータユニットに書き込まれている．書き込み先はSet命令で指定されており，異なる複数のパラメータユニットに同じParam IDのパラメータを書き込める．Eval命令もプロトタイプを参照してキーとなるParam IDやパラメータの型などの情報を得る．

Detectionコンポーネントではパラメータに型の概念を持たせている．これはEval命令を実行する際に大小比較をする可能性があり，整数と文字列，IPアドレスなどを区別する必要があるためである．表9.2に現在ROOKで実装されているパラメータの型の一覧を示している．整数は32ビット以内の整数値，文字列は無制限長のASCII文字データ，IPアドレスはIPv4およびIPv6のIPアドレス，RAWデータは無制限長のバイナリデータを表している．

Set命令でパラメータに書き込む内容やEval命令でパラメータを読み込み比較する内容は，静的なデータ（定義済みデータ）と動的なデータ（トラフィックデータ）が利用できる．静的なデータを利用する場合は，ルール記述の際に`<set>`もしくは`<eval>`タグ内で`data`属性を使う．これによって，比較対象のパラメータの型に変換される．例えばパラ

表 9.2: ROOK で実装されているペイロードベースのプロトコル判定条件

型名	タグ内での表記	説明
整数	`INT`	32 ビット以下の符号無し整数
文字列	`STR`	65000 バイト以下の ASCII 文字列
IP アドレス	`ADDR`	IPv4 / IPv6 アドレス
バイナリデータ	`RAW`	65000 バイト以下のバイナリデータ

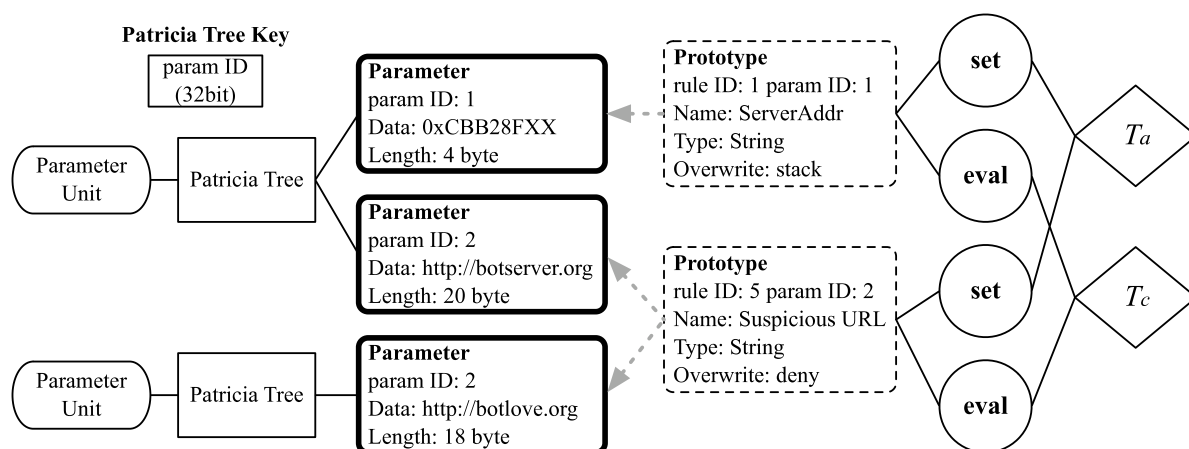

図 9.12: パラメータ操作の概要

メータの型がIP アドレスの場合，ルールに"127.0.0.1"と記述すれば比較用に 32 ビットの符号無し整数に変換される．また，動的にトラフィックデータから書き込み用や比較用のデータを使う場合は，フィルタを利用する．フィルタは条件の比較だけではなく，条件に一致した場合には返値を返す．例えばフィルタ `tcp.dst_port` では宛先ポート番号を返値とするため，`<eval>`タグで `filter="tcp.dst_port"` と記述すれば，パラメータの値と TCP の宛先ポート番号を比較できる．ただし，各フィルタには返値の型が決まっているため，これに一致する型のパラメータとしか比較できない．実装されているフィルタ一覧については付録 A の表 A.2 に示してある．

9.6 Output コンポーネント

Output コンポーネントは 2 次記憶装置への書き出しを制御している．実質的には，各出力形式へ検知結果の情報を変換することと，出力先ごとに適切な設定を管理するのが主な機能となる．

9.6.1 検知結果の出力形式

検知結果の出力形式にはテキスト形式，pcap 形式，syslog 形式が挙げられる．

テキスト形式 一般的なテキストによる形式．ファイルへの出力を想定している．統計処理などの負担はやや大きくなってしまうが，可読性が高いという利点がある．syslog 形式による出力と似ているが，こちらは複数行にわたる表現が容易であるため，一部のペイロードデータや，プロトコル解析に関する情報などを必要に応じて加えやすい．

pcap 形式 tcpdump[76] によって保存される形式．基本的にパケットの未加工データを保存するのみで，付加的な情報は保存できない．そのため，どのような事象があったかを記録するには不向きである．libpcap[71] を使っているプログラムであればほとんどが読み込み可能であるため，より高度なプロトコル解析や，送受信されたデータを正確に追跡できる利点がある．

9.6.2 出力される情報

本実装は複数のセッションルールによって検知処理をしているが，出力される情報は一般的なセキュリティイベント検知機構と類似している．複数のセッションルールを扱う場合，検知条件の一致は 1 つのルールに対して複数回発生するため，各検知結果の関連性が明らかであればルールのテストが容易になる，あるいはネットワークインシデントの追跡に有効であると考えられる．しかし，本実装ではセッション間の相関関係を複雑に表現できるため，検知条件が一致した場合でも，その関係を出力結果上で示すのが難しい．セッションの出現規則も第 8.3.4 節で有限オートマトンのように表記しているが，これはパラメータの使い方次第で何通りも表現可能であり，その全てに対応するのは難しい．そのため，一般的なセキュリティイベント検知機構のようにルールに記述された指示に応じて，トラフィックデータの一部と検知箇所を示すメッセージを表示させている．

出力形式にもよるが，具体的に出力させるべき情報は基本的に以下の通りである．

- ルール名
- セキュリティイベント識別名
- ネットワーク層プロトコル（IPv4，IPv6）
- 送信元／宛先ホスト識別子（IPv4 アドレス，IPv6 アドレス，MAC アドレス）
- トランスポート層プロトコル（TCP，UDP，ICMPv4，ICMPv6）
- 送信元／宛先トランスポート層識別子（TCP/UDP ポート番号，ICMP タイプ）
- セッション層プロトコル（SMTP，POP3，HTTP など）
- ペイロード（TCP セグメント，UDP データグラムなど）

9.7 本実装における処理負荷の考察

ルール数や条件の種類，トラフィック性質の変化が，本実装の処理負荷にどのような影響を与えるかを示すため，Decoding コンポーネントと Detection コンポーネントの計算量を求めた．計算量の導出にはアルゴリズム C[86] を参考とし，平均の計算量と最悪の計算量（最低限保証されている計算量）の 2 つを考慮して求めた．

9.7.1 Decoding コンポーネントにおける計算量

表 9.3: Decoding コンポーネントの計算量一覧

プロトコル	最悪の計算量	平均の計算量
Ethernet	$O(96)$	$O(\log S_ethernet)$
IPv4	$O(64)$	$O(\log S_ipv4)$
IPv6	$O(256)$	$O(\log S_ipv6)$
TCP	$O(32)$	$O(\log S_u)$
UDP	$O(32)$	$O(\log S_u)$
HTTP	$O(1)$	$O(1)$
IRC	$O(1)$	$O(1)$
TFTP	$O(1)$	$O(1)$
DNS	$O(16)$	$O(\log S_u)$

$$O(\log S_{ethernet}) + O(\log S_{ipv4}) + O(\log S_u) + O(1) \tag{9.1}$$

$$O(96) + O(64) + O(32) + O(1) = O(1) \tag{9.2}$$

Decoding コンポーネントの処理負荷はパケットに含まれるプロトコルと，各プロトコルにおけるセッション数に依存する．表9.3 に各プロトコル毎の計算量を示している．$S_ethernet$，S_ipv4，S_ipv6 は各モジュールで管理している全てのセッション数，S_u は下位レイヤーのセッション上に存在するセッション数となる．例えばTCP の場合はある IPv4 セッションか IPv6 セッションが下位セッションとなり，その上に別途ツリーが構成されている．(詳しくは第 9.3.2 節を参照）各プロトコルのセッションはそれぞれパトリシアツリーで管理されている．パトリシアツリーを参照する際の平均の計算量は保持しているノード数の対数，最悪の計算量がキー長となる．そのため，セッションの参照に必要となる計算量は平均がセッション数の対数，最悪がプロトコル毎のキー長 (Ethernet は $O(96)$，IPv4 は $O(64)$，IPv6 は $O(256)$）となる．

計算量は到着したパケットデータによって異なる．例えば，Ethernet，IPv4，TCP，HTTP で構成されれているパケットデータの場合，平均の計算量は式 9.2，最悪の計算量は式 9.2 となる．ここでの $S_{ethernet}$ は管理している全ての Ethernet セッション，S_{ipv4} は管理している全ての IPv4 セッションを表している．S_u は当該 IPv4 セッション上にある

TCP セッション数となる．HTTP セッションは同一 TCP セッション上に 1 つなので，計算量は定数となっている．最悪の場合はそれぞれのキー長に従う定数となる．ただし，最悪の計算量は全ての項が定数であるため，$O(1)$ としてまとめている．

式 9.2 で示されているとおり，Decoding コンポーネントの計算量は最悪の場合でも定数となり，成長しない．そのため，処理の負荷はトラフィックの性質によって影響されにくいことが分かる．

9.7.2 Detection コンポーネントにおける計算量

表 9.4: 多重フィルタの計算量一覧

プロトコル	アルゴリズム	検査対象	平均の計算量	最悪の計算量
IPv4	パトリシアツリー	宛先アドレス	$O(\log T)$	$O(32) = O(1)$
TCP	パトリシアツリー	宛先ポート番号	$O(\log T)$	$O(16) = O(1)$
HTTP	Aho-Corasick	ペイロード	$O(L)$	$O(1500) = O(1)$
DNS	Aho-Corasick	応答クエリ	$O(L)$	$O(1500) = O(1)$

L: パケットの平均長
T: 多重フィルタに登録されているシグネチャ数

検知処理における計算量を図 9.11 の擬似コードから求める．まず，図中の `detect ()` はパケットが 1 度到着する度に呼び出される．そこでパケットの到着数を P とおく．2 行目からの繰り返し回数は，パケットデータに含まれるプロトコルの数に相当する．実際には OSI 参照モデルにおける第 2 層から第 4 層，あるいは第 5 層までなので，3 回もしくは 4 回程度の繰り返しとなるため，計算量の導出において無視する．3 行目，5 行目，6 行目はそれぞれシグネチャの集合の抽出，結合のみなので，各計算量は $O(1)$ とする．これらも固定的な計算量であるため，計算量の導出においては無視する．4 行目の多重フィルタ実行の計算量はプロトコルによって異なる．各プロトコル毎の多重フィルタの計算量は表 9.4 に示しいる．7 行目からの繰り返しは `Sm` と `Ss` に含まれる全ての条件の数を検査している．`Sm` は多重フィルタで検査されるシグネチャの集合であり，この集合に含まれるシグネチャの条件数を M とする．`Ss` は `Sm` に含まれないシグネチャの集合であり，`Ss` のシグネチャに含まれる条件数を S とする．ただし，M は P 個のパケットが入力された，多重フィルタで出力されたシグネチャに含まれる条件の数である．S は 1 つのパケットに対して常に検査される条件の数なので，パケットが P 個到着するまでの間に検査される条件の数は $PS + M$ となる．

多重フィルタの計算量はプロトコル毎に異なるが，どれも条件数に対して増加しない．多重フィルタについては第 9.3.7 節で述べた通り，条件数によって負荷が単調に増加しないための機能となっている．具体的なプロトコル毎の多重フィルタは表 9.4 にまとめている．パトリシアツリーを利用している多重フィルタは，登録されている条件数 T に対して $O(\log T)$，Aho-Corasick アルゴリズムを利用している多重フィルタは，パケットの平均長

L に対して $O(L)$ となる．これらの計算量は最悪の場合に定数となる．IPv4 と TCP のパトリシアツリーは最悪の場合にキー長に従った定数となり，それぞれ $O(32)$ と $O(16)$ となる．また，本実装は第 9.3.1 節などで示したように Ethernet の監視を前提としているため，パケット長は最大で 1500 を想定している．よって L の最大値は 1500 となり，HTTP と DNS における最悪の計算量は定数である $O(1500)$ となる．そのため，多重フィルタの計算量はプロトコルによらず $O(1)$ とする．

条件の種類によっても計算量は異なるが，最悪の計算量は全て定数となる．まず，個別の条件は 1) 計算量が $O(1)$ の条件，2) 計算量が $O(L)$ の条件，3) パラメータを参照する条件の 3 つに分類できる．1 つめの計算量が $O(1)$ の条件とは，パケットヘッダに含まれる値の検査となる．具体例として IPv4 の送信元 IP アドレスの検査や TCP のフラグの検査，ICMP のタイプフィールドの検査が挙げられる．これらの条件は検査時に繰り返しが発生しないため，計算量は常に $O(1)$ となる．2 つめの計算量が $O(L)$ の条件とは，ペイロードなどのパターンマッチを実行する条件である．具体例として TCP のセグメントデータのパターンマッチや，UDP のデータグラムのパターンマッチが挙げられる．これらは多重フィルタの計算量の説明においても示したとおり，L の最大長が 1500 となるため最悪の計算量は $O(1500)$ となる．3 つめのパラメータを参照する条件は，ルールファイルにおける`<eval>`タグに相当する．検査に必要となる動作はパラメータの探索だが，パラメータはパトリシアツリーで管理されているため計算量はパラメータの保持数 N に対して $\log N$ となる．ただし，パラメータ管理のパトリシアツリーは 32 ビットのキーで管理されているため，最悪の計算量は $O(32)$ となる．このように，個別に検査する条件は全て最悪の場合に定数となるため，$O(1)$ として扱う．

$$O(PS + M) \tag{9.3}$$

$$\tag{9.4}$$

以上の議論を元に求めた計算量を式 9.4 に示す．P は到着するパケットの数，S は 1 つのパケットが到着したときに検査される条件の数，M は P 個のパケットに対して多重フィルタが出力したシグネチャに含まれる条件の数である．パケットが到着すると常に検査される条件数 S とパケット数 P による PS の部分は，それぞれ線形に成長する．M は最悪の場合に P と多重フィルタに登録されているシグネチャ数 U で PU となり線形に成長するが，多重フィルタの条件の記述と監視しているトラフィックの傾向によって大きく左右される．

