2004－ＣＳＥＣ－26　(18)
2004／7／20

ポリシに基づく自律ネットワークセキュリティイベント対応システムの構築

白畑　真[†]　南　政樹[††]　村井　純[†,††]

† 慶應義塾大学　政策・メディア研究科
〒 252–8520 神奈川県藤沢市遠藤 5322
†† 慶應義塾大学　環境情報学部
〒 252–8520 神奈川県藤沢市遠藤 5322
E-mail: {true,minami,jun}@sfc.wide.ad.jp

あらまし　本稿では，ネットワーク管理においてIDSの誤検知等に伴う管理コストを軽減するため，統合的なセキュリティイベント対応モデルを提案し，ネットワークの運用ポリシに基づいて自律的にセキュリティイベントに対応するシステムの設計，実装，評価を行った．本手法によって，ネットワーク管理者が運用ポリシをルールとして定義するだけで，IDSやFirewallなどのセキュリティ機器から得られるセキュリティイベントの内容に応じた対応を行うシステムが実現された．ルールにおいては，ホストのOS，稼働しているアプリケーションソフトウェアのバージョン，それらに対する脆弱性等のネットワーク環境情報を動的に反映することで，最新の状況を元にセキュリティイベントを絞り込める．また，セキュリティイベントの内容に応じて，想定される問題に対応するプログラムを設定できる．この結果，ルールに基づいた自律的なセキュリティイベント対応が実現され，管理者よりも迅速に対応が行える．また，イベント対応に要する管理者の負担を軽減できる．

キーワード　IDS，ログ解析，セキュリティイベント，ネットワークマッピング

A Policy-based Autonomous Handling Mechanism for Network Security Events

Shin SHIRAHATA[†], Masaki MINAMI[††], and Jun MURAI[†,††]

† Graduate School of Media and Governance, KEIO University
Endo 5322, Fujisawa-shi, 252–8520 Japan
†† Faculty of Environmental information, KEIO University
Endo 5322, Fujisawa-shi, 252–8520 Japan
E-mail: {true,minami,jun}@sfc.wide.ad.jp

Abstract　In this paper, we propose an integrative security event-handling framework for reducing network administration costs. We designed and implemented policy-based autonomous security event handling system. Our proposed system collects security events from various security devices such as IDS or firewall, and handles events based on operational policy. Then, this system evaluates network environment data such as OS, software version, and vulnerability data. Therefore, our system realizes highly accurate automated security event handing. In the rules, operators can script in event handling conditions and procedure that can run program for handing security event or notify to an operators. As a result, we realize network administration cost reduction, and rapid security event handling by rule-based security event handling system.

Key words　IDS, Log analysis, Security events, Network mapping

1. はじめに

近年，クラッカーやワームによる攻撃が大きな問題となっている．ワームの感染経路として悪用できる脆弱性が公表されてからワームが登場するまでの期間が短くなる傾向にある．セキュリティ対策サービスを提供している Foundstone 社の調査 [1] によると 1999 年には平均 288 日だった期間が，2004 年には平均 10 日間となり，ネットワーク管理者には迅速な脆弱性対策が求められるようになっている．

こういった状況を踏まえ，ネットワークにおけるセキュリティ上の問題を検知する侵入検知システム (Intrusion Detection System. 以下 IDS と呼ぶ) を導入するネットワークが増加している．IDS によって，万が一攻撃を受けた場合でも，管理者が攻撃内容を把握し，対応策を実行できる．

しかし，ネットワークで IDS を運用すると，誤検知や対策済み検知，多重検知など，実際の脅威を伴わないセキュリティイベントが多く発生する．そのため，IDS 管理者は多数のイベントの中から危険性の高いセキュリティイベントを判別する必要があり，多大な運用コストを求められる．

本稿では，セキュリティイベントに対して，各ネットワークの運用ポリシを反映した対応を自律的に実施する手法を提案し，その実装と評価について述べる．

提案手法は，管理者があらかじめ運用ポリシをルールとして記述しておき，システムがルールに基づいて IDS や Firewall 等を始めとしたセキュリティ機器が出力するセキュリティイベントに対応することで，IDS 管理者の負担を軽減する．ルールについては，受動的フィンガープリンティング手法 [2] によるネットワークマッピングやセキュリティスキャナの出力データを動的に反映することにより，高い精度での対応が可能になる．

