Snort ユーザズマニュアル
Snort Release: 2.0.2

Martin Roesch
Chris Green

平成 15 年 12 月 16 日

日本語版翻訳提供: 三井物産株式会社　Sourcefire プロジェクト
日本語版監訳: 白畑 真, 後藤 久

目 次

第1章　Snort概説

本マニュアルは Martin Roesch によって執筆された *Writing Snort Rules* に基づいていますが、現在は Chris Green <cmg@snort.org> によって管理されています。マニュアルの更新や翻訳版ドキュメンテーションへのリンクにつきましては、Chris Green 宛にお寄せください。つまり、よりより良い表現方法をお持ちの場合や、何かマニュアルの内容が古い箇所を発見した場合は、私宛に (訳注:英語で) ご一報ください。

現在、本マニュアルは LaTeX 形式で `doc/snortman.tex` ファイルにありますので、ご希望ならば本マニュアル向けのパッチを投稿できます。ごく小規模なマニュアルの更新でも Snort プロジェクトへの貢献となりますので、ご協力いただければ幸いです。

1.1　さあ始めよう

Snort を使うことは実際、大して難しくはありません。しかし、多くのコマンドラインオプションが用意されており、どれとどれを組み合わせればいいのかが一目瞭然とは限りません。このファイルが初めて Snort を利用するユーザの一助となれば幸いです。

先に進む前に、Snort に関して理解していただきたい基本的概念がいくつかあります。Snort には 3 種類の主要なモードがあります: すなわちスニッファー、パケットロガー、ネットワーク不正侵入検知システムです。スニッファーモードではネットワークを流れるパケットを単に読み込み、入ってくるパケットをコンソール上で次々と表示します。パケットロガーモードではそれらのパケットをディスクに記録します。ネットワーク不正侵入検知モードは、最も複雑かつ設定の変更が可能です。Snort はネットワーク上のトラフィックが、ユーザが定義したルールセットにマッチするかを解析し、その解析結果に基づいて何らかのアクションを実行します。

1.2　スニッファーモード

まずは、基本的な動作から始めましょう。単に TCP/IP パケットのヘッダを画面に出力したい場合は (すなわち、スニッファーモード)、次のようにしてください:

```
./snort -v
```

このコマンドは Snort を実行し、IP および TCP/UDP/ICMP ヘッダのみを表示するだけです。転送中のアプリケーション (層の) データを表示したい場合には、次のようにしてください:

```
./snort -vd
```

この方法では、Snort にヘッダに加えてパケットのデータも表示します。データリンク層のヘッダを含むよう、さらに詳細な表示を行う場合には次のようにしてください:

```
./snort -vde
```

(余談ですが、これらのスイッチは一緒することも、ばらばら、あるいは組み合わせて指定することもできます。先述したコマンドは次のようにも入力できます:

```
./snort -d -v -e
```

が、動作に変わりはありません。)

1.3 パケットロガーモード

さて、これらのコマンドはどれも優れたものですが、パケットをディスクに記録する場合は、ログ保存用ディレクトリを指定する必要があります。そうすれば Snort が自動的にパケットロガーモードに移行します。

```
./snort -dev -l ./log
```

もちろん、この指定はカレントディレクトリに log という名前のディレクトリが作成されていることが前提となります。もしそうでなければ、Snort はエラーメッセージを表示して終了します。Snort がこのモードで動作する場合、認識したパケットをすべて収集し、データグラムに含まれるホスト毎の IP アドレスに基づいたディレクトリ階層に保存します。

単純な-l スイッチだけを指定する場合、Snort がパケットを保存するディレクトリとしてリモートコンピュータのアドレスを利用する場合もあれば、ローカルホストアドレスを利用する場合もあることにお気付きかもしれません。ホームネットワークに関連するログを取るためには、どのネットワークがホームネットワークであるのかを Snort に指定する必要があります。

```
./snort -dev -l ./log -h 192.168.1.0/24
```

このルールは Snort に対し、./log ディレクトリにアプリケーションデータと同様、データリンクと TCP/IP ヘッダを出力して、192.168.1.0 の Class C ネットワークに関連したパケットを記録したいという指定を行います。

入ってくるパケットはすべて、log ディレクトリの下のリモート (192.168.1 以外の) ホストアドレスに基づいた名前のサブディレクトリに保存されます。注意すべき点として、両ホストがホームネットワーク上にある場合は、それぞれのポート番号のうち大きい方に基づいて記録されます。また、ポート番号が同じ場合は送信元アドレスに基づいて記録されます。

高速ネットワーク上にいる場合、あるいは後で解析するためによりコンパクトな形式でパケットを記録したい場合は、バイナリモードでログを取ることを検討してください。バイナリモードではパケットを tcpdump 形式でログ保存用ディレクトリ内の単一バイナリ・ファイルに記録します。

```
./snort -l ./log -b
```

ここで、コマンドラインが変化していることに注目してください。バイナリモードでは全てを単一ファイルに記録します。その結果、出力ディレクトリの構造を指定するためのホームネットワークを指定する必要はなくなります。さらに、バイナリモードではパケットの一部だけでなく、パケット全体が記録されるため、冗長モードで実行する必要もなければ、-d または-e スイッチを指定する必要もありません。Snort をロガーモードで動作させるために必要なことは、-l スイッチによるログ保存用ディレクトリの指定と、デフォルトの ASCII 形式ではなく、バイナリ形式でログを取得する指定を行う-b スイッチだけです。

いったんパケットがバイナリファイルに記録されれば、tcpdump や Ethereal のような tcpdump バイナリ形式をサポートする任意のスニッファープログラムを利用し、そのファイルからパケットを読み込むことができます。Snort でも-r スイッチを利用してプレイバックモードにし、パケットを読み込むこともできます。Snort は tcpdump 形式のファイルを任意の動作モードで処理できます。たとえば、バイナリログファイルを Snort で読み込み、スニッファーモードでパケットを画面にダンプするには、次のようにします:

```
./snort -dv -r packet.log
```

ファイル内のデータは、コマンドラインから利用可能な BPF インタフェースの他に、Snort のパケットロギングおよび不正侵入検知モードなど様々な方法で処理できます。たとえば、ログファイルからの ICMP パケットのみを確認したいのであれば、単にコマンドラインで BPF フィルターを指定すれば、Snort はファイル内の ICMP パケットだけを確認できます。

```
./snort -dvr packet.log icmp
```

BPF インタフェースの詳しい利用方法については、snort および tcpdump の man ページをご参照ください。

1.4 ネットワーク不正侵入検知モード

ネットワーク不正侵入検知 (NIDS) モードを有効にするには (これによって、送信されるパケットを全部記録しなくて済みます)、次のようにします:

```
./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf
```

ここでは snort.conf というのはルールファイルの名前です。snort.conf ファイルで設定されたルールを各パケットに適用して、ファイル内のルールタイプに基づいて動作を決定します。プログラム用の出力ディレクトリが特に指定されない場合、標準の出力先ディレクトリとして/var/log/snort が設定されます。

最後のコマンドラインについて注意すべきことが一点あります。Snort を定常的に IDS として運用する場合は、Snort の性能低下を避けるために、-v スイッチをコマンドラインから取り除くようにしてください。画面はデータ書き込みには遅い場所なので、その間にパケットを取りこぼす可能性があります。

また、大部分のアプリケーションに対してデータリンクヘッダを記録する必要もありませんので、-e スイッチを指定する必要もありません。

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf
```

これで、Snort が最も基本的な NIDS モードで実行されるように設定され、ルールが Snort に伝えるパケットを平易な ASCII 形式でディレクトリ階層構造にログを記録します。

1.4.1 NIDS モード出力オプション

Snort を NIDS モードで動作させる際、出力を設定する方法はたくさんあります。デフォルトのログ取得およびアラートメカニズムは、デコード済み ASCII フォーマットで記録し、フルアラートを利用します。フルアラートメカニズムでは、完全なパケットヘッダの他にアラートメッセージを出力します。2 つのログ取得機能に加えて、他にもいくつかコマンドラインで利用可能なアラート出力モードがあります。

アラートモードはいくぶん複雑です。コマンドラインには 6 つのアラートモードが存在します。すなわち full, fast, socket, syslog, smb(WinPopup), none の 6 つです。これらのモードのうち 4 つはコマンドラインの-A スイッチで指定できます。4 つのオプションは次の通りです:

-A fast 高速アラートモード:タイムスタンプ、アラートメッセージ、送信元および宛先 IP アドレス/ポート番号を含む単純な形式でアラートを出力します。

-A full これはデフォルトのアラートモードです。何も指定しない場合はこのオプションが自動的に利用されます。

-A unsock 別のプログラムがリッスンできるよう、UNIX ソケットにアラートを出力します。

-A none アラートをオフにします。

パケットの記録は標準では ASCII 形式で行われますが、コマンドラインの-b スイッチを用いてバイナリログファイルに記録するよう設定を変更できます。パケットのログ取得をすべて無効にしたい場合は、コマンドラインで-N スイッチを利用します。

設定ファイルを介して利用可能な出力モードについては、第 2.42 節をご参照ください。なお、コマンドラインのロギングオプションは設定ファイルに指定されたあらゆるオプションより優先される点に留意してください。これを活用すれば、コマンドラインから迅速に設定上の問題をデバッグできるようになります。

アラートを syslog に送信するには、“-s” スイッチを利用します。syslog アラートメカニズム向けの標準のファシリティは LOG_AUTHPRIV と LOG_ALERT です。syslog 向けにその他のファシリティを設定したい場合は、rules ファイル内の出力プラグイン・ディレクティブを利用してください。syslog 出力の設定について、詳しくは第 2.5.1 節をご参照ください。

最後に、SMB アラートメカニズムがあります。この機能を利用すると、Snort は Samba に付属している smbclient を呼び出し、Windows マシンに WinPopup アラートメッセージを送信できるようになります。このアラートモードを利用するには、Snort を configure する際に`--enable-smbalerts` スイッチを有効にしてください。

以下は、出力設定例の一部です。

- ログ取得はデフォルト (デコード済み ASCII) 形式で行い、アラートを syslog に送信する:

```
./snort -c snort.conf -l ./log -h 192.168.1.0/24 -s
```

- ログ取得先はデフォルトの/var/log/snort で、アラートを高速アラートファイルに出力する:

```
./snort -c snort.conf -A fast -h 192.168.1.0/24
```

- バイナリファイルにログを取得し、Windows ワークステーションにアラートを送信する:

```
./snort -c snort.conf -b -M WORKSTATIONS
```

1.4.2 高性能設定

Snort を (たとえば、100Mbps 以上の帯域に対応するほどに) **高速化**したい場合、-b と-A fast スイッチまたは-s(syslog) オプションを利用します。これによって、パケットのログを tcpdump 形式で取得し、アラート生成は最小限となります。例:

```
./snort -b -A fast -c snort.conf
```

このような設定において、Snort は 100Mbps LAN 上でも最大約 80Mbps で、同時に複数の探索および攻撃のログを取得できます。この設定では、ログはバイナリ形式の tcpdump 形式ファイルである snort.log ファイルに書き込まれます。このファイルを読み直し、そのデータをおなじみの Snort 形式に変換するには、-r オプションと普段利用するその他のオプションを併用し、データファイルに対して Snort を再度実行するだけです。

例:

```
./snort -d -c snort.conf -l ./log -h 192.168.1.0/24 -r snort.log
```

いったんこれを行えば、データはすべてデコードされた標準形式でログ・ディレクトリに出力されます。すごいと思いませんか？

1.4.3 アラート順序の変更

Snort がパケットにルールを適用するデフォルトの順序を好きになれない利用者もいるはずです。デフォルトのルールの適用順序は、最初に Alert ルール、次に Pass ルール、最後に Log ルールという順序です。この順序はいくぶん分かりにかもしれませんが、これはユーザが百種類の alert ルールを書いた後、わずか 1 つの誤った pass ルールでそれらをすべて無効してしまうようなことを防止する安全設計なのです。ルールタイプについて詳しくは、第 2.2.1 節をご参照ください。

上級ユーザ向けに、デフォルトのルール適用順序を Pass ルール、Alert ルール、Log ルールの順序に変更するために-o オプションが提供されています。