9.7.3 処理負荷軽減についての考察

本実装の負荷を軽減するためには，多重フィルタによる条件の記述に留意しなければならないことが式 9.4 から分かる．多重フィルタを利用できないシグネチャの条件数 S はパケットの到着数が増加した際に，処理性能低下の原因となりやすい．多重フィルタを利用して出力されるシグネチャの条件数 M はパケット数 P に依存しにくいが，頻繁に出現するトラフィックを条件とした場合は PU に近づき，線形に増加していく．

そのため，S を小さくするために可能な限り多重フィルタを利用し，かつ M が増加しにくいように多重フィルタの条件を記述する必要がある．M は最悪の場合には PU として線形に成長してしまうが，出現しにくいトラフィックを条件として多重フィルタを記述すれば，M は定数に近づく．

例えばIPv4のシグネチャを記述する場合，可能な限り宛先IPアドレスを条件に含めることで，そのシグネチャが多重フィルタに登録され，S を小さくできる．宛先IPアドレスも範囲を狭めた条件が望ましく，*192.168.0.0/16* のようにIPアドレスレンジで指定するのではなく，*192.168.0.2* のように限定的に書くのが望ましい．さらに，トラフィックの傾向によって頻出する宛先IPアドレスを指定するよりは，まれにしか出現しない宛先IPアドレスを指定することで，M を小さくできる．このような条件の記述によって，検知処理の負荷を低減できる．

9.8 動作検証

本実装が意図した動作をしているか確認するために，動作検証を実施した．検証用のデータとして，1999 DARPA Intrusion Detection Evaluation[87] において公開されているFifth Week Friday outside tcpdumpのデータセットを利用した．このデータセットには，[26] で提案されているようなIDSによる検知を回避する通信が含まれている．具体的にはIPv4パケットのフラグメンテーション化やTCPパケットのセグメント送信順序を意図的に入れ替えた通信が挙げられる．そのため，通信の内容を正確に把握し，正常に検知処理が実行できているかを確認するために適している．

動作検証はフィルタを利用している条件の検証と，パラメータを利用している条件の検証の2つに分けて実施した．フィルタ機能は一般的なIDSと同様に，各プロトコルのヘッダやペイロードのパターンマッチによって条件が満たされているか確認する．これは他のトラフィック監視ソフトウェアを利用することで，検知結果の正当性を検証する．これは条件を検査する動作が正常であるかを検証する他に，トラフィックの解析や正規化が正常に実行されているかについても検証している．パラメータは本手法における独自の機能であるため，目視によって動作を検証する．

検証に利用した他の実装はtcpdump 3.9.5[76]，tshark 0.99.6[88]，Snort 2.8.0.1[25] の3つである．トラフィックを取得，解析するセキュリティイベント検知機構は，実装によって検知に利用できる項目やプロトコルが異なる．本実装で条件として利用できる項目やプロトコルを網羅するために，3つの実装を検証内容によって使い分けた．

9.8.1 フィルタを利用している条件の検証

フィルタを利用している条件の検証は，各フィルタを単体で動作させた条件と，複数のフィルタを組み合わせた条件に分けて検証した．各フィルタはそれぞれ異なるプロトコルヘッダやペイロードを参照しており，個々の動作検証が必要となる．さらに複数のフィルタを組み合わせた場合は積条件となるため，全ての条件が満たされた場合にのみ指定され

た動作をしているか確認する必要がある．そのため，先に単独のフィルタによる条件の動作を検証し，その後複数のフィルタによる条件の動作を検証する．

単独フィルタによる条件の動作検証に関する実施条件と結果を表 9.5 にまとめた．表に示しているフィルタの引数が本実装での検知条件となり，検証用の条件が検証用実装に読み込ませた条件となる．項目毎に利用した検証用の実装が異なり，それぞれ tcpdump を *T*，snort を *S*，tshark を *W* と示している．tcpdump と snort を使った検証では，それぞれの条件を満たした場合に pcap 形式のファイルで結果を出力する．本実装も条件を満たした場合に pcap 形式で出力する．pcap ファイルを直接比較するのは難しいため，この 2 つの pcap 形式ファイルを tcpdump や tshark で読み込ませテキスト形式で出力し，双方の出力結果を比較する．

検証の結果，表で示しているフィルタについて正常に動作している事が確認できた．表 9.5 における一致の列が `O` となっているフィルタは，他の実装との出力結果が完全に一致した．その他の `P` となっている項目は完全に一致しなかったが，目視によって動作の正当性を確認した．`tcp.flag` では IP パケットのフラグメンテーションが発生しており，tcpdump が出力した結果とは一致しなかったが，本実装はフラグメンテーションを再構築することで正常に TCP のフラグを確認できた．本実装の `dns.query` は DNS の Query 部分に含まれる文字列を条件にしているが，snort では Query 部分の文字列を対象にするという指定ができない．そのため，snort が Answers 部分に `.com` を含んでいる DNS の応答を誤検知しており，出力結果が一致しなかったことを目視で確認した．`http.uri` の検証では tshark を用いて TCP の宛先ポートが 80 番のパケットを出力し，“GET” と “.js” の両方の文字列を含む行を抽出した．そして，本実装によって出力された pcap ファイルも tshark で出力させたところ，結果が一致した．`http.user_agent` の検知結果には，TCP のセグメントが細分化されたトラフィックが含まれていた．snort は検知結果出力時に TCP セグメントのリアセンブルに利用したパケットを全て出力するが，本実装は検知に成功した時点でのパケットしか出力しない．そのため，snort の出力結果には部分的な TCP パケットが多く含まれており一致しなかったが，目視によって等価の検知結果であることを確認した．`tcp.seg_id` は，TCP セッションが確立した後に送受信されたセグメントデータの数である．検証用の引数は “1” であるため，TCP セッション確立後に送信されたセグメントデータを含む最初のパケットを検知する．検証のために tcpdump を用いデータセットを読み込んだ結果をテキストで出力し，egrep コマンドによって`" 1:[0-9]"`，`"ack [12]"`を満たし，`"(0)"`を含まない行を抽出した．この結果と本実装が出力した結果とを比較したところ，tcpdump の出力から抽出した結果には TCP の再送信が含まれていたため一致しなかった．本実装では再送信でも `tcp.seg_id` は増加することを意図しているため，これは正常な動作であったと言える．このように一部出力結果に相違が見られたが，結果として単独フィルタが意図した動作をしていることが検証できた．

複数フィルタによる条件の検証は，2 つの単独フィルタを組み合わせて実施した．積条件として扱われるため，複数条件のなかで 1 つでも異なる条件があれば，条件は満たされなくなる．本実装には `ipv4.src_addr="172.16.113.84"` と `tcp.dst_port="80"`の 2 つの条件を与え，tcpdump には `tcp and src host 172.16.113.84 and dst port 80` という条件を与えた．これによって得られた結果を tcpdump によってテキスト形式に変換

し，diff コマンドで確認したところ完全に一致した．

これらの結果によって，本実装はフィルタを用いた条件から目的のトラフィックを抽出できることを示せた．これは条件を検査する挙動だけではなく，セッションの管理，IP フラグメンテーションや TCP セグメントの再構築，ヘッダの解析といったトラフィックデータの解析や正規化が意図した通りに動作していることがわかる．

9.8.2 パラメータを利用している条件の検証

パラメータを利用している条件の検証は，各実装の出力結果を目視で確認しながら実施した．パラメータ機能は本実装固有の機能であり，他の実装を使って同様の結果を出力させるのは難しい．ただし，パラメータ機能はそれ単独で利用するのではなく，フィルタ機能と併用して利用するように実装されている．そのためフィルタ機能による条件は他実装によって確認し，パラメータ機能を利用する条件が正しく機能していることは目視によって確認した．

パラメータを利用している条件の検証には 2 つのルール $R_{p,q}$ を利用した．それぞれ，R_p を図 9.13 に，R_q を図 9.16 に示している．第 6 章で述べたとおり，本手法で検知条件とすべきセッションの相関関係とはセッションの出現規則とセッション間の情報交換である．本実装でこの 2 つを検知条件として利用できることを示すため，R_p ではセッション出現規則を，R_q ではセッション間の情報交換を条件として利用している．

R_p はあるホストから送信された HTTP リクエストの URI に，“game” という文字列が含まれていた回数を検知条件に利用している．ルールではパラメータ `game_count` を用意している．そして上部の`<trigger>`タグ内では，URI に “game” という文字が出現 (`http.uri="game"`) する度に `game_count` に 1 を書き込んでいる．ただし`<set>`タグにおいて `opt="inc"`としているため，既に書き込まれている数値に 1 を足した数が書き込まれる．下部の`<trigger>`タグでは，URI に “game” という文字が含まれており，かつパラメータ `game_count` が 2 よりも大きかった場合に，`http_game.pcap` へ pcap 形式のデータを出力する．また，検証を容易にするため `ipv4.src_addr="172.16.113.105"` とし，172.16.113.105 から送信されたリクエストに限定している．

R_p の動作を検証するために tshark によってデータセットを解析し，本実装が R_p を適用してデータセットを読み込んだ際の出力と比較した．図 9.14 に tshark によって 172.16.113.105 からの GET リクエストを抽出し，さらに “game” という文字列が URI に含まれていたものを示す．見やすさを優先するために tshark の出力の一部を修正してあり，左からパケットが出現した相対的な時間 (秒数)，送信元 IP アドレス，宛先 IP アドレス，送信された URI の内容となっている．さらに図 9.15 に，R_p を適用した本実装が出力した結果を示す．

図 9.14 と図 9.15 を比較することで，本実装がセッションの出現規則を条件として利用できていることがわかる．まず，図 9.14 における 1 行目，2 行目の HTTP リクエストでパラメータの値が 2 まで増加する．そして 3 用目のリクエストで 3 となり，R_p の`<eval>`タグで示している条件を満たすことで，検知結果を図 9.15 の 1 行目に出力している．パラメータ `game_count` は最後の書き込みから 100 秒後に時間切れで破棄される．そのため図 9.14 の 4 行目，5 行目ではパラメータは 4，5 と増え続け，R_p の`<eval>`タグで示してい

表 9.5: フィルタ機能検証項目の一覧

フィルタ名	フィルタの引数	検証	検証用の条件	一致
ipv4.src_addr	209.67.29.11	T	ip and src host 209.67.29.11	O
ipv4.dst_addr	209.67.29.11	T	ip and dst hsot 209.67.29.11	O
ipv4.proto	6	T	tcp	O
tcp.dst_port	80	T	tcp and dst port 80	O
tcp.src_port	80	T	tcp and src port 80	O
tcp.seg_size	100	S	tcp any any <> any any (dsize: 100;)	O
	1400~	S	tcp any any <> any any (dsize: >1399;)	O
tcp.seg_id	1	T	本文中にて説明	P
tcp.dir	to_server	S	flow:to_server,established;	P
tcp.flag	S,RFAS	T	((tcp[tcpflags] & (tcp-syn|tcp-fin |tcp-ack|tcp-rst)) == tcp-syn)	P
udp.src_port	161	T	udp and src port 161	O
udp.dst_port	161	T	udp and dst port 161	O
udp.content	port	S	udp any any <> any any (content: "port";)	O
udp.size	100	S	udp any any <> any any (dsize: 100;)	O
dns.query	.com.	S	udp any any <> any 53 (content: "|03|com";)	P
dns.answer_query	.blizzard.	S	udp any 53 -> any any (content: "|08|blizzard";)	O
dns.answer_res_data	.blizzard.	S	udp any 53 -> any any (content: "|08|blizzard";)	O
http.uri	.js$	W	本文中にて説明	O
http.user_agent	Lynx	S	tcp any any -> any 80 (pcre:"/User-Agent\: Lynx/i";)	P
http.dl_data	hello	S	tcp any 80 -> any any (content: "hello";)	O
icmp4.type	0	T	'icmp[icmptype] == 0'	O
icmp4.code	3	T	'icmp[icmpcode] == 3'	O
irc.content	hello	S	tcp any any <> any 6660:6669 (content: "hello";)	O

(検証) T: tcpdump，S: Snort，W: tshark

(一致) O: 一致した P: 一致しなかったが挙動として正しいことを確認した

る条件を満し，結果を出力している．しかし，図 9.14 の 6 行目では 5 行目のリクエストから 190 秒以上経過している．そのためパラメータはすでに破棄されており，<eval>タグの条件は満たされず，図 9.15 には出力されていない．図 9.14 の 8 行目では 6 行目，7 行

```
<rule name="Test Rule: Rp">
  <param name="game_count" type="int" mode="allow" />

  <session proto="http">
    <trigger>
      <sig ipv4.src_addr="172.16.113.105" http.uri="game" />
      <set scope="src" name="game_count" data="1" timeout="100" opt="inc" />
    </trigger>

    <trigger>
      <sig ipv4.src_addr="172.16.113.105" http.uri="game" >
        <eval scope="src" name="game_count" data="2" stat="gt" />
      </sig>
      <act type="dump" arg="http_game.pcap" />
    </trigger>
  </session>
</rule>
```

図 9.13: パラメータ機能検証用ルール R_p

目で再びパラメータ game_count が 3 となるため<eval>タグの条件を満たし，図 9.14 の 4 行目に出力されている．その後，図 9.14 の 9 行目から 13 行目まで 100 秒の間に 3 回以上リクエストが発生していないため検知されていない．14 行目で再び 100 秒以内に 3 回目のリクエストが発生しており，図 9.14 の 5 行目に出力されている．この検証によって，本実装がセッションの出現規則を検知条件とし，さらに時間切れの処理が正常に実行されていることを示せた．

R_q は宛先ポート 53 番の UDP パケットに対して，ICMP の Port Unreachable メッセージが返されているかを検知するルールである．パラメータ udp_server を用意し，宛先ポートが 53 番の UDP パケットを発見した場合，宛先 IP アドレスを udp_server に登録する．udp_server は上書きモードが stack に指定されているため，書き込みが複数回発生してもそれぞれの値を独立して保持し続ける．その後，UDP を送信したホストに対して ICMP の Port Unreachable メッセージが送信されてきた場合，udp_server に保持されている IP アドレスと ICMP パケットの送信元 IP アドレスを比較し，一致する IP アドレスを発見した場合に pcap 形式で icmp_port_unreach.pcap に結果を出力する．これによって，送信した UDP パケットに含まれる宛先 IP アドレスを，ICMP Port Unreachable メッセージの検知に利用することになる．

