2005－OS－100 (2)
2005／8／3

# アプリケーション層プロトコルの文脈を考慮したNIDS

山口 聖司† 河野 健二††

†電気通信大学大学院 電気通信学研究科 情報工学専攻
††慶應義塾大学 理工学部 情報工学科
E-mail: sat@zeus.cs.uec.ac.jp, kono@ics.keio.ac.jp

**要旨**

ネットワークから送信される不正なメッセージを検出するためにネットワーク侵入検知システム (NIDS) が広く用いられている．しかし，既存の NIDS は false-positive が頻繁に発生する．本論文では，アプリケーション層プロトコルの状態遷移に基づいて送受信メッセージを検査することにより，従来の NIDS よりも false-positive が少なく，精度が高いと期待される NIDS を提案する．送受信メッセージをアプリケーション層プロトコルで定められた文法に従って解釈し，通信における一連の送受信メッセージから状態の変化を把握する．この方法によりメッセージに含まれる攻撃コードなどを正確に認識する．

# Exploiting the application-layer context for precise NIDS

Satoshi Yamaguchi† Kenji Kono††

†Department of Computer Science, Graduate School of Electro-Communications, University of Electro-Communications
††Department of Information and Computer Science, Keio University
E-mail: sat@zeus.cs.uec.ac.jp, kono@ics.keio.ac.jp

**Abstract**

Network Intrusion Detection System (NIDS) is widely used to detect malicious messages sent from the Internet. Exsiting NIDS has the problem that false-positive warnings are generated quite frequently. This paper proposes a novel sheme of NIDS that inspects messages based on the context of the application-layer protocols. The proposed NIDS interprets messages according to the grammar defined by the protocol, and understands the state transition from a series of messages in the communication. This enables us to define NIDS signatures precisely and thus the proposed NIDS is expected to produce less false-positive warnings.

# 1 はじめに

近年，ネットワークを介した不正な攻撃が急激に増加している．ネットワークに接続されているコンピュータは常に攻撃の脅威にさらされており，コンピュータのセキュリティ対策が重要な課題となっている．

ネットワーク経由で侵入してくる不正な攻撃を検知するために，ネットワーク侵入検知システム (NIDS: Network Intrusion Detection System) が広く導入されている．NIDS は，ネットワーク上を流れるメッセージから不正なものを検出し，ユーザに対して警告を発するセキュリティ・システムである．しかし，一般的に使用されている NIDS は，誤検知が多いという問題がある．SecurityForcus[1] によれば，既存の NIDS をカスタマイズせずにそのまま使用した場合，すべての警告のうちの約 90%が誤検知 (false-positive) であると報告されている．false-positive とは，不正ではないアクセスに対して警告してしまうことをいう．NIDS が不正な攻撃を的確に検知するためには，NIDS の false-positive を減らす必要がある．

既存の NIDS に false-positive が多い理由の 1 つは，NIDS が持つ**シグネチャ**が不正なパケットを厳密に定義できていないことが挙げられる．シグネチャとは，不正なパケットを特徴づけるバイト列をいう．NIDS はこのシグネチャとネットワークから送信されてくるメッセージとを照合することによって検知を行う．従来のシグネチャは単純なバイト列によって表されているため，正当なメッセージであったとしても，そのようなバイト列が偶然含まれている場合はシグネチャに合致してしまう．そのため false-positive が増加する傾向にある．

アプリケーション層プロトコルにおける不正な攻撃メッセージの多くは，RFC などによって定められたプロトコル既約に則して行なわれることが多い．アプリケーション層プロトコルでは，送受信メッセージの文法が定められており，サーバとクライアントの間でやりとりされるメッセージによって状態が変化する．本論文では，定められた文法に則して行なわれる通信の状態遷移を**文脈**と呼び，不正なメッセージを検出するためにアプリケーション層プロトコルの文脈を考慮した NIDS を提案する．

例えば，qpopper 3.0 サーバでは LIST コマンドの引数にオーバーフローしてしまう脆弱性 [2] がある．LIST コマンドは通常，POP3 における認証手続き終了後にのみ受け付けられるメッセージであるので，認証手続き前に攻撃メッセージを送信しても攻撃は成立しない．しかし既存の NIDS では，認証などによる状態変化を考慮しないため，認証前に攻撃メッセージを送信しても検知してしまい，false-positive を発生させてしまう．

