Snortによるサンプリングデータからの不正通信の検出

藤原健志† 永山聖希† 吉田和幸††

近年，インターネットの普及に伴い，ネットワークを通して様々な情報のやり取りが行われている．管理者は管理対象となるネットワークを常に安全に保つためにトラフィックやパケットの監視を行う必要がある．しかし，ネットワークの広帯域化，多様化によりネットワーク上を流れるパケット一つ一つを解析する事は困難になりつつある．そこで我々はパケットサンプリングに基づくトラフィック管理技術「sFlow」をネットワーク型IDS「snort」に適用することを考えた．本論文ではsFlowで取得したデータをSnortに適用させるための実験とその結果について述べる．

Detecting bad traffic by Snort from sampling data

TAKESHI FUJIWARA† TOSHIKI NAGAYAMA† KAZUYUKI YOSHIDA††

Recently, as the Internet spreads, various information is exchanged through the Internet. To keep network safely and securely, network administrators should monitor traffic and the packet. However, analyzing each one of the packet is becoming difficult by the wider bandwidth and the diversification of the network. Then, we thought traffic management technology based on packet sampling "sFlow" was applied to Network type IDS "snort". In this paper, the experiment and the result to apply the data acquired with "sFlow" to "Snort" are described.

1. はじめに

近年，コンピュータネットワークは様々な場面で利用されている．一方でコンピュータウイルス，コンピュータワーム，ホストの存在探索（スキャン），不正侵入等様々な脅威にさらされている．その不正通信への対処は，各ホストの管理者や各ネットワークの管理者に委ねられている．ゆえに，ネットワーク管理者は安全性の高いネットワークを実現するために，ファイアウォール，IDS（Intrusion Detection System：侵入検知システム）やトラフィックの監視ツールを導入することが必要とされる．

IDSはネットワーク上を流れるパケットを取得し，解析することにより意図しないパケットかどうかを判断しそれを検出する．しかし，ネットワークの広帯域化，多様化によりネットワーク上を流れるパケット１つ１つを解析する事は困難になりつつある．我々はこうした背景からパケットサンプリングに基づくトラフィック管理技術「sFlow」とオープンソースで開発が行われているネットワーク型IDS「snort」を組み合わせることで，DoS攻撃やP2P等大量のトラフィックが発生する通信もリアルタイムに低負荷での検出を行うシステムを構築した．本論文ではsFlowデータに対してSnortを適用させるための実験とその結果について述べる．

以下，2，3章では本研究の関連技術であるSnortとトラフィック管理技術について紹介し，4章ではsFlowデータに対してSnortを適用させる．その結果を評価する．5章では本研究の関連研究について紹介する．6章では本研究に関するまとめを述べる．

2. Snort

2.1 Snort概要

Snort[1]は多彩な機能を備えたオープンソースのネットワーク型IDSで，Martin Roesch氏によって開発された．当初Snortはパケットスニファとして開発され，1999年1月にシグネイチャ（不正アクセスのパターン）に基づくパケット解析機能が追加されIDSとしての機能を備えた．その後も機能拡張を続け，プリプロセッサによる異常検出機能の追加，性能や安定性を向上させ続けている．異常検出とは通常の通信やユーザの行動をプロファイルとして記録し，それらと異なった通信や動きを不正として検出を行う方法である．最新の安定版はSnort 2.8.6.1であり，本論文でもSnort 2.8.6.1を用いて研究を行っている．以下ではSnortの基本アーキテクチャを紹介し，侵入検出に関わるプリプロセッサとルールについて説明する．

2.2 Snortの基本アーキテクチャ

Snortはパケットスニファ，プリプロセッサ，検知エンジン，アウトプットプラグインの4つの基本的なコンポーネントから構成されており，次のような動作を行っている．各番号は図 1の番号に対応している．

① パケットキャプチャ処理
（libpcapライブラリ経由でキャプチャを行う）
② パケットを検知エンジンに送るか判断
検知エンジン用にパケットを加工
（プリプロセッサによっては異常検出も行う）
③ ルールに該当するかマッチングを行い，
マッチするとアウトプットプラグインに送る
④ アラートやログを出力

† 大分大学大学院工学研究科
Department of Computer Science and Intelligent System, Oita University
†† 大分大学学術情報拠点情報基盤センター
Center for Academic Information and Library Services, Oita University

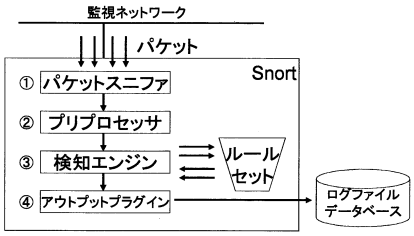


図 1 Snort のアーキテクチャ

2.3 Snort のプリプロセッサ

Snort で監視するパケットは検知エンジンに送られる前に，まずプリプロセッサに送られる．プリプロセッサには大きく分けて二つの役割がある．一つはプロトコルが正しくやり取りされているかの検査（正規化/normalization），もう一つは異常な振る舞いをしていないかを検査する役割を担っている．それぞれで，異常が見つかれば警告を出す．

正規化は，IP パケットのフラグメント（分割）や TCP のようにデータを分割して送ることが可能な場合，正しく検査を行なうには再組み立てが必要である．そのためのプリプロセッサとして frag3，stream5 が用意されている．frag3 は IP フラグメントの再構成を行い，stream5 は TCP ストリームの再構成及び処理状態（ステートフル）の解析を行なう．これら以外にもポートスキャン等のシグネイチャでは検知できない異常な振る舞いを検出するプリプロセッサが Snort には用意されている(表 1).