```
./snort -d -h 192.168.1.0/24 -l ./log -c snort.conf -o
```

1.5 その他

デーモンモードで Snort を実行したい場合は、上記のどんな組み合わせに対してでも-D スイッチを追加できます。ただ注意して頂きたいのは、デーモンに SIGHUP シグナルを送って Snort をリスタートできるようにしたい場合、Snort を起動する際にバイナリへの絶対パスを利用しなければならない点です。例:

```
/usr/local/bin/snort -d -h 192.168.1.0/24 -l \
    /var/log/snortlogs -c /usr/local/etc/snort.conf -s -D
```

セキュリティ上の問題から、相対パスはサポートされていません。

公共のメーリングリストにパケットログを投稿する場合は、-O スイッチを試すとよいでしょう。このスイッチは、パケットの出力に含まれる IP アドレスを隠蔽化します。これは、メーリングリスト上の人々にパケットに含まれる IP アドレスを知られたくない場合に便利です。また、もっぱらホームネットワーク上のホストの IP アドレスを隠蔽化したいのであれば、-O スイッチと-h スイッチを併用してください。これは、攻撃元ホストのアドレスは誰に見られてもかまわないという場合に効果的です。

例:

```
./snort -d -v -r snort.log -O -h 192.168.1.0/24
```

これはログファイルからパケットを読み込み、192.168.1.0/24 Class C ネットワークからのアドレスのみを隠蔽化した状態で、画面にそのパケットをダンプします。

1.6 詳細情報

第 2 章には、設定・ファイルに用意された多数の設定オプションに関する情報がふんだんに盛り込まれています。snort manual ページと

```
snort -?
```

の出力には、様々なモードで Snort を実行させるために役立つ情報が盛り込まれています。なお、多くのシェルでは、?をエスケープするために\?が必要ですのでご注意ください。

Snort Web ページ (`http://www.snort.org`) と Snort ユーザズメーリングリスト (`http://marc.theaimsgroup.com/?l=snort-users`, `snort-users@lists.sourceforge.net`) は、有益なアナウンスのほか、議論の場とサポートを提供する場を提供します。Snort に関する数多くの情報がありますので、飲み物でも飲みながらくつろいでドキュメンテーションやメーリングリスト・アーカイブをお読みください。

第2章　Snortルールを書く
正しいSnortルールの記述方法

2.1　基本

Snort は柔軟性に富み、強力であるにも関わらず、シンプルで軽量なルール記述言語を採用しています。Snort ルールを記述する際に覚えておくと役立つ多くの簡単なガイドラインがあります。

ほとんどの Snort のルールは 1 行で記述されています。1.8 より前のバージョンでは、この方法でしか記述できませんでした。Snort の現行バージョンでは、行の終わりにバックスラッシュ\を追加することで、複数行にわたるルールを記述できます。

Snort のルールは、ルールヘッダ部とルールオプション部という 2 つの論理セクションに分かれています。ルールヘッダ部にはルールのアクション、プロトコル、送信元/ 宛先 IP アドレスおよびネットマスク、送信元/宛先ポート情報が含まれます。ルールオプション部にはアラートメッセージと、ルールに記述されたアクションの実施を判断するために検査するパケットの一部に関する情報が含まれます。

図 2.1 は Snort ルールのサンプルです。

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg:"mountd access";)
```

図 2.1: Snort ルールのサンプル

1 つ目の括弧までの文字列がルールヘッダ部で、括弧で囲まれた部分がルールオプション部です。ルールオプション部の中でコロンの前の単語はオプションキーワードと呼ばれます。ただし、ルールオプション部は必須ではなく、特定のパケットの収集、アラート出力 (または遮断) の際に厳密な定義を行うために利用されます。

ルール全体を構成する要素は、論理積 (AND) 命令文を構成すると見なされます。同時に、Snort ルールファイル群に含まれる様々なルールは大規模な論理和 (OR) 命令文を構成するものと見なされます。

2.1.1　Include 文

include キーワードを利用することで、Snort のコマンドラインで指定されたルールファイルから他のルールファイルを読み込むことができます。この機能は C 言語の#include と同様に機能し、include が含まれるファイルの当該箇所に指定されたファイルの内容を読み込みます。

書式

```
include: <include file path/name>
```

この行末にはセミコロンがないことに注意してください。読み込まれるファイルは事前に定義された変数を独自の変数リファレンスに代入します。Snort ルールファイルにおける変数の定義と利用方法について、詳しくは第 2.1.2 節をご参照ください。

2.1.2 変数

Snort では変数を定義できます。図 2.2 に示したように、var キーワードを用いて簡単に代入を行うことができます。

書式

```
var: <name> <value>
```

```
var MY_NET [192.168.1.0/24,10.1.1.0/24]
alert tcp any any -> $MY_NET any (flags:S; msg:"SYN packet";)
```

図 2.2: 変数の定義および利用例

ルール変数名は様々な方法で修飾できます。$演算子を利用してメタ変数を定義し、変数修飾演算子?および-と共に利用できます。* $var - メタ変数を定義します。* $(var)- 変数 var の内容で代入します。* $(var:-default)-変数 var の内容、または var が未定義の場合は default の値を代入します。* $(var:?message) -変数 var の内容を代入するか、またはエラーメッセージ message を出力して終了します。

これらのルール修飾子の例については、図 2.3 をご参照ください。

```
var MY_NET 192.168.1.0/24
log tcp any any -> $MY_NET 23
```

図 2.3: 高度な変数の利用例

2.1.3 設定

Snort の多くの設定およびコマンドラインオプションは設定ファイルで指定することができます。

書式

```
config <directive> [: <value>]
```

ディレクティブ

order ルールをパスする順序を変更します (snort -o)

alertfile アラート出力ファイルを指定します。例:config alertfile: alerts

classification ルールの分類 (classifications) を行います (表 2.2 を参照)

decode_arp arp デコードを行います (snort -a)

dump_chars_only キャラクターダンプを行います (snort -C)

dump_payload アプリケーション層の情報をダンプします (snort -d)

decode_data_link Layer2 ヘッダをデコードします (snort -e)

bpf_file BPF フィルタを指定します (snort -F). 例: config bpf_file: filename.bpf

set_gid GID を変更します (snort -g). 例: config set_gid: snort_group

daemon デーモンとして fork します (snort -D)

reference_net ホームネットワークを指定します (snort -h). 例: config reference_net: 192.168.1.0/24

interface ネットワークインタフェースを指定します (snort -i). 例: config interface: xl0

alert_with_interface_name アラートにインタフェース名を付加します (snort -I)

logdir ログ出力ディレクトリを指定します (snort -l). 例: config logdir: /var/log/snort

umask 実行時に umask を設定します (snort -m). 例: config umask: 022

pkt_count N パケット後に終了します (snort -n). 例: config pkt_count: 13

nolog ログ取得を行いません。注: アラートは出力されます (snort -N)

obfuscate IP アドレスを隠蔽化します (snort -O)

no_promisc promiscuous モードを利用しません (snort -p)

quiet バナーおよびステータスレポートを無効にします (snort -q)

checksum_mode チェックサムを計算するパケットのタイプ。 値: none, noip, notcp, noicmp, noudp, all

utc タイムスタンプとしてローカル時間の代わりに UTC(協定世界時) を利用します (snort -U)

verbose 標準出力へ冗長なログ出力を行います (snort -v)

dump_payload_verbose 生パケットをデータリンク層からダンプします (snort -X)

show_year タイムスタンプに西暦年を表示します (snort -y)

stateful stream4(たぶん) に対して assurance モードを指定します。表 2.7 も併せて参照して下さい。

min_ttl Snort が処理する最小 TTL を指定します。

disable_decode_alerts Snort のデコード段階で生成されるアラートを無効にします

disable_tcpopt_experimental_alerts 実験的な TCP オプションによって生成されるアラートを無効にします

disable_tcpopt_obsolete_alerts obsolete tcp オプションによって生成されるアラートを無効にします

disable_tcpopt_ttcp_alerts T/TCP オプションによって生成されるアラートを無効にします

disable_tcpopt_alerts (訳注:TCP) オプション長検証アラートを無効にします

disable_ipopt_alerts IP オプション長検証アラートを無効にします

detection 検知エンジンを指定します (例: search-method lowmem)

reference 新しいリファレンスシステムを Snort に追加します

2.2 ルールヘッダ

2.2.1 ルールアクション

ルールヘッダには、パケットが何者なのか、どこから来たか、どんなものか、といったことを定義する情報や、ルールで指定された属性のパケットが姿を現した際に、何を行うかといった定義を行う情報が含まれます。ルール内の最初の項目はルールアクションです。ルールアクションは、ルールの基準に合致するパケットを検知した場合、どのような動作を行うべきかを Snort に伝えます。Snort には alert, log, pass, activate, dynamic の 5 つの有効なデフォルトアクションが用意されています。

1. alert - 指定されたアラート出力方法でアラートを生成し、パケットを記録します
2. log - パケットを記録します
3. pass - パケットを無視します
4. activate - アラートを出力し、別の dynamic ルールを有効にします
5. dynamic - activate ルールによって起動されるまで待機し、log ルールと同様に動作します

さらに独自のルールタイプを定義して、単一または複数の出力プラグインをそれらに関連づけることもできます。また、Snort ルール内のアクションとしてそのルールタイプを利用できます。
次の例では、tcpdump のみに記録するタイプを定義します。

```
ruletype suspicious
{
    type log output
    log_tcpdump: suspicious.log
}
```

この例では、syslog と MySQL データベースに記録するルールタイプを定義します。

```
ruletype redalert
{
      typealert output
      alert_syslog: LOG_AUTH LOG_ALERT
      output database: log, mysql, user=snort dbname=snort host=localhost
}
```

2.2.2 プロトコル

ルール内の次のフィールドはプロトコルです。現在、Snort が不審な挙動に対して解析を行うプロトコルとしては tcp, udp, icmp, ip の 4 つのプロトコルがあります。将来、ARP, IGRP, GRE, OSPF, RIP, IPX といったプロトコルが追加されるでしょう。

2.2.3 IP アドレス

ルールヘッダの次の部分では、指定されたルールの IP アドレスおよびポート番号の情報に対応しています。any キーワードは全アドレスを定義するために利用できます。Snort はルールファイル中の IP アドレスに対するホスト名の問い合わせを行うことはできません。アドレスは、連続した数字から構成される IP アドレス、もし

くは CIDR[3] ブロック形式で指定します。CIDR ブロックはルールの IP アドレスとルールに沿って検査される全パケットのネットマスクを示します。

/24 の CIDR ブロックマスクは Class C ネットワークを意味し、/16 は Class B ネットワークを意味し、/32 は特定のマシンのアドレスを意味します。例えば、192.168.1.0/24 というアドレスと CIDR の組み合わせは、192.168.1.1 から 192.168.1.255 までの連続したアドレス空間を意味しています。宛先アドレスにこの指定を利用する全ルールは、範囲内のどのアドレスに対してもマッチすることになります。CIDR 表記は、わずかな文字数で大きなアドレス空間を指定できる優れた省略表記法といえます。

図 2.1 では、送信元 IP アドレスは全コンピュータに対応し、宛先 IP アドレスは Class C ネットワークの 192.168.1.0 に対応するように設定されています。

IP アドレスに適用可能な演算子として、否定演算子が用意されています。この演算子は Snort に、列記された IP アドレス以外の全 IP アドレスに対応するよう指示します。否定演算子は ! を付けて表記します。たとえば、最初の例に簡単な修正を加えて、図 2.4 に示したように否定演算子を使って、ローカルネットワークの外部を起点とする全トラフィックに対してアラートを発動できます。

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \
     (content: "|00 01 86 a5|"; msg: "external mountd access";)
```

図 2.4: IP アドレス否定ルールの例

このルールでは、内部ネットワーク以外が送信元で、宛先アドレスが内部ネットワークの TCP パケットを意味します。

さらに、IP アドレスのリストを指定することもできます。IP リストはコンマ区切りの IP アドレスと CIDR ブロックを角括弧で囲んで指定します。ただし、現在のところ IP リスト内のアドレスとアドレスの間にスペースを入れることはできません。動作する IP リストの例については、図 2.5 をご参照ください。