本論文で提案する検出方法では，送受信メッセージをプロトコルで定められた文法に従って解釈し，メッセージに含まれる攻撃コードなどを正確に認識する．そして，通信における一連の送受信メッセージから認証などの状態の変化を把握する．そうすることにより，従来の単純なパターンマッチによる検知方法と比べて，より精度の高い検知ができることが期待される．

これより以下，2 章では従来の NIDS で発生する false-positive について述べ，3 章では本論文で提案する NIDS について述べ，4 章ではその実装について述べ，5 章では性能を検証するために行なった実験について述べ，5 章では NIDS の関連研究について述べ，6 章で本論文のまとめと今後の取り組むべき課題について述べる．

# 2 従来の NIDS の誤検知

## 2.1 シグネチャ・マッチング方式

NIDS は，ネットワーク上を流れるメッセージから不正な攻撃を検出し，ユーザに対して警告を発するセキュリティ・システムである．一般的な NIDS は，攻撃メッセージを特徴付けるシグネチャと呼ばれるバイト列を保持しており，そのシグネチャと送信されてきたメッセージとを比較し，合致したメッセージを不正な攻撃と見なして検出を行なう．不正な攻撃に特有なバイト列をシグネチャと呼び，この手法をシグネチャ・マッチング方式という．

シグネチャ・マッチング方式の NIDS の例として Snort[3] がある．例えば Snort には，FTP を利用した root のホーム・ディレクトリへのアクセス [4] を検出をするシグネチャが組み込まれている．FTP ポートに対する送受信メッセージに “CWD ~root” が含まれている場合，Snort がこれを検知してユーザに警告を発するようになっている．Snort では次のシグネチャが登録されている．

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
(msg:"FTP CWD ~root attempt";
flow:to_server,established; content:"CWD";
nocase; content:"~root"; distance:1; nocase;
pcre:"/^CWD\s+~root/smi";
reference:arachnids,318;
reference:cve,1999-0082;
classtype:bad-unknown; sid:336; rev:10;)
```

## 2.2 False-positive の要因

NIDS には誤検知が存在する．特に不正ではないアクセスに対して警告してしまうことを false-positive という．false-positive が頻繁に発生したときに最も重大な問題となるのは，ログファイルに大量の誤検知が出力されてしまうことである．これによって NIDS の管理者が真に重要な警告を見逃してしまう恐れがある．この問題を解決するには NIDS の false-positive を減少させる必要がある．

false-positive の主な原因は，シグネチャが不正なパケットを厳密に定義できていないことによる場合がある．その例を以下に示す．まず，false-positive が発生する脆弱性の例として，qpopper 3.0 サーバにはバッファ溢れの脆弱性 [2] がある．この脆弱性は，メッセージの一覧を得る LIST メッセージの引数に攻撃コードを含む長い文字列を与えると，他のユーザのメールが閲覧できるという脆弱性である．この攻撃を検知するため，Snort には次のシグネチャが登録されている．

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 110
(msg:"POP3 LIST overflow attempt";
flow:to_server,established; content:"LIST";
nocase; isdataat:10,relative;
pcre:"/^LIST\s[^\n]{10}/smi";
reference:bugtraq,948;
reference:cve,2000-0096;
reference:nessus,10197;
classtype:attempted-admin; sid:1937; rev:7;)
```

このシグネチャは，LIST というバイト列の後に 10 バイト以上の引数がある場合，警告を発することを意味する．この脆弱性を利用した不正攻撃は，認証手続きを踏んだ正規のユーザしか行うことはできない．なぜなら，LIST メッセージは POP3 における認証手続き終了後にのみ受け付けられるメッセージであり，認証手続きを経ることなく LIST メッセージを送信しても拒否されるからである．

しかし，Snort のシグネチャでは，すでに認証済みか否かにはかかわらず，LIST という文字列の後に 10 バイト以上の引数が続けば，警告を発する．そのため，正規の認証を経ずに LIST メッセージが送信された場合でも，LIST メッセージのバッファ溢れ脆弱性が攻撃されたかのような警告を発してしまう．

POP3 の LIST コマンドの脆弱性に対する誤検知の原因は，POP3 で定められたプロトコルの状態の変化を考慮していなかった点にある．POP3 のようなユーザ認証を行ってから目的の処理を行うようなプロトコルは，認証の前後でプロトコルの状態が変化するが，検知ではそのことが反映されていない．