表 1 代表的なプリプロセッサ

プリプロセッサ	機能
frag3	IP のデフラグメンテーション
stream5	TCP の再構築・ステートフルの解析
http_inspect	HTTP の正規化
sfportscan	ポートスキャンの検出
smtp	SMTP の正規化
ftp_telnet	Telnet, FTP のネゴシエーションを正規化
ssh	SSH の異常検出
arpspoof	ARP 詐称を検出
bo	Back Orifice のトラフィックを解析

2.4 Snort のルール（シグネイチャ）

Snort には最新の不正アクセス手法を検知するためのルールセットには公式に開発するチーム Sourcefire Vulnerability Research Team(VRT)により検証されたルールセット「Sourcefire VRT Certified Rules」と Snort コミュニティ(Emerging Threats 等)で作成されたルールセット「Community Rules」が存在し，シグネイチャの種類は増え続けている．その結果，数多くのシグネイチャが存在し，それらはルールタイプに従って分類されるようになった．ルールタイプは P2P, バックドア，DoS 攻撃，web に対する不正アクセス，そのほか多くから構成される．それぞれのルールには Snort rule ID (SID)が割り当てられ，番号でルールを識別することができる．表 2 の SID の割り当て範囲から，独自のルールを作る際，SID には 1,000,000 以上の値を割り当てる必要がある．本論文で利用する Snort のルールセットは Sourcefire VRT Certified Rules（3 Aug, 2010）[脚注a]である．

また Snort のルールはルールヘッダとルールボディで構成されている（表 2）．観測パケットの情報がルールヘッダの情報と一致するとルールボディとマッチングが行われ，マッチすればアラートを発する等のルールヘッダで定義されたルールアクションを実行する．

表 2 Snort rule ID の割り当て範囲

範囲	用途
100 未満	予約済み
100 から 1,000,000	VRT が提供するルール用
1,000,000 以上	ローカルルールとして利用可能

2.4.1 ルールヘッダ

ルールヘッダは次の 4 つの要素で構成されている．

① ルールアクション：マッチした場合の処理
② プロトコル：ICMP，TCP，IP，UDP の 4 種類
③ ソース情報：送信元の IP，ポート
④ ディスティネーション情報：宛先の IP，ポート

2.4.2 ルールボディ

ルールボディはルールヘッダの後に置かれ，様々なルールオプションを組み合わせることで不正通信の検出を行う．表 3 に主なルールオプションと本論文に関わるルールオプションを示す[2].

表 3 ルールオプション

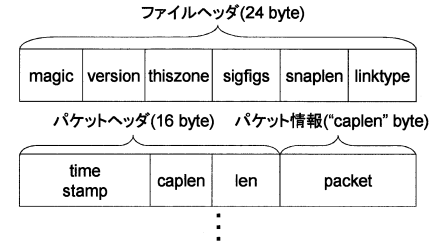
◆ ルール情報オプション (オプション数：9)	
msg	アラートに出力するメッセージを指定
sid	Snort ルールを識別する番号．割り当て範囲は表 2 に示す．
classtype	攻撃の種別に応じた重要度を付けるために用いる．classtype は全て定義されている．
priority	ルールの重要度を示す値．classtype で指定された重要度を上書きできる．
● ペイロードへの検知オプション (オプション数：33)	
content	パケットのペイロード中でマッチングを行う ASCII またはバイナリデータを記述する．
offset	パターンマッチの検索開始位置をペイロードの先頭から何バイトかを指定する．
● ヘッダ等への検知オプション (オプション数：22)	
flags	TCP ヘッダのフラグオプションを検査．
flowbits	フローの追跡を行う．これにより「ログイン状態にある場合のみマッチングを行う」等が可能となる．
flow	フローの方向性を定義するオプション．Stream5 との同調により方向性を特定する．
● その他のオプション(オプション数：11)	

a 現在，無料登録ユーザは Sourcefire VRT Certified Rules の公開から 30 日遅れで入手可能．

3. トラフィック管理技術

3.1 tcpdump / libpcap

tcpdump [3]とはオープンソースのネットワーク分析監視ツールである．導入されたコンピュータのネットワークインタフェースに到達するパケットを表示・記録保存でき，IP,TCP,UDP,ICMP 等の各種プロトコルに対応している．tcpdump はパケットキャプチャ・ライブラリとして libpcap を利用しており，パケットの保存は libpcap のフォーマット（以下，pcap 形式）に従って行われる(図 2)．libpcap は様々なアプリケーションで利用されており，2 章で紹介した Snort にも利用されている．そのため Snort は pcap 形式で保存されたファイルからパケット情報を取得することが可能である．



magic	magic number(A1B2C3D4)：endianの判定
version	libpcapダンプファイルのバージョン
thiszone	GMT(グリニッジ標準時)との時差(±12)を記述
sigfigs	タイムスタンプ精度
snaplen	設定されたパケットキャプチャサイズ
linktype	データリンク層のフレームの形式
timestamp	タイムスタンプ
caplen	キャプチャしたパケットの長さ
len	元のパケットの長さ