```
alert tcp ![192.168.1.0/24,10.1.1.0/24] any -> \
     [192.168.1.0/24,10.1.1.0/24] 111 (content: "|00 01 86 a5|"; \
     msg: "external mountd access";)
```

図 2.5: IP アドレスリスト

2.2.4 ポート番号

ポート番号は全ポート指定、固定ポート指定、範囲指定、否定といった様々な方法で指定できます。“any” は全ポート指定を意味するワイルドカード値です。固定ポート指定は portmapper の 111 番、telnet の 23 番あるいは http の 80 番などのように単一のポート番号で表されます。範囲指定は範囲演算子:で表されます。たとえば図 2.6 で示したように、範囲演算子を様々な意味を持つよう数多くの方法で適用できます。

ポート番号の否定は否定演算子!を利用して指示します。否定演算子は (無を意味することになる any を除き) 他のルールタイプの全てに適用できます。たとえば、いくぶんひねくれた理由で X Window ポートを除く全記録を取りたい場合は、図 2.7 のようにしてください:

2.2.5 方向演算子

方向演算子 “->” は、ルールが適用されるトラフィックの方向を表します。方向演算子の左側にある IP アドレスとポート番号はトラフィックが送られたと考えられる送信元ホスト、方向演算子の右側にあるアドレスとポート情報は宛先ホストを意味します。

```
log udp any any -> 192.168.1.0/24 1:1024 log udp
```

任意のポート番号から、宛先ポート番号の範囲が 1〜1024 番のトラフィック

```
log tcp any any -> 192.168.1.0/24 :6000
```

任意のポート番号から、6000 番以下のポートに送信される TCP トラフィックを記録する

```
log tcp any :1024 -> 192.168.1.0/24 500:
```

1024 番以下の特権ポートから、500 番以上のポート番号へ送信される TCP トラフィックを記録する

図 2.6: ポート範囲の例

```
log tcp any any -> 192.168.1.0/24 !6000:6010
```

図 2.7: ポート否定の例

さらに、“<>” で表記される双方向演算子もあります。この演算子は左右どちらアドレス、ポート番号が送信元または宛先のどちらでもよいことを現します。これは、telnet や POP3 セッションのような通信を記録・解析する場合に便利です。telnet セッションを両方向から記録するために双方向演算子を使った一例を図 2.8 に示します。

なお、“<-” 演算子は存在しませんのでご注意ください。1.8.7 より前のバージョンの Snort では、方向演算子に対する適切なエラーチェック機能を実装していなかったため、多くの人々が誤ったトークンを利用していました。ルールの読み込みの整合性を確保するため、“<-” は存在しません。

```
log tcp !192.168.1.0/24 any <> 192.168.1.0/24 23
```

図 2.8: 双方向演算子を利用した Snort ルール

2.2.6 Activate/Dynamic ルール

注: Activate および Dynamic ルールは、徐々にタギングに道を譲り、廃止される方向にあります。Snort の将来のバージョンでは、activate/dynamic ルールは改良されたタギング機能によって完全に置き換えられるでしょう。詳細については、第 2.3.32 節をご参照ください。

activate/dynamic ルールの組み合わせは Snort に強力な機能をもたらします。一連のパケットに対してあるルールのアクションが実行された際に、そのルールをきっかとして別のルールを呼び出すことが可能です。これは、特定のルールが呼び出された際、その結果に基づいて Snort を動作させたい場合に非常に役立ちます。

activate ルールは *必須*オプションフィールドとして activates フィールドがある点を除けば alert ルールのように動作します。dynamic ルールは log ルールと同様に機能しますが、オプションフィールドとして activated_by フィールドがある点が異なります。また、dynamic ルールにはもう 1 つの必須フィールドとして count フィールドが必要です。

activate ルールは alert そっくりですが、特定のネットワーク上のイベントが発生した場合にルールを追加指定できます。dynamic ルールは log ルールそっくりですが、特定の activate ルール ID が存在するときに有効化される点で異なります。

総括すれば、activate/dynamic ルールは図 2.9 のようになります。

これらのルールは IMAP のバッファオーバーフローを検知した場合、アラートを出力し、$HOME_NET の外部から$HOME_NET 内の 143 番ポート宛てに入ってくる最初の 50 パケットを収集するように指示します。バッファオーバーフローが仕組まれ、(訳注:攻撃が) 成功してしまった場合、同じサービスポート宛てに送信される次

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA; \
     content: "|E8C0FFFFFF|/bin"; activates: 1;  \
     msg: "IMAP buffer overflow!\";)
dynamic tcp !$HOME_NET any -> $HOME_NET 143 (activated_by: 1; count: 50;)
```

図 2.9: activate/dynamic ルールの例

の (およそ)50 パケットの中に有用なデータが含まれる可能性は非常に高いため、後々の解析のためにこれらのパケットを収集する価値が十分にあります。

2.3 ルールオプション

ルールオプションは Snort 不正侵入検知エンジンの心臓部で、力強さと柔軟性に使いやすさを兼ね備えています。すべての Snort ルールオプションはセミコロン ";" 文字で区切られます。また、ルールオプションはコロン ":" でキーワードと引数を区切ります。

利用可能なキーワード

msg アラートやパケットログ内にメッセージを出力します

logto 標準の出力ファイルではなくユーザが指定したファイル名にパケットを記録します

ttl IP ヘッダの TTL フィールドの値を検査します

tos IP ヘッダの TOS フィールドの値を検査します

id 特定の値に関して IP ヘッダのフラグメント ID フィールドを検査します

ipoption 特定のコードに関して IP option フィールドを監視します

fragbits IP ヘッダのフラグメンテーションビットを検査します

dsize パケットの値とペイロード長を (訳注:を計算して) 検査します

flags ある種の TCP フラグを検査します

seq 特定の値に関して TCP シーケンス番号フィールドを検査します

ack 特定の値に関して TCP 応答確認フィールドを検査します

window 特定の値に関して TCP ウィンドウフィールドを検査します

itype 特定の値に関して ICMP type フィールドを検査します

icode 特定の値に関して ICMP code フィールドを検査します

icmp_id 特定の値に関して ICMP ECHO ID フィールドを検査します

icmp_seq 特定の値に関して ICMP ECHO シーケンス番号を検査します

content パケットのペイロード内のパターンを検索します

content-list パケットのペイロード内の一連のパターンを検索します

offset content オプションのための修飾子。パターンマッチ処理のオフセットを設定します

depth content オプションのための修飾子。パターンマッチ処理の検索対象の限界 (maximum search depth) を設定します。

nocase 直前に指定された content の文字列の大文字/小文字を区別しないようにします

session 所定のセッションに関するアプリケーション層の情報をダンプします

rpc 特定のアプリケーション/プロシージャコールに関する RPC サービスを監視します

resp アクティブレスポンスを行います (接続切断など)

react アクティブレスポンスを行います (Web サイトをブロックする)

reference 外部の攻撃参照 ID

sid Snort ルール ID

rev ルールのリビジョン番号

classtype ルール分類識別子

priority ルール重大度識別子

uricontent パケットの URI 部分のパターンを検索します

tag ルールに対する高度な記録アクション

ip_proto IP ヘッダのプロトコル値

sameip 送信元 IP が宛先 IP と同じかどうか判断する

stateless ストリーム状態に関係なく有効とする

regex ワイルドカード・パターンマッチング

byte_test 数値評価

distance スペースを無視するよう部分的なパターンマッチングを強制

within カウント内に収まるよう部分的なパターンマッチングを強制

byte_test 数値パターン検査

byte_jump 数値パターン検査とオフセット調整

2.3.1 Msg

msg ルールオプションはログ取得およびアラートエンジンに、パケットダンプと共に出力すべき、またはアラートに対して出力すべきメッセージを伝えます。このルールオプションはシンプルな文字列で、区切り文字を表示する際 (たとえば、セミコロン ";") にエスケープ文字として\を利用します。さもなれば、Snort のルール構文解析系を混乱させてしまいます。

書式

```
msg: "<message text>";
```

2.3.2 Logto

logto オプションはこのルールをきっかけとして Snort が記録したパケットを特別なログファイルに記録するよう設定します。このオプションは特に、NMAP によるアクティビティ、HTTP CGI スキャンなどから得られるデータを結合する場合に便利です。ただし、Snort がバイナリ形式のログ取得モードで動作している場合には、本オプションは機能しませんのでご注意ください。

書式

```
logto:"filename";
```

2.3.3 TTL

このルールオプションは、検査の対象となる特定の TTL(time-to-live) 値を検査するために利用します。実施される検査では完全に一致した場合にのみ合格となります。このオプションキーワードは本来 traceroute 試行の検知を目的としたものでした。

書式

```
ttl:<number>;
```

2.3.4 TOS

tos キーワードを使うことで、IP ヘッダ中の TOS フィールドの特定の値をチェックすることができます。実施される検査では完全に一致した場合にのみ合格となります。

書式

```
tos: <number>;
```

2.3.5 ID

このオプションキーワードは、IP ヘッダフラグメント ID フィールドが一致しているかどうかを検査するために利用します。一部のハッキングツール (とその他のプログラム) はこのフィールドを様々な目的のために利用します。たとえば、31337 という値は一部のハッカーの間で非常に人気があります。この値や他のハッカー番号と呼ばれる番号を検査するためにシンプルなルールを適切に設定することで、ハッカーに対抗することができます。

書式

```
id: <number>;
```

2.3.6 Ipoption

パケット内に IP オプションが存在する場合、本オプションはソースルーティングのようなある特定のオプションを検索します。このオプションに対して有効な引数は次の通りです。

- rr - Record route

- eol - End of list
- nop - No op
- ts - Time Stamp
- sec - IP security option
- lsrr - Loose source routing
- ssrr - Strict source routing
- satid - Stream identifier

最もよく監視対象となる IP オプションは、ストリクトソースルーティングおよびルーズソースルーティングです。これらは広い範囲で使われているインターネットアプリケーションでは利用されていません。このオプションでは、ルール毎に単一のオプションのみを指定できます。

書式

```
ipopts: option;
```

2.3.7 Fragbits

このルールは IP ヘッダ内のフラグメントおよび予約ビットを検査します。Reserved Bit(RB)、More Fragments(MF) ビット、Don't Fragment(DF) ビットの 3 ビットをチェックすることができます。これらのビットは様々な組み合わせでチェックすることができます。特定のビットを表すために、次の値を利用できます。すなわち * R - Reserved Bit * D - DF ビット*M - MF ビットです。

また、指定されたビットに対する論理的な一致条件を定義するために修飾子を利用できます。すなわち、* — ALL フラグ。指定されたビットに値が設定されており、なおかつ他に値が設定されている場合** — ANY フラグ。指定されたビットのどれかに値が設定されている場合* — NOT フラグ。指定されたビットに値が設定されていない場合です。

書式

```
fragbits: <bitvalues>;
```

```
alert tcp !$HOME_NET any -> $HOME_NET any (fragbits: R+; \
      msg: "Rerserved bit set!";)
```

図 2.10: フラグビット検知の利用例

2.3.8 Dsize

dsize オプションはパケットペイロードサイズを検査するために利用します。このオプションにはどんな値でも指定でき、値の範囲や限界を示すために大なり記号 “>”、小なり記号 “<” も指定できます。たとえば、特定のサービスがある大きさのバッファを持っていることが分かっている場合、バッファオーバーフローの企みを監視するためにこのオプションを利用できます。このオプションはペイロード内容のチェックよりも遥かに高速にバッファオーバーフローを検査できるという利点も備えています。

また、値の範囲をチェックするためにこのオプションを利用できます。たとえば、dsize:400<>500 はすべてのパケットのペイロード部分から 400〜500 バイトを返します。

これらのチェックはストリーム再構築済みパケットに対しては常に偽の値を返します。

書式

```
dsize: [<>]<number>[<><number>];
```

注: >と<演算子はオプションです。

2.3.9 Content

content キーワードは、Snort の最も重要な機能のうちの 1 つです。このキーワードを利用すれば、ユーザはパケットペイロード内の特定の内容を検索するルールを設定し、そのデータに基づく対応策を呼び出すことができます。content オプションのパターンマッチが行われる場合、常に Boyer-Moore 法によるパターンマッチ関数が呼び出され、パケットの内容に対して (かなり計算量の多い) 検査が実施されます。引数であるデータ文字列に合致するデータが、パケットのペイロード内のいかなる部分にでも含まれる場合は、この検査が合格となり、残りのルールオプションの検査が実行されます。この検査は大文字と小文字を区別する点に留意してください。

content キーワードに対するオプションデータはいくぶんか複雑です。このキーワードではテキストとバイナリデータが混在できます。バイナリデータは一般にパイプ (|) 文字で囲まれ、バイトコードで表記されます。バイトコードはバイナリデータを 16 進数で表現する方法で、複雑なバイナリデータを記述できるするための優れた省略表記法といえます。図 2.11 は Snort ルールにテキストとバイナリデータが混在している例を示したものです。