また，もう 1 つの例として，HTTP を利用した親ディレクトリへの不正なアクセスがある．これは，リクエストされた URI に ../ を含めることによって，リンクの貼られていないディレクトリにアクセスするものである．クライアントから受け取った URI が正しいものかチェックする機構がない一部のウェブ・アプリケーションでこのような問題がある．これを利用することで，システム上の機密情報が漏洩してしまうことがある．この攻撃を検知するために Snort では，以下のようなシグネチャが用意されている．

```
alert tcp $EXTERNAL_NET any ->
$HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC http directory traversal";
flow:to_server,established; content:"../";
reference:arachnids,297;
classtype:attempted-recon; sid:1113; rev:5;)
```

Snort ではこの不正なアクセスに対して，HTTP ポートに送信されたメッセージの中に ../ が含まれていた場合に警告を発するようなシグネチャを備えている．しかし，Snort のシグネチャでは，リクエストされている URI に ../ が含まれているか否かにはかかわらず，メッセージ中のどこかに ../ が含まれていれば警告を発する．そのため，偶然に ../ というバイト列を含むメッセージが送信されたときにも全て検知してしまうことになる．

この false-positive の原因は，HTTP の文法を考慮しなかった点にある．Snort でもリクエスト URI などに含まれる不正な攻撃コードを検知することはできるが，プロトコル定義に基づいて送受信メッセージを記述するための汎用的な方法を提供してはいない．送受信メッセージにはプロトコルで定められた文法があるが，検知ではそのことが反映されていない．

# 3 アプリケーション層プロトコルの文脈を考慮したNIDS

## 3.1 文脈を考慮したNIDS

アプリケーション層プロトコルにおける不正な攻撃メッセージの多くは，RFCなどによって定められたプロトコル既約に則して行なわれることが多い．アプリケーション層プロトコルでは，送受信メッセージの文法が定められており，サーバとクライアントの間でやりとりされるメッセージによって状態が変化する．本論文では，定められた文法に則して行なわれる通信の状態遷移を**文脈**と呼び，不正なメッセージを検出するためにアプリケーション層プロトコルの文脈を考慮したNIDSを提案する．前述のPOP3の例では，プロトコルの状態の変化を考慮していなかったがためにfalse-positiveが発生してしまっていたが，本論文の検知方法では，認証後にだけLISTコマンドを用いたオーバーフロー攻撃を検知するためfalse-positiveは減少する．またHTTPの例では，プロトコルで定められた文法を考慮していなかったがために false-positive が発生していたが，本論文の検知方法では，送受信メッセージの中にURIの一部として . . / が表われた時にだけ検知するようにするためfalse-positiveは減少する．

本論文で提案する検出方法では，送受信メッセージをプロトコルで定められた文法に従って解釈し，リクエストされているURIなどを正確に認識する．そして，サーバ・クライアント間でやりとりされる一連のメッセージから状態の変化を把握する．そうすることにより，従来の単純なパターンマッチによる検知方法と比べて，より精度の高い検知ができることが期待される．

## 3.2 文脈を取り入れたシグネチャ

### 3.2.1 シグネチャの定義

本論文のNIDSでは，アプリケーション層プロトコルに従って送受信メッセージを解釈するため，監視対象となるアプリケーション層プロトコルの**振舞い**を定義しなければならない．振舞いとは，プロトコル既約によって定義された文法と状態遷移のことをいう．本論文のプロトコルの振舞いの定義として，1) メッセージ・フォーマット，2) 時系列に沿ったメッセージのやりとり，3) システムの状態遷移，を定めることとする．これを**プロトコル定義**と呼ぶ．

NIDSがプロトコル定義に従って送受信メッセージを解釈するためには，プロトコルの振舞いを正しく記述できなければならない．そこで本研究で提案するNIDSでは，プロトコル定義を記述するためのシグネチャ記述言語を設計した．