図 2 pcap 形式のファイルフォーマット

3.2 sFlow

sFlow[4]は指定したインターフェースを通過するパケットに対してあらかじめ設定されたサンプリングレートでパケットを観測する．観測されたパケットは設定されたパケットサイズに切り詰められ，sFlow データグラムというパケットに格納される．sFlow データグラムには，sFlow のヘッダ，および，サンプリングされたパケットが複数個格納されている．また sFlow データグラムには，上記のデータ以外にも，Simple Network Management Protocol (SNMP)を利用したときに取得できるルータの各統計情報も送信するため，機器のトラフィック量等を格納することもできる．完成した sFlow データグラムはコレクタと呼ばれる収集サーバに UDP パケット(sFlow パケット)として送信され，コレクタ側は sflowtool [5]等のツールを用いることで sFlow パケットを受信し解析する．sflowtool では受信した sFlow パケットからサンプリングされたパケット情報を取得し pcap 形式でパケットを保存することが可能であることから，tcpdump 等他のツールを用いてパケットの観測も可能である．またサンプリングされたパケットはそのまま記録されるため，パケットのヘッダ情報等は元のパケットの情報を保持している．

4. sFlow と Snort による不正通信の検出

4.1 実験環境

本論文における sFlow データは図 3 のように FW の外側に設置された LAN-SW から収集を行なう．また検出精度を確認するために FW 外側の LAN-SW からミラーリングを行い，パケットをフルダンプし実験を行なう．表 4 に sFlow データの収集を行う LAN-SW の機器名称，sFlow に関する設定についてまとめる．

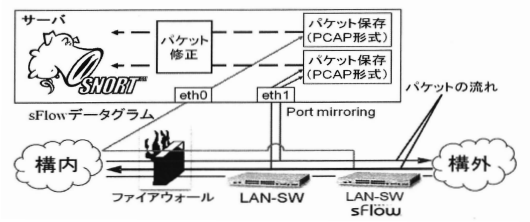


図 3 実験環境

表 4 sFlow 対応 LAN-SW

機器・設定	詳細
LAN-SW	Stackable FESX424+2XG
サンプリングレート	1 / 1024
取得パケットサイズ	先頭 128 byte
収集対象パケット	構内へのインバウンドトラフィック
収集期間	2009/08/01 ～

4.1.1 Snort の設定

表 5 に 4.4 節以降の実験で用いられる Snort の設定において重要な設定を記す．また実験ではプリプロセッサと検知エンジンによるアラートの結果を調査したいため，パケットデコーダによるアラートは発しないように設定する．パケットデコーダは libpcap ライブラリを経由してキャプチャしたパケットをデータリンク層からトランスポート層レベルまでの解析とデコード処理を行い，プリプロセッサにパケットを渡す．各層でどのようなプロトコルのパケットであるか等の解析を行うが，パケットのチェックサムやパケット長等で問題があればアラートを発する．

4.2 sFlow データの Snort への適用に関する問題点

sFlow データを Snort への適用するにあたり次のような問題点が挙げられる．

1. sFlow はパケットを取得する際にパケットサイズを先頭 256byte 以下に切り詰める．そのため sFlow データを Snort に適用させても検知エンジンに届く前に破棄される．
2. サンプリングが行われるため，TCP ストリームの再構築やステートフルな解析は不可能である．
3. sFlow データはサンプリングされ，先頭 256byte 以下に切り詰められるため，Snort に適用させた場合に誤検出の発生や検出不可能な不正通信が存在するかもしれない．

表 5 Snort の設定

設定	説明・設定
checksum_mode	チェックサムの計算.
	チェックサムの計算を行わないように設定する.
Stream5_global	Stream5 の全体設定.
	TCP, UDP を Stream5 の監視対象とし, ICMP は Stream5 の監視対象から外す.
timeout (stream5_tcp,udp)	TCP, UDP のセッションの監視を止めるまでのアイドル時間の設定する.
	sFlow データに対しては 1 時間を設定し, フルダンプデータに対しては 180 秒を設定する.
non_rfc_char (http_inspec)	非 RFC 文字が存在する場合にアラート
	データに付加する 0x00 がアラートとなるため無効化する.
oversize_dir_length (http_inspec)	ディレクトリを示す文字列の最大値
	"/"が付加されないため無効化する.