2. 既存手法と問題点

本稿では，ネットワーク管理者が各ノードに管理権限を委譲している中規模以上のネットワーク，例えば大学などのキャンパスネットワーク環境を想定する．この環境では，IP アドレスの割り当て方式として，静的割り当てと動的に割り当ての両方が混在しており，各ノード管理者が自律的にノードの構成変更などを行うものとする．

2.1 シグネチャ型 IDS

シグネチャ型 IDS はあらかじめ検出すべきトラフィックのパターンをシグネチャを設定しておき，その条件に合致した通信が検出されるとアラートなどのセキュリティイベントを発生させる．シグネチャ型 IDS は，緊急度を各シグネチャに定義することが一般的であるため，シグネチャの緊急度がイベントの緊急度となる．例えば，サーバへ管理者権限で侵入する攻撃を検出するシグネチャに高い緊急度が付加されている場合，攻撃の成否とは無関係に，高い緊急度のセキュリティイベントが発生する．

既存のシグネチャ型 IDS は，各ネットワークで稼働しているノードに関する情報を持たないため，攻撃を受けた際のリスクを誤って評価してしまう．従って，実害を伴わないセキュリティイベントにより，ネットワーク管理者の IDS の運用コストが増加する問題がある．

例えば，セキュリティパッチが適用されワームに感染する恐れのないノードに対し，ワームのアタックが行われた場合，IDS はパッチの適用の有無にかかわらずアタックを検知する．このため，管理者に対して過剰なイベントが通知される．この問題を対策済み検知と呼ぶ．

また一方で，Apache で稼働する Web サーバに対して，IIS の脆弱性を狙った攻撃が行われたとしても，被害が発生するリスクはないにも関わらず，IDS に当該イベントが検知される．本稿では，上記の問題を過剰検知と呼ぶ．このように，既存のシグネチャ型 IDS は，ノードに対して影響を及ぼすリスクを考慮していない．

2.2 Target-Based IDS

Target-based IDS [3] は，あらかじめセキュリティスキャナなどを利用して監視対象のノードのプロファイルを作成しておき，IDS のルールと統合することで，監視対象にとって脅威となるイベントをフィルタする IDS である．Target-based IDS の利用により，従来の IDS で存在した対策済み検知や，過剰検知等の問題をある程度解決できる．

Target-Based IDS は，監視対象ノードのプロファイルにより，セキュリティイベント中の重要度に対する精度が左右されるため，正確なプロファイルの作成が必要である．しかし，Target-Based IDS のプロファイル作成に連携できるツールが限定されていることが多い．このため，ツールが未対応のアプリケーションが稼働している場合や，未対応のセキュリティホールが存在する場合には，IDS が正しく脅威を判別できず，攻撃を見逃す可能性がある．

また，ネットワークにおける脅威は常に変化し続けるため，プロファイルが古くなると，かえって脅威を見逃してしまう恐れがある．例えば，外部から内部ネットワークにノードを持ち込こめる環境では，セキュリティ上の脆弱性が存在したままのノードが持ち込まれ，ワームの感染の原因となる可能性がある．また，プロファイル作成後に新たな脆弱性が発覚した場合には，Target-Based IDS がその脆弱性に対する影響を含むようにプロファイルを更新する必要がある．

2.3 既存の手法のまとめ

シグネチャ型 IDS はノードに対して影響を及ぼすリスクを考慮していないため，冗長なセキュリティイベントが多量に発生し，運用コストが上昇する問題がある．また，Target-Based IDS はネットワーク環境の動的な変化に対応しない場合，ノードに対して影響を及ぼすリスクを正しく評価できないため，攻撃の見逃しが発生する問題がある．

3. 問題解決に必要な条件

第 2 節で述べた問題を踏まえ，問題を解決するために，各ネットワークごとの運用ポリシを反映したセキュリティイベント対応を自律的に行う手法の実現に必要な条件を定める．