シグネチャ記述言語では，アプリケーション層プロトコルをオートマトンの形で表わし，その振舞いを定義する．オートマトンの各状態には，どのようなメッセージが送受信されるかを記述し，さらにそのメッセージが送受信されたときの遷移先の状態を記述する．状態遷移のトリガとなるメッセージの定義には正規表現を用いる．このオートマトンを**プロトコル・オートマトン**と呼ぶことにする．オートマトンと正規表現を用いるとPOP3，SMTP，HTTP，FTPなどの一般に使われているプロトコルを記述できることはTCPストリーム・フィルタリング[5]やApril[6]といった他の研究によって既に確認されている．TCPストリーム・フィルタリングは，プロトコル既約に反するメッセージをフィルタリングする機構であり，Aprilはプロトコル定義から通信部分のソースコードを自動で生成する機構である．

図1は，HTTP/1.0でウェブ・ページを取得するときにやりとりされるメッセージの状態遷移を表わしたプロトコル・オートマトンである．HTTP/1.0ではまず，サーバとクライアントの間で接続が確立すると，クライアントからリクエスト・メッセージが送信される．リクエスト・メッセージはリクエスト・ラインなどから成る．リクエスト・メッセージが送信された後，サーバからレスポンス・メッセージが送信される．レスポンス・メッセージはステータス・ライン，複数のヘッダー・フィールド，ウェブ・ページなどのメッセージ・ボディから成る．こののようにして，プロトコルをオートマトンの形で表わすこととする．

シグネチャ記述言語ではまず，各状態の定義を行なう．状態は`state`によって状態名を宣言し，{}内でその状態における処理を記述する．{}内では，送受信されるメッセージによって次にどの状態に遷移するのかを記述する．例えば，< *message* >をサーバが受信することで次の状態< *state* >に遷移する場合は，<: < *message* > -> < *state* >と記述する．このとき，< *message* >は正規表現を用いる．<: はサーバへの送信を表わし，:> はクライアントへの送信を表わしている．メッセージによって遷移

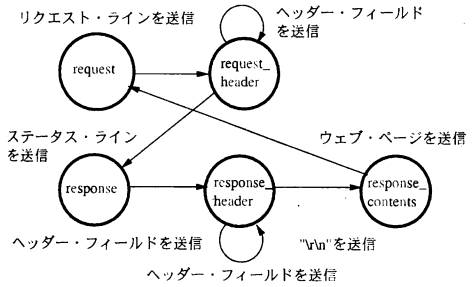


図 1: HTTP のオートマトン

先の分岐がある場合は，| を用いて表わす．また，正規表現によってマッチした文字列は変数に代入することができ，その場合，< *message* >:$< *var* > と記述する．正規表現のパターンは `regexpr` によって名前を付けることもできる．以上にようにしてプロトコル・オートマトンの記述できる．

さらに本論文の NIDS では，以上のプロトコル・オートマトンの記述中に検知すべき攻撃メッセージを記述する．具体的には，変数に代入された文字列が不正な攻撃コードを含んでいる場合には警告を出すように指示する．対象となる変数 $< *var* > を `match` $< *var* > `with` によって宣言し，その変数の中に 不正な攻撃コード < *code* > があるときに $< *var* >: < *message* > と記述することで警告 < *message* > を出す．

### 3.2.2 シグネチャの定義例

図 2 は HTTP/1.0 の定義を記述したものである．1 行目でプロトコル名とポート番号を定義している．3, 4 行目では正規表現のパターンを定義している．6 行目以降はメッセージの送受信の流れを定義している．6 ～ 11 行目では状態 req によって，クライアントから送信されるリクエスト・メッセージを定義している．7 行目では送受信メッセージが ``GET ([a-zA-Z\./]+) HTTP/'' ver ``\.]*'' crlf にマッチした場合は状態 req_header に遷移することを表わしている．また () で囲まれた正規表現にマッチした文字列は変数に代入することができ，7 行目では () で囲まれた [a-zA-Z\./]+ を $uri に代入している．