4.3 sFlow データの Snort への適用方法の検討

4.3.1 パケット情報を修正

sFlow から得られたデータを PCAP 形式で保存し, Snort に適用させたがアラートを殆ど発生させなかった. この原因を調査したところ, 殆どのパケットが Snort の検知エンジンに届く前に破棄されていることがわかった. 3.2 節で記述したように sFlow で取得したパケットは設定したパケットサイズに切り詰められるため, 実際のパケットサイズと異なるためである.

そこで取得したパケット情報に次のような修正を行い, Snort に適用し検出率の良いもので運用を行う.

- 各ヘッダ情報のパケットサイズを切り詰められた状態に合わせたサイズに修正.
- パケットサイズを切り詰められたことにより破棄された部分に適当なデータを付加し元のパケット情報を再現.

4.3.2 ルールの変更

sFlow ではサンプリングによって全てのパケットが取得できないため TCP ストリームの再構築やステートフルな解析は不可能である. このことから Stream5 と同調を行うオプションをもつルールの検出が不可能となる. そこでルールに対して次のような修正を行う.

- flowbits オプションの無効化する.
- flow オプションの無効化する.
- 修正したルールの sid に 1,000,000 を加える.

flowbits はルールを複数定義し, 監視しているセッションのパケット①がルール A にマッチすれば状態 A をセットし, 状態 A がセットされていればルール B にてパケット②のマッチングを行う. 状態 A をセットするためのパケットを取りこぼした場合にはルール B にマッチングするパケットが流れてもマッチングが行われないため, パケットサンプリングを行う sFlow での検出は難しい. 本論文では flowbits オプションを無効化したルールに修正し, 検出できるかを試みる.

flow オプションは表 6 のような内容となっており, TCP では 3way-handshake が確立された状態でしかマッチングを行わない等が可能となる. しかし, sFlow では 3way-handshake のパケットが全て取得できるとは限らず, 現在の sFlow の収集対象パケットはインバウンドトラフィックのみであることから確立されたかの確認は不可能である. 本論文では, 現在のルールの flow オプションを全て無効化したルールに修正する.

また表 2 より修正したルールは sid に 1,000,000 を加え, ローカルルールとして運用を行う. ただしフルダンプしたデータに対しては. 通常の VRT が提供するルールで検出を行う. これらのルールの修正は誤検出に繋がるため, 今後これらの修正を行わずに検出可能にする仕組みが必要となる.

表 6 flow オプション

オプション	内容
stateless	状態に関係しない.
established	確立されたセッションのパケット.
to_client	クライアント向きに送られるパケット.
to_server	サーバ向きに送られるパケット.
from_client	クライアントから送られるパケット.
from_server	サーバから送られるパケット.
only_stream	再構築されたパケットや確立された通信を流れるパケット.
no_stream	再構築されたパケットや確立された通信を流れるパケットを除外したもの.

4.4 実験 1. Snort に有効な形式の調査

本節では sFlow で取得したデータを Snort へ適用可能な形に変換するために, 様々なパケットの状態で Snort による解析を行った.

4.4.1 調査方法

フルダンプしたデータの 1 つ 1 つのパケットに対して, 次のような修正を行い Snort による解析を行なう.

【pcap】pcap ヘッダとパケット情報に対する修正点

1. 「 Raw 」
   修正なし.
2. 「 sFlow 」
   sFlow で取得するデータと同じ状態にする. pcap ヘッダのキャプチャサイズを 256byte 以下に修正し, 先頭 256 byte 以外のデータを破棄する.
3. 「 ChangeLen 」
   状態「sFlow」から pcap ヘッダの元のパケットサイズの情報をキャプチャサイズに変更する.
4. 「 Padding 」
   先頭 256 byte 以外のデータを破棄し, 破棄した部分に NULL 文字 (0x00) を付加 (パディング).

【ヘッダ】IP,TCP,UDP ヘッダに対する修正点

1. 「 - 」
   修正なし.

2. 「 IP 」
   IP ヘッダのパケット長の情報をキャプチャサイズに合わせた情報に修正する
3. 「 IP + UDP 」
   IP, UDP ヘッダのパケット長の情報をキャプチャサイズに合わせた情報に修正する.
4. 「 checksum 」[脚注b]
   IP, UDP ヘッダのパケット長に関する情報をキャプチャサイズに合わせた情報に修正し, 各ヘッダのチェックサムを再計算する.

本実験では sFlow の規格から最大のキャプチャサイズが 256 byte であるため, 状態「sFlow」ではパケットサイズを先頭 256byte に切り詰めた. また状態「Padding」ではヘッダ情報を修正しないときパケット長が正しい値となるため, ヘッダ情報が「IP」,「UDP」の場合は修正の必要がないため記述しない.