以下に，問題解決に必要な条件を示す．

条件 1　ネットワーク状況の動的な認知

条件 2　複数の IDS や Firewall に対応可能であること

条件 3　運用ポリシの反映

3.1　ネットワーク状況の動的な認知

本稿では，ネットワーク環境データをセキュリティイベント中におけるノードの付加的な情報と定義する．具体的には，ノード上で稼働している OS をはじめとするソフトウェアのバージョンデータ，パッチ適用データ，未対策のセキュリティ上の脆弱性などがネットワーク環境データである．

過剰検知や対策済み検知の問題を解決するには，セキュリティイベントに加えてこれらのネットワーク環境データを利用し，応用する必要がある．このため，ネットワーク環境データには，セキュリティスキャナの他，ネットワークマッピングツール，パッチ管理ツールの出力データの正規化が必要である．

各ノードの管理者に管理権限が委譲されたネットワーク環境においては，ノードごとにパッチの適用やソフトウェアのアップデートが行われる．従って，IDS 管理者が全ノードのセキュリティ状態をリアルタイムに把握するのは非常に難しい．また，パッチの適用忘れなどにより，管理者が予期せぬうちにウィルスやワームに感染する場合があるため，定期的なスキャンに加えて常時監視が必要である．

さらに，動的な IP アドレス割り当てを行っている環境下においては，IP アドレスを元にホスト情報の取得を行うと，不正確な結果となる場合がある．動的に割り当てられた IP アドレスが，時間の経過に伴い別のノードに割り当てられる可能性があるからである．

例えば，ある時点で当該 IP アドレスを利用しているノードがセキュリティパッチを適用していても，その後にその IP アドレスを利用するノードはセキュリティパッチを未適用である場合も含めて想定すべきである．

従って，ネットワーク環境データをリアルタイムに把握し，個々のノードにおけるリスク要因を把握する必要がある．

3.2　複数の IDS や Firewall に対応可能であること

中規模以上のネットワークでは，IDS や Firewall 等のセキュリティイベントを生成する機器が複数設置されているケースが多い．セキュリティ機器のログを統合的に管理することで，ネットワーク管理者はネットワーク上で発生している事象の把握が容易となる．

また，複数ベンダの機器によるネットワーク構築が行われるケースもある．セキュリティイベントを統合管理するには，ベンダや機種ごとに異なる IDS や Firewall のイベントの表現形式の差異を吸収できる必要がある．

3.3　運用ポリシの反映

通常，ネットワークは独自の運用ポリシを元に運用されるが，IDS や Firewall だけでは運用ポリシの反映に限界がある．たとえば，原則的にネットワーク内部のノードが外部の SMTP サーバと通信することを許可するが，ウィルスメールや spam メールを送信しているクライアントに限り，ネットワークの外部と通信を禁止するポリシがあるとする．その場合，本システムがウィルスメールを送信しているというイベントを認識し，そのノードからの通信をフィルタするといった対応ルールが必要になる．

運用ポリシに基づき，どのような手順で対応を行うのかを定めたルールがあれば，システムが自律的にセキュリティイベントへ対応できるため，管理者の負担が軽減される．

4.　設　　計

本節では，問題解決のための条件を元に，問題を解決するシステムの設計を行った．図 1 に本システムの概要を示す．

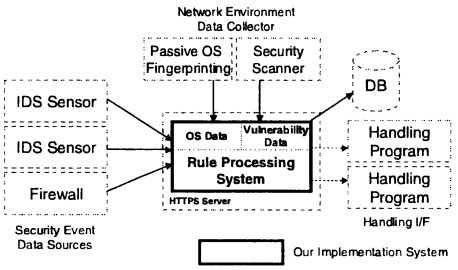

図 1　システムの概要

本手法を実現するシステムは，セキュリティイベント部 (Security Event Data Sources)，ネットワーク環境データ部 (Network Environment Data Collector)，ルール処理部 (Rule Processing System)，イベント対応部 (Handling I/F) によって構成される．

4.1　セキュリティイベント部

セキュリティイベント部は，複数の IDS や Firewall などのセキュリティ機器からセキュリティイベントを収集し，その結果をルール処理部に提供する部分である．

セキュリティイベント部では，条件 2 を満たすため，複数のセキュリティ機器に対応する必要がある．しかし，セキュリティ機器の出力するデータ形式は様々である．そこで，必要に応じてルール処理部が解釈可能なデータ形式に正規化することとする．また，各ツールのデータ出力を正規化したデータをノード識別子の付加データとして定義する．