またこのプロトコル・オートマトンの記述には NIDS の検知情報も記述されている．記述してあるのは，前述の HTTP を利用した親ディレクトリへの不正なアクセスである．8, 9 行目で，変数 $uri の中に代入された URI 文字列が ../ であるかを検査する．もしも ../ であった場合は，directory traversal というメッセージでユーザに警告を発する．このように記述することで，アプリケーション層プロトコルの文脈を考慮した検知を定義することができる．

```
1  protocol http(80);
2
3  regexpr ver = ``HTTP/[0-9\.]*'';
4  regexpr crlf = ``\r\n''
5
6  state req {
7  <: ``GET ([a-zA-Z\./]+)'' ver crlf:$uri {
8    match $uri with
9    ``\.\./'': ``directory traversal'';
10   } -> req_header;
11 }
12
13 state req_header {
14 <: ``Content-Length: ([0-9]+)'' crlf:$len
15    -> res_header;
16 | ``[a-zA-Z0-9:- ]+\r\n'' -> resp_header;
17 | crlf -> resp;
18 }
19
20 state resp {
21 :> ``HTTP/'' ver [a-zA-Z]+'' crlf
22    -> resp_header;
23 }
24
25 state resp_header {
26 :> ``[a-zA-Z0-9:-]+'' crlf -> resp_header;
27 | crlf -> resp_contents
28 }
29
30 state resp_contents {
31 :> ".*"[== $len] -> request;
32 }
```

図 2: シグネチャの記述例

## 3.3 マッチング・エンジン

本論文の NIDS では，マッチング・エンジンと呼ばれる部分においてプロトコル定義に基づいた攻撃メッセージの検知を行なう．プロトコル定義を記述したシグネチャは一度，バイトコードに変換され，その後 NIDS によって読み込まれる．読み込まれたバイトコードは，NIDS の中でオートマトンを用いた内部表現で表わされる．各状態ではマッチングすべきメッセージのパターンを持ち，次に遷移する状態が連結されている．マッチング・エンジンでは，送受信メッセージと現在の状態に登録されているパ

ターンとを順次マッチングしていき，次に遷移する状態を決定する．また，状態遷移の中で不正な攻撃コードの検知をした場合にはユーザに警告を発する．

# 4 実装

本論文のNIDSは現在，Linux上で実装されている．本研究のNIDSは，プロトコル定義を記述するためのシグネチャ記述言語と，アプリケーションの送受信メッセージを横取りする通信関数ラッパと，プロトコル定義を解釈しながら送受信メッセージの監視を行なうマッチング・エンジンから構成されている．図3にシステムの構成を示す．

本論文のNIDSはまず，アプリケーションの呼び出す通信関数をフックすることで送受信メッセージを横取りする．そして，得られたメッセージはマッチング・エンジンに渡さる．そこではシグネチャに表わされたプロトコル定義と照らし合わせて通信の状態を解釈し，メッセージの検査が行なわれる．不正なメッセージがあった場合には，ユーザに対して警告を発する．

NIDSのシグネチャは本来，NIDSの外部から取り込まれるものであるが，今回の実装ではシグネチャはプログラム中に含まれているものとして実装を行った．

## 4.1 通信ライブラリのフック

通信関数ラッパは，アプリケーションの呼び出す通信ライブラリをフックすることで送受信メッセージを横取りする．通信ライブラリ呼び出しのフックは共有ライブラリのプリロード機構を用いて実現した．多くのUNIXシステムでは，共有ライブラリのプリロードは環境変数LD_PRELOADを用いて実現されている．LD_PRELOADに設定された共有ライブラリ内の関数は，標準ライブラリのものよりも先に呼び出されるため，独自の機能を追加することができる．そのため，本論文のNIDSは，既存のサーバ・アプリケーション，通信ライブラリ，OSを改変する必要がない．

現在までに実装したNIDSでは，ソケット通信に関する12種類の関数をフックしている．監視対象となる記述子を特定するために socket(), bind(), accept() などの関数をフックしている．また，送受信メッセージの検査を行なうために，送受信のための関数 recv(), send() などをフックしている．

## 4.2 コネクション管理

本論文のNIDSは通信用関数をフックしている．プログラムではコネクション毎の通信を管理するために，通信関数が生成するソケット用ファイル・ディスクリプタを監視している．

NIDSはソケットによって行われる通信がサーバとクライアントのどちらに対して行われるものなのかを知る必要がある．本研究のNIDSでは通信の向きを知るために，呼ばれた通信関数の種類に判別している．サーバは socket(), bind(), listen(), accept() という一連のシステム・コールによってクライアントからのアクセスを受けつける．またクライアントは socket(), connect() によってサーバにアクセスする．このシステム・コールの呼び出し順に注目し，生成されるソケット用ファイル・ディスクリプタの通信の向きを判別するようにし，そのファイル・ディスクリプタによって送受信されるメッセージがサーバからクライアントへのものか，クライアントからサーバへのものかを判別するようにした．