R_q を適用した本実装による出力結果と，tcpdump を利用して抽出した UDP パケットの宛先ポート 53 番に対する ICMP Port Unreachable メッセージを比較し，本実装が正確に動作していることを確認した．図 9.17 には tcpdump によって ICMP Port Unreachable メッセージを出力し，さらに UDP の宛先ポート 53 番に対するメッセージを grep によって抽出した．一方，図 9.18 では R_q を適用した本実装によって出力された結果を示している．ICMP Port Unreachable メッセージにはエラーの原因となったパケットの情報が含ま

```
   57.989702 172.16.113.105 -> 207.25.71.141 HTTP GET /images/fronticons/icon_games.
   58.474961 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/fleming_fl
   58.802444 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
   71.780447 172.16.113.105 -> 207.25.71.141 HTTP GET /inside_game/magazine/basketba
   72.890674 172.16.113.105 -> 207.25.71.141 HTTP GET /inside_game/magazine/basketba
  269.096336 172.16.113.105 -> 207.25.71.141 HTTP GET /images/fronticons/icon_games.
  269.553201 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/fleming_fl
  270.015678 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
  783.033689 172.16.113.105 -> 166.77.13.117 HTTP GET /images/fun_and_games.gif HTTP
 6662.842704 172.16.113.105 -> 165.254.124.9 HTTP GET /icons/game.gif HTTP/1.0
 6813.056031 172.16.113.105 -> 165.254.124.9 HTTP GET /icons/game.gif HTTP/1.0
16618.171004 172.16.113.105 -> 207.25.71.141 HTTP GET /images/fronticons/icon_games.
16618.984047 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/fleming_fl
16619.405953 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
```

図 9.14: URI に “game” を含む 172.16.113.105 からの HTTP GET リクエスト

```
   58.802444 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
   71.780447 172.16.113.105 -> 207.25.71.141 HTTP GET /inside_game/magazine/basketba
   72.890674 172.16.113.105 -> 207.25.71.141 HTTP GET /inside_game/magazine/basketba
  270.015678 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
16619.405953 172.16.113.105 -> 207.25.71.141 HTTP GET /images/inside_game/farber_nhl
```

図 9.15: R_p を適用した本実装による出力結果

れているため，これを利用して tcpdump は UDP の宛先ポート 53 番のパケットに対するメッセージだと判断している．しかし，本実装が利用したルール R_q は図 9.16 に示したとおり，パケットに含まれる条件として ICMP タイプと ICMP コードしか検査していない．そのため，本実装が ICMP Port Unreachable メッセージが送信される前の UDP パケットに着目し，目的とする ICMP メッセージを抽出していることがわかる．図 9.17 と図 9.18 が完全に一致しているため，本実装がセッション間の情報交換を利用して検知できることが示せた．

9.9 まとめ

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の検知を実現するために，セキュリティイベント検知機構の ROOK を実装した．C 言語によって約 34,000 行で実装されている．ROOK はネットワークトラフィックを収集・解析し，セキュリティイベントを検知する．複数のセッションルールとパラメータによってセッション間の相関関係を表し，解析されたトラフィックデータと比較している．セッション情報の管理やパラメータの管理では計算量を考慮した実装となっており，検知処理も一括した条件検査が可能なフレームワークを実装している．これによって処理負荷が増大しないようになっている．一方，各プロトコルの解析モジュールもモジュール間の依存関係を最小にするなど拡張性

```
<rule name="Test Rule: Rq">
  <param name="udp_server" type="addr" mode="stack" />

  <session proto="udp">
    <trigger>
      <sig udp.dst_port="53" />
      <set scope="src" name="udp_server" filter="ipv4.dst_addr" timeout="10" />
    </trigger>
  </session>

  <session proto="icmp4">
    <trigger>
      <sig icmp4.type="3" icmp4.code="3">
        <eval scope="dst" name="udp_server" filter="ipv4.src_addr" stat="eq" />
      </sig>
      <act type="dump" arg="icmp_port_unreach.pcap" />
    </trigger>
  </session>
</rule>
```

図 9.16: パラメータ機能検証用ルール R_q

```
22:37:51.964151 IP 172.16.112.50 > 153.10.8.174: ICMP 172.16.112.50 udp port 53 unre
00:28:17.164175 IP 192.168.1.20 > 192.168.1.10: ICMP 192.168.1.20 udp port 53 unreac
00:28:17.164261 IP 192.168.1.20 > 192.168.1.10: ICMP 192.168.1.20 udp port 53 unreac
00:28:18.596220 IP 192.168.1.20 > 172.16.112.20: ICMP 192.168.1.20 udp port 53 unrea
00:28:19.589639 IP 192.168.1.20 > 172.16.112.20: ICMP 192.168.1.20 udp port 53 unrea
```

図 9.17: tcpdump により抽出した UDP の宛先ポート 53 に対する ICMP Port Unreachable メッセージ

```
22:37:51.964151 IP 172.16.112.50 > 153.10.8.174: ICMP 172.16.112.50 udp port 53 unre
00:28:17.164175 IP 192.168.1.20 > 192.168.1.10: ICMP 192.168.1.20 udp port 53 unreac
00:28:17.164261 IP 192.168.1.20 > 192.168.1.10: ICMP 192.168.1.20 udp port 53 unreac
00:28:18.596220 IP 192.168.1.20 > 172.16.112.20: ICMP 192.168.1.20 udp port 53 unrea
00:28:19.589639 IP 192.168.1.20 > 172.16.112.20: ICMP 192.168.1.20 udp port 53 unrea
```

図 9.18: R_p を適用した本実装による出力結果

も考慮しており，今後様々な機能を追加できるようになっている．この実装によって，実ネットワークにおける複数セッションを利用したセキュリティイベントが検知できるようになった．

第10章　評価

本論文で提案・実装した複数セッションの相関関係を利用するセキュリティイベント検知機構の有効性を示すために，検知精度，処理性能の評価実験を実施し，関連研究との比較を定性評価としてまとめた．第5章で述べた要求事項のうち，検知精度と規模性は評価実験によって実ネットワークでの運用に必要な機能を満たしていることを示す．特に既存の明確性が高いセキュリティイベント検知機構であるIDSと比較して，検知精度が十分に改善されていることを示す．検知結果の明確性と検知結果の柔軟性については第8.3.4節で示したが，関連研究と比較した結果を改めて定性評価として示す．

検知精度と処理性能の評価に用いるトラフィックのデータセットは5種類あり ($N_{1,2,3,4,5}$)，検知精度の評価では $N_{1,2,3}$ を，処理性能の評価では $N_{4,5}$ を使用した．実運用ネットワークから取得したデータセット ($N_{4,5}$) を利用した．$N_{1,4}$ は著者が所属する研究会のネットワークから取得したトラフィックである．N_2 は200名程度が参観するカンファレンスのネットワークから取得したトラフィックである．N_3 はあるバックボーントラフィックを観測した．N_4 は1999 DARPA Intrusion Detection Evaluation[87] のFifth Week Friday，outside tcpdumpデータセット (N_5) を利用した．付録Bにおいて，各データセットのトラフィック傾向を示している．

10.1　検知精度の評価

検知精度評価ではFalse Positive，False Negativeの発生を調べ，本手法を用いることで高い精度のセキュリティイベント検知が実現できることを示す．第2.3.3節で示したとお

表10.1: 評価用データセット概要

	収集開始日時	期間	パケット数	データ長	説明
N_1	2007-12-02 21:12	77時間	164.16M	83.56GB	研究ネットワーク
N_2	2007-09-12 10:00	12時間	163.86M	77.15GB	カンファレンスのネットワーク
N_3	2008-01-08 01:40	10時間	1516.31M	700.21GB	バックボーンネットワーク
N_4	2007-12-16 17:23	1時間	1.34M	953.17MB	N_1 と同じネットワークより取得
N_5	1999-04-09 08:00	22時間	2.65M	969.94MB	[87] (Fifth Week Friday outside tcpdump data)

り，セキュリティイベント検知における誤検知は False Positive と False Negative の 2 種類であり，この 2 つが無い，もしくは十分に少なければ検知精度が高いと言える．そのため，検知精度の評価では検知ルールの動作確認として False Negative が発生するかを確認し，主要な評価として False Positive が発生するかを確認する．

False Negative が発生しないことを示すためには，検知対象となるセキュリティイベントを含むトラフィックを意図的に入力し，ルール通りに検知できるかを検証すればよい．入力するトラフィックは同じセキュリティイベントに対して複数種類用意する．これによって，正常にトラフィックが入力されている状態であれば，False Negative が発生しない事を示せる．

False Positive が発生しないことを示すために，様々な環境のトラフィックを入力して検知処理を実行させた結果を確認する．トラフィックデータは環境によって異なる傾向を示す．そのため，どのような傾向のトラフィックデータでも誤検知をおこさないかを確認するためには，複数の環境のトラフィックデータによって検証する必要がある．

10.1.1 評価条件

本評価に用いるルールは $R_{1,2,3,4,5}$ の 5 種類である．ルールの詳細は付録 C で示している．

R_1 (ボットの検知): 疑わしいドメイン名の問い合わせと IRC，HTTP による活動 R_1 はボットやスパイウェアに利用されるドメイン名の問い合わせを発見し，その後に IRC サーバへの接続や Windows で実行されるファイルのダウンロードが発生しないかを確認するルールである．多くのボットネットでは DNS を利用して C&C サーバの IP アドレスを探索する構造になっている．そのため，ボットが頻繁に利用するドメイン名のリストを作成し，そのドメイン名の問い合わせに着目することでボットの部分的な活動を発見できる．しかし，他のソフトウェアでもこのような疑わしいドメイン名の問い合わせを実行する場合がある．例としては，SMTP[18] において HELO コマンドと共に送信されるドメイン名をメールサーバが問い合わせる状況が挙げられる．そのため，R_1 では疑わしいドメインの問い合わせに対する応答に着目している．該当するドメイン名の応答があった場合，回答レコードに含まれる A レコードを抽出しパラメータ `cc_server` に書き込む．その後，問い合わせ元のホストから A レコードに含まれていた IP アドレスに対して，IRC による接続，もしくは HTTP による Windows 実行ファイルのダウンロードが発生した場合に，問い合わせ元のホストがボットとして活動していると判断する．疑わしいドメイン名の一覧は BLEEDING EDGE THREATS で公開されている blackhole.conf[89] の 2007 年 12 月 9 日の版を参照した．

R_2 (ボットの検知): 実行ファイルのダウンロードと IRC への接続 R_2 は実行ファイルのダウンロードが発生した直後に，IRC への接続が発生したかを確認する．ボットは自身を自動的に更新する機能を持ち，主に HTTP を利用して Windows の実行ファイルを取得している．しかし，Windows の実行ファイルのダウンロードは一般的に発生する可能性があるトラフィックであるため，これを検知条件としてしまうと誤検知が多発してしま

う．そのため，R_2 では実行ファイルのダウンロード直後に，ダウンロードしたホストから IRC の接続が発生するかを確認している．更新用の実行ファイルのダウンロード後は，当該ファイルを起動させ，IRC によって C&C サーバへ接続する場合が多い．R_2 では実行ファイルのダウンロード後，そのホストから 30 秒以内に IRC 接続があった場合にボットの活動と判断する．

R_3 (ボットの検知): IRC によるコマンドの送信とスキャンの実行　R_3 は IRC によって C&C サーバから送信されたスキャンの命令を発見し，命令を受信したホストがスキャンを開始したか確認する．多くのボットは IRC によって C&C サーバから命令を受信するため，IRC のメッセージに含まれるスキャン開始の命令を発見することで，ボットの活動をとらえる．ただし，C&C からの命令は多様であるため，命令の発見は抽象的な条件にならざるを得ない．そのため R_3 では命令があった直後に，命令を受信したと見られるホストからスキャンに相当する通信が発生しないかを監視する．R_3 では TCP の宛先ポート番号が 135，139，445，2967，2968 番である通信をスキャンに相当する通信としている．

R_4 (ボットの検知): IRC による URL の送信と HTTP による実行ファイルのダウンロード　R_4 は IRC によって C&C サーバから送信された更新用プログラムの URL を発見し，その URL から Windows の実行ファイルがダウンロードされたかを確認する．ボットの更新用プログラムの設置場所は頻繁に変更されるため，その都度 URL を示す場合がある．更新用の命令で出現する URL には変わった特徴が見られないため，検知条件として使うのは難しい．そのため R_4 では IRC サーバから送られた URL をパラメータに記憶し，HTTP で送信されるリクエストと一致するか確認する．さらに，ダウンロードされるのが Windows の実行ファイルであるかを確認し，これらの条件に一致すればボットの活動であると判断する．

R_5 (P2P ファイル交換ソフトウェアの検知): Winnyp の活動　winnyp の検知は第 7 章でも行ったが，より大規模な評価をするために厳密なルールを R_5 で設定した．まず，TCP セッションを確立してから 1400 バイト以上の 5 パケットが送受信される．この 5 パケットは全て同じ TCP のセグメントサイズとなっている．さらに，最初の 5 パケットに送信と受信が含まれていなければならない．この 3 つの条件を満たすことで，winnyp に類似したセッションであると判断する．そして，このセッションが 5 回発生した場合に winnyp の活動であると判断する．最後の類似セッションが発生してから 300 秒以上経過した場合は，パラメータが時間切れで消滅するため，5 回発生しても winnyp の活動とは判定されない．

また，本評価ではネットワークトポロジと監視地点が結果に影響するため，$N_{1,2,3}$ のデータセットの収集環境と収集地点を付録 B の図 B.3，図 B.4，図 B.5 にまとめている．

10.1.2 検知ルールの動作検証

評価に用いるルールによって，検知対象としているセキュリティイベントを実際に検知できるかを確認した．それぞれ検知対象のセキュリティイベントが含まれるトラフィックデータを収集し，ルールを適用した本実装に入力する．これによって検知対象のセキュリティイベントが発見できれば，False Negative が発生しないことを確認できる．

ボット用ルールの検証 ボット用の検知ルール $R_{1,2,3,4}$ の確認は，ハニーポットから採取された検体を利用して実施した．2007 年 4 月から 2007 年 11 月までの間，4,000 程度の IP アドレス上で Nepenthes[90] を動作させて収集した検体から，54 個を抽出し利用した．これらを実験環境において実行し，発生したトラフィックデータを保存した．このトラフィックデータを本実装に入力することで，ルールを検証した．ボットの活動を検知できたルールの種類と，検体を AVG Free Edition 7.5[91] によって検査した結果を表 10.2 にまとめる．37 個の検体によるトラフィックデータからは $R_{1,2,3,4}$ のいずれかのルールでボットを検知できたが，27 個の検体によるトラフィックデータからは検知できなかった．これは 27 個の検体が DNS の問い合わせの失敗や，問い合わせ結果の IP アドレスに到達性が無いなどの理由で C&C サーバに接続できなかったためである．これは 27 個の検体のトラフィックデータを全て Wireshark[88] に入力し，目視で確認した．一方，ルールに合致した 37 個の検体についても意図した動作によって検知しているかを確認した．これにより，C&C サーバに接続するボットは $R_{1,2,3,4}$ のルールにおいて False Negative が発生しないことを確認した．

P2P ファイル交換ソフトウェア検知ルールの検証 P2P ファイル交換ソフトウェア用の検知ルール R_5 の確認は，実際に P2P ファイル交換ソフトウェアの winnyp を起動し観測されたトラフィックデータを利用した．3 種類のトラフィックデータを収集し，R_5 を適用したところ，全てのトラフィックデータを winnyp として判定した．これにより，R_5 で False Negative が発生しないことを確認した．