なお、1 つのルール内に複数の content ルールを指定できますので、誤検出 (false positives; 偽陽性) がより少ないルールを書けます。

content ルールにおいては、下記の文字をエスケープする必要がありますのでご注意ください。

```
: ; \ "
```

ルールの先頭に ! を設定すると、指定された内容を含まないパケットに対してアラートが出力されます。これは特定のパターンに一致しないパケットに対してアラートを出力するルールを記述する際に便利です。

書式

```
content: [!] "<content string>";
```

```
alert tcp any any -> 192.168.1.0/24 143 (content:"|90C8 C0FF FFFF|/bin/sh"; \
                                         msg:"IMAP buffer overflow!";)
```

図 2.11: content ルールオプションにバイナリバイトコードとテキストが混在する例

```
alert tcp any any -> 192.168.1.0/24 21 (content: !"GET"; depth: 3; nocase; \
          dsize: >100; msg: "Long Non-Get FTP command!";)
```

図 2.12: 否定の例

2.3.10 Offset

offset ルールオプションは、content オプションキーワードを利用するルールの修飾子として利用します。本キーワードは、パターンマッチ機能の検索開始位置をパケットのペイロードの先頭から変更します。この機能はペイロードの最初の 4 バイト内にコンテンツ検索の対象となる文字列が決して検知されない場合に CGI スキャン検知ルールのような事象に対して非常に効果的です (訳注: CGI スキャンのペイロードの先頭には HEAD / HTTP/1.0.. のような文字列が必ず存在すると考えられるため、これらを検索対象としない設定です)。ただし、offset 値を厳格に設定にすると、攻撃を見逃す恐れがありますのでご注意ください。また、このルールオプションキーワードは content ルールオプションと併用する必要があります。content、offset、depth 検索ルールを組み合わせた例については、図 2.13 をご参照ください。

書式

```
offset: <number>;
```

2.3.11 Depth

depth は content ルールオプションのもう 1 つの修飾子です。この修飾子は content パターンマッチ機能において、検索対象の先頭から検索を行う終端 (search depth) を定義する。この修飾子は、与えられた一連のコンテンツに対し、想定される検索対象を越える非効率的な検索を抑制する際に役立ちます (つまり、Web 向きのパケット内の cgi-bin/phf を検索する場合に、最初の 20 バイト以降のペイロードを検索するために時間を無駄にする必要はおそらくなくなります!)。content, offset, depth 検索ルールを組み合わせた例については、図 2.13 をご参照ください。

書式

```
depth: <number>;
```

```
alert tcp any any -> 192.168.1.0/24 80 (content: "cgi-bin/phf"; \
    offset: 3; depth: 22; msg: "CGI-PHF access";)
```

図 2.13: content, offset, depth search ルールを組み合わせた例

2.3.12 Nocase

nocase オプションは content ルール内の大文字と小文字を区別しないようにするため利用します。このオプションはルールにおいて単体で指定され、パケットのペイロードと比較される ASCII 文字は大文字・小文字関係なく扱われます。

書式

```
nocase;
```

```
alert tcp any any -> 192.168.1.0/24 21 (content:"USER root"; \
    nocase; msg: "FTP root user access attempt";)
```

図 2.14: nocase 修飾子を含む content ルール

2.3.13 Flags

このルールは TCP フラグが一致するかについて検査します。現在、Snort では 9 種類のフラグ変数に対応しています。

F FIN (TCP フラグバイト内の最下位ビット)

S SYN

R RST

P PSH

A ACK

U URG

2 予約ビット 2

1 予約ビット 1 (TCP フラグバイト内の最上位ビット)

0 フラグ未設定

指定されたフラグに対するマッチング定義を指定できる論理演算子も用意されています:

+ ALL フラグ、指定されたフラグとその他すべてのフラグに対してマッチする

***** ANY フラグ、指定されたフラグのいずれかに対してマッチする

! NOT フラグ、指定されたフラグがパケット内に存在しない場合にマッチする

IP スタックに対するフィンガープリンティングの試みや、その他不審な活動といった異常な挙動を検知する目的で予約ビットを利用できます。図 2.15 は SYN-FIN スキャン検知ルールを示したものです。

ECN のように以前の予約ビット 1 番、2 番と共に送信される SYN パケットといったセッション開始パケットに対応できるルールを書くには、オプションマスクを指定する必要があります (訳注: ECN は RFC3168 で定義されたネットワークの混雑状況を明示的に相手に伝える仕組みで、ECN の CWR フラグと ECE フラグは以前、予約フラグだった領域を利用している。以前の Snort には ECN フラグの処理に問題があった。詳細については [6] を参照のこと)。フラグ値 S,12 を検知するルールの場合、予約ビットの値に関係なく SYN パケットを検知できます。

書式

```
flags: <flag values>[,mask value];

alert any any -> 192.168.1.0/24 any (flags: SF,12; msg: "Possible SYN FIN scan";)
```

図 2.15: フラグ指定例

2.3.14 Seq

このルールオプションは TCP シーケンス番号を照会するものです。基本的に、このオプションはパケットが静的なシーケンス番号セットを含んでいるかどうかを検知しますので、滅多に利用されることはありません。このオプションは万全を期して盛り込まれたものです。

書式

```
seq: <number>;
```

2.3.15 Ack

ack ルールオプションキーワードは TCP ヘッダの応答確認 (acknowledge) フィールドを照会するものです。このルールは NMAP[1, 2] TCP ping を検知するという実用的な目的を担っています。NMAP TCP ping はネットワークホストが動作しているかを確認するために、このフィールドの値をゼロにした TCP ACK フラグ付きパケットを送信します。この活動を検知するためのルールを図 2.16 に示します。

書式

```
ack: <number>;
```

```
alert any any -> 192.168.1.0/24 any (flags: A; ack: 0; msg: "NMAP TCP ping";)
```

図 2.16: TCP ACK フィールド利用例

2.3.16 Window

このルールオプションは TCP ウィンドウサイズを照会します。このオプションは静的なウィンドウサイズをチェックします。この機能により、バックドアの類がハードコーディングされたウィンドウサイズの値を利用する場合に、TCP ウィンドウサイズオプションを利用して特定の値をチェックできます。

書式

```
window:[!]<number>;
```

2.3.17 Itype

このルールは ICMP タイプフィールドの値を検査します。このルールでは、フィールド内の数値が設定されます。設定できる値の一覧については、Snort に含まれる decode.h ファイル、または ICMP のリファレンスを参照してください。なお、注目すべき点としては、DoS 攻撃やフラッド攻撃で時折利用される無効な ICMP タイプの値を検知できるよう、範囲外の値を設定できることが挙げられます。

書式

```
itype: <number>;
```

2.3.18 Icode

icode ルールオプションキーワードは itype ルールとほぼ同様、ルールに数値を設定するだけで、Snort は指定された ICMP コードの値を含むトラフィックをすべて検知します。また、不審なトラフィックを検知するために、範囲外の値も設定できます。

書式

```
icode: <number>;
```

2.3.19 Session

session キーワードはバージョン 1.3.1.1 で追加された機能で、TCP セッションからユーザのデータを抽出するために利用します。ユーザが telnet、rlogin、ftp、あるいは Web セッションにおいてユーザが何を入力しているかを確認する場合に非常に便利です。session ルールのオプションとして、printable と all の 2 つのキーワードが引数として利用できます。printable キーワードはユーザが通常閲覧したり、入力できるデータのみを出力します。all キーワードは printable キーワードで出力できない文字列を 16 進数に変換します。この機能を利用すると Snort の性能低下を招く恐れがあるため、高負荷環境での利用は避けた方がよいでしょう。後処理用バイナリ (tcpdump 形式) のログファイルの方がお勧めです。telnet セッション記録ルールのよい例として、図 2.17 をご参照ください。

書式

```
session: [printable|all];
```

```
log tcp any any <> 192.168.1.0/24 23 (session: printable;)
```

図 2.17: printable telnet セッションデータの記録

2.3.20 Icmp_id

icmp_id オプションは、ICMP ECHO パケットの特定の ICMP ID の値を検査します。一部の [84] 隠密チャネル (covert channel) を用いるプログラムは静的な ICMP フィールドを通信時に利用するため、このルールが役立ちます。この特殊なプラグインは、[85]Max Version 氏による stacheldraht(訳注:DDoS 攻撃ツール) 検知ルールを実現するために開発されましたが、様々な潜在的攻撃を検知する際においても間違いなく役立つでしょう。

書式

```
icmp_id: <number>;
```

2.3.21 Icmp_seq

icmp_seq オプションは、ICMP ECHO パケットの特定の ICMP シーケンス番号の値を検査します。一部の [84] 隠密チャネル (covert channel) を用いるプログラムは静的な ICMP フィールドを通信時に利用するため、このルールが役立ちます。この特殊なプラグインは、[85]Max Version 氏による stacheldraht(訳注:DDoS 攻撃ツール) 検知ルールを実現するために開発されましたが、様々な潜在的攻撃を検知する際においても間違いなく役立つでしょう。(すでにお気付きかもしれませんが、このフィールドに関する情報は icmp_id の説明とほぼ同じものです。)

書式

```
icmp_seq: <number>;
```

2.3.22 RPC

このオプションは RPC リクエストを調査し、アプリケーション、プロシージャ、プログラムバージョンを自動的にデコードし、これらの 3 つの変数が一致すれば合格したことを示します。オプションの形式はアプリケーション、プロシージャ、バージョンです。プロシージャおよびバージョン番号についてはワイルドカード*が指定できます。

書式

```
rpc: <数値, [数値|*], [数値|*]>;
```

```
alert tcp any any -> 192.168.1.0/24 111 (rpc: 100000,*,3;\
      msg:"RPC getport (TCP)";)

alert udp any any -> 192.168.1.0/24 111 (rpc: 100000,*,3;\
      msg:"RPC getport (UDP)";)

alert udp any any -> 192.168.1.0/24 111 (rpc: 100083,*,*; msg:"RPC ttdb";)

alert udp any any -> 192.168.1.0/24 111 (rpc: 100232,10,*;
      msg:"RPC sadmin";)
```

図 2.18: さまざまな RPC 呼び出しのアラート

2.3.23 Resp

resp キーワードは、Snort のルールにマッチしたトラフィックに対してフレキシブル・レスポンス (FlexResp - 柔軟性に富んだ応答) を実行します。FlexResp コードによって、Snort は目障りなコネクションを動的に切断できます。このモジュールに対しては以下の引数が有効です:

rst_snd - 送信側ソケットに TCP-RST パケットを送信します
rst_rcv - 受信側ソケットに TCP-RST パケットを送信します
rst_all - 両方向に TCP-RST パケットを送信します
icmp_net - 送信側に ICMP_NET_UNREACH を送信します
icmp_host - 送信側に ICMP_HOST_UNREACH を送信します
icmp_port - 送信側に ICMP_PORT_UNREACH を送信します
icmp_all - 送信側に上記の ICMP パケットをすべて送信します

対象ホストに複数のレスポンスを組み合わせて送信することができます。複数の引数はコンマで区切ってください。

書式

```
resp: <resp_modifier[, resp_modifier...];
```

注意

フレキシブル・レスポンスの使用の際には、細心の注意を払ってください。次のようなルールを定義しただけで、Snort は簡単に無限ループ状態に陥ってしまいます。

```
alert tcp any any -> 192.168.1.1/24 any (msg: "aiee!";  resp: rst_all;)
```

通常のネットワークトラフィックを妨害することも簡単にできてしまいます。

```
alert tcp any any -> 192.168.1.0/24 1524 (flags: S;  \
   resp: rst_all; msg: "Root shell backdoor attempt";)

alert udp any any -> 192.168.1.0/24 31 (resp: icmp_port,icmp_host; \
   msg: "Hacker's Paradise access attempt";)
```