また，プロトコル定義に基づく通信の状態はコネクション毎に管理されている．通信の状態の情報には，通信の向き，プロトコル・オートマトンにおける現在の状態，現在の正規表現のマッチング情報などが含まれている．

# 5 実験

本研究で実装したNIDSによるオーバーヘッドを計測するため，NIDSを組み込んだウェブ・サーバとNIDSを組み込んでいない通常のウェブ・サーバとの性能比較を行った．

サーバとクライアントのスペックを表1に示す．実験ではサーバとクライアントのコンピュータを用意し，それぞれを1000Base-Tのスイッチを経由して接続した．サーバにはApache (ver.2.0.52) を用い，クライアントにはApacheに付属のベンチマーク・プログラムであるApacheBench (ver.2.0.41) を用いた．Apacheの設定はすべてデフォルトで，HTTPにおけるKeepAliveを有効にして計測を行なった．

同時接続崇を1から10まで変動させ，8 Kbyteの

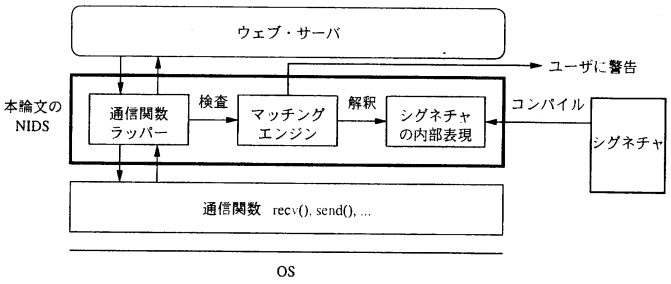


図 3: システム構成

ファイルを 100 回取得するときのスループットと，1 リクエストあたりの通信時間を測定した．実験は 10 回行なった平均値をとった．NIDS は HTTP1.0 の GET メソッドのルールを適用し，そのルールの中には 5 つの脆弱性に対応した検知情報を加えた．その脆弱性を表 2 に示す．

スループットと通信時間による性能評価の実験結果を図 4 と図 5 に示す．スループットは接続数が 1 のときに約 9% 低下し，接続数が 2 以上のときは最大で約 34% 低下してほぼ一定の値となった．通信時間は接続数が 1 のときに約 9% 増大し，最大で約 50% の割合いで増大している．同時接続数が 1 の場合は，同時に処理する正規表現のマッチングの処理が少ないため，少しだけ性能低下が見られる．同時接続数が増加するに従って，同時に処理する正規表現のマッチング処理が増加するため，通信の遅延が増大している．

NIDS は内部で正規表現のマッチングを行っている．NIDS のマッチング・エンジンにおいて，一部のマッチングを行なわずに動作させたところ，性能低下が大きく減少した．このことから，NIDS による性能低下の原因は正規表現によるマッチングであると考えている．本研究で実装した NIDS は，GNU の標準ライブラリに含まれている正規表現ライブラリを使用している．今後はこれをアプリケーション層プロトコルの振舞いを定義するために作成された正規表現ライブラリに置き換えることで，より高速に動作するものと考えている．

表 1: サーバとクライアントのスペック

| | サーバ | クライアント |
|---|---|---|
| OS | Linux ver.2.6.9 | Linux ver.2.6.9 |
| CPU | Pentium 4 3GHz | Pentium 4 2.40GHz |
| 主記憶 | 512 Mbyte | 1Gbyte |

表 2: NIDS に組み込んだ脆弱性

| 検知内容 |
|---|
| URI に「/site/eg/source.asp」を含む. |
| URI に「conf/httpd.conf」を含む. |
| URI に「?M=D」を含む. |
| URI に「/server-info」を含む. |
| URI に「/server-status」を含む. |

## 6 関連研究

NIDS の関連研究として，Snort, Bro[7], Shield[8] などがある．Snort は軽量かつ設置が容易な NIDS で，シグネチャ・マッチング方式の検知方法を採用している．独自のルール記述言語によってシグネチャを記述し，不正な送受信メッセージを定義するために正規表現を用いている．しかし Snort では，本論文で提案するプロトロルの状態遷移を考慮した検知が行なえない．