10.1.3 評価結果

トラフィックデータ $N_{1,2,3}$ を用いて，False Positive の発生を確認した．検知用ルールは $R_{1,2,3,4,5}$ を全て適用し，それぞれのトラフィックデータを入力した．結果として出力された内容を表 10.3 にまとめた．

評価の結果，各ルールによってボットや Winnyp の活動は検知されなかった．表 10.3 では各ルールにおいて R'_n が活動の部分的な痕跡，R''_n 以降が各ソフトウェアの実質的な活動を示している．$R_{1,2,3,4}$ では，いずれかのデータセットにおいて部分的な活動の痕跡が発見されている．しかし，R'_n と関連した R''_n 以降の条件は発見されなかった．これにより，データセット $N_{1,2,3}$ の環境では False Positive が発生しないことが確認された．R_5 は単一セッションにおけるルールを厳密に記述したため，N_1 で Winnyp に類似するトラフィックとして R'_5 が 1 件検知されたが，1 セッションだけであったために R''_5 は発見されなかった．

表 10.2: ボット検知用ルールによって検知できたボットの種類一覧

検知したルール	AVG Free Edition による検体の検査結果
R_1	Trojan horse BackDoor.Generic3.UW
R_1,R_3,R_1	Trojan horse BackDoor.Generic4.YYN
R_1,R_3	Trojan horse BackDoor.Generic6.RV
R_1,R_3	Trojan horse BackDoor.Generic7.WFE
R_1,R_3	Trojan horse BackDoor.Generic8.BOA
R_1,R_3	Trojan horse BackDoor.Generic8.NLT
R_1	Trojan horse BackDoor.Robobot.CW
R_1,R_3	Trojan horse BackDoor.Robobot.FM
R_1	Trojan horse Generic4.RPM
R_1,R_3	Trojan horse Generic5.BSS
R_1	Trojan horse IRC/BackDoor.SdBot.YRR
R_1	Trojan horse IRC/BackDoor.SdBot2.LCZ
R_1,R_3	Trojan horse IRC/BackDoor.SdBot2.MVZ
R_1,R_3	Trojan horse IRC/BackDoor.SdBot2.MVZ
R_1,R_3	Trojan horse IRC/BackDoor.SdBot2.WEX
R_1,R_3	Trojan horse IRC/BackDoor.SdBot3.DEI
R_1	Trojan horse IRC/BackDoor.SdBot3.OCW
R_1,R_3	Trojan horse IRC/BackDoor.SdBot3.SIS
R_1,R_3	Trojan horse IRC/BackDoor.Sdbot2.NEV
R_1,R_3	Trojan horse SHeur.DBB
R_1,R_3	Trojan horse SHeur.ETE
R_1,R_3	Trojan horse SHeur.ETE
R_1,R_3	Trojan horse SHeur.KRN
R_1,R_3	Trojan horse SHeur.KRN
R_1,R_3	Trojan horse Sheur.JSD
R_1	Virus found Win32/Parite
R_1	Virus found Win32/Sality
R_1,R_3	Virus found Win32/Sality
R_1,R_2,R_3,R_4	Virus found Win32/Virut
R_1,R_3	Virus identified Packed.Pestil
R_1,R_2,R_3,R_4	Virus identified Win32/Virut.A
R_1,R_2,R_4	Virus identified Win32/Virut.A
R_1	Virus identified Worm/Agobot.CTJ
R_1,R_3,R_1,R_3	Virus identified Worm/Agobot.FVY
R_1,R_2	Virus identified Worm/Bobax.X
R_1,R_4	Virus identified Worm/Bobax.X
R_1	Virus identified Worm/Padobot.Z

表 10.3: 精度評価結果

ルール		N_1	N_2	N_3
R_1	R_1': 疑わしいドメインの問い合わせに対する応答	622	422	1,779
	R_1'': R_1' に含まれる A レコードのアドレスから HTTP による “.exe” を含む URL のリクエスト	0	0	0
	R_1''': R_1' に含まれる A レコードのアドレスから HTTP による Windows 実行形式ファイルのダウンロード	0	0	0
	R_1'''': R_1' に含まれる A レコードのアドレスに対する IRC の接続	0	0	0
R_2	R_2': Windows 実行形式ファイルのダウンロード	219	172	805
	R_2'': R_2' の発生から 30 秒以内の IRC 接続	0	0	0
R_3	R_3': スキャンの開始命令に類似した IRC メッセージ	4	0	3,578
	R_3'': R_3' 発生後のスキャンに相当する通信が 10 回発生	0	0	0
R_4	R_4': IRC のメッセージに URL が出現	44	434	2138
	R_4'': 120 秒以内に R_4' で出現した URL への HTTP リクエスト	0	0	0
	R_4''': R_4'' の HTTP リクエストによる Windows 実行形式ファイルのダウンロード	0	0	0
R_5	R_5': Winnyp に類似したセッションの発生	1	0	-
	R_5': R_5' が同一ホストで 5 回以上発生	0	0	-

既存の IDS で検知を試みた場合，条件として記述できるのは 1 つめの条件である R_n' となり，表 10.3 の結果では誤検知が相当数になってしまう事が分かる．そのため，本評価における R_1'，R_2'，R_3'，R_4'，R_5' の検知数が，既存の IDS における False Positive の数と同義であるとみなせる．R_1'，R_2' は全てのデータセットで 100 件以上発生している．トラフィックデータの収集期間が最も長い N_1 でも 77 時間であるため，例えば R_2' の結果が False Positive であるかを確認するためには，1 日辺り 70 回あまりの確認作業が必要となる．1 回にかかる時間が 10 分と仮定しても 11 時間超の時間が必要となるため，運用の効率は悪い．R_5' は N_1 において 1 件検知されているのみだが，それでも同様のルールが 100 個になれば，確認回数は日に 30〜40 回となってしまう．本手法では各結果において False Positive を発生させておらず，確認に必要な負担を大幅に減らせたと言える．

ただし，データセット N_3 においては，R_5 は R_5' が 72 件，R_5'' が 7 件発見されたが記述していない．これは，検知したホストが筆者の管理外のホストであるため，実際に Winnyp を利用しているかを確認することができなかった．そのため，結果として記載していない．

図 10.1: 性能評価用構成

表 10.4: 性能評価に使用したホストの構成

	トラフィック送信ホスト	実験ホスト
CPU	Intel(R) Xeon(TM) CPU 3.06GHz	AMD Opteron 2.8GHz
メモリ	4GB	4GB
HDD	SCSI 336G (ext3, RAID 5)	SATA 1.5TB（ext3, RAID 0, 750G x 2)
OS	Debian GNU/Linux (kernel 2.6.12)	Debian GNU/Linux (kernel 2.6.18)
NIC	Broadcom Tigon3	Broadcom NetXtreme II BCM5706
Link	1000Base-T	

10.2 性能評価

本実装の規模性を評価するため，トラフィックの処理性能を調査した．トラフィック性能を測る上で，無作為に作成したパケットデータを使用するとパケットサイズやプロトコルの分布が通常のネットワークと大きく異なってしまう可能性が高い．また，特に TCP などでは正しくセッションが構成されない通信が多数発生してしまう．そのため，実ネットワークと比較して極端に良い結果や悪い結果になってしまう可能性がある．これを回避するために，本評価では実ネットワークのトラフィックを利用した．ただし，性能評価のために送信速度を調整しながら負荷のかかり方を調査した．

具体的には，本実装の性能評価として再現した実ネットワークのトラフィックを入力し，パケットを損失することなく処理できるかを調査した．評価はトラフィック送信ホストと実験用ホストと分離し実験用ホストで本実装を動作させ，パケットの損失率を調査した．

```
echo 33554432 > /proc/sys/net/core/rmem_default
echo 33554432 > /proc/sys/net/core/rmem_max
echo 10000 > /proc/sys/net/core/netdev_max_backlog
```

図 10.2: 実験ホストのチューニング

10.2.1 性能評価のねらい

本節で述べる性能評価では，パラメータの保持数や参照数が検知処理に及ぼす影響を調査し，実際の運用環境において十分にトラフィックを処理できるかどうかを調査する．本論文で提案している手法は，従来のIDSに実装されているセッションやトラフィックの特徴を検知条件とする機能の他に，パラメータの管理機能が実装されている．パラメータはセッション間の相関関係を示すために必要となる機能だが，この機能によって従来のIDSより処理性能が低下する可能性がある．

本実装はパラメータを参照する検知条件と参照しない検知条件を記述できるが，パラメータを参照しない条件による検知処理の負荷は他のIDS実装と大きな違いはない．例えばSnortではトランスポート層のプロトコル（TCP，UDP，ICMP）とTCPの宛先ポート番号から検知に利用するルールセットを選択している．Bro[77]も各プロトコル毎にルールセットを分割する実装となっている．これは第9.5.1節で示した検知処理の擬似コード（図9.11）の3行目や5行目の処理に相当する．さらに，Snortではルールの中で`content`や`uricontent`という項目により文字列の出現パターンを条件として定義している．これらの条件はAho-Corasickアルゴリズムを改変した実装によって一括した検知処理を実行している．本実装では，このような処理は擬似コードの4行目の処理に相当している．

そのため，パラメータの読み込み必要とするルールが処理に与える影響を調べることで，本手法特有の負荷がどの程度であるかを示せる．評価において実験用ホストがトラフィックを受信する際，パラメータの読み込み，書き込みを大量に発生させることを目的としたルールを適用して本実装を動作させる．これによって擬似的に大量のパラメータを保持している状態を再現している．そのような状態で正常にパケットが受信できるかを確認した．

10.2.2 性能評価構成

図10.1は性能評価に用いたネットワーク構成，表10.4には各ホストの構成を示している．2ホスト間のリンクは1000Base-Tによって直接接続した．トラフィック送信ホストからはデータセット $N_{4,5}$ をそれぞれtcpreply[92]を使って送信した．トラフィックの送信はtcpreplayの`-p`オプションを使用して送信するppsを10,000から150,000まで，10,000ずつ変化させて実行した．実際に送信されたパケット数はtcpreplayの出力結果から判断し，グラフ作成時に利用した．実験ホスト側では本実装によって送信されてくるトラフィックを監視し，処理したパケット数を記憶する．送出したパケット数と処理したパケット数の差から，パケット損失率を判断した．

また，本実装を動作させる実験ホストはチューニングのためカーネルの設定を一部変更している．変更の際には[93]を参考にし，図10.2のように設定した．これによってネットワークインターフェースから入力されたデータを保持できるバッファを多くし，一定の処理の偏りに対応できるようにしている．

```
<rule name="Eval Rule a">
 <param name="sent_com_dns_query" type="bool" mode="allow" />

 <session proto="dns">
  <trigger>
   <sig dns.query="com" />
   <set scope="src" data="1" name="sent_com_dns_query" timeout="120" />
  </trigger>
 </session>

 <session proto="http">
  <trigger>
   <sig http.method="GET" >
    <eval scope="src" data="1" name="sent_com_dns_query" stat="eq" />
   </sig>
  </trigger>
 </session>
</rule>
```

図 10.3: 評価用ルール R_a

```
<rule name="Eval Rule b">
 <param name="init_size" type="int" mode="stack" />

 <session proto="tcp">
  <trigger>
   <sig tcp.seg_id="1" />
   <set scope="src" filter="tcp.seg_size" name="init_size" timeout="120" />
  </trigger>

  <trigger>
   <sig tcp.seg_id="1" >
    <eval scope="src" filter="tcp.seg_size" stat="eq" name="init_size" />
   </sig>
  </trigger>
 </session>
</rule>
```

図 10.4: 評価用ルール R_b

10.2.3 性能評価条件

図 10.3，図 10.4，図 10.5 はそれぞれ評価用ルール R_a，R_b，R_c を表している．これらは評価のため意図的にパラメータ操作を大量に発生させるルールである．そのため検知しようとしている対象はセキュリティイベントではなく，意味のない事象である．それぞれ 1 つのパラメータを操作するルールとなっており，書き込みと読み込みを両方発生させるようにしている．

R_a は DNS の問い合わせで，クエリの内容に”com” を含むパケットを発見した場合にパ

```
<rule name="Eval Rule c">
 <param name="html_count" type="int" mode="allow" />

 <session proto="http">
  <trigger>
   <sig http.uri="\.html$"/>
   <set scope="src" data="1" opt="inc" name="html_count" timeout="600" />
  </trigger>
 </session>

 <session proto="tcp">
  <trigger>
   <sig tcp.dst_port="80" tcp.seg_id="1">
    <eval scope="src" data="10" name="html_count" stat="gt" />
   </sig>
  </trigger>
 </session>
</rule>
```

図 10.5: 評価用ルール R_c

ラメータを書き込んでいる．パラメータが bool 型なので，1 が書き込まれるのみである．上書きモードは”allow” なので，何度でも上書きできる．ただし，このルールでは 1 しか書き込み命令が無いため，1 以外の値が書き込まれることはない．その後，HTTP セッションにおいて”GET” メソッドのリクエストが発生した場合に，パラメータを検査している．

R_b は TCP のセグメントデータを持つ最初のパケットを発見した場合に，書き込みと読み込みを両方実行する．書き込みではセグメントデータのサイズをパラメータに書き込む．パラメータは INT 型で，上書きモードは積み上げ型になっている．そのため書き込まれた後，時間切れになる 120 秒の間に異なる値が書き込まれた場合，書き込み前の値と書き込み後の値を両方保持する．セグメントデータのサイズの値は書き込まれると同時に，パラメータに保持してある他のセグメントデータのサイズと比較される．

R_c は HTTP のリクエストで”.html” を持つ URL が含まれていた場合，パラメータに書き込みをする．書き込むパラメータは INT 型で上書き可能となっている．さらに，書き込む際は `opt="inc"` と指定してあり，これは既に書き込まれているパラメータの値に対する加算を意味する．例えばすでに 4 という値が書き込まれていた場合，この書き込み命令によってパラメータは 5 という値になる．一方，読み込みは宛先ポートが 80 番で最初のセグメントデータの場合に実行される．

これらのルールを使って実験するにあたり，各データセットのパケットが正常に入力された場合に，書き込み数と読み込み数がどの程度発生するか事前に調査した．調査の結果は表 10.5 に示している．正常に入力された場合，データセット N_4 では 40,000 個以上，N_5 では 170,000 個以上のパラメータを保持することとなる．また，読み込み回数もデータセット N_4 で 35,000 回以上，N_5 で 350,000 回以上発生している．例えばクラス B のネットワークを監視する際に，全ての内部ホストに 2 個から 3 個の状態を保持したいと仮定すると必要となるパラメータは 120,000 個から 180,000 個程度である．170,000 個のパラメー