4.2　ネットワーク環境データ部

ネットワーク環境データ部は，前節の条件 1 で定めたネットワーク状況の動的な認知を実現するため，セキュリティスキャナや各種ネットワークマッピングツールからネットワーク環境データを収集し，データの更新を行う．

また，本システムにおいては，動的な IP アドレス割り当て環境下における同一 IP アドレスの別ノードへの再割り当て，各ノードにおけるソフトウェアのバージョンアップやパッチの適用などによる状況の変化などのネットワーク環境データの頻繁な更新が想定されるため，システムを再起動することなくネットワーク環境データが更新できる必要がある．

4.3　ルール処理部

ルール処理部においては，条件 3 で定めた運用ポリシの反映のため，ルール内でセキュリティイベントに含まれるデータおよびネットワーク環境データを変数として扱えるように定義し，制御構文ではこれらの変数を参照したポリシを記述可能にする．

また，条件1を満たすため，イベントに対応するたびにネットワーク環境データを参照する．図2にセキュリティイベントのデータ構造を示す．

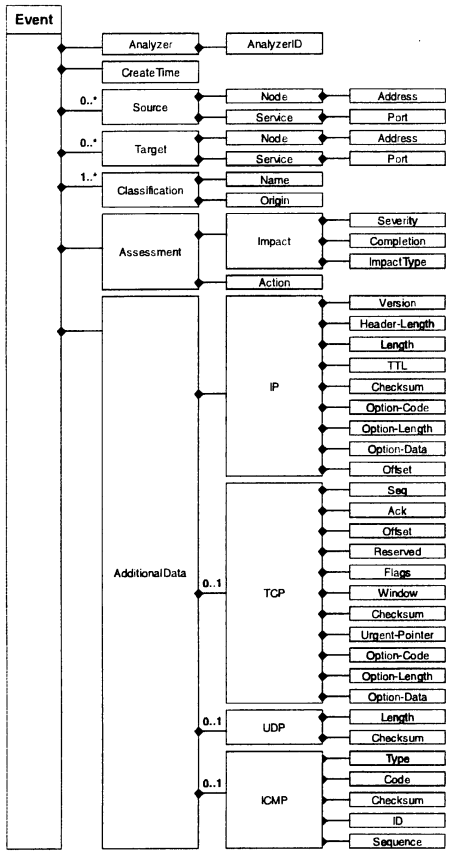

図2　セキュリティイベントのデータ構造

同時に複数のセキュリティ機器からイベントデータを取得するため，同時接続に対応できるようにする．さらに，イベントデータのデータソースおよびルール処理部の各ホストのなりすましを防止するため，認証や特定が容易なプロトコルを用いる．

4.4　イベント対応部

イベント対応部においては，ルール処理部での結果に基づき，当該セキュリティイベントに対応する外部プログラムを必要に応じて起動可能にする．この際，ルール処理部からセキュリティイベントに含まれるデータやネットワーク環境データを引き継ぎ，外部プログラムが利用可能にするものとする．

5.　実　　装

本節では，第4節で述べたシステムの実装について述べる．本実装は，Debian GNU/Linux testing (kernel 2.4.26)，Apache HTTP Server 1.3.29 (mod_ssl 2.8.14，mod_air 0.9.8 の各モジュールを利用)，PostgreSQL 7.4，Snort 2.0.2 上で行った．図3に本実装の概要を示す．

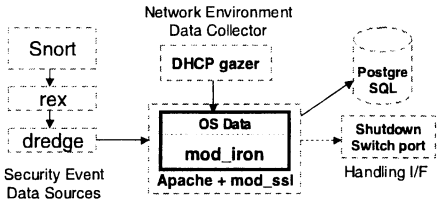

図3　本実装の概要

5.1　セキュリティイベント部

本実装ではネットワーク型IDSであるSnortをセキュリティ機器として利用した．また，セキュリティイベントの正規化のため，Snortの出力したログをXML形式に変換するAirCERT [4] プロジェクトのツールであるrex [5] と，変換されたデータをセキュリティイベント部に送信するためにdredge [5] を用いた．伝送プロトコルには，セキュリティイベントの盗聴や，第三者のなりすましを防止するため，HTTPSを用いた．