本実験は 2010 年 09 月 04 日(土) 00:00:00 ~ 23:59:59 に取得したパケット (157,336,412 パケット[脚注c]) を対象とする. 発生したアラートの正検出, 誤検出の判断は, 状態「Raw」のデータに対して Snort がアラートを発したパケットを全て正検出とし, それ以外を誤検出とする. また, 状態「Raw」で検出したパケットが他の状態でどれ程検出されたかを確認するために, 次のような式を用いて一致率と誤検出率を計算する.

- 一致率　= 正検出 ÷ 状態「Raw」検出数
- 不一致率 = 誤検出 ÷ 状態「Raw」検出数

4.4.2 実験結果

表 7 に各状態のアラート関する件数, 一致率, 不一致率を記し,
表 8 にアラートの種類の数を記す.
表 9 に正検出したアラートを分類ごとにその数と種類の数を記す. 比較を行うために次に 4 つの状態のみを記述する.

- 「Raw」「-」 フルダンプした結果
- 「sFlow」「-」 sFlow データをそのまま適用させた場合を想定
- 「sFlow」「IP, UDP」 一致率が高く, 不一致率が低いため
- 「Padding」「-」 正検出率が最も高いため

この実験結果より次のことがわかった.

1. 状態「sFlow」と状態「ChangeLen」が全く同じ結果であることから, pcap ヘッダのパケットサイズは Snort による解析には何も影響しない.
2. 各状態の「checksum」は IP ヘッダ, UDP ヘッダが正しい値であるものと結果が同じことから, Snort の設定においてチェックサムの計算を行わないならば, チェックサムの再計算は不要である.

表 7 実験結果 (アラートの件数)

状態		アラート数			一致率	不一致率
pcap	ヘッダ	検出数	正検出	誤検出		
Raw	-	44,167	44,167	-	100%	-
sFlow	-	10,066	6,722	3,344	15.2%	7.6%
	IP	17,830	9,028	8,802	20.4%	19.9%
	IP+UDP	47,678	38,961	8,717	88.2%	19.7%
	checksum	47,678	38,961	8,717	88.2%	19.7%
ChangeLen	-	10,066	6,722	3,344	15.2%	7.6%
	IP	17,830	9,028	8,802	20.4%	19.9%
	IP+UDP	47,678	38,965	8,713	88.2%	19.7%
	checksum	47,678	38,965	8,713	88.2%	19.7%
Padding	-	86,151	39,925	46,226	90.4%	104.7%
	checksum	89,940	39,074	50,866	88.5%	115.2%

表 8 実験結果 (アラートの種類)

状態		アラート数		
pcap	ヘッダ	検出数	正検出	誤検出
Raw	-	42	42	-
sFlow	-	15	14	8
	IP	23	22	11
	IP+UDP	24	23	10
	checksum	24	23	10
ChangeLen	-	15	14	8
	IP	23	22	11
	IP+UDP	24	23	10
	checksum	24	23	10
Padding	-	38	30	11
	checksum	36	26	19

3. 状態「sFlow」ではヘッダ情報の「修正なし」に比べ「IP,UDP ヘッダのパケット長を修正」の方が検出性能の良いことから, IP, UDP ヘッダ内のパケット長は元のパケットサイズではなく, キャプチャサイズに合わせた修正をする必要がある. 状態「Padding」ではデータを付加するだけで IP,UDP ヘッダの情報に修正する必要がない.
4. 最も正検出の数と種類が多いのは状態「Padding」のヘッダ情報修正なし (以下, Padding 方式) である. さらに表 9 から Raw を除いた他の状態で検出しているアラートは全て検出している.

4.4.2.1. Padding 方式における誤検出

Padding 方式における誤検出を調べたところ, 次のアラートが 43,508 件(94.1%)を占めていた.

・VOIP-SIP from header field buffer overflow attempt

このアラートは VoIP(Voice over Internet Protocol)と呼ばれる IP ネットワーク上で音声通信を行えるようにするために開発された技術におけるアラートである. ルールでは送信先ポート : 5600 のパケットにおいてパケット情報の特定の位置から「ラインフィード(0x0A)とキャリッジリターン(0x0D)以外の ASCII コード」(=改行コード以外の文字コード) が連続して 256 バイト以

b 本実験での snort の設定では UDP, TCP の checksum の確認はしない.
c Received (186,673,231 packets) Dropped by kernel (29,336,791 packets)

表 9 実験結果（正検出・アラート分類）

パケット状態 / Classtype [priority]	正検出数（種類）			
	Raw	sFlow	sFlow	Padding
	-	-	IP, UDP	-
Executable Code was Detected [1]	6,350(9)	22 (1)	2,341(7)	2,345(9)
Attempted Administrator Privilege Gain [1]	32 (3)	-	-	-
Attempted User Privilege Gain [1]	20 (4)	-	-	1 (1)
A System Call was Detected [2]	36 (2)	-	5(2)	5(2)
Misc activity [3]	8 (3)	-	-	-
アラート [priority]	正検出数			
SQL version overflow attempt [1]	29,810	-	29,810	29,810
ATTACK-RESPONSES 403 Forbidden [2]	3	-	-	3
SPYWARE-PUT Trackware casalemedia [2]	2	-	-	-
ORACLE misparsed login response [2]	1	-	-	-
プリプロセッサ	正検出数（種類）			
http_inspect	4(1)	-	-	-
portscan	3,250(8)	2,244(7)	2,360(6)	3,250(8)
frag3	11 (2)	8 (2)	11 (2)	11 (2)
smtp	174(1)	-	-	36(1)
ftp_telnet	77 (4)	55 (3)	35(3)	65(4)
Ssh	4,399(1)	4,393(1)	4,399(1)	4,399(1)