表 10.5: パラメータの保持数と参照回数

		N_4	N_5
R_a のパラメータ	書き込み数	14304	1812
	読み込み数	6885	109654
R_b のパラメータ	書き込み数	26095	168651
	読み込み数	26095	168651
R_c のパラメータ	書き込み数	376	1319
	読み込み数	3660	108485

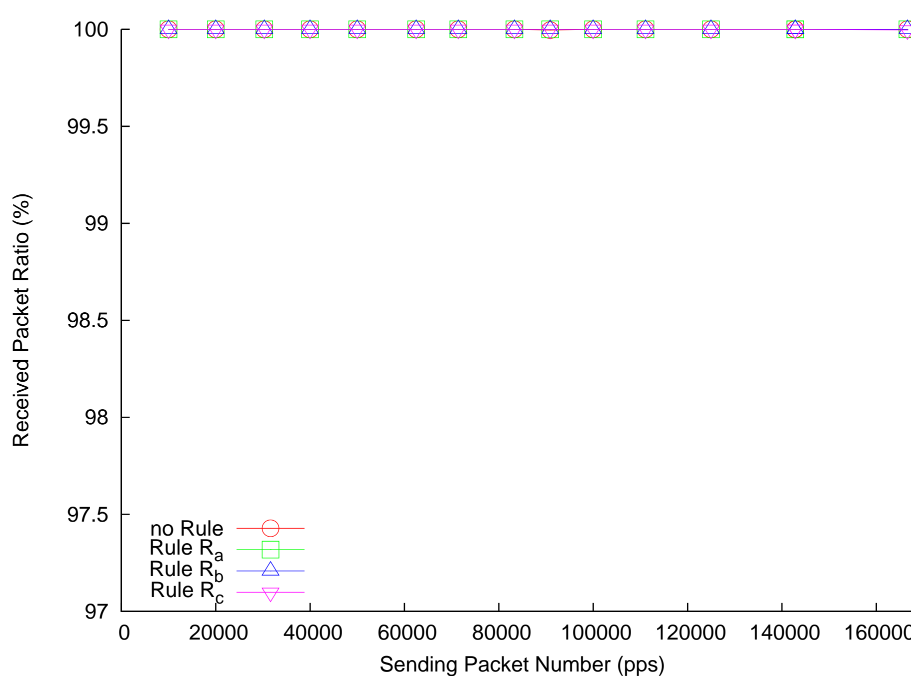

図 10.6: データセット (N_4) を用いた性能評価実験結果

タを保持できれば，全てのホストに 2〜3 個の状態を保持させるという要求を満たせる．

10.2.4 評価結果

図 10.6 は N_4，図 10.7 は N_5 を用いた実験結果をそれぞれグラフで示している．グラフはそれぞれ X 軸がパケットの送信速度 (pps)，Y 軸がパケットの受信率である．パケットの送信速度は tcpreplay によって示された pps を元に描画している．Y 軸のパケット受信率は送信したパケット数と本実装が受信したパケット数を比較し，全て受信できていれば 100%，パケットの損失が発生していれば値が下がっている．Y 軸は下限が 97%，上限が 100% となっている．グラフはそれぞれ，ルール無し (No Rule)，R_a を適用した場合の実

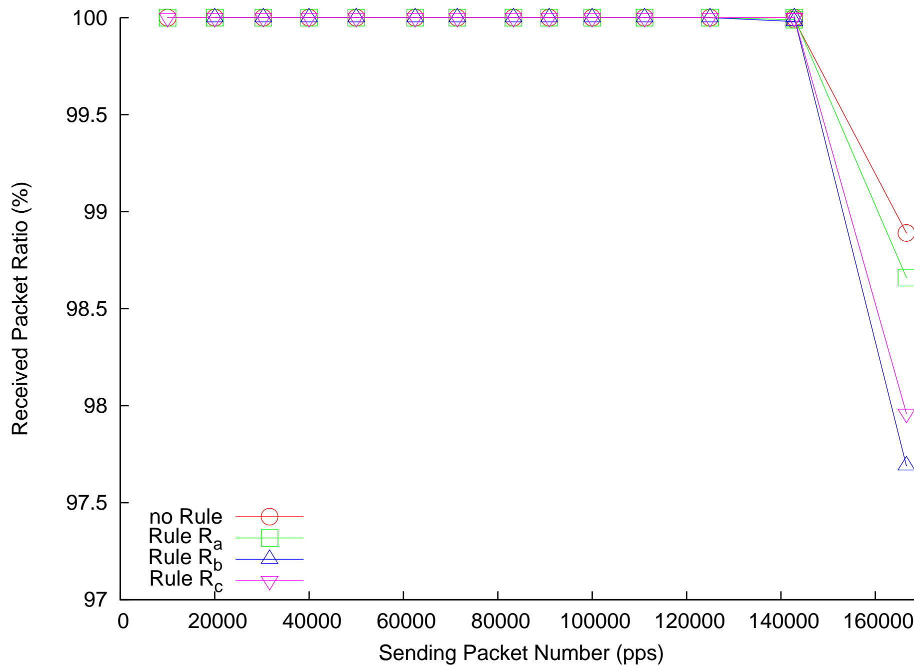

図 10.7: データセット N_5 を用いた性能評価実験結果

行結果 (Rule R_a)，R_b を適用した場合の実行結果 (Rule R_b)，R_c を適用した場合の実行結果 (Rule R_c) で別のラインとなっている．

図 10.6 では，全てのグラフが 100%を維持している．このことから，データセット N_4 の環境では 160,000pps までは各 R_a，R_b，R_c のルールを適用してもほとんどパケット損失が発生しない事を示している．

一方，図 10.7 では 160,000pps 付近で 1%から 2.5%あまりのパケット損失が発生している．ただし，ここではルール無しのグラフも他のグラフと同様に傾いている．これはデータセット N_5 の環境において 160,000pps の入力があった場合，パケット損失の 1%ほどの原因はパラメータ機能とは無関係であることを示している．パラメータはルールに記述された読み込みと書き込みをするため，ルールを読み込ませない実行結果をとった No Rule のグラフはパラメータ操作が存在しない．そのため，このパケット損失の原因は Decoding コンポーネントによるものであると推測される．ただし，140,000pps まではほとんどパケット損失が発生していないため，140,000pps までの負荷には耐えられる事を示している．

10.3 関連研究との比較

第 5 章で述べた要求事項をもとに，本手法の有効性を第 4 章で挙げた関連研究と比較した．比較した一覧は表 10.6 にまとめている．比較項目は検知結果とネットワークインシデントとの対応が明らかであるかを示す明確性，False Positive，False Negative が十分に少なく検知できているかを示す精度，柔軟に様々な検知対象についてのセキュリティイ

表 10.6: 関連研究との比較一覧

	明確性	精度	柔軟性	規模性
Traffic Classification	-	-	-	O
Anomaly Detection	-	-	-	P
Correlation Analysis	O	O	P	-
Network Behavior Analysis	P	P	P	O
本手法	O	O	O	O

O: 優れている，-: 劣っている，P: 部分的に優れている

ベントを検出できるかを示す柔軟性，そして広帯域なネットワークでも機能するかどうかを示す規模性の 4 項目である．比較対象は Traffic Classification，Anomaly Detection，Correlation Analysis の 3 種類を取り上げた．

Traffic Classification は，本来トラフィックエンジニアリングを目的とした手法である．そのため，バックボーンなどの広帯域ネットワークでの利用が想定されており，処理負荷は低いものが多く，希望性に優れていると言える．しかし高い精度は求められているが，確実な検知は求められておらず，いくらかの誤検知は許容される傾向がある．同様に明確性についても確実に動作しているソフトウェアを検知する事を目的としていない場合が多い．よって多種法と比較して明確性，精度が高いとは言い難い．また，固定的なアルゴリズムを使うため，検知対象が増加した場合の対応も難しい．

Anomaly Detection はネットワークトラフィックの異常からセキュリティイベントを検出する．そのため，未知のセキュリティイベントでも検知できるというメリットがあるが，検知結果が何を示すかを知るためには調査が必要になってしまい，明確性は低い．また，異常検知であるために正常と異常を区別する閾値が必要となる．閾値はネットワーク環境や接続しているホストの状態に応じて変化してしまうため，精度を保つのは難しいと言える．柔軟性は Traffic Classification と同様にアルゴリズムを変更しなければならないため，柔軟性が高いとは言えない．規模性はアルゴリズムに依存するが，多くの手法は負荷が低いものになっている．

Correlation Analysis はルールを基準として検知するため，明確性に優れている．また，複数のセキュリティイベントを組み合わせることにより発生している事象を正確に捉えやすいため，精度も優れていると言える．柔軟性についてはルールの追加，変更により多くのセキュリティイベントに対応できる可能性を持つが，第 4.2 節で述べたように，セキュリティイベントの出現規則制御が逐次的なものに着目している実装が多い．また，セキュリティイベントに含まれる情報の相互利用も送信元，宛先 IP アドレスや送信元，宛先ポート番号などに限られており，部分的に柔軟性に欠ける．Correlation Analysis は検知された複数のセキュリティイベントから新しいセキュリティイベントを検知するという性質上，検知に必要となる情報をセキュリティイベントとして保持しておく必要がある．特に頻繁に発生する事象を保持しようとすると二次記憶装置を圧迫してしまう可能性があり，規模性の面からは優れているとは言えない．

Network Behavior Analysis は sFlow を用いており，参照できるトラフィックデータが限られている．そのため，各ホストの挙動を正確に捉えるのは難しく，精度や明確性に問題が生じてしまう．基本的な動作は単純であるため，検知対象を拡大するための柔軟性には部分的に優れる．検知処理のための sFlow を収集する際，サンプリングしたデータからも検知できるため規模性には優れていると言える．

本手法は Correlation Analysis と同様，ルールに基づいてセキュリティイベントを検知する．ルールもトラフィックの特徴について様々な内容が記述できるため，精度は高く明確性も高い．これは第 10.1 節の評価結果から明らかである．ルールの柔軟性は Correlation Analysis より高い．基本的に逐次的な出現規則をセキュリティイベントの検知条件として扱う Correlation Analysis と比べて，第 8.3.4 節で示したように様々な事象の出現規則を表現できる．また第 9.3.6 節において，様々なトラフィックデータを複数セッション間で相互利用できることを示した．そのため，柔軟性は高いと言える．規模性は第 10.2 節で示したとおり，広帯域ネットワークで利用できることを示している．

以上の結果から，これまで検知が困難だったセキュリティイベントに対して，関連手法と比較しても本研究が優位であることと言える．

10.4 まとめ

本論文で提案・実装した複数セッションの相関関係を利用するセキュリティイベント検知機構の有効性を示すために，検知精度，処理性能の評価実験を実施し，関連研究との比較を定性評価としてまとめた．

検知精度の評価結果として，従来の IDS で検知を試みた場合と比較した場合，False Positive の発生率を大幅に抑えることができた．また，動作に不備があったボットは検知できなかったが，攻撃者の意図通りに動作したボットは検知できたため，False Negative も起こりにくい事を示した．これによって本手法は明確性の高いルールでも高い精度で検知する事を示した．

性能評価では，状態を記憶するためのパラメータを大量に持たせたとしても，140,000pps までは正常に動作することを示した．パラメータを 170,000 個保持した状態で，140,000pps まで正常にパケットを受信できている．パケットの平均長を 100 バイトとしても 112Mbps の処理が可能であるため，一般的な企業や家庭ネットワークにおいては十分に利用可能であると考えられる．

定性評価として，特に明確な検知が可能であることと柔軟な検知が可能であることを述べた．複数セッションを検知対象とできるため，ソフトウェアの振る舞いそのものを検知条件とでき，セキュリティイベントの原因となる脅威を特定しやすい．さらに複雑なセッションの出現規則も制御できるため，多様なセキュリティイベントに対応できる．

以上の結果から，本手法を用いることでボットや P2P ファイル交換ソフトウェアによるセキュリティイベントを容易に検知できるようになったと言える．

第11章　結論

11.1　本論文のまとめ

本論文では複数セッションの相関関係に着目したネットワーク上のセキュリティイベント検知手法を提案し，設計・実装した．これによって従来は検知が難しかったボットやP2P ソフトの高精度な検知を実現した．

既存のセキュリティイベント検知機構で発見が難しいセキュリティイベントは2つある．1つ目は暗号化された通信を使用するソフトウェアのセキュリティイベントである．既存の代表的なセキュリティイベント検知機構であるIDSの多くはトラフィックのパターンを検知条件にしているため，暗号化された通信には対応できない．特に自身の隠蔽を目的としたソフトウェアでは，コンテンツだけではなくヘッダやプロトコルそのものを暗号化しており，検知が非常に困難となっている．2つ目は他のソフトウェアに類似した通信を利用するソフトウェアのセキュリティイベントである．これはマルウェアに多く見られる手法で，一般的なWebブラウザやIRCソフト，FTPクライアントと類似した通信を使い，悪意ある活動を行う．既存のIDSで他のソフトウェアと類似した通信からセキュリティイベントの検知を試みると，悪意のないソフトウェアとの判断がつかず，誤検知を多発してしまう．

本論文ではこの問題を解決するために，複数セッションの相関関係に着目したセキュリティイベント検知手法を提案する．セッションとはホスト間の通信をプロトコル毎に分類したものを指す．本論文では複数セッションの相関関係を**セッションの出現規則**と**セッション内に含まれる情報の相互利用**の2つと定義している．セッションの出現規則とは，出現するセッションの種類の順番や，セッション出現の繰り返しを指す．セッション内に含まれる情報の相互利用とは，あるセッションを発見するために，異なるセッションの含まれるヘッダやペイロード情報の参照を指す．提案手法では，決定的な検知ができるトラフィックのパターンマッチと，複数セッションの相関関係の両者を組み合わせてセキュリティイベントを検知する．既存手法は1つのセッションに含まれる情報のみを検知の手がかりとしたが，複数セッションを利用することによってソフトウェアの振る舞いをより正確に捉えることができる．提案手法を実際の運用ネットワークで利用するために，複数セッションの相関関係を利用できるセキュリティイベント検知機構を設計，実装した．

これまで検知が困難だったセキュリティイベントに対して，本実装が有効であることを示すために検知精度と処理性能の評価を実施した．検知精度の評価では，ボットやP2Pファイル交換ソフトウェアの検知におけるFlase PositiveとFalse Negativeの発生率を調査した．False Negativeの確認ではハニーポットで収集したボットやP2Pファイル交換ソフトウェアを実際に動作させ，False Positiveの確認では実際の運用ネットワークを利用