図 2.19: FlexResp の利用例

2.3.24 Content-list

content-list キーワードを利用すると、複数の content の文字列をわずか 1 つの content オプションで指定できるようになります。図 2.20 に示した通り、検索対象となるパターンは content-list ファイルに一行づつ記述する必要があります。このように記述しない場合には、content の文字列すべてが標準の content ディレクティブに対する引数として扱われてしまうことになります。このオプションは react キーワードの基準となります。

```
# adult sites
"porn"
"porn"
"adults"
"hard core"
"www.pornsite.com"
```

図 2.20: 成人向け content-list ファイルの例

書式

```
content-list: <file_name>;
```

2.3.25 React

この機能を利用することにより、ネットワークトラフィックを生成するループ状態に容易に陥ってしまう危険性があることを警告しておきます。

フレックス・レスポンス (FlexResp) を基にした react キーワードは、Snort ルールに合致したトラフィックに対する柔軟なリアクションを実現します。基本的なリアクションは、New York Times、Slashdot、あるいはもっと重要な問題—つまり Napster やポルノサイトのようなユーザがアクセスしようとしているサイトをブロックすることです。

FlexResp コードによって、Snort は目障りなコネクションを動的に切断、もしくはブラウザに対して視覚的な通知を送信できます (warn 修飾子は近日中に提供されます)。この通知には独自の注釈を含めることができます。このオプションで利用できる引数 (基本的な修飾子) は下記の通りです。

- block - コネクションを遮断し、視覚的な通知を送信する。
- warn - 視覚的な警告通知を送信する (近日中に提供されます)。

基本的な引数と以下の引数 (追加的な修飾子) を組み合わせることができます:

- msg - 遮断の際の視覚的通知に msg オプションの文字列を含めます。
- proxy: <port_nr> - 視覚的通知を送信する際にプロキシポートを利用する (近日中に提供されます)。

複数の引数を追加する場合にはコンマで区切ります。react キーワードはオプションリストの最後に記述してください。

書式

```
react: <react_basic_modifier[, react_additional_modifier]>;
```

```
alert tcp any any <> 192.168.1.0/24 80 (content: "bad.htm"; \
    msg: "Not for children!"; react: block, msg;)
```

図 2.21: react 利用例

2.3.26 Reference

reference キーワードを利用すれば、ルールに外部の攻撃識別システムへのリファレンスを含めることができます。現在、このプラグインは固有の URL の他に、特定のシステムもいくつかサポートしています。この機能は出力プラグインに対し、生成されるアラートに関するより詳細な情報を提供するために利用します。

また、sid に基づいてアラートの説明に索引を付加するシステムについては、`http://www.snort.org/snort-db/` も必ず確認してください (第 2.3.27 参照)。

表 2.1: サポートしているシステム

システム	URL プリフィックス
Bugtraq	http://www.securityfocus.com/bid/
CVE	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
Arachnids	http://www.whitehats.com/info/IDS
McAfee	http://vil.nai.com/vil/dispVirus.asp?virus_k=
url	http://

書式

```
reference: <id system>,<id>; [reference: <id system>,<id>;]
```

```
alert tcp any any -> any 7070 (msg: "IDS411/dos-realaudio"; \
   flags: AP; content: "|fff4 fffd 06|"; reference: arachNIDS,IDS411;)

alert tcp any any -> any 21 (msg: "IDS287/ftp-wuftp260-venglin-linux"; \
   flags: AP; content: "|31c031db 31c9b046 cd80 31c031db|"; \
   reference: arachNIDS,IDS287; reference: bugtraq,1387; \
   reference: cve,CAN-2000-1574; )
```

図 2.22: リファレンス利用例

2.3.27 Sid

sid キーワードは Snort のルールを識別するために利用します。この情報に基づいて、出力プラグインが容易にルールを識別できるようになります。使用例については図 2.23 をご参照ください。sid の範囲は次にように割り当てられます。

- <100 将来のために予約
- 100-1,000,000 Snort の配布物に含まれるルール
- >1,000,000 ローカルルールとして利用可能

sid-msg.map というファイルに、Snort のルール id と msg タグのマッピングが含まれています。これは出力後の処理において id からアラートメッセージを割り当てる際に利用します。

書式

```
sid: <snort rules id>;

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 \
        (msg:"WEB-IIS File permission canonicalization"; \
         uricontent:"/scripts/../.."; \
         flags: A+; nocase; sid:983; rev:1;)
```

図 2.23: sid の利用例

2.3.28 Rev

rev キーワードはルールのリビジョンを識別するために利用します。リビジョンと Snort のルール ID を併用することで、シグネチャと解説を最新の情報に更新できます。利用例については、図 2.23 をご参照ください。

書式

```
rev: <revision integer>
```

2.3.29 Classtype

classtype キーワードは、攻撃の種類別に優先度に基づいてアラートを分類します。ユーザは各タイプのルール分類ごとに優先度を指定できます。また、分類されたルールにはデフォルト優先度セットが設定されます。

書式

```
classtype: <class name>;
```

ルール分類は `classification.config` ファイルで定義されます。この設定ファイルは以下の構文を利用します。

```
config classification:  <class name>,<class description>,<default priority>
```

Snort に含まれる標準の分類を表 2.2 に示します。標準の分類は現在、3 つのデフォルトの優先度に従って順序付けられています。デフォルトルールセットにおいて、優先度 1 は最も深刻度の高い優先度レベルで、優先度 4 はもっとも低いレベルです。

表 2.2: 優先度高に分類 - 優先度 1

Classtype	Description
attempted-admin	Attempted Administrator Privilege Gain
attempted-user	Attempted User Privilege Gain
shellcode-detect	Executable code was detected
successful-admin	Successful Administrator Privilege Gain
successful-user	Successful User Privilege Gain
trojan-activity	A Network Trojan was detected
unsuccessful-user	Unsuccessful User Privilege Gain
web-application-attack	Web Application Attack

```
alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; \
    dsize: >128; classtype:attempted-admin; priority:10 );

alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
   content:"expn root"; nocase; classtype:attempted-recon;)
```

図 2.24: classtype ルールの例

2.3.30 Priority

priority タグは影響度のレベルをルールに割り当てます。classtype ルールは、priority ルールで上書きできるデフォルトの優先度を割り当てます。分類ルールと併用した場合の例については、図 2.24 をご参照ください。単独の利用例については、図 2.25 をご参照ください。

書式

```
priority: <priority integer>;
```

表 2.3: 優先度中に分類 - 優先度 2

Classtype	Description
attempted-dos	Attempted Denial of Service
attempted-recon	Attempted Information Leak
bad-unknown	Potentially Bad Traffic
denial-of-service	Detection of a Denial of Service Attack
misc-attack	Misc Attack
non-standard-protocol	Detection of a non-standard protocol or event
rpc-portmap-decode	Decode of an RPC Query
successful-dos	Denial of Service
successful-recon-largescale	Large Scale Information Leak
successful-recon-limited	Information Leak
suspicious-filename-detect	A suspicious filename was detected
suspicious-login	An attempted login using a suspicious username was detected
system-call-detect	A system call was detected
unusual-client-port-connection	A client was using an unusual port
web-application-activity	access to a potentially vulnerable web application

表 2.4: 優先度低に分類 - 優先度 3

Classification	Description
icmp-event	Generic ICMP event
misc-activity	Misc activity
network-scan	Detection of a Network Scan
not-suspicious	Not Suspicious Traffic
protocol-command-decode	Generic Protocol Command Decode
string-detect	A suspicious string was detected
unknown	Unknown Traffic

```
alert TCP any any -> any 80 (msg: "WEB-MISC phf attempt"; flags:A+; \
      content: "/cgi-bin/phf"; priority:10;)
```

図 2.25: priority ルールの例

2.3.31 Uricontent

uricontent ルールを利用すれば、リクエストの URI 部分のみを検索対象とすることができます。この機能によって、サーバのデータファイルからの誤検知を発生させることなく、攻撃のリクエスト部分のみを検索できるようになります。この機能に対するパラメータについては、第 2.3.9 節の content ルールオプションをご参照ください。

このオプションは、第 2.4.1 節に指定された HTTP デコーダと連動して機能します。

書式

```
uricontent:[!]<content string>;
```

2.3.32 Tag

tag キーワードを利用すれば、ルールが呼び出した単一のパケットだけではなく、それ以外のログも取れるようになります。いったんルールが呼び出せれると、送信元ホストや宛先ホストに関係している付加的トラフィックに “タグ” が付加されます。タグが付けられたトラフィックは応答コードや攻撃後のトラフィックの解析を可能にするために記録されます。“タグ” 付きアラートは通常のアラートと同様、同じ出力プラグインに送信されますが、これらの特別なアラートを適切に処理するのは出力プラグインの責任となります。ただし、現在のところ、第 2.5.7 節のデータベース出力プラグインは “タグ” 付きアラートを適切に処理できません。利用例については、図 2.26 をご参照ください。

書式

```
tag: <type>, <count>, <metric>, [direction]
```

type

session ルールを呼び出したセッションのパケットを記録します

host タグが付加されたホストからのパケットを記録します ([direction] 修飾子を利用します)

count count を様々な単位数として指定できます。単位は<metric>フィールドで指定します。

metric

packets <count>パケットだけホスト/セッションにタグを付加します

seconds <count>秒だけホスト/セッションにタグを付加します

```
alert tcp !$HOME_NET any -> $HOME_NET 143 (flags: A+; \
      content: "|e8 c0ff ffff|/bin/sh";  tag: host, 300, packets, src; \
      msg: "IMAP Buffer overflow, tagging!";)

alert tcp !$HOME_NET any -> $HOME_NET 23 (flags: S; \
     tag: session, 10, seconds; msg: "incoming telnet session";)
```

図 2.26: タグキーワード例

2.3.33 IP proto

ip_proto キーワードを利用すれば、IP プロトコルヘッダに対する照会が可能になります。名前を用いて指定できるプロトコルの一覧については、/etc/protocols をご参照ください。ルール内で IP プロトコルを指定する方法にご注意ください。

書式

```
ip_proto:[!] <name or number>;
```

```
alert ip !$HOME_NET any -> $HOME_NET any \
         (msg: "IGMP traffic detected";  ip_proto: igmp;)
```

図 2.27: IP Proto の例

2.3.34 Same IP

sameip キーワードを利用すれば、送信元 IP が宛先 IP と同じかどうかを検査できます。

書式

```
sameip;
```

```
alert ip $HOME_NET any -> $HOME_NET any (msg: "SRC IP == DST IP"; sameip;)
```

図 2.28: Same IP 利用例

2.3.35 Regex

このモジュールは現在開発中のため、実運用環境のルールセットで利用すべきではありません。したがって、これを利用してアラートが設定された場合、エラー状態が引き起されてしまいます。

2.3.36 Flow

flow ルールオプションは TCP ストリーム再構築機能と組み合わせて利用します (第 2.4.5 節参照)。このオプションを利用すれば、特定のトラフィックフローの方向に限定してルールを適用できるようになります。

この結、ルールをクライアントまたはサーバにのみ適用できるようになります。また、Web ページを閲覧する $HOME_NET 内のクライアントに関連したパケットを、$HOME_NET 内で稼働しているサーバのものと区別できるようになります。

established キーワードは、確立された TCP 接続を示すために多くの場所で利用されてきた `flags: A+`を置き換えます。

オプション

to_client A から B へのサーバ応答を元に呼び出しを行います

to_server A から B へのクライアント要求を元に呼び出しを行います

from_client A から B へのクライアント要求を元に呼び出しを行います

from_server A から B へのサービス応答を元に呼び出しを行います

established 確立された TCP 接続を元に呼び出しを行います

stateless ストリームプロセッサの状態に関係なく呼び出しを行います (マシンのクラッシュを狙ったパケットに対して効果的です)。

no_stream 再構築されたストリームパケットに対して呼び出しを行いません (dsize や stream4 に対して効果的)。

only_stream 再構築されたストリームパケットに対してのみ呼び出しを行います

書式

```
flow:[to_client|to_server|from_client| \
  from_server|established|stateless|no_stream|only_stream]}
```

```
alert tcp !$HOME_NET any -> $HOME_NET 21 (flow: from_client; \
       content: "CWD incoming"; nocase; \
       msg: "cd incoming detected"; )

alert tcp !$HOME_NET 0 -> $HOME_NET 0 \
      (msg: "Port 0 TCP traffic"; flow: stateless;)
```