Bro もまた独自の記述言語によってシグネチャを記述し，正規表現によるマッチングを行なう NIDS である．Bro は不正なバイト列を探すだけでなく，プロトコルを考慮に入れたコネクションの解析ができる．しかし Bro のプロトコル解析部は，アプリケーション層プロトコルごとに C 言語を用いて作成しなければならない．これでは，数多く存在するアプリケーション層プロトコルに対応するために，それぞれ別の解析プログラムを用意しなければならない．本論文の NIDS では，プロトコルごとの定義をシグネチャ記述言語によって定義するため，プロト

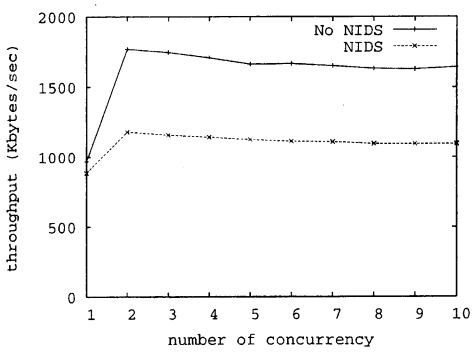


図 4: スループットによる性能比較

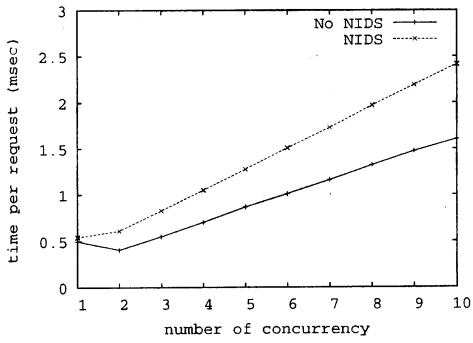


図 5: 通信時間による性能比較

コルごとに解析プログラムを用意する必要はない．

Shield は攻撃対象となるサーバの脆弱性を認識し，この脆弱性を突く攻撃を排除するフィルタを記述する．フィルタの記述には専用言語を使用し，プロトコルの状態遷移をトレースしながら攻撃パターンを検出する．Shild では脆弱性ごとにシグネチャを定義するが，本研究では汎用的なプロトコル記述にもとづいたシグネチャの定義を行なうため，本論文とは異なる．

## 7 まとめと今後の課題

既存の NIDS には多くの誤検知が発生する．その理由の 1 つとして，シグネチャが不正なパケットを厳密に定義できていないことが挙げられる．従来のシグネチャは単純なバイト列によって表されているため，正当なパケットでも合致することがあり，このことが false-positive を増加させてしまうのである．

本論文では，NIDS の誤検知を減少させるために，アプリケーション層プロトコルの状態を考慮する NIDS を提案した．NIDS がサーバとクライアントとの送受信メッセージを読み取り，アプリケーション層プロトコルの定義に従って通信内容を解釈することで，現在行なわれている通信の状態を追跡する．NIDS は通信の状態を取り入れることで検知精度の向上が期待される．

今後の課題としては，検知精度が実際にどれほど向上するのかを検証する必要がある．実際の通信内容を本論文の NIDS と既存の NIDS に再送し，誤検知発生率の比較を行なう予定である．

## 参考文献

[1] Tim, K.: Strategies to Reduce False Positives and False Negatives in NIDS, http://securityforcus.com/infocus/1463 (2001).

[2] CVE: CVE-2000-0096 (2000).

[3] Roesch, M.: Snort - Lightweight Intrusion Detection for Networks, *Proc. 13th Systems Administration Conference (LISA)*, USENIX Association, pp. 229–238 (1999).

[4] CVE: CVE-1999-0082 (1999).

[5] 河野健二, 品川高廣, ラハト・カビル: TCP ストリームに対するフィルタリングによるインターネット・サーバの安全性向上, 情報処理学会論文誌: コンピューティングシステム, Vol. 46, No. SIG4(ACS9), pp. 33–44 (2005).

[6] 河野健二: アプリケーション層プロトコルの実現を容易にするフレームワーク, 情報処理学会論文誌: プログラミング, Vol. 44, No. SIG2(PRO16), pp. 25–35 (2003).

[7] Paxson, V.: Bro: A System for Detection Network Intruder in Real-Time, *Computer Networks*, Vol. 31, No. 23-24, pp. 2435–2463 (1999).

[8] Wang, H. J., Guo, C., Simon, D. R. and Zugenmaier, A.: Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits, *ACM SIGCOMM '04* (2004).