した．その結果，既存の IDS と比較して誤検知を大幅に低減させることに成功した．また，性能評価では相関関係を示すための情報を大量に保持した状態で，正常にパケットを受信できるかを確認した．送信するパケットの送信速度を上げたところ，170,000 個以上の情報を保持していても 140,000pps 以上の性能で処理できた．平均パケット長が 500 バイトのネットワークであれば，500Mbps 以上で処理が可能であり一般的な企業や組織のネットワークでも運用できることを示している．さらに，他の関連研究と比較してネットワークインシデントとの関連が明確であること，柔軟な検知条件を指定できることを述べた．以上の評価により，提案手法を用いた本実装がボットや P2P ファイル交換ソフトウェアの検知に有用であることを示した．

本論文で提案，実現したセキュリティイベント検知手法を用いることによって，ネットワークインシデントの検出に必要となる負担を抑制した．ネットワークインシデント発生後に被害を極小化するための事後対策は，ネットワークインシデントの早急な検出によって効果を発揮するが，従来の手法では早急に検出するために多大なコストが必要である．そのため，対策に費用をかけられる大規模な組織でなければ検出と事後対策の実現は困難であった．しかし，本論文の提案手法の実現によりネットワークインシデント検出にかかるコストを少なくできる．これはネットワーク管理者がセキュリティイベントを調査する負担が少なくなるだけではなく，知識・経験の必要性も従来より低くなる．これによって中規模，小規模な組織でも検出と事後対策を運用が可能になる．さらに，高い精度を利用して自動的な事後対策を可能とすれば，一般家庭などにおいても多層的な対策を実現できる．このように，本研究はネットワークセキュリティ対策において意義がある．

11.2 今後の課題と展望

11.2.1 ルール作成時のデバッグ

本手法で利用する検知ルールは他のセキュリティイベント検知機構と比較して柔軟性が高くなっている反面，複雑さが増している．そのため本実装を利用するネットワーク管理者の意図通りにセキュリティイベントを検知するためには，従来手法以上にルールが適切に記述できているかどうかを確認する作業が必要となる．現在は開発用に適宜動作の確認コードを入れているが，これは一般のネットワーク管理者には難しい作業となる．

そのため，記述したルールが適切に動作しているかを対話的に確認する機能が必要となる．コンセプトとしてはプログラムの動作確認をするデバッガに相当する．パケットデータの読み込みタイミングの制御や，ある読み込み時点でのパラメータの値の確認，あるパケットデータに対する検知条件の判定結果などを逐次的に確認できる機能が求められる．

11.2.2 ルール記述方法の問題

本実装ではセキュリティイベント検知用のルールを非常に柔軟に記述できる設計・実装となっている．そのため，同様のセキュリティイベントを発見する場合でも複数のルール

記述方法が存在する．セキュリティイベントの複数セッションの関係を適切に記述できていれば異なる記述方法でも正確に検知できるが，記述方法によって処理性能が低下してしまう可能性がある．

これを解決するために，記述されたルールを最適化する，あるいは適切な記述方法を示す必要がある．性能低下の原因としては過剰なパラメータの利用や，1 ルール内に膨大な数のセッションルールを記述するなどが考えられる．実際には過剰なパラメータや膨大な数のセッションルールを使わなくても検知できる場合に，パラメータやセッションルールの使い方を最適化する方法が考えられる．あるいは，ある挙動を示すために余計な記述について記述者に助言を与える機能が必要になる．

11.2.3 ルール作成の自動化

本論文で検知対象の 1 つとして挙げてるボットは，アップデートによる活動パターンの変化によっても検知が難しくなっている．ボットは自らの機能拡張が可能であり，攻撃者の指示により新しいボットのソフトウェアに更新する．これによって，ウィルス対策ソフトのファイルパターンマッチが困難になり，未知のボットに対応し切れていないという現状がある．

本論文での提案手法もルールを用いた検知であるため，この問題点は解決されていないが，[94] などでも機械的な解析が進んでいる．また，仮想的にインターネット環境を再現し，ボットの活動を解析する手法 [62] も研究されている．このような技術と連携し，亜種発生から短い時間で検知が可能になる機構が必要となる．

11.2.4 検知処理実装の高速化

今後，本実装を実運用ネットワークで利用するにあたり，ルール数の増加は避けられない問題である．第 11.2.3 節でも述べた通り，脅威の数は日々増加している．それにともなって検知すべきセキュリティイベントも増加しているため，膨大なルールに耐えられる実装でなければならない．

本実装では条件検査を一括してできる多重フィルタ機（第 9.3.7 節）や，セッションルールの分離性を元にした検知処理の並列化が可能となっている．しかしこれらは実装の負担が大きく，まだ実装が完了していない．多重フィルタは一部のモジュールのみで実装されており，検知は直列処理となっている．最終的には読み込んだルールに存在する条件の種類や数をもとに，最適な並列化や高速化を実行できる必要があり，今後も実装を続ける必要がある．

11.2.5 IPv4/IPv6 混在環境や NAT 利用環境への対応

本実装は IPv4/IPv6 混在環境や NAT 利用環境では意図した動作をしない可能性がある．本実装はネットワーク上でホストを識別するために，主に IPv4 アドレス，IPv6 アドレ

スを使用している．IPv4 アドレス，IPv6 アドレスは共にホストのネットワークインターフェースを特定できるが，1 つのネットワークインターフェースには複数の IP アドレスがつく可能性がある．特に IPv4，IPv6 の混在環境では IPv4 アドレスと IPv6 アドレスの両方が割り当てられるため，本実装は IPv4 による通信と IPv6 による通信を別ホストによる通信と認識してしまう．また，NAT を利用しているネットワークを上流で監視した場合，複数ホストからの通信が 1 つの IP アドレスで通信しているように見えてしまう．そのため複数のセッションを関連づける際に，正確に 1 つのホストを追跡するのが困難になってしまう．

これは今後の IPv4 の枯渇や IPv6 の普及により問題になると考えられる．2007 年には JPNIC より IPv4 アドレス在庫枯渇問題に関する検討報告書 [95] が公開されており，IPv4 から IPv6 への移行期において IPv4 と IPv6 の混在環境普及と NAT 環境の普及が予想されている．このような背景から複数の識別子を持つホストや，1 つの識別子で通信する複数ホストを適切に監視するための手法が必要となる．

11.2.6 未知のボットに対する応用

本実装を応用することで，未知の脅威を高い精度で発見できる可能性がある．これはセッション間の相関関係を抽象化することによって，亜種に対応することで実現できると考えられる．

本論文では既に詳細な動作が判明しているボットを正確に検知することを目的としていた．これは第 6.3 節で述べたとおり，セキュリティイベントを発生させる脅威を決定的に捉えることによって，ネットワーク管理者の運用負担を減らすことを目的としている．ただし，ボットは従来のコンピュータウィルスと比較しても亜種が発生する頻度が高く，多い場合は 1 週間に数回更新される場合もある．そのため，ルールの作成が間に合わずボットの活動を発見できない可能性がある．

しかし，ボットは亜種が発生しても基本的な動作は変化しにくい特徴があり，これを利用することで未知のボットを発見できる可能性がある．ボットは実行ファイルのパターンマッチを利用するウィルス対策ソフトの検知を回避するために実行ファイルの細部を更新したり，接続する C&C サーバを変更するために実行ファイル内に持つ値を更新する．本論文で示した検知用ルールでも，C&C サーバの変更や命令の書式変更があった場合には検知できなくなる．これに対応するため，ボットの動作を抽象化したルールを作成する．ボットは詳細な動作が変化しても，C&C サーバに接続して命令を受信し，何種類かの悪意ある行動を開始するという動作の流れは変化しない場合が多い．1 セッションに着目していた従来手法でこのような動作の検知を試みると，多くの誤検知を多発してしまう可能性がある．しかし，複数セッションを利用することでホストの動作を包括的に捉えられるため，誤検知率の低下が期待できる．

謝辞

本論文の執筆にあたり，本当に多くの方々にお世話になりました．この場をお借りして，お礼を述べさせていただきたいと思います．

まず，本論文の作成にあたり御指導いただきました慶應義塾大学環境情報学部教授村井純博士に感謝します．直接お時間を頂ける機会は少なかった物の，修論や今後の研究の進め方についても貴重な意見を頂きました．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．特に本年度は研究について多くのご意見を頂き，研究の戦略的な進め方などについてご指導いただきました．一方で，研究について助言を頂いただけでなく，本論文の評価に必要となる実験環境の提供をしていただけるなど，研究の後押しをしていただきました．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．氏には厳しいながらも，研究や実装の本質を捉えた非常に鋭い意見をたくさん頂きました．そのおかげで，自分の研究について大きい局面から細部までの様々な視点から客観的に見て研究を改善して行けたと思っています．また，大変お忙しい中にも関わらず修論中間発表に参加して頂きました．ありがとうございました．

IIJ 技術研究所主幹研究員 長健二郎博士，慶應義塾大学環境情報学部 学部専任講師 重近範行博士，東京大学 関谷勇司博士, 石原知洋氏，堀場勝広氏にも実験の準備において様々なご協力を頂きました．この場を借りてお礼申し上げます．

また，かねてより研究を進める上での助言などを頂いてきました，慶應義塾大学環境情報学部准教授楠本 博之博士，同学部専任講師 湧川隆次博士，同学部専任講師 バンミーター・ロドニー博士，慶應義塾大学大学院政策・メディア研究科講師 吉藤英明博士に感謝します．ありがとうございまいた．

慶應義塾大学大学院政策・メディア研究科 小原泰弘氏に感謝します．本論文を執筆するに辺り，氏には計算量の議論について大いに助けて頂きました．氏は私が研究室に入った当時から研究グループのリーダを務められており，技術的な内容だけではなく研究に対する心構えなど，いろいろなことを学ばせて頂きました．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，同研究科博士課程 白畑真氏に特別の感謝をします．両氏には研究室に入った時より長らくお世話になってきました．南政樹氏には学部時代より指導を頂き，無理難題に挑戦するという気構えを教えて頂きました．修士課程に入ってからも，自身がお忙しい中で論文執筆や研究そのものをどのようにすすめるべきかについて相談に乗っていただき，様々な助言を頂きました．また，白畑真氏にはセキュリティに関する分野について様々な事をご教授いただきました．膨大な知識と経験を背景とした技術的な指導だけではなく，セキュリティの考え方について，とても多くのことを学ばせて頂きました．両氏無くしてはこの修士論文はなし

得ませんでした．本当に，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村井研究室のメンバーである植原啓介博士，三次仁博士，鈴木茂哉氏，稲葉達也氏，中根雅文氏，土本康生氏，川喜田佑介氏，杉本信太氏，佐藤雅明氏，片岡広太郎氏，三屋光史朗氏，海崎良氏，岡田耕司氏，工藤紀篤氏，久松剛氏，三島和宏氏，松園和久氏，大藪勇輝氏，中村友一氏，Achmad Basuki 氏，苧阪浩輔氏，遠峰隆史氏，松谷健史氏，金井瑛氏，空閑洋平氏，奥村佑介氏，本多倫夫氏，尾崎隆亮氏，佐藤龍氏，江村桂吾氏，佐藤泰介氏，六田佳祐氏，黒宮佑介氏，波多野敏明氏，上原雄貴氏，朝永愛子女史，そして OB の東京大学今泉英明博士，Cisco Systems 小椋康平氏，NTT 研究所 豊野剛氏，慶應義塾大学 DMC 機構の遠山緑生氏，大久保成氏に感謝いたします．特に尾崎隆亮氏は氏自身の卒論が大変な時に私の投稿論文の英語についてコメントして頂きました．波多野敏明氏，上原雄貴氏，朝永愛子女史にはそれぞれ掲載するデータの加工や論文の修正を手伝って頂きました．大藪勇輝氏，中村友一氏，Achmad Basuki 氏，苧阪浩輔氏は共に修士論文の執筆に向けて，時に励まし合い，時に愚痴りあってきました．ありがとうございました．

米ブリザード社に感謝します．御社が発売するゲームはどれも秀逸なものばかりで，修論執筆の際に行き詰まった精神状態を潤し，時にストレス解消に貢献してくれました．

最後に，大学入学から 6 年間に渡る研究生活で，私を支え続けてくれた父と母，そして細井ゆりな女史に感謝します．

参考文献

[1] Bruce Schneier. Beyond Fear: Thinking Sensibly About Security in an Uncertain World. *Springer*, Aug 2003.

[2] CERT Advisory CA-2003-04 :MS-SQL Server worm. *http://www.cert.org/advisories/CA-2003-04.html*, Jan 2003.

[3] CERT Advisory CA-2003-20 W32/Blaster worm. *http://www.cert.org/advisories/CA-2003-20.html*, Aug 2003.

[4] 独立行政法人 情報処理推進機構. 国内・海外におけるコンピュータウィルス被害状況調査. *http://www.ipa.go.jp/security/fy15/reports/virus-survey/documents/2003_calc_model.pdf*, Apr 2004.

[5] 金子 勇. *Winny* **の技術**, chapter Appendix C. アスキー, 10 2005.

[6] 株式会社イプシ・マーケティング研究所. 2006 年 国内における情報セキュリティ事象被害状況調査報告書, Aug 2007. http://www.ipa.go.jp/security/fy18/reports/virus-survey/documents/2006_virus_domestic.pdf.

[7] Bruce Schneier. Secrets and Lies: Digital Security in a Networked World. *John Wiley & Sons*, Aug 2000.

[8] Chris McNab. Network Security Assessment. *O'REILLY*, Mar 2004.

[9] Mark Cooper, Stephen Northcutt , Matt Fearnow , Karen Frederick. Intrusion Signatures and Analysis. *Person Education*, Jan 2001.

[10] Stephen Northcutt, Donald McLachlan, Judy Novak. Network Intrusion Detection: An Analystś Handbook (2nd Edition). *Paperback*, Sep 2000.

[11] ISO Standards. ISO/IEC 17799:2005, Information technology – Security techniques – Code of practice for information security management.

[12] Wenke Lee, Salvatore Stolfo, and Kui Mok. Mining in a data-flow environment: Experience in network intrusion detection. In Surajit Chaudhuri and David Madigan, editors, *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pages 114–124, 1999.

[13] CERT Advisory CA-1996-26 Denial-of-Service Attack via ping. *http://www.cert.org/advisories/CA-1996-26.html*, Dec 1996.

[14] Chris Anley. Advanced SQL Injection In SQL Server Applications. Next Generation Security Software Ltd, 2002.

[15] H. Debar, D. Curry, B. Feinstein. RFC 4765: The Intrusion Detection Message Exchange Format (IDMEF), Mar 2007. http://www.ietf.org/rfc/rfc4765.txt.

[16] UNYUN. **ハッカー・プログラミング大全**. 株式会社データハウス, 2001.

[17] IETF. Request For Comments. *http://www.ietf.org/rfc.html*.

[18] Jonathan B. Postel. RFC 821: Simple Mail Transfer Protocol. *http://www.ietf.org/rfc/rfc821.txt*, Aug 1982.

[19] R. Nelson. Some Observations on Implementations of the Post Office Protocol (POP3). *http://www.ietf.org/rfc/rfc1957.txt*, Jun.