5.2　ネットワーク環境データ部

ネットワーク環境データ部では，ネットワークマッピングツールや，セキュリティスキャナの結果をルールで解釈可能な形式で出力する．

本実装では，IPアドレスなどのノード識別子の付加的データを定義し，動的に更新することで，ルール処理部からネットワーク環境データを参照可能にした．従って，IPアドレスなどのノードの識別子データを含む任意のプログラムの出力を本システムが利用できる形式に正規化することで，本実装においてそのデータを活用できる．

図4に，IPアドレスが192.168.23.4のマシンで稼働するOSがWindows XPであることを示すデータの記述例を示す．

```
<rule set var="192.168.23.4.osname" value="Windows XP" />
```

図4　ノード識別子 "192.168.23.4" にOSデータを定義した例

また，任意のプログラムのデータが活用可能なことを示すために，ネットワーク環境のデータソースとして受動的OSフィンガープリンティングツールであるDHCP gazer [7] を利用したネットワークマッピングを行い，動的にネットワーク環境データを反映した．

5.3　ルール処理部

ルール処理部は，セキュリティイベント部から送信されたセキュリティイベントを管理者が運用ポリシに従って記述したルールに基づき処理する．本実装では，セキュリティイベントの記述に，Snortと親和性の高いXMLベースのセキュリティイベント記述言語であるSNML [6] を利用した．

また，HTTPSプロトコルでセキュリティイベントを収集するサーバとして，Apacheにmod_sslモジュールを組み込んで利用した．さらに，送信されたセキュリティイベントデータ，ネットワーク環境データの解釈とルールを処理するモジュール

である “mod_iron” を実装した．

mod_iron モジュールのうち，本システムが利用するデータ形式の SNML の解釈機能は AirCERT プロジェクトの mod_air [5] モジュールを，ルールの実行処理は mod_include [12] モジュールを元に実装を行った．

mod_air モジュールは外部から送信されてきたセキュリティイベントデータを解釈し，データベースに格納するモジュールである．mod_iron モジュールにおいては，新規にセキュリティイベントの各データをプロセスの環境変数に設定し，ルールからイベントに含まれるデータを参照可能とする機能拡張を行った．

ルール処理部は mod_include を元に実装を行った．本実装においては，ポリシの変更やネットワーク環境データ部で得られたデータを動的に反映するため，セキュリティイベントを処理するたびにルールファイルを読み込み，解釈する．

図 5 に，IDS のイベント名に “Worm” が含まれ，イベント発生元のソース IP アドレスのノードの OS 名が Windows が始まるルールの例を示す．

```
<?xml version="1.0" encoding="utf-8"?>
<config>
  <rule if expr="($alert.classification.name = /Worm/) &&
  (${alert.source.node.address.ipv4-addr}.${osname} = /^Windows/)" />
  <rule exec cmd="/usr/local/bin/ex_handler $alert.source.ipv4-addr" />
  <rule endif />
</config>
```

図 5　ルールの例

mod_iron モジュールの構成を図 6 に示す．なお，図中の太字は今回の実装において新規追加もしくは改変した関数である．

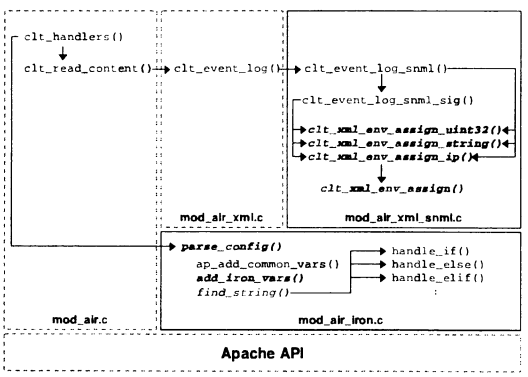

図 6　mod_iron の実装

5.4　イベント対応部

mod_iron モジュールのセキュリティイベントデータ解釈機能で実装した関数について述べる．clt_xml_env_assign_uint32() 関数，clt_xml_env_assign_string() 関数，clt_xml_env_assign_ip() 関数は，それぞれ 32 ビットの unsigned int 型，文字列型，IPv4 アドレスのデータを解釈するための関数である．clt_xml_env_assign() 関数は，これらの関数を元にセキュリティイベントの内容をプロセスの環境変数として設定する．次に，ルールの処理を行う関数として，parse_config() 関数を本ルール形式を処理できるよう変更し，add_iron_vars() 関数で，環境変数の初期化を行った．