図 2.29: フローの利用例

2.3.37 Fragoffset

fragoffset キーワードを利用すれば、10 進数の値と IP フラグメントオフセットフィールドを比較できます。IP セッションの最初のフラグメントを全て捕捉するためには、fragoffset を 0 に設定した状態で、More fragments オプションを検索し、fragbits キーワードを利用する。

書式

```
fragoffset:[<|>]<number>
```

```
alert ip any any -> any any \
      (msg: "First Fragment"; fragbits: M; fragoffset: 0;)
```

図 2.30: Fragoffset 利用例

2.3.38 Rawbytes

rawbytes キーワードを利用すれば、デコードされた telnet データを正規化されていないデータとして処理できます。この機能により、プリプロセッサと独立して telnet のネゴシエーションコードを検索できるようになります。このキーワードは、前第 2.3.9 節の content オプションの修飾子として動作します。

書式

```
rawbytes;
```

```
alert tcp any any -> any any (msg: "Telnet NOP"; content: "|FF F1|"; rawbytes;)
```

図 2.31: rawbytes 利用例

2.3.39 distance

distance キーワードは、Content(第 2.3.9 節参照) を利用したパターンマッチの間隔が少なくとも N バイト空いていることを確認する content 修飾子です。このキーワードは within(第 2.3.40 節) ルールオプションと共に利用します。

図 2.32 でリストアップされたルールは、ÄBCDE.{1}EFGHo̎の正規表現に相当します。

書式

```
distance: <byte count>;
```

```
alert tcp any any -> any any (content: "2 Patterns"; \
          content: "ABCDE"; content: "EFGH"; distance: 1;)
```

図 2.32: distance 利用例

2.3.40 Within

within キーワードは、Content(第 2.3.9 節参照) を利用したコンテンツマッチ間の間隔が多くとも N バイト以下に維持されていることを確認する content 修飾子です。このキーワードは distance(第 2.3.39 節) ルールオプションと共に利用します。

図 2.33 にリストアップされたルールは、ABCDE のマッチから 10 バイト以内に検索対象を制限します。

書式

```
within: <byte count>;
```

```
alert tcp any any -> any any (content: "2 Patterns"; \
          content: "ABCDE"; content: "EFGH"; within: 10;)
```

図 2.33: within 利用例

2.3.41 Byte_Test

byte フィールドを特定の値に対して (演算子を使って) 検査します。2 進数の値を検査したり、代表的なバイトストリングを 2 進数の値に変換して、それらを検査したりすることができます。

書式

```
byte_test: <bytes_to_convert>, <operator>, <value>, <offset> \
        [, [relative],[big],[little],[string],[hex],[dec],[oct]]
```

bytes_to_convert パケットから取り込むバイト数

operator 値を検査するために実行する操作 (<,>,=,!)

value 変換された値と比較を行う値

offset 処理を開始するペイロードのバイト数

relative 最後に行ったパターンマッチに関連する offset を利用

big big endian としてデータを処理 (デフォルト値)

little little endian としてデータを処理

string データがパケット内に文字列形式で記憶

hex 変換された文字列データを 16 進数で表現

dec 変換された文字列データを 10 進数で表現

oct 変換された文字列データを 8 進数で表現

2.3.42 Byte_Jump

Byte_Jump オプションは一定のバイト数を取り込んで、それらを数値表現に変換し、(さらなるパターンマッチング/バイト検査を行えるようにするため) doe_ptr を指定された数値分、ジャンプさせるために利用します。これにより、ネットワークデータ内で検知された数値を考慮した相対的なパターンマッチが可能になります。

書式

```
byte_jump: <bytes_to_convert>, <offset> \
        [, [relative],[big],[little],[string],[hex],[dec],[oct],[align]]
```

bytes_to_convert パケットから取り込むバイト数

offset 処理を開始するペイロードのバイト数

relative 最後に行ったパターンマッチに関連する offset を利用

big big endian としてデータを処理 (デフォルト値)

little little endian としてデータを処理

string データがパケット内に文字列形式で記憶

hex 変換された文字列データを 16 進数で表現

dec 変換された文字列データを 10 進数で表現

oct 変換された文字列データを 8 進数で表現

align 変換されたバイト数を次の 32 ビット境界まで概算

```
alert udp $EXTERNAL_NET any -> $HOME_NET any \
       (msg:"AMD procedure 7 plog overflow "; \
       content: "|00 04 93 F3|"; \
       content: "|00 00 00 07|"; distance: 4; within: 4; \
       byte_test: 4,>, 1000, 20, relative;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any \
     (msg:"AMD procedure 7 plog overflow "; \
     content: "|00 04 93 F3|"; \
     content: "|00 00 00 07|"; distance: 4; within: 4; \
     byte_test: 4, >,1000, 20, relative;)

alert udp any any -> any 1234 \
     (byte_test: 4, =, 1234, 0, string, dec; \
     msg: "got 1234!";)

alert udp any any -> any 1235 \
     (byte_test: 3, =, 123, 0, string, dec; \
     msg: "got 123!";)

alert udp any any -> any 1236 \
     (byte_test: 2, =, 12, 0, string, dec; \
     msg: "got 12!";)

alert udp any any -> any 1237 \
     (byte_test: 10, =, 1234567890, 0, string, dec; \
     msg: "got 1234567890!";)

alert udp any any -> any 1238 \
     (byte_test: 8, =, 0xdeadbeef, 0, string, hex; \
     msg: "got DEADBEEF!";)
```

図 2.34: Byte Test 利用例

```
alert udp any any -> any 32770:34000 (content: "|00 01 86 B8|"; \
              content: "|00 00 00 01|"; distance: 4; within: 4; \
              byte_jump: 4, 12, relative, align; \
              byte_test: 4, >, 900, 20, relative; \
              msg: "statd format string buffer overflow";)
```

図 2.35: Byte Jump 利用例

2.4 プリプロセッサ

プリプロセッサは Snort のバージョン 1.5 から導入されました。プリプロセッサ機能を通じて、ユーザやプログラマはモジュール型のプラグインを容易に Snort に組み込み、機能を拡張できます。プリプロセッサのコードは侵入検知エンジンが呼び出される前に実行されますが、パケットがデコードされた後に実行されます。この機構により、パケットの修正や分析を帯域外 (out of band) 方式で行うことができます。

preprocessor キーワードを通じてプリプロセッサの読み込み・設定を行います。Snort ルール内の preprocessor ディレクティブの形式は次の通りです。

```
preprocessor <name>: <options>

preprocessor minfrag: 128
```

図 2.36: Preprocessor ディレクティブ形式の例

2.4.1 HTTP デコード

HTTP の URI 文字列を処理し、そのデータを確かな ASCII 文字列に変換するために、HTTP デコード機能が利用できます。たとえば、HTTP 規格は%20 の文字列が単一の空白 (例:) として解釈されるよう、文字に対する 16 進エンコーディングを定義しています。Web サーバは、様々な規格に準拠するように書かれているだけでなく、おびただしい種類のクライアントにも対応できるよう設計されています。Microsoft の Web サーバは特定のバグだけに対応するだけでなく、付加的な種類のエンコード方式も処理できます。

表 2.5: HTTP デコードオプション

オプション	目的	Web サーバ
unicode	Multibyte encoding standard	IIS (all versions 3+)
iis_alt_unicode	%u### encodings	IIS
double_encode	IIS encoding bugs	IIS 3,4,5 versions prior to MS01-44
iis_flip_slash	interpret \ as /	IIS
full_whitespace	interpret tabs as spaces	Apache

書式

```
http_decode:<port list> [unicode] [iis_alt_unicode]\
             double_encode] [iis_flip_slash] [full_whitespace]

preprocessor http_decode: 80 8080 unicode iis_flip_slash iis_alt_unicode
```

図 2.37: HTTP デコードディレクティブの形式例

2.4.2 Portscan Detector

Snort Portscan プリプロセッサは Patric Mullen 氏によって開発されました。

Snort Portscan プリプロセッサの機能

- 単一の送信元 IP からのポートスキャンを開始から終了まで、標準のログ取得機能で記録します。
- ログファイルを指定する場合に、スキャンの種類の他に、スキャンの対象となった宛先 IP アドレスとポート番号も記録します。

ポートスキャンは、T 秒間に P を超えるポート番号への TCP 接続試行、または T 秒間に P を超えるポート番号に送信される UDP パケットと定義されます。ポート番号は複数の宛先 IP アドレスに拡張でき、複数の IP に渡る同一のポート番号も対象となります。現在のバージョンでは単独->単独、または単独->多数のポートスキャンを検知できます。次のフルリリースでは、分散型ポートスキャン (複数->単独または複数->複数) に対応する予定です。ポートスキャンは NULL, FIN, SYNFIN, XMAS などの単一のステルススキャンパケットとも定義されます。これは、Snort の標準配布ファイルに含まれる scan-lib から、ステルススキャンパケットの部分をコメントアウトすべきであることを意味しています (訳注:バージョン 2.0.2 の配布物には当該ファイルを確認できなかったため、この記述は古い可能性がある)。ポートスキャンモジュールを利用する際のメリットとしては、これらのアラートが各パケットごとにではなく、各スキャンごとに出力される点が挙げられます。外部ログ取得機能を利用すれば、ログファイルを参照することでスキャンのテクニックやタイプを確認できます。

このモジュールに対する引数は次の通りです。

- ポートスキャンを監視する対象のネットワーク/CIDR ブロック
- 検知期間中にアクセスを受けたポートの数
- ポートへのアクセスをカウントする閾値を考慮する検知期間の秒数。
- アラートを出力するログ出力先のディレクトリ/ファイル名。アラートは標準の alert ファイルにも書き込まれます

書式

```
portscan: <monitor network> <number of ports> <detection period> <file path>

preprocessor portscan: 192.168.1.0/24 5 7 /var/log/portscan.log
```

図 2.38: Portscan Preprocessor 設定例

2.4.3 Portscan Ignorehosts

もう 1 つの Patric Mullen 氏から提供されたモジュールは、ポートスキャン検知システムの挙動を変更するものです。運用中のサーバ (NTP, NFS, DNS サーバなど) がポートスキャン検知機構を誤動作させる傾向がある場合は、ポートスキャン検知プラグインに特定のホストからの TCP SYN や UDP ポートスキャンを無視するよう設定できます。このモジュールに対する引数は、無視する対象の IP アドレス/CIDR ブロックのリストです。

書式

```
portscan-ignorehosts: <host list>
```

```
preprocessor portscan-ignorehosts: 192.168.1.5/32 192.168.3.0/24
```

図 2.39: Portscan Ignorehosts モジュールの設定例

2.4.4 Frag2

Snort 1.8 で導入された Frag2 は新しい IP フラグメント再構築プリプロセッサです。Frag2 は (訳注:以前利用されていた)defrag プリプロセッサを置き換えるよう設計されています。このフラグメント再構築機構は、メモリの利用効率を高めるとともに、Snort の他の部分で利用されているものと同じメモリ管理ルーチンを利用することを目的としています。

Frag2 ではメモリ利用量とフラグメント化タイムアウトオプションを設定可能です。引数が未設定の場合、frag2 は 4194304 バイト (4MB) のデフォルトメモリと 60 秒のタイムアウト時間を利用します。タイムアウト時間は、まだ再構築が行われていないフラグメントを廃棄するまでの待ち時間を設定する際に利用します。

Snort 1.8.7 では、fragroute のような回避テクニックの利用を検知するために効果的なオプションがいくつか追加されました。

書式

```
preprocessor frag2: [memcap <xxx>], [timeout <xx>], [min_ttl <xx>], \
                    [detect_state_problems], [ttl_limit <xx>]
```

timeout <seconds> ステートテーブルに不活発なストリームを保持している時間、セッションが再び活動していることが確認されれば、掃き出し済みセッションが自動的に再び取り込まれます。デフォルトの値は 30 秒です。

memcap <bytes> メモリ容量をバイト数で設定し、この限界を超過すると積極的に frag2 が不活発な再構築済みパケットを取り除きます。デフォルトの値は 4MB です。

detect_state_problems 重複フラグメントのようなイベントに関するアラートを有効にします

min_ttl frag2 が受け付ける最低の TTL の値を設定します

ttl_limit アラートを出力しないようにする誤差値を設定します (初期フラグメント TTL +/- TTL リミット)

```
preprocessor frag2: memcap 16777216, timeout 30
```