[20] ASCII Table and Description. *http://www.lookuptables.com/*.

[21] Richard Bejtlich. *EXTRUSION DETECTION*, chapter 1-10. Addison-Wesley, 2005.

[22] Salvador Mandujano. Outbound intrusion detection.

[23] 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*, pages 296–305, 1990.

[24] WheelGroup Corporation. NetRanger, 1994.

[25] Martin Roesch. Snort. *http://www.snort.org*, 12 1998.

[26] Thomas H. Ptacek, Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Jan 1998. http://insecure.org/stf/secnet_ids/secnet_ids.html.

[27] Paul Bacher,Thorsten Holz,Markus Kotter,Georg Wicherski. Know your enemy: Tracking botnets, May 2005. http://www.honeynet.org/papers/bots/.

[28] Cyber-TA Research and Development Project. SRI Honeynet and BotHunter Malware Analysis Automatic Summary Analysis Table, 2005.

[29] David Geer. Malicious bots threaten network security. *Computer*, 38(1):18–20, 2005.

[30] Moheeb Abu Rajab and Jay Zarfoss and Fabian Monrose and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52, New York, NY, USA, 2006. ACM.
http://doi.acm.org/10.1145/1177080.1177086.

[31] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, Request for Comments 2616.
ftp://ftp.isi.edu/in-notes/rfc2616.txt, Jun 1999.

[32] Kaspersky Lab. Kaspersky free online virus scan, 1997.
http://www.kaspersky.com/scanforvirus.

[33] Microsoft Corporation. Microsoft Windows. http://www.microsoft.com/.

[34] Bram Cohen. BitTorrent. http://www.bittorrent.com/.

[35] eDonkey2000. http://web.archive.org/web/20010213200827/www.edonkey2000.com/overview.html.

[36] Gnuttela.com. http://www.gnutella.com/.

[37] 鵜飼 裕司. P2P ソフト Share の暗号を解析，ネットワーク可視化システムを開発. ITpro, Jan 2007.
http://itpro.nikkeibp.co.jp/article/Watcher/20070122/259207/.

[38] Symantec. W32.hllw.antinny, Aug 2003.
http://www.symantec.com/region/jp/sarcj/data/w/w32.hllw.antinny.html.

[39] Andrew W. Moore, Denis Zuevy. Internet Traffic Classification Using Bayesian Analysis Techniques. *SIGMETRICS Ó5*, 2005.

[40] P-CUBE. Approaches To Controlling Peer-to-Peer Traffic: A Technical Analysis.
http://www.p-cube.com/doc_root/products/Engage/WP_Approaches_Controlling_P2P_Traffic_31403.pdf.

[41] Karagiannis, Thomas and Papagiannaki, Konstantina and Faloutsos, Michalis. Blinc: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, New York, NY, USA, 2005. ACM Press.

[42] Risto Vaarandi. SEC - a Lightweight Event Correlation Tool. *Proceedings of the 2002 IEEE Workshop on IP Operations and Management*, pages 111–115, 2002.

[43] Giovanni Vigna,Richard A. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. *ACSAC*, 1998.

[44] Klaus Julisch IBM Research, Zurich Research Laboratory, Rschlikon, Switzerland. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, Nov 2003.

[45] OSSIM Open Source Security Information Management. `http://www.ossim.net/`.

[46] ArcSight. `http://www.arcsight.com/`.

[47] LLC QoSient. Argus, 2000. `http://qosient.com/argus/`.

[48] ArcSight. Using Advanced Event Correlation to Improve Enterprise Security, Compliance and Business Posture. White Paper, Apr 2007.

[49] 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`.

[50] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[51] Sourcefire. *Snort Users Manual 2.8.0*, 2007. `http://www.snort.org/docs/snort_htmanuals/htmanual_2.4/node22.html`.

[52] Network Flight Recorder. *`http://www.nfr.com`*.

[53] 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*, 10(1):39–46, 2006.

[54] 秀誠 朝長 and 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***]*, 2006(129):13–18, 20061208.

[55] James R. Binkley. An algorithm for anomaly-based botnet detection. *SRUTI '06*, pages 43–48.

[56] 竹森敬祐，三宅優，中尾康二，菅谷史昭，笹瀬巌. 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*, pages 816–825, Jul 2004.

[57] John Jerrim. Benefits of Flow Analysis Using sFlow: Network Visibility, Security and Integrity, 2006. `http://www.lancope.com/resource/`.

[58] Lancope. The Role of Network Behavior Analysis & Response Systems in the Enterprise, 2006.

[59] 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.

[60] 財団法人インターネット協会, editor. **インターネット白書** *2007*. インプレス R&D, Jun 2007.

[61] Internet Assigned Numbers Authority. Port numbers, 2007. http://www.iana.org/assignments/port-numbers.

[62] 須藤 年章 and 富士原 圭. 仮想インターネットを用いたボットネット挙動解析システムの評価. *Computer Security Symposium 2006*, 2006.

[63] Bleeding edge of snort: spyware-dns.rules, Aug 2007. http://www.bleedingsnort.com/.

[64] Salman A. Baset and Henning G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *IEEE Infocom*, 2006.

[65] R. Jain and S. Routhier. Packet Trains-Measurements and a New Model for Computer Network Traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):986–995, Sep 1986. http://www.cs.wustl.edu/~jain/papers/train.htm.

[66] Juniper Networks Inc. NetScreen IDP 100/500. http://www.netscreen.com/pdf/NS5000DS_datasheet.pdf.

[67] Robin Sommer, Vern Paxson. Enhancing byte-level network intrusion detection signature with context. *ACM*, 2003.

[68] 水谷正慶, 白畑真, 南政樹, 村井純. Session-based IDS の設計と実装. **電子情報通信学会 次世代ネットワーク論文特集**, Mar 2005.

[69] 藤田 直行. 侵入検知ポリシ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**, J88-D-1:391–400, 2005.

[70] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, 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*, 1996.

[71] LIBPCAP. *http://www.tcpdump.org/*.

[72] Daniel Veillard. libxml. Sep 1999. `http://xmlsoft.org/`.

[73] Philip Hazel. PCRE - Perl Compatible Regular Expressions. sep 1997. `http://www.pcre.org/`.

[74] Yoann Vandoorselaere. Prelude-IDS - The Hybrid IDS framework. *http://www.prelude-ids.org/*.

[75] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, Francois Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition), Feb 2004. `http://www.w3.org/TR/2004/REC-xml-20040204/`.

[76] LBNL's Network Research Group. TCPDUMP. `http://www.tcpdump.org/`.

[77] Lawrence Berkeley National Laboratory. Bro Intrusion Detection System, 2003-2007. `http://www.bro-ids.org/`.

[78] Rafal Wojtczuk. libnids, 1999. `http://libnids.sourceforge.net/`.

[79] Donald R. Morrison. Patricia practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4):514–534, 1968.

[80] Bleeding Edge of Snort, Oct 2004. `http://www.bleedingsnort.com/`.

[81] A.V. Aho and M.J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[82] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 512–521, New York, NY, USA, 2004. ACM.

[83] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: automated construction of application signatures. In *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202, New York, NY, USA, 2005. ACM.

[84] Larry Wall. Perl, 1987. `http://www.perl.com/`.

[85] アンドレアス・ロイター ジム・グレイ. **トランザクション処理 -概念と技法-**, volume 1. 日経 BP, Oct 2001.

[86] R. セジウィック. **アルゴリズム** *C*. 近代科学社, 2004. (翻訳) 野下 浩平, 佐藤 創, 星守, 田口 東.

[87] Lincoln Laboratory, Massachusetts Institute of Technology. 1999 DARPA Intrusion Detection Evaluation Data Set, 1999.
http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html.

[88] Gerald Combs. Wireshark. http://www.wireshark.org/.

[89] Bleeding edge threats: blackhole.conf, Nov 2007. http://doc.bleedingthreats.net/pub/Main/SnortConfSamples/blackhole.conf.

[90] Nepenthes - finest collection -. http://nepenthes.mwcollect.org/.

[91] Grisoft. AVG Free Edition. http://free.grisoft.com/.

[92] Aaron Turner. Tcpreplay, 2005. http://tcpreplay.synfin.net/trac/.

[93] Fabian Schneider, Jorg Wallerich, Anja Feldmann, Robin Sommer. High Performance Packet Capture, 2006.
http://www.net.t-labs.tu-berlin.de/research/bpcs/.

[94] 独立行政法人 情報処理推進機構. ウイルス情報ipedia, 2006.
https://isec.ipa.go.jp/zha-virusdb/web/Top.php.

[95] IP アドレス管理政策検討ワーキンググループ, IPv4 アドレス枯渇克服策検討ワーキンググループ. IPv4 アドレス在庫枯渇問題に関する検討報告書（第一次）, 2007.

付 録A　ルールの記述

表 A.1 は ROOK の実装で読み込むルールを作成する際，各タグに記述できる属性を表でまとめている．値の内容の列がカンマ区切りになっている属性は，区切られている各値の中から選択する．初期値の列が N/A となっている属性は，初期値が存在せず記述が省略できない属性である．

表 A.2 は ROOK で実装されているフィルタの一覧である．条件として記述するフィルタ名，フィルタを実行した場合の計算量，フィルタを実行した場合の返値，返値の説明を示している．返値はパラメータに値を保存する場合に利用する．

表 A.1: 各タグの属性一覧

タグ名	属性名	値の内容	初期値	説明
Param	name	任意のパラメータ名	N/A	ルール内で一意のパラメータ名を指定する
	type	INT, STRING, RAW, BOOL, ADDR	N/A	パラメータの型を指定する
	mode	allow, deny, stack	allow	上書きモードを指定する
Session	proto	プロトコル名	N/A	対象とするプロトコルを指定する．指定できるのは第 9.3.1 節で示したモジュール名 (例:TCP)
Stat	name	状態名	N/A	ルール内で一意の状態名を指定する (ただし，start と end は予約済み)
Confirm	pri	優先度	10	0 から 100 の間で優先度を指定する．数字が小さいほど優先度が高い
Trigger	from	状態名	start	遷移元の状態名を指定する
	to	状態名	end	遷移先の状態名を指定する
Sig	フィルタ	–	–	後述
Set	name	パラメータ名	N/A	書き込むパラメータを名前で指定する
	scope	session, src, dst, global	N/A	パラメータを書き込む対象を指定する
	filter	フィルタ名	N/A	パラメータをトラフィックの解析結果をもとに書き込む場合，データを抽出するフィルタを指定する (data との併用不可)
	key	フィルタ引数	–	filter 属性を使う際に引数を指定する必要があれば指定できる
	data	書き込みデータ	–	書き込むデータを予め定義できる (filter との併用不可)
	timeout	時間切れ秒数	N/A	パラメータを持続させる時間を秒数で指定する
Eval	name	パラメータ名	N/A	比較するパラメータを名前で指定する
	scope	session, src, dst, global	N/A	パラメータを比較する対象を指定する
	filter	フィルタ名	N/A	パラメータをトラフィックの解析結果をもとに比較する場合，データを抽出するフィルタを指定する (data との併用不可)
	key	フィルタ引数	–	filter 属性を使う際に引数を指定する必要があれば指定できる
	data	比較データ	–	比較するデータを予め定義できる (filter との併用不可)
Unset	name	パラメータ名	N/A	削除するパラメータを名前で指定する
	scope	session, src, dst, global	N/A	パラメータを削除する対象を指定する

表 A.2: ROOK で実装されているフィルタ一覧

フィルタ名	計算量	返値の型	返値
ipv4.src_addr	$O(1)$	IP アドレス	IPv4 パケットの送信元 IP アドレス
ipv4.dst_addr	$O(1)$	IP アドレス	IPv4 パケットの宛先 IP アドレス
ipv4.proto	$O(1)$	整数値	IPv4 パケットの Protocol フィールド
ipv6.src_addr	$O(1)$	IP アドレス	IPv4 パケットの送信元 IP アドレス
ipv6.dst_addr	$O(1)$	IP アドレス	IPv4 パケットの宛先 IP アドレス
ipv6.proto	$O(1)$	整数値	IPv6 パケットの Next Field の値
tcp.dst_port	$O(1)$	整数値	TCP パケットの送信元ポート番号
tcp.src_port	$O(1)$	整数値	TCP パケットの宛先ポート番号
tcp.seg_size	$O(1)$	整数値	TCP パケットのセグメントサイズ
tcp.seg_id	$O(1)$	整数値	TCP パケットの到着セグメント番号
tcp.dir	$O(1)$	文字列	TCP パケットの送信方向．サーバ宛かクライアント宛
tcp.content	$O(L)$	文字列	TCP のセグメントデータ．正規表現を使用可能
tcp.flag	$O(1)$	文字列	TCP フラグ
tcp.proto	$O(1)$	文字列	TCP の上位プロトコル
udp.src_port	$O(1)$	整数値	UDP の送信元ポート番号
udp.dst_port	$O(1)$	整数値	UDP の宛先ポート番号
udp.content	$O(L)$	RAW	UDP のデータグラム
udp.size	$O(1)$	RAW	UDP のデータグラムサイズ
udp.proto	$O(1)$	文字列	UDP の上位プロトコル
dns.query	$O(L)$	文字列	DNS のクエリー
dns.answer_query	$O(1)$	文字列	DNS の Answer 部分に含まれるクエリ
dns.answer_res_data	$O(1)$	IP アドレス	DNS の応答内容
dns.answer_res_type	$O(1)$	整数値	DNS の応答タイプ
dns.rep_code	$O(1)$	整数値	DNS の応答コード
http.uri	$O(1)$	文字列	HTTP リクエストに含まれる URI
http.user_agent	$O(1)$	文字列	HTTP リクエストに含まれる User Agent
http.method	$O(1)$	文字列	HTTP リクエストのメソッド
http.dl_data	$O(L)$	RAW	HTTP の応答データ
http.ul_data	$O(L)$	RAW	HTTP の送信データ
icmp4.type	$O(1)$	整数値	ICMP タイプ
icmp4.code	$O(1)$	整数値	ICMP コード
icmp4.size	$O(1)$	整数値	ICMP データ部のサイズ
irc.content	$O(L)$	文字列	IRC のメッセージに含まれる文字列
tftp.opcode	$O(1)$	文字列	TFTP の OP コード
tftp.name	$O(1)$	文字列	TFTP のファイル名