上存在した場合にアラートを発するものであった．他の誤検出に関しても同様に特定の ASCII コード以外が一定以上連続して存在した場合のアラートであった．

4.4.2.2. Padding 方式の付加するデータを変更

Padding 方式において付加するデータをラインフィード(0x0A)に変更し実験を行ったところ，誤検出率が大幅に改善されることが確認できた．しかし，今度は ftp_telnet のプリプロセッサから次のアラートが 10,211 件(91.3%)の誤検出を発生させた．

・FTP response message was too long

FTP は ASCII コードをベースとしたコマンドとレスポンスによって実現されている．レスポンスはレスポンスコード（3 桁の応答結果を示す番号）とメッセージやステータスから構成される．メッセージが規定サイズ(256 byte)よりも大きくなってしまったため誤検出されていた．これらのことから FTP に関しては 0x00 のコードを付加するのが適当であると考えられる．

4.4.2.3. 実験結果のまとめ

表 10 に今回の実験結果をまとめた．比較を行うために表 7，表 8 から 3 つを抜き出し，新たに調査した 2 つに関して記す．表の各状態は次の通りである．

- Raw : 状態「Raw」
- sFlow : 状態「sFlow」 - 「IP+UDP」
- Padding<0x00>: Padding 方式(0x00 を付加)

表 10 実験結果・まとめ

状態	アラート数（種類）			一致率	不一致率
	検出数	正検出	誤検出		
Raw	44,167 (42)	44,167 (42)	-	100%	-
sFlow	47,678 (24)	38,961 (24)	8,717(10)	88.2%	19.7%
Padding<0x00>	86,151 (38)	39,925 (30)	46,226 (23)	90.4%	104.7%
Padding<0x0A>	51,071 (33)	39,889(29)	11,182 (5)	90.3%	25.3%
Padding <New>	40,860 (33)	39,889(29)	971 (4)	90.3%	2.2%

- Padding<0x0A>: Padding 方式(0x0A を付加)
- Padding<New>: Padding 方式(FTP の場合 0x00, 他の場合 0x0A を付加)

4.4.3 考察

sFlow 取得データに対しては次の 2 つの処理を行なうことで Snort に対して有効なデータを作成できる．

① IP, UDP ヘッダのパケット長に関する情報をキャプチャサイズに合わせたパケット長に修正
② 元のパケット情報から破棄された部分に特定のデータを付加する

今回の実験では①と②で正検出数に大きな違いはないが，②のほうが多くの種類を正検出している．さらに誤検出に関しても付加するデータによって①よりも誤検出が少なくなる．ポートやアプリケーションを考慮することで誤検出を減少させられることから，今後更に誤検出を減らすことが期待できる．

4.5 実験 2. 取得パケットサイズの影響

sFlow ではパケットサイズを設定したパケットサイズに切り詰められる．本校に存在する sFlow 対応 LAN-SW の設定可能な範囲は 0~256 byte となっており，デフォルトでは 128 byte が設定されている．取得パケットサイズの影響を調査し，今後必要ならば設定サイズを変更する．

4.5.1 実験方法

実験 1 で行ったようにパケットサイズを設定サイズまで切り詰め，各パケットサイズの結果を比較する．パケットの修正は 4.4 節でもっとも正検出が多かった Padding 方式で修正を行う．ただし，誤検出を減らすために付加するデータは FTP 通信では 0x00, 他の場合は 0x0A とする．

比較するパケットサイズは次の 5 つである．

1. 「Raw」: 修正なし
2. 「256」: 先頭 256 byte
3. 「192」: 先頭 192 byte
4. 「128」: 先頭 128 byte
5. 「Head」: ヘッダ情報のみ(IP, TCP, UDP, ICMP)

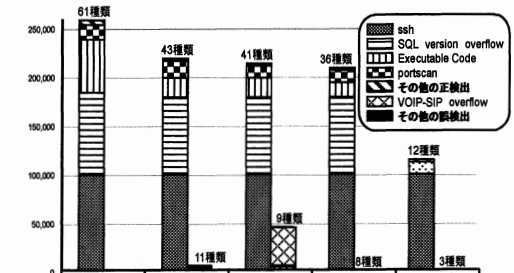
本実験は 2010 年 09 月 03 日(金) 00:00:00 ~ 2010 年 09 月 05 日(日) 23:59:59 の 3 日間に取得したパケット（622,336,898 パケット[脚注d]）を対象とする． 発生

d Received (730,499,105 packets) Dropped (108,062,364 packets)