図 2.40: Frag2 プリプロセッサの設定

2.4.5 Stream4

stream4 モジュールは Snort に TCP ストリームの再構築およびステートフル解析機能を提供します。堅牢なストリーム再構築機能を通じて、Snort は stick や snot 攻撃のような“ ステートレス ”攻撃を無視できるようになります。stream4 は同時に 256 を超える TCP ストリームを追跡できる大規模なスケーラビリティも提供します。stream4 はデフォルトの設定で 32,768 の同時 TCP 接続を処理できる拡張性を備えているはずです。

Stream4 は stream4 プリプロセッサ、stream4 に関連する再構築プラグインの 2 つの構成可能なモジュールが含まれています。以下に stream_assemble のオプションをリストアップします。

Stream4 形式

```
preprocessor stream4: [noinspect], [keepstats], [timeout <seconds>], \
            [memcap <bytes>], [detect_scans], [detect_state_problems], \
            [disable_evasion_alerts], [ttl_limit <count>]
```

noinspect ステートフル検査を無効にします

keepstats <logdir>/session.log にセッションサマリー情報を記録します

timeout <seconds> ステートテーブルに不活発なストリームを保持している時間、セッションが再び活動していることが確認されれば、掃き出し済みセッションが自動的に再び取り込まれます。デフォルトの値は 30 秒です。

memcap <bytes> メモリ容量をバイト数で設定し、この限界を超過すると積極的に stream4 が不活発な再構築済みパケットを取り除きます。デフォルトの値は 8MB です。

detect_scans ポートスキャンに対するアラートを有効にします。

detect_state_problems 回避的 RST パケット (evasive RST packets)、データにおける SYN パケット、範囲外のウィンドウシーケンス番号といったストリームイベントに対するアラートを有効にします。

disable_evasion_alerts TCP オーバーラップなどのイベントに対してアラートを無効にします

ttl_limit アラートを出力しないようにする誤差値を設定します

Stream4_Reassemble 形式

```
preprocessor stream4_reassemble: [clientonly], [serveronly],\
                                 [noalerts], [ports <portlist>]
```

clientonly コネクションのクライアント側にのみ再構築を提供します

serveronly コネクションのサーバ側にのみ再構築を提供します

noalerts 挿入または回避攻撃 (insertion or evasion attacks) の可能性があるイベントに対してアラートを出力しません

ports <portlist> - 再構築を実行するポートのリストを空白で区切りで指定します。all はすべてのポート番号に再構築を提供します。デフォルト値ではポート番号 21 23 25 53 80 110 111 143 および 513 向けに再構築を提供します。

注

snort.conf ファイル内で引数を設定せずに、stream4 と stream4_reassemble ディレクティブを設定した場合、表 2.6 と表 2.7 に示したデフォルトの値が設定されます。

stream4 は新しいコマンドラインスイッチとして “-z” を導入しています。TCP トラフィックにおいて-z スイッチを指定した場合、Snort はスリーウェイハンドシェイクを通じて確立されたストリーム、または協調的な双方向の活動が観測された場合のストリーム (つまりあるトラフィックが一方向になってしまい、RST または FIN 以外のトラフィックが送信者に戻ってくることが確認された場合) に対してのみアラートを発動します。-z が有効な状態の場合、Snort は完全に TCP ベース stick/snot 攻撃を無視します。

表 2.6: Stream4 のデフォルト値

Option	Default
Session Timeout	30 seconds
Session Memory Cap	8388608 bytes
Stateful Inspection	ACTIVE
Stream Stats	INACTIVE
State Problem Alerts	INACTIVE
Portscan Alerts	INACTIVE

表 2.7: Stream4_reassemble のデフォルト値

Option	Default
Reassemble Client	ACTIVE
Reassemble Server	INACTIVE
Reassemble Ports	21 23 25 53 80 143 110 111 513 1433
Reassembly Alerts	ACTIVE

2.4.6 Conversation

Conversation プリプロセッサを利用すると、Snort は TCP において *spp_stream4* で行われている方法ではなく、プロトコルに基づいて対話の状況を得ることができます。将来的には、これによってバイトカウントやファーストトーカー (first talker) の状況に対応するルールを記述できるようになるでしょう。

現在、Conversation プリプロセッサは stream4 と同じメモリ防御メカニズムを利用しており、DoS 攻撃の際に自衛できます。

Conversation プリプロセッサは、ネットワーク上で許容されない IP プロトコルを含んだパケットを受信するとアラートメッセージを生成できます。allowed_ip_protocols で許容するプロトコル番号のリストを設定すると、許容されないパケットを受け取った場合にアラートを発動して、そのパケットを記録します。

書式

```
preprocessor conversation: [allowed_ip_protocols <protonumbers|all>], \
                           [timeout <sec>], [alert_odd_protocols], \
                           [max_conversations <number>]
```

表 2.8: Conversation デフォルト値

Option	Default
allowed_ip_protocols	all
timeout	60
alert_odd_protocols	disabled
max_conversations	65335

2.4.7 Portscan2

このモジュールを利用ポートスキャンを検知できます。このモジュールは対話 (conversation) が新しく行われているものなのかを知るために、conversation プリプロセッサ (第 2.4.6 節) を必要とします。

このモジュールは高速 nmap スキャンのような高速スキャンの捕捉を意図したものです。

書式

```
preprocessor portscan2:  [scanners_max <num>], [targets_max <num>], \
                         [target_limit <num>], [port_limit <num>], \
                         [timeout <sec>]
```

scaners_max 同時にサポートするネットワークをスキャンするホストの数

targets_max ホストを表すために割り当てるべきノードの数

target_limit スキャンが呼び出される前にスキャナーが通信するホストの数

port_limit スキャナーが呼び出される前にスキャナーが通信するポートの数

timeout スキャナーの活動を忘れ去る秒数

表 2.9: Portscan2 のデフォルト値

Option	Default
scanners_max	1000
targets_max	1000
target_limit	5
port_limit	20
timeout	60

2.4.8 Portscan2 Ignoreports

これら二つのプリプロセッサは Portscan2 プリプロセッサの動作を一部変更し、特定の TCP および UDP ポートを宛先や送信元とするアラートを無視するよう設定するものです。

特定のポートを**宛先**とするアラートを無視するには、portscan2-ignoreports-to を利用します。特定のポートを**送信元**とするアラートを無視するには、portscan2-ignoreports-from を利用します。これら二つのディレクティブは snort.conf の portscan2 preprocessor の指定より後の行に設定する必要があります。また、ポート番号は空白区切りのリストで指定する必要があります。

書式

```
preprocessor portscan2-ignoreports-from: <port list>
preprocessor portscan2-ignoreports-to: <port list>
```

```
preprocessor portscan2-ignoreports-from: 53  80
preprocessor portscan2-ignoreports-to: 80 1080
```

図 2.41: Portscan2 Ignoreports モジュールの設定例

例

2.4.9 Telnet デコード

telnet_decode プリプロセッサを利用すれば、Snort はセッションデータからの telnet の制御文字列を正規化できます。1.9.0 以降の Snort では引数として (プリプロセッサを) 利用するポートのリストを指定します。また 1.9.0 では、Snort はパケットそのものから個々のデータバッファに正規化することで、rawbytes content 修飾子 (第 2.3.38 節) を使って生データを記録。または検査できます。

デフォルトの設定ではポート番号 21, 23, 25, 119 で動作します。

書式

```
preprocessor telnet_decode: <ports>
```

2.4.10 RPC デコード

rpc_decode プリプロセッサは、パケットをパケットバッファに正規化することで、複数のフラグメント化した RPC の記録を、単一のフラグメント化していない記録に正規化します。stream4 が有効になっている場合、クライアント側のトラフィックのみを処理します。デフォルト設定では、ポート番号 111 と 32771 で動作します。

書式

```
preprocessor rpc_decode: <ports> [ alert_fragments ] \
   [no_alert_multiple_requests] [no_alert_large_fragments] \
   [no_alert_incomplete]
```

2.4.11 Perf Monitor

このモジュールは、パフォーマンス統計データを元に、Snort のさまざまな状況を調整するために利用します。このモジュールの出力形式や引数形式は予告なしに変更されることがあります。

2.4.12 Http Flow

このモジュールを利用すれば、Snort は HTTP ヘッダの後の HTTP サーバーの応答を無視できるようになります。

2.5 出力モジュール

出力モジュールはバージョン 1.6 から新たに導入された機能です。出力モジュールを利用することで、Snort の書式や提示方法の柔軟性が高まります。出力モジュールはプリプロセッサや侵入検知エンジンに続き、Snort のアラートやログ取得サブシステムが呼び出された際に実行されます。ルールファイル中のディレクティブの形式はプリプロセッサのものに非常に類似しています。

表 2.10: RPC Decoder オプション

オプション	目的
alert_fragments	フラグメント化した RPC 記録に対してアラートを出力する
no_alert_multiple_requests	1 つのパケット内に複数の記録が存在する場合でもアラートを出力しない
no_alert_large_fragments	フラグメント化した記録の合計が 1 つのパケットを上回った場合でもアラートを出力しない
no_alert_incomplete	単一のフラグメント記録が 1 つのパケットのサイズを上回った場合でもアラートを出力しない

Snort 設定ファイルには複数の出力プラグインを指定することができます。同一のタイプ (log、alert) に複数のプラグインが指定されている場合は、イベントが発生した時点で、プラグインが順次スタックされ、呼び出されます。標準のログ取得およびアラートシステムと同様に、出力プラグインはそのデータをデフォルト設定の/var/log/snort に出力するか、あるいは (-l コマンドラインスイッチを利用して) ユーザが指定したディレクトリに出力します。

出力モジュールは、ルールファイルに output キーワードを指定することで実行時に読み込まれます。

```
output <name>: <options>

output alert_syslog: LOG_AUTH LOG_ALERT
```

図 2.42: 出力モジュールの設定例

2.5.1 Alert_syslog

このモジュールは syslog 機能にアラートを送信します (-s コマンドラインスイッチに酷似しています)。また、このモジュールを利用すれば、ユーザは Snort のルールファイル内でログ取得ファシリティや優先度を指定できるようになり、アラートを記録する際の柔軟性が高まります。

有効なキーワード

オプション

- LOG_CONS
- LOG_NDELAY
- LOG_PERROR
- LOG_PID

ファシリティ

- LOG_AUTH
- LOG_AUTHPRIV
- LOG_DAEMON
- LOG_LOCAL0

- LOG_LOCAL1
- LOG_LOCAL2
- LOG_LOCAL3
- LOG_LOCAL4
- LOG_LOCAL5
- LOG_LOCAL6
- LOG_LOCAL7
- LOG_USER

プライオリティ

- LOG_EMERG
- LOG_ALERT
- LOG_CRIT
- LOG_ERR
- LOG_WARNING
- LOG_NOTICE
- LOG_INFO
- LOG_DEBUG

書式

```
alert_syslog: <facility> <priority> <options>
```

2.5.2 Alert_fast

これは Snort のアラートを高速な 1 行形式で指定された出力ファイルに出力します。この方式はパケットヘッダのすべてを出力ファイルに出力する必要がないため、完全なアラートよりも高速にアラートを出力できる方式です。

書式

```
alert_fast: <output filename>

output alert_fast: alert.fast
```

図 2.43: 高速アラートの設定

2.5.3 Alert_full

パケットヘッダ全体を含んだ Snort のアラートメッセージを出力します。アラートは標準のログ取得先ディレクトリ (/var/log/snort)、またはコマンドラインで指定されたログ取得先ディレクトリに書き込まれます。

ログ取得先ディレクトリの下には、IP アドレス毎にディレクトリが 1 つ作成されます。これらのファイルは、アラートを引き起こしたパケットのデコード済みパケットダンプとなります。これらのファイルを作成するようにすると、Snort の性能が大幅に低下します。この出力方式は、ほとんどトラフィックがない状況以外にはお勧めできません。

書式

```
alert_full: <output filename>

output alert_full: alert.full
```

図 2.44: フルアラート設定

2.5.4 Alert_smb

このプラグインは、WinPopup アラートメッセージをこの出力プラグインの引数として指定されたファイルで指定された NETBIOS マシンに送信します。このプラグインは (一般に root 権限である)Snort と同じ特権レベルで、外部の実行形式のバイナリ (smbclient) を実行するため、このプラグインの使用はお勧めできません。Workstation ファイルの書式は、アラートを受信させたいホストの NETBIOS 名を 1 行につき 1 つ記述したリストです。