付 録B　評価に利用したデータセットの詳細

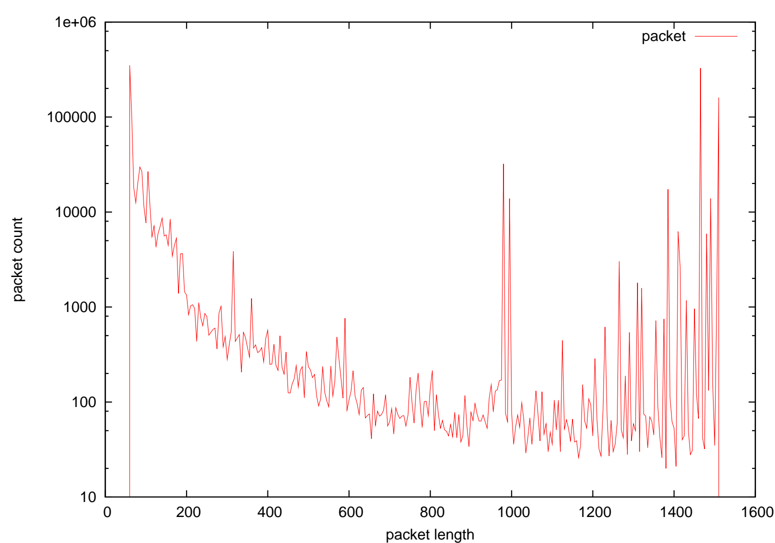

図 B.1: データセット N_4 のパケットサイズの分布

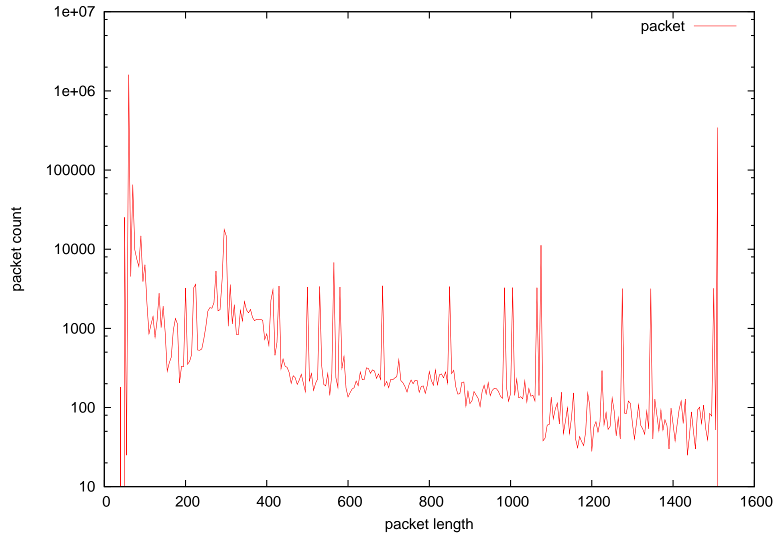

図 B.2: データセット N_5 のパケットサイズの分布

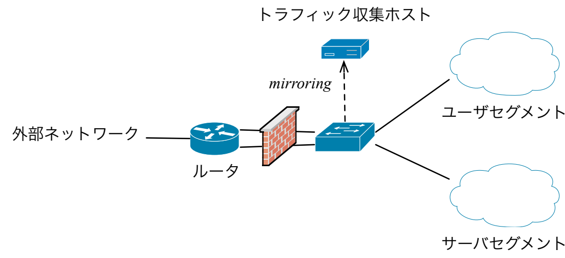

図 B.3: 評価用データセット収集環境 (N_1)

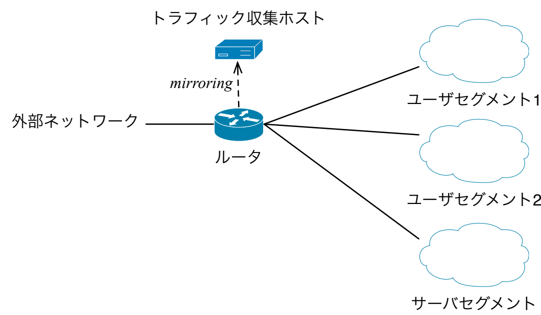

図 B.4: 評価用データセット収集環境 (N_2)

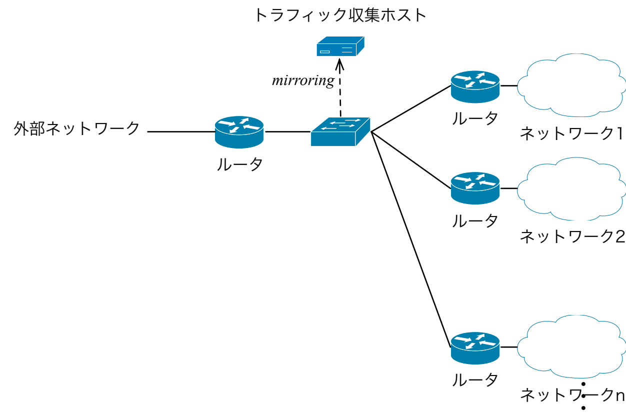

図 B.5: 評価用データセット収集環境 (N_3)

付 録C　精度評価用ルール一覧

```
<!------------------- Rule 1 --------------------------------------------->
<rule name="R1">
  <param name="cc_server" type="addr" mode="stack" />

  <session proto="dns">
    <trigger>
      <sig dns.answer_query="example.com" />
      <set scope="dst" name="cc_server" filter="dns.answer_res_data" key="example.com:A" timeout="1200" />
      <act type="msg" arg="detect example.com" />
    </trigger>
    <!-- 80543 件の疑わしいドメイン名のシグネチャ -->
  </session>

  <session proto="tcp">
    <trigger>
      <sig tcp.dir="to_server">
        <eval scope="src" filter="ipv4.dst_addr" name="cc_server" stat="eq" />
      </sig>
      <act type="dump" arg="sustcp.pcap" />
    </trigger>
  </session>

  <session proto="http">
    <trigger>
      <sig tcp.dir="to_client" http.dl_data="MZ.*This program cannot be run in DOS mode" >
        <eval scope="dst" filter="ipv4.src_addr" name="cc_server" stat="eq" />
      </sig>
      <sig tcp.dir="to_client" http.dl_data="MZ.*This program must be run under Win32" >
        <eval scope="dst" filter="ipv4.src_addr" name="cc_server" stat="eq" />
      </sig>
      <act type="msg" arg="[success] Detected Bot HTTP Activity (MS executable file)" />
      <unset scope="dst" name="cc_server" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_server" http.uri="\.exe">
        <eval scope="src" filter="ipv4.dst_addr" name="cc_server" stat="eq" />
      </sig>
      <act type="msg" arg="[success] Detected Bot HTTP Activity (exe download)" />
      <unset scope="dst" name="cc_server" />
    </trigger>
  </session>

  <session proto="irc">
    <trigger>
      <sig tcp.dir="to_server">
        <eval scope="src" filter="ipv4.dst_addr" name="cc_server" stat="eq" />
      </sig>
      <unset scope="src" name="cc_server" />
      <act type="msg" arg="[success] Detected Bot IRC Activity"/>
      <unset scope="dst" name="cc_server" />
    </trigger>
```

```
  </session>
</rule>

<!------------------- Rule 2 -------------------------------------------->
<rule name="R2">
  <param name="download" type="bool" mode="allow" />

 <session proto="http">
   <trigger>
     <sig tcp.dir="to_client" http.dl_data="MZ.*This program cannot be run in DOS mode" />
     <sig tcp.dir="to_client" http.dl_data="MZ.*This program must be run under Win32" />
     <set scope="src" data="1" name="download" timeout="30" />
     <act type="msg" arg="detected .exe file downlaod" />
   </trigger>
 </session>

  <session proto="irc">
    <trigger>
      <sig tcp.dir="to_server" >
        <eval scope="src" data="1" name="download" stat="eq" />
      </sig>
      <unset scope="src" name="download" />
      <act type="msg" arg="[success] Detect Bot Node Activity"/>
    </trigger>
  </session>
</rule>

<!------------------- Rule 3 -------------------------------------------->
<rule name="R3">
  <param name="scan_445" type="bool" mode="allow" />
  <param name="scan_135" type="bool" mode="allow" />
  <param name="scan_139" type="bool" mode="allow" />
  <param name="scan_296x" type="bool" mode="allow" />
  <param name="counter" type="addr" mode="stack" />

  <session proto="irc">
    <trigger>
      <sig tcp.dir="to_client" irc.content="445 " />
      <set scope="dst" name="scan_445" data="1" timeout="120" />
      <act type="msg" arg="recv msg like command (command 445)" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_client" irc.content="135 " />
      <set scope="dst" name="scan_135" data="1" timeout="120" />
      <act type="msg" arg="recv msg like command (command 135)" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_client" irc.content="139 " />
      <set scope="dst" name="scan_139" data="1" timeout="120" />
      <act type="msg" arg="recv msg like command (command 139)" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_client" irc.content="msass " />
      <set scope="dst" name="scan_135" data="1" timeout="120" />
      <set scope="dst" name="scan_139" data="1" timeout="120" />
      <set scope="dst" name="scan_445" data="1" timeout="120" />
      <act type="msg" arg="recv command (msass)" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_client" irc.content="ipscan " />
      <set scope="dst" name="scan_135" data="1" timeout="120" />
```

```
      <set scope="dst" name="scan_139" data="1" timeout="120" />
      <set scope="dst" name="scan_445" data="1" timeout="120" />
      <set scope="dst" name="scan_296x" data="1" timeout="120" />
      <act type="msg" arg="recv command (ipscan)" />
    </trigger>

    <trigger>
      <sig tcp.dir="to_client" irc.content="scanall " />
      <set scope="dst" name="scan_135" data="1" timeout="120" />
      <set scope="dst" name="scan_139" data="1" timeout="120" />
      <set scope="dst" name="scan_445" data="1" timeout="120" />
      <set scope="dst" name="scan_296x" data="1" timeout="120" />
      <act type="msg" arg="recv command (scanall)" />
    </trigger>

  </session>

  <session proto="tcp">
    <trigger>
      <sig tcp.dst_port="135" tcp.flag="s,sarf">
        <eval scope="src" name="scan_135" data="1" stat="eq" />
      </sig>
      <set scope="src"  name="counter" filter="ipv4.dst_addr" timeout="60" />
    </trigger>

    <trigger>
      <sig tcp.dst_port="139" tcp.flag="s,sarf">
        <eval scope="src" name="scan_139" data="1" stat="eq" />
      </sig>
      <set scope="src"  name="counter" filter="ipv4.dst_addr" timeout="60" />
    </trigger>

    <trigger>
      <sig tcp.dst_port="445" tcp.flag="s,sarf">
        <eval scope="src" name="scan_445" data="1" stat="eq" />
      </sig>
      <set scope="src"  name="counter" filter="ipv4.dst_addr" timeout="60" />
    </trigger>

    <trigger>
      <sig tcp.dst_port="2967~2968" tcp.flag="s,sarf">
        <eval scope="src" name="scan_296x" data="1" stat="eq" />
      </sig>
      <set scope="src"  name="counter" filter="ipv4.dst_addr" timeout="60" />
    </trigger>

    <trigger>
      <sig>
        <eval scope="src" name="counter" data="10" stat="gt" opt="count" />
      </sig>
      <act type="msg" arg="[success] bot order, probe" />
    </trigger>
  </session>
</rule>

<!------------------- Rule 4 -------------------------------------------->
<rule name="R4">
  <param name="url_path" type="str" mode="allow" />

  <session proto="irc">
    <trigger>
      <sig tcp.dir="to_client" irc.content="http://\S+" />
      <set scope="dst" filter="irc.content" key="http://[A-Za-z0-9\.\-:]+(/\S+)"
   name="url_path" timeout="120" />
      <act type="msg" arg="URL appeared in IRC message" />
```

```
    </trigger>
  </session>

  <session proto="http">
    <trigger to="sent_req">
      <sig tcp.dir="to_server">
        <eval scope="src" filter="http.uri" name="url_path" stat="eq" />
      </sig>
      <act type="msg" arg="Match URI" />
    </trigger>

    <trigger from="sent_req">
      <sig tcp.dir="to_client" http.dl_data="MZ.*This program cannot be run in DOS mode." />
      <act type="msg" arg="[success] Detected Malware download" />
    </trigger>
  </session>
</rule>

<!------------------- Rule 5 --------------------------------------------->
<rule name="R5">
  <param name="pkt_size" type="int" mode="allow" />
  <param name="pkt_count" type="int" mode="allow" />
  <param name="ssn_count" type="addr" mode="stack" />
  <param name="client_sent" type="bool" mode="allow" />
  <param name="server_sent" type="bool" mode="allow" />

  <session proto="tcp">
    <trigger to="track">
      <sig tcp.seg_id="1" tcp.seg_size="1400~" />
      <set scope="session" name="pkt_size" filter="tcp.seg_size" timeout="30" />
    </trigger>

    <trigger from="track" to="track">
      <sig tcp.seg_id="2~4" tcp.seg_size="1400~" tcp.dir="to_client">
        <eval scope="session" name="pkt_size" filter="tcp.seg_size" stat="eq" />
      </sig>
      <set scope="session" name="pkt_count" data="1" opt="inc" timeout="30" />
      <set scope="session" name="server_sent" data="1" timeout="120" />
    </trigger>

    <trigger from="track" to="track">
      <sig tcp.seg_id="2~4" tcp.seg_size="1400~" tcp.dir="to_server">
        <eval scope="session" name="pkt_size" filter="tcp.seg_size" stat="eq" />
      </sig>
      <set scope="session" name="pkt_count" data="1" opt="inc" timeout="30" />
      <set scope="session" name="client_sent" data="1" timeout="120" />
    </trigger>


    <trigger from="track">
      <sig tcp.seg_id="2~5" tcp.seg_size="1400~">
        <eval scope="session" name="pkt_size" filter="tcp.seg_size" stat="ne" />
      </sig>
      <unset scope="session" name="pkt_count" />
      <unset scope="session" name="pkt_size"  />
    </trigger>

    <trigger from="track">
      <sig tcp.seg_id="5">
        <eval scope="session" name="pkt_size" filter="tcp.seg_size" stat="eq" />
        <eval scope="session" name="pkt_count" data="3" stat="eq" />
        <eval scope="session" name="client_sent" data="1" stat="eq" />
        <eval scope="session" name="server_sent" data="1" stat="eq" />
      </sig>
      <set scope="src" name="ssn_count" filter="ipv4.dst_addr" timeout="600" />
```

```
      <set scope="dst" name="ssn_count" filter="ipv4.src_addr" timeout="600" />
      <act type="msg" arg="detect session like winnyp" />
      <act type="dump" arg="like_winny.pcap" />
    </trigger>
  </session>

  <session proto="tcp">
    <trigger>
      <sig tcp.seg_id="1"><eval scope="src" name="ssn_count" data="5" opt="count" stat="gt" /></sig>
      <unset scope="src" name="ssn_count" />
      <act type="msg" arg="[success] Winnyp Detection (Src)" />
    </trigger>

    <trigger>
      <sig tcp.seg_id="1"><eval scope="dst" name="ssn_count" data="5" opt="count" stat="gt" /></sig>
      <unset scope="dst" name="ssn_count" />
      <act type="msg" arg="[success] Winnyp Detection (Dst)" />
    </trigger>
  </session>

</rule>
```