したアラートの正検出，誤検出の判断は，「Raw」のデータに対して Snort がアラートを発したパケットを全て正検出とし，それ以外を誤検出とする．

4.5.2 実験結果

図 4 に正検出・誤検出のアラートの合計を示し，アラートの割合も示す．また図 5 では「Raw」の結果を100%としたときの正検出のアラートの減少率を示す．



	Raw	256	192	128	Head
正検出	256,566	216,577	212,754	209,685	115,889
誤検出	0	4,123	47,858	53	116

図 4 実験結果：正検出と誤検出（3 日分）

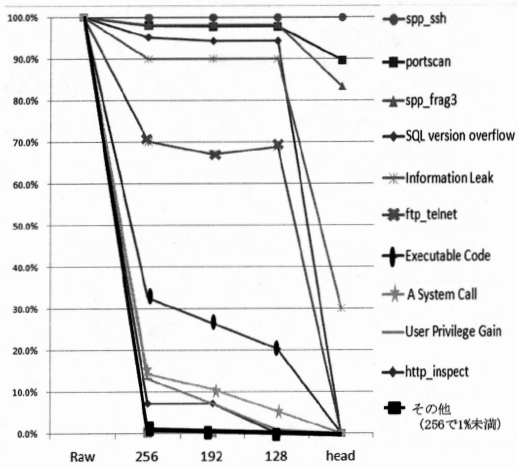


図 5 実験結果：正検出のアラート減少率

この実験結果より次のことが分かった．

1. 「Raw」に比べ 256byte 以下のデータで検出可能なアラートは 20 種類程減少していることが確認できた．これは 256byte 以上のパケット情報によるマッチングが必要なためである．
2. 検出数，種類とも「256」，「192」，「128」の順に多いが「Head」のように大きな差は存在しない．
3. 256 byte 以下に切り詰めても大量のアラートを発生させているパケットの検出率は落ちていない．これは 128byte 以内のパケット情報でのマッチング（24 種類）もしくはヘッダ情報に基づいた検出（12 種類）が行われているためである．
4. 「192」で誤検出数が多くなったように，切り詰めるサイズによってはパケットの重要なコードが途中で切れてしまい，誤検出されている場合がある．
5. sFlow で取得したデータに対してヘッダ情報だけの収集では検知できないアラートが約 10 万件(24 種類以上)存在する．

4.5.3 考察

sFlow で取得するパケット情報はヘッダ情報以降の情報を持っており Snort での検出に大きく影響する．また sFlow の取得パケットサイズは大きい方が検出率は高くなるが，正検出数に関しては 128byte ~ 256byte の間では全体的に大きな差はなく，256byte の時点から検出数が減少しているものは ExecutableCode を除き 3 日で 100 件以下のアラートのみであった．それらの通信はサンプリングが行われた場合，検出が難しいことから監視対象外とする．

ExecutableCode に関しては 256byte の時に約 15,000 件，128byte の時に約 10,000 件となっているが，256byte 取得する際のサンプリングレートは LAN-SW への負荷を考慮し，128byte で設定するサンプリングレートよりも低くする必要がある．そのため今後はサンプリングレートと取得するパケットサイズの均衡を考慮し設定の変更を行っていく必要がある．

また誤検出に関しては，パケット内の重要なコードの部分で切れた場合等では単純なデータを付加するだけでは対処できない．

4.6 実験 3. sFlow データの適用

実際に収集した sFlow データに Snort を適用させ，検出されたアラートを調査する．

4.6.1 実験方法

sFlowデータに対して実験2と同じPadding方式でパケットを修正し実験を行う．実験データは 2009 年 08 月 01 日～2010 年 07 月 31 日(152,266,950 パケット)に対して行うものと，実験 2 で行った期間と同じ 2010 年 09 月 03 日 ~ 2010 年 09 月 05 日(729,646 パケット)に対して行う．Snort で用いる検出ルールは 4.3.2 で修正を行ったルールである．

4.6.2 実験結果

表 11 に 2010/08/01 ~ 2010/07/31 の sFlow データに対して Snort を適用させた結果を示し，

表 12 に 2010/09/03 ~ 2010/09/05 の sFlow データに対して Snort を適用させた結果と比較用に実験 2 の結果を示す．

実験結果から次のことがわかった．

1. 1 年分のデータに対して実行した結果，様々な種類のアラートが検出された．現段階では誤検出も存在するが，多くのアラートを発することが可能なことから，sFlow と Snort を組み合わせても多くの不正通信を検出できると考えられる．
2. 多くのプリプロセッサがアラートを発していることから，サンプリングされたデータに対してもプリプロセッサは有効である．