イベント対応部は，ルール処理部の判断結果に応じて，セキュリティイベントに含まれるデータを環境変数に設定し，その直後に外部プログラムを呼び出す．外部プログラムは，イベント対応部から渡された環境変数を参照し，セキュリティイベントの内容に応じて対応できる．

イベント対応部プログラムのサンプルとして，SNMP を用いて L2 スイッチのポートを停止するスクリプトを実装した．

6.　評　　価

提案手法が第 3 節で挙げた設計条件を満足しているかどうかを評価した．また，セキュリティイベントをサーバに送信してから，応答が得られるまでの時間を評価し，複雑なルールが記述されている場合のパフォーマンスを評価した．

6.1　定性評価

提案手法が第 3 節で挙げた設計条件を満足しているかについて検証する．

条件 1 で挙げた「ネットワーク状況の動的な認知」については，ネットワーク環境データの収集に用いるソフトウェアに依存するものの，受動的フィンガープリンティング手法を用いたネットワークマッピングを実施することにより，ネットワーク上を流れるトラフィックを元に動的にネットワーク状況を認知することで実現されている．また，セキュリティスキャナ，ネットワークマッピングやパッチ管理ツールの出力データを活用し，ネットワーク環境データを取得できることからも，条件を満足していると考えられる．

条件 2 で挙げた「複数の IDS や Firewall に対応可能であること」については，セキュリティイベントを収集しているノードにおいて，セキュリティイベントデータの正規化を行うことで実現されている．また，イベント対応部が稼働する HTTPS サーバは，複数のクライアントから送信されたイベントデータを同時に処理できる．

条件 3 で挙げた「運用ポリシの反映」については，本手法において管理者がネットワークにおける運用ポリシをルールとして記述し，ポリシに基づいた自律的なイベント対応を実現している．上記のことから，条件を満足していると考えられる．

6.2　ルール内の条件数に伴う応答速度の変化

提案手法においては，ルールの一部にネットワーク環境データを定義するため，ルールの記述が複雑化が予想される．そのため，条件数の異なるルールを用い，セキュリティイベントをサーバに送信してから，サーバから処理完了応答が得られるまでの時間がどのように変化するかを評価した．処理完了応答が得られるまでの時間の大部分は，ルールファイルの処理に占められる．そこでルールファイルに記述する条件数の増加に伴い，ルールファイルの処理時間がどのように変化するかを測定した．