書式

```
alert_smb: <alert workstation filename>

output alert_smb: workstation.list
```

図 2.45: SMB アラートの設定

2.5.5 Alert_unixsock

UNIX ドメインソケットを設定し、アラートレポートを UNIX ドメインソケットに送信します。外部プログラムやプロセスは、このソケットをリッスンし、Snort のアラートやパケットデータをリアルタイムに受信できます。現在、この機能は実験段階のインタフェースです。

書式

```
alert_unixsock
```

```
output alert_unixsock
```

図 2.46: UnixSock alert 設定

2.5.6 Log_tcpdump

log_tcpdump モジュールは、パケットを tcpdump 形式のファイルに記録します。この機能は、後日、収集されたトラフィックの分析を tcpdump 形式のファイル読み込みに対応した多数のツールで行う場合に有効です。

このモジュールでは、出力ファイル名を単一の引数として指定できます。ファイル名の末尾には UNIX タイプスタンプの秒数が付加されることに注意してください。このことにより、個別の Snort の実行結果のデータをはっきり区別できます。

書式

```
log_tcpdump: <output filename>

output log_tcpdump: snort.log
```

図 2.47: Tcpdump 出力モジュールの設定例

2.5.7 Database

Jed Pickel 氏から提供された本モジュールは、各種 SQL データベースに Snort データを送信します。このモジュールのインストール、設定方法に関する詳細は [91]Incident.org Web ページで入手できます。このプラグインに対する引数は、ログ保存先のデータベース名とパラメータリストです。パラメータはパラメータ=引数という初期で指定されます。利用例については、図 2.48 をご参照ください。

書式

```
database: <log | alert>, <database type>, <parameter list>
```

以下のパラメータが利用できます。

host 接続先のホスト。ゼロ長以外の文字列が指定された場合、TCP/IP 通信を利用します。ホスト名の指定がない場合、ローカル UNIX ドメインソケットを通じて接続します。

port 接続先のサーバホストのポート番号、または UNIX ドメイン接続用のソケットファイル名の拡張子。

dbname データベース名。

user ユーザ認証用のデータベースユーザ名

password データベースがパスワード認証を要求する場合に利用するパスワード

sensor_name この Snort センサーに対する独自の名前を指定します。名前を指定しない場合は、名前が自動的に設定されます。

encoding パケットペイロードおよびオプションデータは 2 進数 (バイナリ) であるため、それをデータベースに保存するシンプルで移植可能な方法は存在しません。また、BLOBS はデータベース間の移植性の問題か

ら、利用していません。そこで、以下のエンコーディングオプションを提供していますので、お好みのオプションを選択できます。各オプションにはそれぞれ次のような長所と短所があります。

hex (デフォルト) バイナリを 16 進文字列として表現します。

保存要件 — バイナリサイズの 2 倍

検索性 — 非常に優れる

対人可読性 — 真の geek でもない限り読み取り不可能。後処理が必要

base64 バイナリデータを base64 文字列として表現します。

保存要件 — バイナリサイズの約 1.3 倍

検索性 — 後処理なしには不可能

対人可読性 — 読み取り不可能。後処理が必要

ascii バイナリデータを ASCII 文字列として表現します。このオプションはオプション中唯一、(訳注:オリジナルの) データを失うオプションです。非 ASCII データは.. として表現されます。なお、このオプションを選択しても、IP および TCP オプションに対するデータは 16 進数として表現されます。なぜならば、そのデータが ASCII であることに意味をなさないからです。

保存要件 — 一部の文字列がエスケープされるため (&,<,>)、バイナリより若干大きめ

検索性 — テキスト文字列の検索は非常に良いが、バイナリの検索は不可能

対人可読性 — 非常に優れる

detail どの程度詳細なデータを保存しますか? オプションは次の通りです。

full (デフォルト) アラートの引き金となったパケットの全詳細を記録します (IP/TCP オプションとペイロードを含む)

fast 最低限のデータだけを記録します。このオプションを選択する場合、解析アプリケーションでの可能性の一部を著しく制限されますが、それでも一部のアプリケーションにとって最良の選択肢です。このオプションでは次のフィールドが記録されます - (timestamp, signiture, source ip, destination ip, source port, destination port, tcp flags, protocol)

さらに、ログ取得方式やデータベースタイプを定義する必要があります。ログ取得のタイプには log と alert という 2 つのタイプがあります。log タイプを設定すると、データベースに対するログ取得機能をプログラムのログ機能に追加します。タイプを log に設定した場合、ログ出力チェーンでプラグインが呼び出されます。タイプを alert に設定すると、プラグインをプログラム内のアラート出力チェーンに追加します。

現在のバージョンのプラグインは以下の 4 つのデータベースタイプに対応しています。MySQL, PostgreSQL, Oracle, unixODBC 準拠データベースの 4 種類が対応データベースです。利用するデータベースに適合したタイプを設定してください。

注: この出力プラグインには tag キーワードを用いて生成されたアラートを処理する機能がありません。詳細については、第 2.3.32 節をご参照ください。

```
output database: log, mysql, dbname=snort user=snort host=localhost password=xyz
```

図 2.48: データベース出力プラグインの設定

2.5.8 CSV

CSV 出力プラグインを利用すれば、データベースに容易にインポート可能な書式でアラートデータを出力することが可能となります。このプラグインにはファイルへのフルパス、出力書式オプションの 2 つの引数が必要です。

書式オプションのリストは次の通りです。書式オプションがデフォルト設定の場合、書式オプションリストに記載された順に出力されます。

- timestamp
- msg
- proto
- src
- srcport
- dst
- dstport
- ethsrc
- ethdst
- ethlen
- tcpflags
- tcpseq
- tcpack
- tcplen
- tcpwindow
- ttl
- tos
- id
- dgmlen
- iplen
- icmptype
- icmpcode
- icmpid
- icmpseq

```
output alert_CSV: /var/log/alert.csv default

output alert_CSV: /var/log/alert.csv timestamp, msg
```

図 2.49: CSV 出力の設定

書式

```
output alert_CSV: <filename> <format>
```

2.5.9 Unified

統合 (Unified) 出力プラグインは、Snort イベントを記録する最速方式を目指したものです。本プラグインは、イベントを alert ファイルと packet log ファイルに記録します。alert ファイルには、イベントの高いレベルの詳細情報が含まれています (ip, protocol, port, message id)。一方、log ファイルには詳細なパケット情報が含まれます (関連する event id を含むパケットダンプ)。

両ファイルとも spo_unified.h に記載されているバイナリ形式で書かれています。Barnyard が利用可能になれば、Barnyard は現行の出力プラグインを新しいアーキテクチャに組み込み、ログ記録を行います。近いうちに、統合出力フォーマットは活動量の多いセンサーの Snort データを記録する標準方式となるでことしょう。Snort はリアルタイムなデータ収集にフォーカスする一方で、Barnyard はセンサーの効率を低下させることなく、複雑なログ記録方式を可能にします。

各ファイルを書き込む際に、ファイル名の末尾には UNIX タイプスタンプの秒数が付加されることにご注意ください。

書式

```
output alert_unified: <file name>

output log_unified: <file name>

output alert_unified: snort.alert

output log_unified: snort.log
```

図 2.50: Unified 設定例

2.5.10 Log Null

特定のタイプのトラフィックに対しては、アラートは出力するもののパケットログを登録しないルールを作成できると好都合な場合があります。Snort 1.8.2 では、log_null プラグインが導入されました。これは、-N コマンドラインオプションに相当するものですが、ルールタイプ内で動作させることができます。

書式

```
output log_null

output log_null  # like using snort -N

ruletype info {
    type alert
    output alert_fast: info.alert
    output log_null
}
```

図 2.51: Log Null 利用例

2.6 適切なルールの書き方

効率と速度を最大限に高める Snort のルールを書くために、、念頭に置くべき一般的概念がいくつかあります。

適切なルールには contents が含まれています。2.0 検知エンジンは、第 1 段階で設定別のパターンマッチを実行することにより、Snort の動作を少々変更します。content オプションが長ければ長いほど、マッチはより “正確” になります。ルールに content オプションが含まれていなければ、システム全体の性能が低下してしまいます。

ルールを書き込む場合に、exploit の詳細 (ここではこのシェルコード) ではなく、脆弱性に的を絞った (たとえば、1025 以上の offset でこの手順を呼び出すことなど) ルールを書くように心掛けてください。

content ルールは大文字と小文字を区別します (ただし、nocase オプションを利用する場合を除く)。

Content ルールが大文字と小文字を区別し、多くのプログラムではコマンドを表示するために一般に大文字を利用することを念頭に置いてください。FTP はその好例といえます。次の 2 つのルールをご覧ください。

```
alert tcp any any -> 192.168.1.0/24 21 (content: "user root"; \
      msg: "FTP root login";)

alert tcp any any -> 192.168.1.0/24 21 (content: "USER root";\
      msg: "FTP root login";)
```

上記の 2 つ目のルールはほぼすべての自動的な root ログイン試行を検知しますが、user に小文字を利用する試行を検知できないので注意する必要があります。インターネットデーモンは入力として受け取るものに関して寛大です。ルールを書く際には、プロトコルが受け付けるものを把握できれば、攻撃を見逃す件数を最小限に抑えることが可能となります。

第3章　Snortの開発について

本章は現在、プレースホルダー段階にあります。将来、新しい検知プラグインやプリプロセッサの作成方法に関するリファレンスが盛り込まれることでしょう。エンドユーザは本章を必ずしも読む必要はありません。本章は、何が起こっているのかを開発者が敏速に把握できるように用意されたものです。

Snort 開発に支援の手をさしのべるつもりがおありでしたら、CVS の HEAD ブランチをご利用ください。我々は、もっぱら stable ブランチ宛にパッチを提出される方がいるために困っています (御自身の IDS 目的のためにパッチを書き込んでいるようですので)。バグフィックスについては、STABLE 宛に提出してください。機能については、HEAD 宛に提出してください。

3.1　パッチの提出

Snort へのパッチを `snort-devel@lists.sourceforge.net` メーリングリストに、CC 先を cmg@snort.org にして、パッチの件名: <subject>を添えてお送りください。

パッチのサイズが 20K に満たない場合は、gzip で圧縮しないでください。コマンド `diff -Nu snort-orig snort-new` を使ってパッチをしてください。

3.2　Snortデータフロー

最初に、トラフィックはネットワークから libpcap を通じて収集されます。パケットは最初にリンク層のプロトコル解析のため、パケットの構造体にデータを書き込み、一連のデコーダルーチンを通り抜け、TCP や UDP ポートなどのためにさらにデコードされます。

次に、パケットは登録済みプリプロセッサセットへと送られます。各プリプロセッサは、そのパケットを調査するべきものかか確認します。

さらに、パケットは侵入検知エンジンへと送られます。侵入検知エンジンは Snort ルールファイルに列挙された様々なオプションに基づき各パケットをチェックします。各キーワードのオプションはプラグインです。これによって、拡張性が確保されます。

3.2.1　プリプロセッサ

たとえば、パケットに TCP ヘッダが含まれていない場合に、TCP 解析プリプロセッサは簡単に処理から戻ってくることができます。以下の説明で確認できます。

```
if (p->tcph==NULL)
   return;
```

同様に、パケットに “再構築済み” またはログ取得済みのフラグを付けるため、多くの packet_flags が利用できます。PKT_*定数のリストについては src/decode.h をご確認ください。。

3.2.2　検知プラグイン

基本的に、既存の出力プラグインを調査し、それを新しい項目にコピーし、多少の変更を行います。後で、我々はこれらの多少の変更の内容を文書化します。

3.2.3　出力プラグイン

総じて、新しい出力プラグインは Snort プロジェクトではなく、Barnyard プロジェクトに回されるべきです。現在、我々は利用可能な出力オプションに関して調整している最中です。

関連図書

[1] http://packetstorm.securify.com/mag/phrack/phrack49/P49-06

[2] http://www.nmap.org

[3] http://public.pacbell.net/dedicated/cidr.html

[4] http://www.whitehats.com

[5] http://www.incident.org/snortdb

[6] http://www.sourcefire.com/services/advisories/sa032803.html