表 11 実験結果: 2010/08/01/~2010/07/31

アラート	結果	アラート	結果
全プリプロセッサ	16,638(24)	全てのルール	440,078(134)
http_inspect	1,473(3)	Potentially Bad Traffic	2,069(02)
portscan	9,653(8)	Information Leak	375(07)
frag3	75(3)	P2P, Messenger	6,312(25)
ftp_telnet	750(8)	A Network Trojan	21,756 (18)
ssh	3,213(1)	SPYWARE-PUT	3,333(03)
bo	1(1)	Executable Code	5,458(08)
SQL version overflow	291,701(1)	その他(71)	109,074

表 12 実験結果: 2010/09/03/~2010/09/05

アラート	実験 2 (Raw)	実験 2 (128byte)	実験 3 (sFlow)
全アラート数 (種類)	256,566(61)	209,685(36)	1,636 (16)
SQL version overflow	87,707	82,640	1,294
Executable Code	48,526	9,831	17
User Privilege Gain	691	1	1
Information Leak	10	9	17
ssh	103,901	103,806	42
portscan	13,433	13,130	78
A Network Trojan	-	-	85 (6)
その他	-	-	105(3)

3. 2009 年 12 月に BitTorrent によるトラフィックが発生しており，sFlow データからその時の通信が Snort で検出された. このことから P2P 等の大量のパケットが発生する不正通信の検出は可能である.
4. 表 12 より実験 2 で検出されなかったアラートが検出された. このアラートはパケット 4.3.2 で変更したルールによって発生した誤検出である．A Network Trojan の classtype のアラートはバックドアが作成されている可能性を示唆するアラートで，ルールオプションで flowbits を利用している．通常のルールではマッチングするパケットが来た場合に flowbits で状態 A をセットし，状態 A がセットされていた場合にアラートを発するのだが，そのオプションを無効化したルールで運用を行ったため，こうした誤検出が見つかった.

4.6.3 考察

sFlow を利用した不正通信の検出では，DoS 攻撃や P2P といったパケットが大量に発生する不正通信の検出は容易であるが，非常に少ないパケットで行われる不正通信の検出は難しい．サンプリングレートの変更等を行って低帯域で行われる不正通信の検出を行うことが課題となる．また修正したルールでは flowbits を無効化した影響で検出される誤検出が非常に多いことからルールの修正を見直す必要がある.

5. 関連研究

奈良先端科学技術大学院大学の斎田佳輝氏らは NetFlow・sFlow によるネットワークトラフィック監視システムが提案した[6]. この研究ではヘッダ情報の修正により snort への適応を行っていた．本研究では実験結果より sFlow データへの適当なデータの付加により snort への適応を行っている．この方法によりヘッダ情報の修正だけでは検出できない不正通信の検出を可能とします．ただし，付加するデータによっては誤検出に繋がる可能性があることから，付加するデータで最適なものを調査する必要がある．また InMon 社からは sFlow パケットに Snort ルールを利用した簡易的な IDS の機能を持ったツール[7]が販売されているが，プリプロセッサ等による検出はできない.

6. おわりに

本論文では sFlow データを Snort に適用させるための方法を提示し，実データに対して適用を試みた．結果は Snort に対して有効なパケットデータに修正する方法として適当なデータを付加する方法が最も有効であることが分かった．また sFlow で取得するデータはサンプリングデータとなるためステートフルな解析が不可能なことから一部のルールを修正し実験を行った. 実験結果から sFlow データの Snort への適用は可能であり，sFlow データはパケット量が非常に少ないため Snort を運用するサーバへの負荷が小さく，高い検出率を実現できれば非常に有用であると考えられる．また sFlow データは長期の保存が容易であり，ネットワークの構成の変更やファイアウォールの更新後に，変更前と後で攻撃の様子を比較する際に有用である．ただし低帯域で行われる不正通信の検出は難しく，それらの通信に対応するためにはサンプリングレートの変更や取得するパケットサイズの変更等で収集する情報量を増やす必要があると考えられる.

現段階では付加するデータは単純なコードを付加するだけのため誤検出が存在し，ルールの修正は誤検出につながっている．今後の課題として誤検出を減らすために付加する適当なデータの探索とルールの変更を最小限に抑えセッション情報を利用するパケットの検出を行う方法を確立する必要がある.

参考文献

1 Snort, http://www.snort.org/
2 SNORT Users Manual 2.8.6 , April 26, 2010 http://www.snort.org/assets/140/snort_manual_2_8_6.pdf
3 TCPDUMP/LIBPCAP public repository http://www.tcpdump.org/
4 Peter Phaal, Sonia Panchen, and Neil McKee. RFC3176. InMon Corporation's sFlow: A Method for Monitoring Traffic in a Switched and Routed Networks.
5 sflowtool, http://www.inmon.com/
6 斎田佳輝, 新井イスマイル, 市川本浩, 藤川和利, 砂原秀樹, “sFlow によるネットワークトラフィック監視システムの提案”, 情報処理学会, マルチメディア, 分散, 協調とモバイル(DICOMO2005)シンポジウム論文集, pp.761-764, Jul.2005.
7 InMon TrafficSentinel, http://www.marubeni-sys.com/network/inmon/