Apache HTTP Server は複数のプロセスを起動し，同時に複数のコネクションを処理可能であるが，今回のテストにおいて

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SNML-Message>
<SNML-Message version="0.3">
<sensor encoding="hex" detail="full">
  <file>rex</file>
  <hostname>localhost</hostname>
</sensor>
<event>
  <signature id="2003" revision="2">MS-SQL Worm propagation attempt
  </signature>
  <reference system="bugtraq">5311</reference>
  <reference system="bugtraq">5310</reference>
  <timestamp>2003-12-08 19:51:24+0900</timestamp>
   <packet>
    <iphdr saddr="10.73.177.162" daddr="192.168.3.169" ver="4" hlen=
    "20" tos="0" len="404" id="3635" flags="0" ttl="113" proto="17">
     <udphdr sport="1053" dport="1434" len="376">
     </udphdr>
    </iphdr>
   </packet>
</event>
</SNML-Message>
```

図 7　評価で利用した SNML メッセージ

は単一プロセスあたりの性能を計測するため，単一プロセスで計測を行った．以下のデータの処理時間は，HTTPS の接続確立後，mod_air モジュールに処理が移ってから，処理を終了するまでの時間を示す．なお，データ転送プロトコルには TLSv1 を用い，暗号化および鍵交換方式には DHE-RSA-AES256-SHA を用いた．

図 7 に，評価用に用いた SNML メッセージを示す．

ルールファイルには，以下の内容を設定して比較を行った．

（1）　処理内容を含まないルール

（2）　条件分岐を 500 回繰り返すルール

（3）　条件分岐を 1,000 回繰り返すルール

（4）　条件分岐を 1,500 回繰り返すルール

（5）　条件分岐を 2,000 回繰り返すルール

各ルールあたりの処理時間を図 8 に示す．

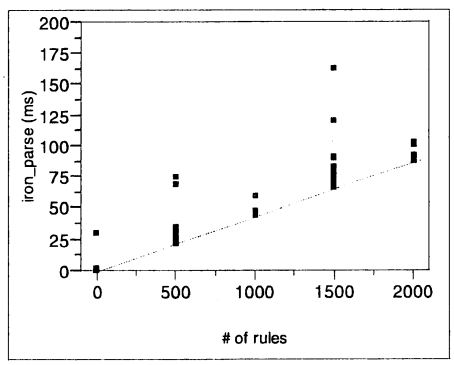

図 8　ルールあたりの処理時間

ルール内に設定された条件分岐の数に伴い，応答速度は直線的に増加すると予想される．これらの結果から近似曲線をみると，

$$time(t) = -0.35448 + 0.0436516 rules$$

となり 1 ルールの処理におよそ 0.04ms を要しているといえる．

この直線を結果にあてはめた場合，決定係数 R^2 は 0.998166 となり，この結果が直線に当てはまっているといえる．

条件分岐を 2,000 回繰り返すルールは，クラス C のネットワークで各ノードに 8 条件を設定したルールを記述した状況に相当する．この条件下において，設定したルールの処理時間は 80ms となり，1 プロセスで 1 時間あたり 45,000 件のセキュリティイベントを処理できる．

従って，実装したシステムは十分な応答速度でセキュリティイベントに対応しているといえる．

7. ま と め

本稿では，IDS などから得られるセキュリティイベントと，セキュリティスキャナやフィンガープリンティングツールから得られるネットワーク環境データを統合的に集約し，ネットワークの運用ポリシに基づき対応するシステムを提案した．

また，本システムを実装し，IDS のイベントデータを受信した際に，ルールに基づき自律的にセキュリティイベントデータに対応していることを確認した．

今後は，複数のセキュリティイベントデータソースやネットワーク環境データを用いた際の性能や，データの統合方法について検討を行う．また，複数のセキュリティイベント間の相関分析について実装，評価する予定である．

文　献

[1] Foundstone, Inc. “Computer Vulnerability-to-Worm Cycle Compressing Dramatically”, 2004 年 5 月, http://www.foundstone.com/company/pressrelease_template.htm?indexid=132

[2] Craig Smith, Peter Grundl, Subterrain Siphon Project. “Know Your Enemy: Passive Fingerprinting”, 2001 年 9 月 http://project.honeynet.org/papers/finger/

[3] TechTarget, “Taking Aim ... Target-based IDSes squelch network noise to pinpoint the alerts you really care about.”, http://infosecuritymag.techtarget.com/ss/0,295796,sid6_iss306_art540,00.html

[4] CERT Coordination Center, “AirCERT”, http://www.cert.org/kb/aircert/

[5] Roman Danyliw, Sean Levy, Brian Trammell and Andrew Kompanek, “AirCERT: The Definitive Guide”, http://aircert.sourceforge.net/docs/manual/index.html

[6] Roman Danyliw, “Snort XML Output Plugin”, http://www.cert.org/kb/snortxml/

[7] 白畑　真，土本 康生，村井　純，“DHCP を用いた受動的フィンガープリンティング手法の提案と実装”，情報処理学会研究報告 2003-DPS-111,2003 年 2 月

[8] Mark Burgess, “Cfengine: a site configuration engine”, USENIX Computing Systems, Vol 8, No. 3 1995., 1995 年

[9] 笠井真理子，萱島信，渡辺義則，中野喜之，“統合セキュリティ運用管理システムにおける設定内容生成機能の提案と実装”，情報処理学会研究報告 2003-DPS-111,2003 年 2 月

[10] Martin Roesch, “Snort: Lightweight Intrusion Detection for Networks”, 13th Systems Administration Conference (LISA '99), 1999 年 11 月

[11] D. Curry, H. Debar and B. Feinstein, “Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition”, http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-11.txt, 2004 年 1 月

[12] The Apache Software Foundation, ”Apache module mod_include”, http://httpd.apache.org/docs/mod/mod_include.html